7

Indoor Simultaneous Localization and Mapping

7.1 Introduction to Simultaneous Localization and Mapping

A map is a key element for indoor navigation. According to the *Cambridge Dictionary*, a map is "a drawing that gives you a particular type of information about a particular area." In other words, the composition of features associated with coordinate positions can be regarded as a map. Currently, most indoor mapping methods use simultaneous localization and mapping (SLAM) to build the first indoor map. This chapter will first introduce mapping methods using Light Detection and Ranging (LiDAR), which can provide decimeter-to-centimeter accuracy in depth measurements to generate accurate 3-D representations of indoor areas. However, the cost of LiDAR could hinder its applications in indoor navigation. In general, a camera is a low-cost option visually similar to the perception of our eyes. The second half of this chapter introduces visual SLAM for both monocular and depth cameras. The integration of LiDAR and camera is briefly discussed as a professional indoor mapping method that provides realistic 3-D indoor maps for augmented reality applications. Finally, the roles of IMU in SLAM are discussed to provide readers with the core benefits of this integration. A flowchart of a general SLAM, which includes odometry and mapping, is shown in Figure 7.1.

AU: The figures and callouts do not line up up. We fixed the captions and placed them where there were in original file, but I didn't change the callouts in the CE file.

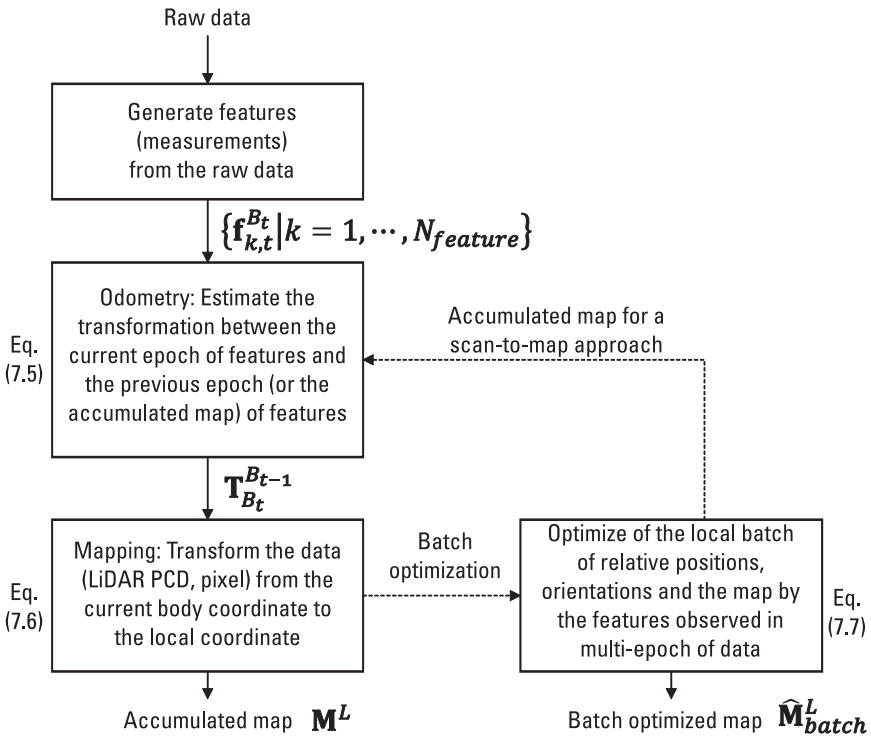

Figure 7.1 The generalized flowchart of SLAM.

7.1.1 The Historical Development of SLAM

SLAM is a computational problem that involves constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. SLAM is fundamental to autonomous systems navigating in previously unexplored environments.

The SLAM problem was first conceptualized in the 1980s, with significant early contributions coming from researchers such as Hugh Durrant-Whyte and John J. Leonard, who built on the work of Smith and Cheeseman [1]. The initial theoretical foundations were established using EKF, marking the beginning of what we now call the classical era of SLAM research.

The development of SLAM can be broadly categorized into several eras:

1. *Classical era (1986–2004):* This period focused on establishing the theoretical foundations of SLAM. The research was dominated by filter-based approaches, particularly EKF-SLAM [1] and particle

filters. A significant breakthrough came with FastSLAM [2], which used Rao-Blackwellized particle filters to efficiently solve SLAM by factoring the joint posterior into a product of a robot path posterior and landmark posteriors conditioned on the path.

2. *Algorithmic-analysis era (2004–2015):* During this period, researchers focused on understanding computational complexity, consistency, and convergence properties. Graph-based optimization methods emerged, and researchers began exploring how to handle large-scale environments and long-term operation. Key developments included Square Root SAM [3] and GraphSLAM [4].
3. *Robust-perception era (2015–present):* The current era is characterized by focusing on real-world robustness and adaptability. Modern SLAM systems are designed to work in dynamic, changing environments, with research emphasizing semantic understanding, object-level mapping, and deep learning integration. Key developments in this era include Oriented FAST and Rotated BRIEF (ORB)-SLAM [5], Large-Scale Direct Monocular (LSD)-SLAM [6], and LiDAR-based methods such as LiDAR Odometry and Mapping (LOAM) [7] and LiDAR Inertial Odometry-Smoothing and Mapping (LIO-SAM) [8].

{AU: Edits correct?}

Figure 7.2 provides a visual timeline of the evolution of SLAM technologies from 1990 to 2020. This figure illustrates the progression through three distinct eras, each represented by a different colored line. The classical era begins with EKF-SLAM in 1990 and extends to FastSLAM in 2002. The

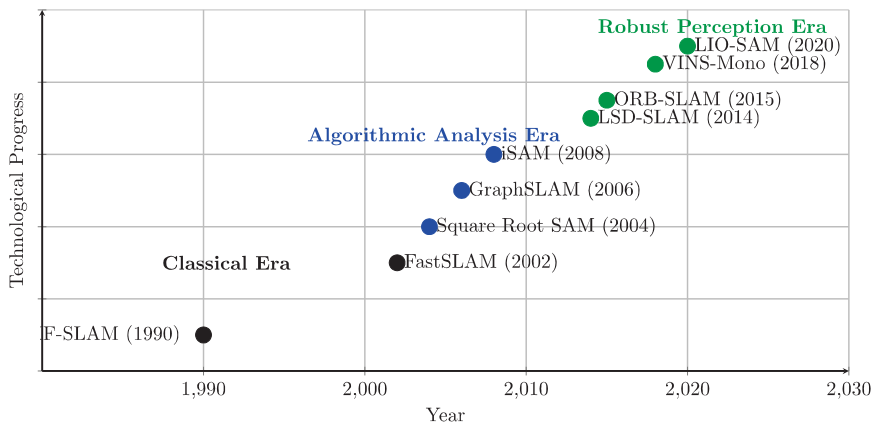

Figure 7.2 Evolution of SLAM technologies over time, showing key algorithms and their approximate relative impact.

{AU: Spell out all acronyms at first mention}

algorithmic-analysis era spans from Square Root SAM in 2004 to various developments including GraphSLAM and iSAM. The most recent robust perception era includes modern approaches like LSD-SLAM, ORB-SLAM, VINS-Mono, and LIO-SAM. The vertical axis represents technological progress, showing how each era has built upon the foundations laid by previous research while introducing new paradigms and capabilities.

7.1.2 SLAM Frameworks and Evolution

The evolution of SLAM frameworks closely follows the hardware and algorithmic advancements in robotics and computer vision. Here, we provide an overview of major SLAM frameworks that have shaped the field.

- *GraphSLAM (2006):* This approach formulated SLAM as a graph optimization problem. Poses and landmarks form the nodes of a graph, while measurements create edges (constraints) between nodes. The SLAM problem is then solved by finding the configuration of nodes that best satisfies all constraints [4]. GraphSLAM represented a significant paradigm shift from filter-based methods by recasting the entire SLAM problem into a sparse graph structure. By exploiting this sparsity, GraphSLAM achieves better computational efficiency for large-scale environments compared to earlier approaches. The method constructs information matrices where nonzero elements correspond to directly related poses and landmarks, allowing for efficient optimization using sparse linear algebra techniques. This graph-based representation also provides a natural framework for incorporating loop closures and handling uncertainty in a principled manner. GraphSLAM's formulation has become foundational for many modern SLAM systems, as it scales well to large environments and complex scenarios by focusing computational resources on the most informative constraints.
- *Parallel tracking and mapping (PTAM) (2007):* This introduced the idea of splitting tracking and mapping into separate threads, allowing real-time operation even with more computationally intensive mapping processes. PTAM was primarily designed for augmented reality applications but influenced many subsequent visual SLAM systems [9]. The key innovation of PTAM was recognizing that tracking camera motion could operate at a frame rate with minimal computational resources, while the more complex task of map building and refinement could run asynchronously at lower frequencies. This

parallel architecture enabled the use of computationally expensive bundle adjustment techniques in real-time applications for the first time. PTAM also introduced several practical innovations including a keyframe-based mapping approach, a map initialization procedure using a stereo initialization from a five-point algorithm, and an efficient relocalization system based on a modified version of randomized trees. By demonstrating that accurate camera tracking could be achieved in real time on consumer hardware, PTAM sparked a revolution in visual SLAM research and applications, particularly in the AR/VR domain where low latency is critical.

- *LSD-SLAM (2014):* Unlike feature-based methods, LSD-SLAM directly works with image intensities, creating semi-dense depth maps. It efficiently handles scale drift and loop closures, making it suitable for large-scale environments [6]. As a direct method, LSD-SLAM represented a significant departure from the feature-based paradigm by operating directly on image intensities rather than extracted features. This approach allows it to exploit more of the available image information, particularly in texture-poor regions where feature detection often fails. LSD-SLAM builds semi-dense depth maps by estimating depths only at pixels with sufficient image gradient, striking a balance between computational efficiency and information utilization. One of its major contributions is a novel scale-aware image alignment algorithm that addresses the inherent scale ambiguity in monocular vision. LSD-SLAM also incorporates a pose graph optimization framework for global consistency and loop closure, allowing it to create large-scale maps with reduced drift. By demonstrating that direct methods could operate efficiently at scale, LSD-SLAM inspired subsequent research in direct and semidirect SLAM approaches that seek to maximize the use of available visual information.
- *ORB-SLAM (2015):* This feature-based system uses ORB features for all SLAM tasks: tracking, mapping, relocalization, and loop closing. Its robust design has made it one of the most popular visual SLAM systems, with versions for monocular, stereo, and RGB-D cameras [5]. ORB-SLAM represents the culmination of years of research in feature-based SLAM, integrating best practices and novel solutions into a comprehensive framework. The system employs a carefully designed three-thread architecture separating tracking, local mapping, and loop closing tasks. Its use of ORB features provides a good balance of computational efficiency and descriptive power, enabling real-time

{AU: Spell out all acronyms at first mention}

performance on modest hardware while maintaining robust feature matching across different viewpoints and lighting conditions. ORB-SLAM implements an efficient keyframe-based bundle adjustment approach for local map optimization, combined with a pose graph optimization for global consistency. The system also incorporates an innovative map initialization method that automatically detects either planar or general scenes, adapting its initialization strategy accordingly. These capabilities, combined with careful engineering for robustness, have made ORB-SLAM a standard baseline in the field and the foundation for numerous subsequent systems.

- *LOAM (2014):* This algorithm optimizes for low-drift and real-time performance without using GPS or IMU data. It extracts plane and edge features from point clouds and uses them for fast scan-to-scan odometry estimation and scan-to-map matching [7]. LOAM introduced a novel approach to LiDAR SLAM by decoupling the odometry and mapping processes into two parallel algorithms running at different frequencies. The high-frequency odometry algorithm extracts edge and planar features from the point cloud and matches them to the previous scan to estimate motion, while the lower-frequency mapping algorithm registers these features to a global map for refinement and loop closure. This dual-rate approach enables LOAM to achieve both real-time performance and high accuracy. LOAM's feature extraction methodology is particularly innovative, using the distinctiveness of local surface geometry to classify points as edge or planar features, which reduces the computational burden of point cloud registration while maintaining sufficient constraints for accurate pose estimation. The system also incorporates distortion correction to account for sensor motion during scanning, which is crucial for accurate mapping during continuous movement. LOAM's exceptional performance in the KITTI odometry benchmark demonstrated that LiDAR-only SLAM could achieve accuracy comparable to or exceeding systems using additional sensors such as IMU or GPS, establishing it as a foundational approach for subsequent LiDAR SLAM research.
- *VINS-Mono (2018):* This is a robust monocular visual-inertial state estimator. It tightly couples visual and inertial measurements to handle challenging scenarios like dynamic environments and fast motion [10]. VINS-Mono represents a significant advancement in visual-inertial fusion, addressing the limitations of vision-only systems while maintaining computational efficiency. The system employs a sliding

{AU: Spell out all acronyms at first mention}

window-based optimization framework that tightly integrates visual features and pre-integrated IMU measurements to estimate the full state including position, velocity, orientation, and sensor biases. VINS-Mono incorporates a robust initialization procedure that can reliably estimate initial states without requiring specific motions or known patterns. One of its key innovations is the incorporation of a 4-DOF pose graph for loop closure that maintains the gravity direction constraint from IMU, enabling more accurate global consistency. The system also features online spatial and temporal calibration capabilities, automatically determining the transformation between camera and IMU as well as time offsets between sensor measurements. VINS-Mono's robust outlier rejection strategies and failure detection mechanisms make it particularly suitable for challenging real-world deployments where lighting conditions vary and dynamic objects are common. Its modular design and open-source implementation have made it a popular foundation for autonomous drone navigation, AR applications, and research in visual-inertial fusion.

- *LIO-SAM (2020):* This builds on LOAM by tightly coupling LiDAR and IMU data using factor graph optimization. It incorporates IMU pre-integration and efficient loop closure to achieve high accuracy with real-time performance [8]. LIO-SAM addresses several key limitations of previous LiDAR SLAM systems through its innovative factor graph formulation and sensor fusion approach. Unlike loosely coupled methods that use IMU only for initial state estimation, LIO-SAM fully integrates IMU constraints throughout the optimization process, enabling robust performance during rapid motion and featureless environments. The system employs a factor graph architecture with four types of factors: IMU pre-integration factors, odometry factors, GPS factors (when available), and loop closure factors. A key innovation is its dual-model design, maintaining two separate factor graphs: one for global mapping that preserves all constraints, and another for real-time odometry that is periodically reset to ensure computational efficiency. LIO-SAM also introduces scan context-based loop closure detection that leverages the 360° field of view of mechanical LiDARs for robust place recognition. The system's ability to incorporate GNSS factors when available allows for georeferenced mapping, while still maintaining high accuracy in GNSS-denied environments. By achieving both high accuracy and real-time performance on modest hardware, LIO-SAM has enabled new applications in autonomous navigation,

particularly for robots operating in complex 3-D environments where robust localization is essential.

The progression from early filter-based approaches to modern optimization-based methods has been driven by several factors:

- *Computational resources:* Modern computers allow for more complex algorithms and real-time processing of dense sensor data. The exponential growth in computing power, particularly in embedded systems and mobile processors, has enabled the deployment of sophisticated SLAM algorithms. The development of specialized hardware accelerators for visual processing and machine learning tasks has further expanded the capabilities of real-time SLAM systems.
- *Sensor technology:* The development of affordable, high-quality sensors such as RGB-D cameras and 3-D LiDARs has enabled more accurate environment perception. The dramatic reduction in cost and size of 3-D sensing technologies has been crucial for widespread SLAM adoption. Solid-state LiDARs have made high-precision depth sensing more accessible, while the mass production of RGB-D cameras initially driven by the gaming industry has provided affordable depth sensing for indoor applications. Time-of-flight cameras have improved depth accuracy in close-range applications, and event-based cameras with microsecond temporal resolution have opened new possibilities for high-speed motion tracking. Additionally, improvements in MEMS-based inertial sensors have made high-quality IMUs available at consumer price points, facilitating visual-inertial fusion approaches that enhance robustness and accuracy.
- *Algorithmic innovations:* New optimization techniques, feature extraction methods, and loop closure strategies have improved SLAM robustness and accuracy. Mathematical breakthroughs in sparse optimization, manifold representation of rotations, and factor graph formulations have enabled more efficient and accurate state estimation. Learned feature descriptors leveraging deep neural networks have significantly improved feature-matching robustness across varying conditions. Semantic SLAM approaches that incorporate object recognition and scene understanding are bridging the gap between geometric mapping and higher-level environmental representation. Novel outlier rejection techniques and robust estimation methods have enhanced performance in dynamic environments with moving

objects. The incorporation of probabilistic approaches for handling uncertainty continues to improve reliability in challenging conditions, while recent advances in differentiable programming enable end-to-end optimization of entire SLAM pipelines.

- *Multisensor fusion:* Modern SLAM systems often integrate data from multiple sensors (e.g., camera, LiDAR, IMU) to overcome the limitations of individual sensors. Complementary sensor characteristics can be leveraged to enhance robustness; cameras provide rich texture and color information, LiDARs offer precise depth measurements regardless of lighting, and IMUs provide high-frequency motion estimates resistant to visual degradation. Sophisticated fusion frameworks using factor graphs, Kalman filter variants, or deep learning approaches can optimally combine these heterogeneous data sources. Tight coupling of sensors at the measurement level, rather than simply fusing outputs, has proven particularly effective for handling challenging scenarios such as rapid motion or feature-poor environments. Recent research has explored adaptive fusion strategies that dynamically adjust sensor weighting based on environmental conditions and sensor health, further improving reliability across diverse operating conditions. The integration of additional sensors such as wheel encoders, barometers, magnetometers, and GNSS receivers provides even more constraints for robust state estimation in specific application domains.

7.1.3 Comparison of Popular SLAM Implementations

Different SLAM implementations have their strengths and weaknesses, making them suitable for different applications. Table 7.1 provides a comparison of some popular SLAM frameworks based on various aspects.

Table 7.2 compares the pipeline components of LiDAR-based and vision-based SLAM systems.

Currently, the evolution of SLAM continues with active research in several directions:

1. *Semantic SLAM:* Incorporating object-level and semantic understanding into SLAM systems [11].
2. *Deep learning integration:* Using neural networks for feature extraction, depth estimation, and loop closure detection [12].
3. *Dynamic environment handling:* Developing methods to handle moving objects and changing environments [13].

Table 7.1
Basic Comparison of Popular SLAM Implementations

Aspect	ORB-SLAM	LSD-SLAM	LOAM	VINS-Mono	LIO-SAM
Sensor type	Camera (Mono/ Stereo/ RGB-D)	Monocular camera	LiDAR	Monocular camera + IMU	LiDAR + IMU
Feature type	Sparse features (ORB)	Semi-dense depth maps	Edge and planar features	Visual features (optical flow)	Edge and planar features
Map representation	Sparse point cloud	Semi-dense depth maps	Point cloud	Sparse point cloud	Point cloud
Localization accuracy	High (with loop closure)	Medium	High	Very high	Very high
Computational cost	Medium	Medium-high	Medium-high	High	Medium-high
Loop closure	Yes	Yes	No (original version)	Yes	Yes
Scale recovery	No (monocular)	No (monocular)	Yes	Yes (using IMU)	Yes
Real-time performance	Yes	Yes (with GPU)	Yes	Yes	Yes
Environment types	Well-textured	Well-textured	Structured environments	Various	Various
Major advantages	Accurate, robust loop closure	Works in large environments	Good for unmanned vehicles	Robust to dynamic objects	High accuracy, leverages IMU
Major limitations	Fails in textureless areas	Requires GPU for real-time	Motion distortion issues	Initialization challenges	Sensor synchronization requirements
Reference	[5]	[6]	[7]	[10]	[8]

4. *Resource-constrained SLAM:* Creating efficient algorithms for mobile and embedded devices [14].
5. *Event-based SLAM:* Leveraging neuromorphic cameras (event cameras) for high-speed, high-dynamic range visual SLAM [15].

Table 7.2
Comparison of LiDAR-Based and Vision-Based SLAM Pipelines

Pipeline Component	LiDAR SLAM	Visual SLAM
Data preprocessing	Point cloud downsampling Motion distortion correction Ground/outlier removal	Image rectification Lens distortion correction Exposure adjustment
Feature extraction	Edge features Planar features Corner points Normal distributions (NDT)	Corner features (FAST, Harris) Blob features (SIFT, SURF, ORB) Direct methods (photometric error)
Data association	Nearest neighbor matching ICP matching Feature-to-feature matching	Feature descriptor matching Optical flow tracking Epipolar constraints
Motion estimation	Point-to-point registration Point-to-plane registration NDT registration	Essential/fundamental matrix Perspective-n-Point (PnP) Homography (for planar scenes)
Back-end optimization	Factor graph optimization Pose graph optimization Bundle adjustment (with IMU)	Bundle adjustment Pose graph optimization Key frame-based optimization
Loop closure	ICP-based scan matching Place recognition using point cloud Scan context matching	Bag of Words (BoW) DBoW2/DBoW3 Visual place recognition
Map representation	Point cloud map Surfel map Occupancy grid/voxel grid	Sparse point cloud Semi-dense depth maps Dense volumetric (TSDF)
Multisensor fusion	LiDAR-IMU tight coupling LiDAR-visual integration GNSS integration	Visual-inertial fusion Multicamera systems Visual-GNSS integration

These advancements are pushing SLAM systems toward greater autonomy, robustness, and applicability across diverse scenarios, from autonomous driving to AR/VR applications.

7.2 General Mathematical Model of SLAM

First, the features (which are the measurements used for the coming estimation, denoted by **f**) are extracted from the raw data. If this is the first epoch, the device's (i.e., camera and LiDAR) body local coordinate is used as the local map coordinate for the upcoming estimation, denoted by B in the superscript. Note that the features extracted at the first epoch are used as the initial map. The movement of an agent can be estimated by the transformation between features from adjacent epochs, incorporating a rotation $\mathbf{C}$ and a translation $\Delta\mathbf{x}$ in 3-D space (from the subscript to the superscript coordinate systems). The feature transformation can be represented by the coordinate transformation, using

$$\mathbf{f}_{k,t}^{B_{t-1}} = \mathbf{C}_{B_t}^{B_{t-1}}\mathbf{f}_{k,t}^{B_t} + \Delta\mathbf{x}_{B_t}^{B_{t-1}} \tag{7.1}$$

where the superscripts B_t and B_{t-1} denote corresponding to the local body coordinate system at the epochs t and $t-1$, respectively. For multiple epochs, the transformation is usually described by matrix multiplication in a compact manner, as follows:

$$\begin{bmatrix} \mathbf{f}_{k,t}^{B_{t-1}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{B_t}^{B_{t-1}} & \Delta\mathbf{x}_{B_t}^{B_{t-1}} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{f}_{k,t}^{B_t} \\ 1 \end{bmatrix} \tag{7.2}$$

For convenience, the transformation is simply represented by the following multiplication, where the feature vector will be automatically extended with a row of 1 at the bottom during the multiplication.

$$\mathbf{f}_{k,t}^{B_{t-1}} = \mathbf{T}_{B_t}^{B_{t-1}}\mathbf{f}_{k,t}^{B_t} \tag{7.3}$$

$$\mathbf{T}_{B_t}^{B_{t-1}} = \begin{bmatrix} \mathbf{C}_{B_t}^{B_{t-1}} & \Delta\mathbf{x}_{B_t}^{B_{t-1}} \\ 0 & 1 \end{bmatrix} \tag{7.4}$$

where $\mathbf{T}$ is the transformation matrix between the previous and current epoch of features is estimated, also known as a pose in robotics field. It is then used to transform a kth feature from the current epoch of the device's body coordinate, $\mathbf{f}_{k,t}^{B_t}$, back to the kth feature of the previous epoch of device's body coordinate, $\mathbf{f}_{k,t}^{B_{t-1}}$. Thus, the odometry can be obtained by optimizing $\mathbf{T}_{B_t}^{B_{t-1}}$ to align the features from the previous epoch with those from the current one after the transformation, as below.

$$\hat{\mathbf{T}}_{B_t}^{B_{t-1}} = \arg\min_{\mathbf{T}_{B_t}^{B_{t-1}}} \sum_{k=1}^{N_{\text{feature}}} \left\| \mathbf{f}_{k,t-1}^{B_{t-1}} - \mathbf{T}_{B_t}^{B_{t-1}} \mathbf{f}_{k,t}^{B_t} \right\|^2 \tag{7.5}$$

For the mapping stage, the features at different epochs are all transformed to the local map coordinate to form the map $\mathbf{M}^L$ as below.

$$\mathbf{M}^L = \left\{ \mathbf{M}_t^L | \forall t \right\} \tag{7.6}$$

with $\mathbf{M}_t^L = \left\{ \mathbf{f}_{k,t}^L | k = 1, \cdots, N_{\text{feature}} \right\}$ and $\mathbf{f}_{k,t}^L = \mathbf{T}_{B_0}^L \left(\prod_1^t \mathbf{T}_{B_t}^{B_{t-1}} \right) \mathbf{f}_{k,t}^{B_t}$.

Then both odometry and mapping are completed, meaning the SLAM is achieved. SLAM will make use of the fact that all the features are mutually observed in multi-epoch of the data. It means that SLAM can further apply a batch optimization considering the co-visibility in a window containing several scans. The estimated transformation matrix and the map earlier are used as initial guesses for this batch optimization. The transformations in a window are batch optimized as follows.

$$\begin{gathered} \hat{\mathbf{T}}_{\text{window}} = \arg\min_{\mathbf{T}_{\text{window}}} \sum_{t=i}^{i+w} \sum_{k=1}^{N_{\text{feature}}} \left\| \mathbf{M}_{\text{window}}^L - \mathbf{T}_{B_0}^L \left(\prod_1^t \mathbf{T}_{B_t}^{B_{t-1}} \right) \mathbf{f}_{k,t}^{B_t} \right\|^2 \\ \mathbf{T}_{\text{window}} = \left\{ \mathbf{T}_{B_t}^{B_{t-1}} | \forall t \in [i, i+w] \right\} \end{gathered} \tag{7.7}$$

where i and $i + w$ denote the index of the first scan and the last scan in the window, respectively. $\mathbf{M}_{\text{window}}^L$ denotes the window map defined as $\{\mathbf{M}_t^L | \forall t \in [i, i + w]\}$. It is worth mentioning that for the constraint construction of the certain pose $\mathbf{T}_{B_t}^{B_{t-1}}$, the map points contributed by the corresponding scan $\mathbf{M}_t^L$ should be excluded from $\mathbf{M}_{\text{window}}^L$ during the match.

Finally, the batch optimized map can be obtained as $\hat{\mathbf{M}}_{\text{batch}}^L = \left\{ \hat{\mathbf{M}}_t^L | \forall t \right\}$. $\hat{\mathbf{M}}_t^L$ is registered by the batch optimized transformation from (7.7). It is different from (7.6) where the map is accumulated by the pose from odometry (7.5).

The maps built by these methods are mostly in a local coordinate. The connection between indoor maps to outdoor maps requires the coordination transformation from the local map coordinate to the world geodetic coordinate. If the transformation between the world coordinate and the local map coordinate is available (which is usually given by a GNSS receiver), the SLAM map can be transformed to the world coordinate to provide absolute map information.

$$\mathbf{M}^W = \mathbf{T}_L^W \mathbf{M}^L \tag{7.8}$$

Table 7.3 provides an overview of the relationship between measurements, model, and the state to be estimated. Table 7.3 focuses on LiDAR and visual odometry, which are the primary methods discussed in this chapter. The formulas listed in Table 7.3 will be detailed in the following sections.

7.3 LiDAR SLAM

A LiDAR sensor transmits pulsed light wave from a channel and receives a reflected wave that hits an object. There are two types of popular LiDAR: mechanical LiDAR, and solid-state LiDAR. In general, the former one has a sparser point cloud data (PCD) but wider coverage of field of view, while the latter one has a dense PCD but narrower coverage of field of view. The distance between the LiDAR and object can therefore be determined by a two-way TOA ranging method as shown in the left of Figure 7.4. If there are multiple channels, it is important to distinguish the reflection is coming from a specific channel, resulting in a sequence of binary codes (pulses) that has a characteristic of a low cross-correlation is usually used. Since it requires a few pulses to for the correlation, the LiDAR ranger usually has a requirement on the minimum distance between the sensor and an object. For a mechanical 3-D LiDAR, a servomotor will rotate all the channels in 360° to receive the reflections from its surrounding environment in 360°, forming a point cloud as shown in the right of Figure 7.4. The point cloud is to describe as the distances between all the objects (which cause the reflections) and the LiDAR sensor and it is defined as follows.

$$\mathbf{P}_t = \begin{bmatrix} \mathbf{p}_1 & \cdots & \mathbf{p}_i \end{bmatrix} = \begin{bmatrix} x_1 & \cdots & x_i \\ y_1 & \cdots & y_i \\ z_1 & \cdots & z_i \end{bmatrix}, i \in \text{the received reflections} \quad (7.9)$$

Table 7.3
Comparison of LiDAR and Visual Odometry in Terms of Measurement, Model, and State

Methods	Measurement	Model	State
LiDAR	$\mathbf{p}_{i,t}^{B_t}$	$\mathbf{p}_{j,t-1}^{B_{t-1}} = \mathbf{C}_{B_t}^{B_{t-1}} \mathbf{p}_{i,t}^{B_t} + \Delta \mathbf{x}_{B_t}^{B_{t-1}}$	$\mathbf{T}_{B_t}^{B_{t-1}} = \begin{bmatrix} \mathbf{C}_{B_t}^{B_{t-1}} & \Delta \mathbf{x}_{B_t}^{B_{t-1}} \\ \mathbf{0} & 1 \end{bmatrix}$
Monocular camera	$\mathbf{p}_{k,t+\Delta t}^{I}$ $\mathbf{p}_{k,t}^{I}$	$\left(\mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{I}\right)^T \mathbf{t}^{\wedge} \mathbf{R} \mathbf{K}^{-1} \mathbf{p}_{k,t}^{I} = 0$	$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ and depth of features

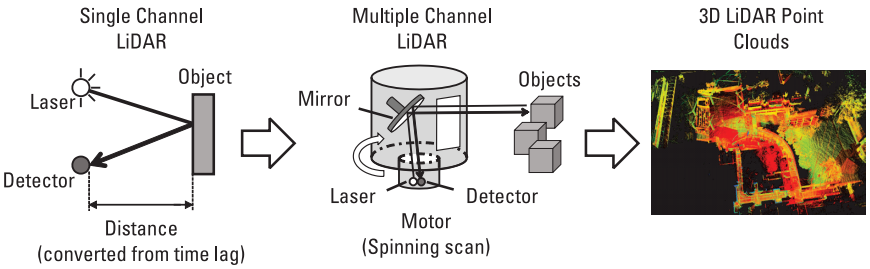

Figure 7.3 Mechanical LiDAR sensing principle.

where the subscript i denotes the index of the point scanned in epoch t and (x_i, y_i, z_i) denotes the location of a point in the local body coordinate of the LiDAR at epoch t. The point cloud, $\mathbf{P}_t$, is therefore the measurement of LiDAR. The model is the surrounding environment, as shown in Figure 7.5. In the following section, a conventional point-based and two state-of-the-art feature-based LiDAR models are introduced. The challenges of LiDAR SLAM are also discussed [16].

7.3.1 Point Cloud-Based LiDAR SLAM: ICP

The most well-known conventional method is iterative closest point (ICP). It minimizes the point-to-point distance between two scans of LiDAR PCD. The previous scan can be regarded as a model of surrounding environment and the current scan is regarded as the measurement. ICP can estimate the transformation matrix between the two scans $\mathbf{T}_{B_t}^{B_{t-1}}$ when the minimum distance is achieved by iterations. It is described as follows.

$$\hat{\mathbf{T}}_{B_t}^{B_{t-1}} = \arg\min_{\mathbf{T}_{B_t}^{B_{t-1}}} \sum_{i=1}^{N_{\text{model}}} \sum_{j=1}^{N_{\text{measurement}}} b_{i,j} \left\| \mathbf{p}_{i,t-1}^{B_{t-1}} - \mathbf{T}_{B_t}^{B_{t-1}} \mathbf{p}_{j,t}^{B_t} \right\|^2 \tag{7.10}$$

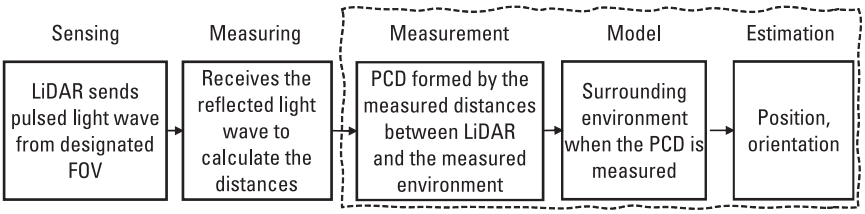

Figure 7.4 The measurements, models, and estimation of using LiDAR point cloud data for odometry.

where N_{model} and $N_{\text{measurement}}$ denote the number of points in the previous and current scans, respectively. $\mathbf{p}_{i,t-1}^{B_{t-1}}$ and $\mathbf{p}_{j,t}^{B_t}$ are the points in the previous and current scans, respectively. $b_{i,j}$ is a binary selector of the paired points, meaning that, if $\mathbf{p}_{i,t-1}^{B_{t-1}}$ and $\mathbf{p}_{j,t}^{B_t}$ is a pair, then $b_{i,j} = 1$ and otherwise $b_{i,j} = 0$ To determine whether the two points in the two scans is a pair, generally speaking, a threshold, $d_{\text{threshold}}$, of the distance of the two points is used to make the judgment.

$$b_{i,j} = \begin{cases} 0, \text{ if } \left\| \mathbf{p}_{i,t-1}^{B_{t-1}} - \left(\mathbf{T}_{B_t}^{B_{t-1}} \right)_0 \mathbf{p}_{j,t}^{B_t} \right\|_2 > d_{\text{threshold}} \\ 1, \text{ otherwise} \end{cases} \tag{7.11}$$

where the subscript 0 for $\left(\mathbf{T}_{B_t}^{B_{t-1}}\right)$ denotes initial guesses. The value of the threshold is based on applications, for example, 1m is used for the application of LiDAR in an indoor parking lot. For a high (depending on the velocity of the LiDAR) scan rate of LiDAR, the null initial guess is fine. One way to obtain a better initial guess is to apply a constant translational and rotational velocities assumption to predict the transformation matrix of the next epoch of time to solve (7.10), a nonlinear optimizer solver as mentioned Section 2.4. To reduce the computational load, the rotation matrix can be solved first by singular value decomposition (SVD), and then the translation matrix can be solved by substituting $\hat{\mathbf{C}}_{B_t}^{B_{t-1}}$ into $\Delta\hat{\mathbf{x}}_{B_t}^{B_{t-1}} = \overline{\mathbf{p}}_{t-1}^{B_{t-1}} - \left(\hat{\mathbf{C}}_{B_t}^{B_{t-1}} \overline{\mathbf{p}}_t^{B_t}\right)$ [17]. The $\overline{\mathbf{p}}_{t-1}^{B_{t-1}}$ and $\overline{\mathbf{p}}_t^{B_t}$ denotes the geometric center of the points in the previous scan and the current scan, respectively. As a scan of LiDAR PCD can easily contain more than 10,000 points using a typical mechanical 3-D LiDAR, the computational load of the conventional ICP is high, meaning it could not be used in the applications that have limited computing resources. A straightforward idea to maintain the SLAM performance but to reduce the computational load is to maintain the observability of the measurement model but to reduce the number of points used. A voxel grid filter can be used to sampling the dense raw points [18]. Another solution is to exploit the representative features from the dense raw points where only partial of the distinct features are used for the point registration.

{AU: Is this the correct section number?}

Finally, either a direct assembly of the PCD, for example, (7.6), or a batch estimation considering all the PCD, for example, (7.7), can be used to obtain the accumulated PCD map $\mathbf{M} = \{\mathbf{p}_1^L \cdots \mathbf{p}_{N\text{point}}^L\}$ where N_{point} denotes the total number of the registered points.

To maintain the observability of the measurement model of the LiDAR SLAM, an idea is to extract "features" from the PCD. Two popular methods

are: (1) to represent the raw points inside a voxel by a normal distribution, and (2) to extract 2-D planes and 1-D edges from the PCD.

7.3.2 Feature-Based LiDAR SLAM: NDT

The normal distribution transform (NDT) method represents the points inside a voxel as a 3-D normal distribution as shown in Figure 7.6 [19]. In other words, it describes the probability of finding a point in a certain position by a set of normal distributions [20].

For kth voxel, the average $\overline{\mathbf{p}}_{k,t-1}^{B_{t-1}}$ and covariance matrix $\mathbf{\Sigma}_{k,t-1}^{B_{t-1}}$ of the points inside the voxel are calculated:

$$\overline{\mathbf{p}}_{k,t-1}^{B_{t-1}} = \frac{1}{n_k}\sum_{i=1}^{n_k}\mathbf{p}_{i,t-1}^{B_{t-1}} \tag{7.12}$$

$$\mathbf{\Sigma}_{k,t-1}^{B_{t-1}} = \frac{1}{n_k - 1}\sum_{i=1}^{n_k}\left(\mathbf{p}_{i,t-1}^{B_{t-1}} - \overline{\mathbf{p}}_{k,t-1}^{B_{t-1}}\right)\left(\mathbf{p}_{i,t-1}^{B_{t-1}} - \overline{\mathbf{p}}_{k,t-1}^{B_{t-1}}\right)^T \tag{7.13}$$

where n_k denotes the total number of points inside the kth voxel. To apply these normal distribution features in SLAM, the transformation matrix $\mathbf{T}_{B_t}^{B_{t-1}}$ is associated with the NDT measurement model. Comparing to ICP, the NDT method models the surrounding environment by voxels that represented by normal distributions. If the points in the current scan can achieve a maximum joint probability with a given transformation matrix $\hat{\mathbf{T}}_{B_t}^{B_{t-1}}$, then the given $\hat{\mathbf{T}}_{B_t}^{B_{t-1}}$ is solution with the maximum likelihood. The likelihood (which can be regarded as a score) of voxel k referring to $\mathbf{T}_{B_t}^{B_{t-1}}$ can be described as:

$$\text{likelihood}_k\left(\mathbf{T}_{B_t}^{B_{t-1}}\right) = \prod_{j=1}^{N_{\text{measurement}}}\exp\left(-\frac{\left\|\mathbf{T}_{B_t}^{B_{t-1}}\mathbf{p}_{j,t}^{B_t} - \overline{\mathbf{p}}_{k,t-1}^{B_{t-1}}\right\|_{\mathbf{\Sigma}_{k,t-1}^{B_{t-1}}}^2}{2}\right) \tag{7.14}$$

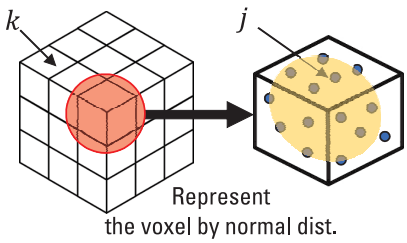

Figure 7.5 Illustration of representing points as normal distributions.

where $\|\mathbf{r}\|_{\Sigma}^{2}$ denotes $\mathbf{r}^{T}\Sigma^{-1}\mathbf{r}$. Considering all the voxel, the joint likelihood becomes:

$$\text{likelihood}\left(\mathbf{T}_{B_t}^{B_{t-1}}\right)=\prod_{k=1}^{N_{\text{voxel}}}\prod_{j=1}^{N_{\text{measurement}}}\exp\left(-\frac{\left\|\mathbf{T}_{B_t}^{B_{t-1}}\mathbf{p}_{j,t}^{B_t}-\overline{\mathbf{p}}_{k,t-1}^{B_{t-1}}\right\|_{\Sigma_{k,t-1}^{B_{t-1}}}^{2}}{2}\right) \tag{7.15}$$

Since the likelihood is Gaussian distributed, the MLE of (7.15) can be estimated by a nonlinear LS estimation.

$$\hat{\mathbf{T}}_{B_t}^{B_{t-1}}=\arg\min_{\mathbf{T}_{B_t}^{B_{t-1}}}\frac{1}{2}\sum_{k=1}^{N_{\text{voxel}}}\sum_{j=1}^{N_{\text{measurement}}}\left\|\mathbf{T}_{B_t}^{B_{t-1}}\mathbf{p}_{j,t}^{B_t}-\overline{\mathbf{p}}_{k,t-1}^{B_{t-1}}\right\|_{\Sigma_{k,t-1}^{B_{t-1}}}^{2} \tag{7.16}$$

The NDT method divides the surrounding environment into fix-sized cells. This may lead to discontinuities in the model representation since some voxel may contain very little points. The insufficient constraints could be problematic in calculating the Jacobian matrix of the NDT measurement model [21]. A method that combines the ideas of ICP and NDT is proposed and called generalized ICP [22]. It first associates a group of points between the previous and current scans and then models the group of points by normal distributions and finally does the distribution-to-distribution matching.

7.3.3 Feature-Based LiDAR SLAM—LOAM: Plane and Edge

A state-of-the-art LiDAR method is LOAM [7]. Its aims are: (1) to maintain a low-drift of the mapped PCD map, and (2) to reduce the computational load of SLAM. In the previously mentioned ICP and NDT, the two SLAMs are matching the previous and current scans (i.e., a scan-to-scan matching). These scan-to-scan matchings only consider the points in the vicinity of the two scans, meaning that the points (features) at far are not really used in the calculation of the transformation; see (7.10) and (7.16). This results in drifting to the conventional ICP and NDT maps. LOAM proposed a scan-to-map approach (refer to the dashed line in the upper right of Figure 7.1). The map means a locally registered map during the SLAM process. This local map is the aggregated PCD of all previous scans so that it also considers the points (features) at far. This scan-to-map approach can effectively reduce the drift. However, this approach drastically increases the computational load; hence, the LOAM has to reduce the computational load at the same time. To achieve this aim, LOAM extracts representative edge and planar features from PCD to reduce the redundancy of the dense point clouds while maintaining the

{AU: "at far" correct?; please reword for clarity}

{AU: Please reword "at far" for clarity}

model observability. The extraction accords to the curvature of a few segmented points in a current scan $\mathbf{P}_t$. A point is selected as an edge point if its curvature value is larger than a predetermined threshold, or classified as a planar point by a smaller curvature, as Figure 7.7 shows.

To further reduce the computation load of the SLAM, the lower update rate of the scan-to-map approach is desired. This reduction of the update rate results in another challenge; a more stringent requirement on the quality of the initial guess comparing it to a scan-to-scan approach. In LOAM, a scan-to-scan odometry is used to provide that initial guess for the later scan-to-map matching. Thus, it is a two-step approach. Equation (7.17) describes the rough transformation estimated by the odometry.

$$\hat{\mathbf{T}}_{\text{odom}} = \arg\min_{\mathbf{T}_{\text{odom}}} \frac{1}{2}\left\{ \sum_{k=1}^{N^{\text{edge}}} \left\| \omega_{(k)} f\left(\mathbf{p}_{(k,t)}^{\text{edge}}, \mathbf{T}_{\text{odom}}\right) \right\|^2 + \sum_{k=1}^{N^{\text{plane}}} \left\| \omega_{(k)} f\left(\mathbf{p}_{(k,t)}^{\text{plane}}, \mathbf{T}_{\text{odom}}\right) \right\|^2 \right\} \tag{7.17}$$

Figure 7.6 Illustration of extracted edge (black points) and planar points (gray boxes) for scan-to-map concerning a frame of LiDAR point clouds (gray points).

where $\omega_{(k)}$ denotes the bisquare weight, setting as $1 - 1.8 \times f(\mathbf{p}_{(k,t)}, \mathbf{T}_{\text{odom}})$ [7]. $f\left(\mathbf{p}^{\text{edge}}_{(k,t)}, \mathbf{T}_{\text{odom}}\right)$ denotes a point-to-line distance, which is calculated as follows:

$$f\left(\mathbf{p}^{\text{edge}}_{(k,t)}, \mathbf{T}_{\text{odom}}\right) = \frac{\left\| \left(\mathbf{T}_{\text{odom}}\mathbf{p}^{\text{edge}}_{(k,t)} - \mathbf{p}^{\text{edge}}_{(j,t-1)}\right) \times \left(\mathbf{T}_{\text{odom}}\mathbf{p}^{\text{edge}}_{(k,t)} - \mathbf{p}^{\text{edge}}_{(l,t-1)}\right) \right\|_2}{\left\| \mathbf{p}^{\text{edge}}_{(j,t-1)} - \mathbf{p}^{edge}_{(l,t-1)} \right\|_2} \tag{7.18}$$

where $\mathbf{p}^{\text{edge}}_{(k,t)}$ denotes the kth edge feature points on the current scan. $\mathbf{p}^{\text{edge}}_{(j,t-1)}$ and $\mathbf{p}^{\text{edge}}_{(l,t-1)}$ denote two closest edge features of $\mathbf{p}^{\text{edge}}_{(k,t)}$ in the previous scan that are assumed to be on the corresponding line segment of $\mathbf{p}^{\text{edge}}_{(k,t)}$. As shown in Figure 7.8, $\mathbf{p}^{\text{edge}}_{(j,t-1)}$ and $\mathbf{p}^{\text{edge}}_{(l,t-1)}$ are selected from adjacent rings so that vertical lines orthogonal to the laser beam are exploited. Such lines are free from motion distortion compared with that parallel to the laser beam.

In (7.19), $f\left(\mathbf{p}^{\text{plane}}_{(k,t)}, \mathbf{T}_{\text{odom}}\right)$ is formulated as the point-to-plane distance for planar feature points as follows:

$$f\left(\mathbf{p}^{\text{plane}}_{(k,t)}, \mathbf{T}_{\text{odom}}\right) = \frac{\left\| \begin{array}{c} \left(\mathbf{T}_{\text{odom}}\mathbf{p}^{\text{plane}}_{(k,t)} - \mathbf{p}^{\text{plane}}_{(j,t-1)}\right) \\ \left(\left(\mathbf{p}^{\text{plane}}_{(j,t-1)} - \mathbf{p}^{\text{plane}}_{(l,t-1)}\right) \times \left(\mathbf{p}^{\text{plane}}_{(j,t-1)} - \mathbf{p}^{\text{plane}}_{(m,t-1)}\right)\right) \end{array} \right\|_2}{\left\| \left(\mathbf{p}^{\text{plane}}_{(j,t-1)} - \mathbf{p}^{\text{plane}}_{(l,t-1)}\right) \times \left(\mathbf{p}^{\text{plane}}_{(j,t-1)} - \mathbf{p}^{\text{plane}}_{(m,t-1)}\right) \right\|_2} \tag{7.19}$$

where $\mathbf{p}^{\text{plane}}_{(k,t)}$ denotes the kth planar feature points on the current scan. $\mathbf{p}^{\text{plane}}_{(j,t-1)}$, $\mathbf{p}^{\text{plane}}_{(l,t-1)}$, and $\mathbf{p}^{\text{plane}}_{(m,t-1)}$ denotes the three closest planar feature points of $\mathbf{p}^{\text{plane}}_{(k,t)}$ in the previous scan which are assumed be on the corresponding planar patch of $\mathbf{p}^{\text{plane}}_{(k,t)}$. As shown in Figure 7.8, $\mathbf{p}^{\text{plane}}_{(j,t-1)}$ and $\mathbf{p}^{\text{plane}}_{(l,t-1)}$ are selected from the same

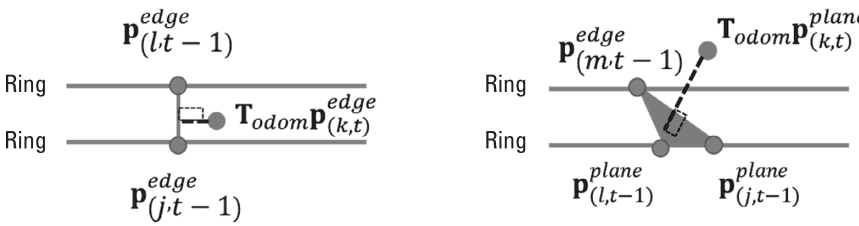

Figure 7.7 Illustration of point-to-line distance (left) and point-to-plane (right) distance for scan-to-scan odometry.

ring while $\mathbf{p}_{(m,t-1)}^{\text{plane}}$ are selected from an adjacent ring. Consequently, the three points are not colinear to form a planar patch.

Then the rough transformation $\hat{\mathbf{T}}_{\text{odometry}}$ is used to estimate a refined transformation by the scan-to-map approach. The objective function is formulated as (7.17). However, the point-to-line (plane) distance is that between $\mathbf{p}_{(k,t)}^{\text{edge}}$ $\left(\mathbf{p}_{(k,t)}^{\text{plane}}\right)$ and the line (plane) fitted by its closest points selected from the map rather the previous scan.

Finally, the refined transformation will be used in the batch optimization as mentioned in (7.7).

7.3.4 Challenges of LiDAR SLAM

For IPIN applications, adverse weather conditions are not a problem for LiDAR SLAM. The two major challenges are motion distortion and the presence of dynamic objects.

7.3.4.1 Motion Distortion

Ideally speaking, LiDAR SLAM assumes that all the points are scanned at the same location of the ego-vehicle. In reality, the movement of the ego-vehicle while the LiDAR is scanning will cause a motion distortion to the scanned LiDAR PCD. This phenomenon is illustrated in Figures 7.9 and 7.10. As can be seen when LiDAR stays static as the blue dot in the left part of Figure 7.9, the scanning pattern remains as a circle. While when LiDAR moves from the blue dot to the green dot as in the right part of Figure 7.9, the scanning pattern is distorted. This is a factor resulting in the drift of accumulated LiDAR PCD map. This distortion can be compensated for if the rotation rate and acceleration of the ego-vehicle are roughly known. A popular method is to integrate with IMU to provide such measurements to compensate the distortion [23]. The poses of the LiDAR during the scan sweep are predicted. As shown in the following equation, each point is transformed to the starting moment of the scan so that the motion distortion is compensated for:

$$\tilde{\mathbf{p}}_i = \mathbf{T}_i \mathbf{p}_i \tag{7.20}$$

where $\mathbf{p}_i$ denotes the ith point produced by the LiDAR during a scan sweep at the LiDAR body frame. $\mathbf{T}_i$ is the LiDAR pose currently related to scan starting time. $\tilde{\mathbf{p}}_i$ denotes the point with distortion compensation. $\mathbf{T}_i$ is predicted as:

$$\mathbf{T}_i = \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \exp\left(\boldsymbol{\omega} \cdot \Delta t_i\right)^{\wedge} & \mathbf{v} \cdot \Delta t_i \\ 0 & 1 \end{bmatrix} \tag{7.21}$$

where Δt_i denotes the time increment when $\mathbf{p}_i$ is achieved relative to the scan starting point. $\boldsymbol{\omega}$ and $\mathbf{v}$ are the average rotation rate and average velocity provided by IMU during Δt_i, respectively. ( )^ represents the skew symmetry matrix of a vector which transforms the rotation vector to the rotation matrix with further exponential transformation.

7.3.4.2 Dynamic Objects

A major assumption made in LiDAR SLAM is that the surrounding environment is static, meaning that there are only still objects. In other words, LiDAR SLAM assumes that the change of the distances between the ego-vehicle to the surrounding objects is due to the relative transformation of ego-vehicle the between two epochs. As Figure 7.11 shows, the position change of the dynamic object is not due to the position change of the ego-vehicle, resulting in the estimated relative transformation of ego-vehicle not being accurate. This is particularly a problem when LiDAR SLAM is used in urban areas where the number of dynamic objects is excessive [24].

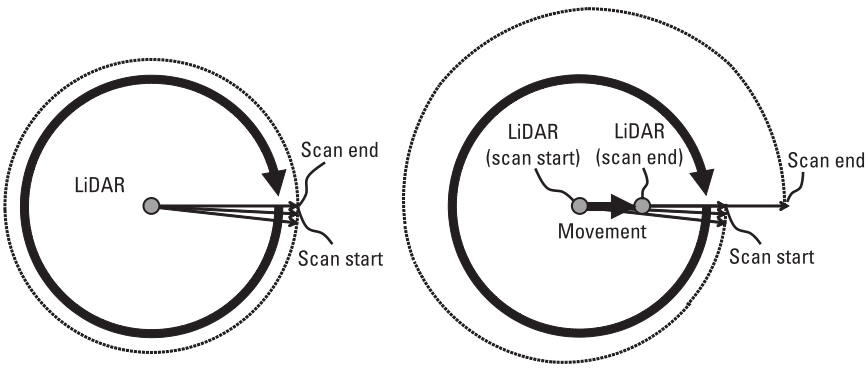

Figure 7.8 Illustration of motion distortion due to LiDAR movement.

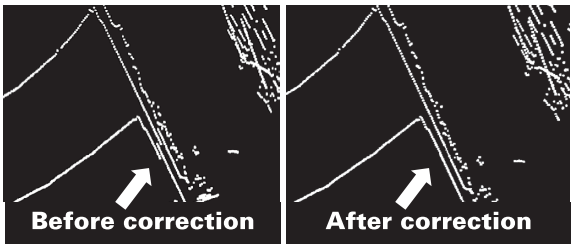

Figure 7.9 Motion distortion before (left) and after (right) applying correction.

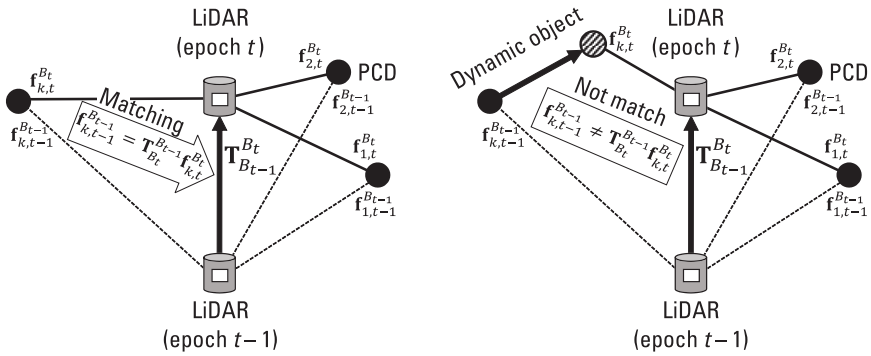

Figure 7.10 LiDAR odometry without (left) and with (right) the impact of a dynamic object.

There are several approaches to mitigate the effect. One of the intuitive approaches is to detect the dynamic object using advanced object recognition approach [25] to exclude the point cloud that came from the dynamic object before its use in LiDAR SLAM. Another approach is to apply a robust estimator (such as the ones introduced in Section 2.3) to mitigate the impact of these outlier measurements [26].

{AU: Is this the correct section?}

7.3.5 Close-Loop Constraints

{AU: Should this be "Closed-Loop", here and throughout?}

A loop closure should be detected once the sensor carrier arrives at the same place it has been. A Euclidean distance-based loop closure detection approach is proposed [8]. For the latest estimated pose $\mathbf{T}_j$, the closest prior pose $\mathbf{T}_i$ is searched in the surrounding Euclidean space with a certain size. $\mathbf{T}_i$ is selected as the loop pair for $\mathbf{T}_j$ if the Euclidean distance between them is smaller than a predetermined threshold. The transformation $\Delta\mathbf{T}'_{ij}$ between the loop pair is achieved by matching the i scan with the sub-map composed of scans around the j one. The close-loop constraint is formulated as the residual between the observed pose increment $\Delta\mathbf{T}'_{ij}$ and the practical one derived from the state as follows:

$$\mathbf{f}\left(\mathbf{T}_i, \mathbf{T}_j, \Delta\mathbf{T}'_{ij}\right) = \ln\left(\Delta\mathbf{T}'^{-1}_{ij} \mathbf{T}_i^{-1} \mathbf{T}_j\right)^{\vee} \tag{7.22}$$

For implementation feasibility (e.g., derivative calculation of the residual relative to the state), the residual is transformed from the matrix format to the

Lie algebra format by $\ln(\ )^{\vee}$. $\ln(\ )$ transforms the transformation matrix to the skew-symmetric matrix constructed from its corresponding Lie algebra. $(\ )^{\vee}$ recover the Lie algebra from its skew symmetric matrix.

7.4 Visual SLAM

There are two popular types of cameras that are used for visual SLAM for IPIN applications, monocular and RGB-D cameras, which will be introduced next. Stereo camera-based SLAM is not introduced in the book for the following reasons. The stereo camera-based approach uses the baseline between the left and right cameras to calculate the disparity to reconstruct the depth of the images. However, the baseline is usually very short (e.g., the stereo camera on smartphones), resulting in the reconstructed depth being inaccurate for objects at long distances. The poor depth measurements limit its performance in odometry and SLAM. As a result, a monocular camera-based SLAM is usually used even when a stereo camera is available. In other words, only the left or right camera is used. As a result, the monocular camera-based SLAM is still the most popular approach.

{AU: Edits correct?}

7.4.1 Monocular Camera

Compared with LiDAR, a monocular camera cannot directly measure the distance (also known as "depth" in image processing) between the sensed objects and the camera. To use a monocular camera in a visual odometry, it has to use a sequence of 2-D images (t and $t+1$) to estimate the transformation matrix of the camera pose between two epochs of time. In other words, the change of a same feature in two images can be used to infer the change of the camera pose as shown in Figure 7.15. The overall flow of a visual SLAM can be simplified as in Figure 7.16, for which each of the functions will be introduced next.

{AU: Call out Figures 7.12, 7.13, and 7.14 sequentially in the text}

7.4.1.1 Feature Detection and Tracking

As the visual odometry makes use of the change of a same feature in two images to infer the change of the camera pose, it is utterly important to detect distinctive features from an image. Obviously, corner points are much more distinctive than lines since it can be easily identified in a 2-D image by our eyes (see Figure 7.17).

The corner point can be detected by a widely applied method, the Shi-Tomasi method [27]. Its idea is to check whether there are two lines in a

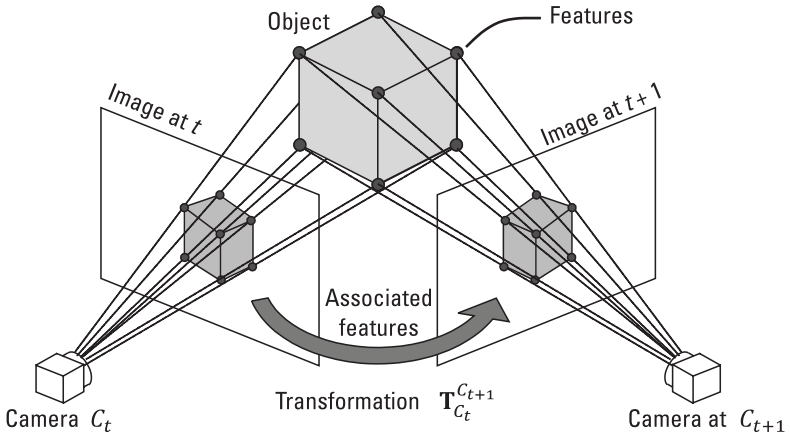

Figure 7.11 Illustration of the idea of estimating a transformation matrix using the detected and tracking features on a sequence of images.

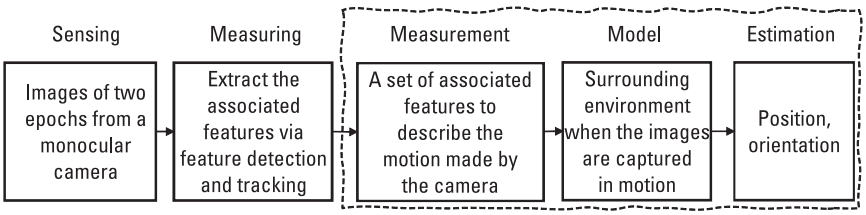

Figure 7.12 The measurements, models and estimation of using an image for a monocular camera based odometry.

searched area of the image. First, the image is transformed from red-green-blue (RGB) to a grayscale image for making only one value in each pixel. Then a 2-D window with the size of w_u and w_v in u- and v-axes is used as a region of interest (ROI) to find corner points, as shown in Figure 7.18.

The goal here is to find the corner point, meaning the intersection of two lines. Thus, the representation of the lines using the changes of grayscale

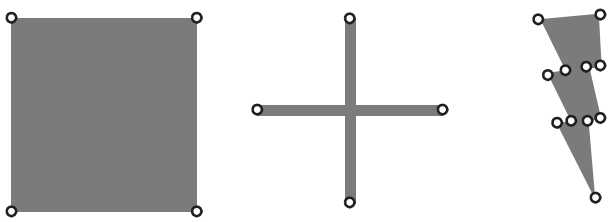

Figure 7.13 Illustration of corner points are distinctive in a 2D plane.

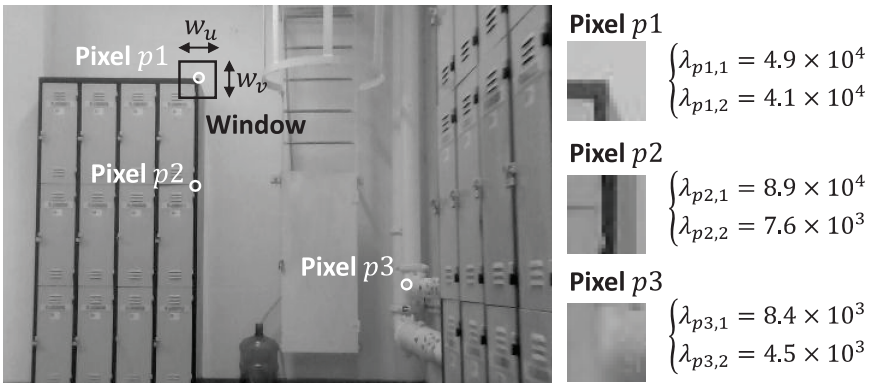

Figure 7.14 (a) Illustration of a 2D window used as a ROI to the searched corner points, and (b) associated eigenvalues when there is either a line or a corner in the window.

values insides the window is important. The grayscale change $E_p(u_p, v_p)$ within the ROI of the searched pixel can be calculated as:

$$E_p\left(u_p,v_p\right) = \sum_{w_u}\sum_{w_v} W\left(w_u,w_v\right)\left[I\left(u_p + w_u, v_p + w_v\right) - I\left(u_p,v_p\right)\right]^2 \quad (7.23)$$

where I denotes a grayscale image. $u_p$and v_p denote the location of the pixel index p used to search the corner points, which is the center of the searching window. $I(u_p, v_p)$ denotes a specific grayscale value. $w(w_u, w_v)$ is the weighting of the corresponding pixel within this window. The calculation of (7.23) can be regarded as a weighted convolution process between the $I(u_p, v_p)$ and the surrounding grayscale value in this window. In general, the window will be set as a quadratic Gaussian distribution with the window center as the origin.

By linearizing the change of grayscale values in the window using a first-order Taylor series expansion, the following equation can be derived:

$$I\left(u_p + w_u, v_p + w_v\right) \approx I\left(u_p,v_p\right) + I_{w_u} w_u + I_{w_v} w_v \quad (7.24)$$

where $I_{w_u} = \partial I(u_p, v_p)/\partial u_p$ and $I_{w_v} = \partial I(u_p, v_p)/\partial v_p$. I_{w_u} and I_{w_v} are the gradient of the gray value changes according to pixel location directions in u and v, respectively (i.e., the partial differential of I). By substituting (7.24) into (7.23), the grayscale change within the window can be modeled as:

$$E_p\left(u_p,v_p\right) \approx \begin{bmatrix} u_p & v_p \end{bmatrix} \mathbf{G} \begin{bmatrix} u_p \\ v_p \end{bmatrix} \quad (7.25)$$

$$\mathbf{G} = \sum_{w_u}\sum_{w_v} W(w_u, w_v)\begin{bmatrix} I_{w_u}^2 & I_{w_u} I_{w_v} \\ I_{w_v} I_{w_u} & I_{w_v}^2 \end{bmatrix} = \mathbf{U}^{-1}\begin{bmatrix} \lambda_{p,1} & 0 \\ 0 & \lambda_{p,2} \end{bmatrix}\mathbf{U} \tag{7.26}$$

If there is a corner point in the window, both the eigenvalues in the derived 2-D model should be larger than a certain value as shown in the right of Figure 7.18. Note that an eigenvector of a model is a nonzero vector that changes at most when the model is applied to by a state value. Eigenvalues are then used to indicate how much the vectors are scaled. In this example, the vectors can be understood as the major lines in the window. Thus, an SVD is applied in the model to obtain the eigenvalues, $\lambda_{p,1}$ and $\lambda_{p,2}$. If both the eigenvalues are larger than a predefined threshold, $\lambda_{\min}$, then it is concluded that there is a corner point in this window. All the pixels p in the image are scanned to find all the detected corner points $\mathbf{f}$. For the example in Figure 7.18 with a uniform weighting W within the window, the detection of features (i.e., corner points) is achieved by

$$\sum_{w_u}\sum_{w_v}\begin{bmatrix} I_{w_u}^2 & I_{w_u} I_{w_v} \\ I_{w_v} I_{w_u} & I_{w_v}^2 \end{bmatrix} = \mathbf{U}^{-1}\begin{bmatrix} \lambda_{p,1} & 0 \\ 0 & \lambda_{p,2} \end{bmatrix}\mathbf{U} \tag{7.27}$$

$$\mathbf{f} = \left\{ p | \lambda_{p,1} > \lambda_{\min}, \& \lambda_{p,2} > \lambda_{\min} \right\} \tag{7.28}$$

As mentioned in Figure 7.15, the key of a visual odometry to use a 2-D correspondence of images to infer the change of the 3-D pose (3-D position and 3-D orientation) of a camera. A popular and computational efficient method to obtain the 2-D correspondence of the detected features is Optical Flow with the Lucas-Kanade method [28]. Its idea is as follows and is shown in Figure 7.19.

For each kth detected feature in the image of the current epoch, located at $[u_k \quad v_k]^{\mathrm{T}} \in \mathbf{f}$, a window in the image of the current epoch is used to check

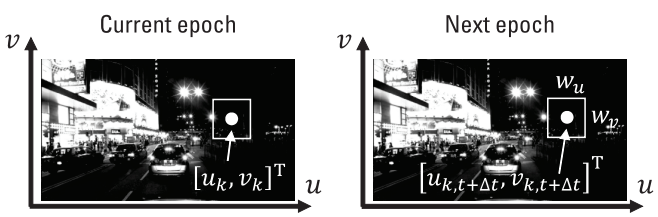

Figure 7.15 Illustration of searching whether the detected feature k in the current epoch exists in the image of the next epoch.

whether the feature still exists in the image of the next epoch. Note that the window here is used to track feature movements, which is different from the ROI window in (7.23) for detecting features. If the grayscale value changes dramatically over a threshold $\varepsilon_{\text{grayscale}}$, it is assumed that the detected feature no longer exists in the current image. It should be noted if the brightness of the image change may cause the loss of detected features, which is one of the reasons that monocular camera-based visual odometry is less robust.

$$\varepsilon_k = \sum_{w_u}\sum_{w_v}\left(I_t\left(u_k, v_k\right) - I_{t+\Delta t}\left(u_k + w_u, v_k + w_v\right)\right)^2 \tag{7.29}$$

$$\mathbf{f}_{\text{track}} = \left\{\begin{bmatrix} u_k & v_k \end{bmatrix}^{\mathrm{T}} \middle| \varepsilon_k < \varepsilon_{\text{grayscale}}\right\} \tag{7.30}$$

The next step is to derive the pixel change Δu_k and Δv_k of a tracked feature k in $\mathbf{f}_{\text{track}}$. Note that the pixel change is the 2-D correspondence that is used as a measurement mentioned in Figure 7.16. If the constant brightness assumption can be made, then, for each k, the grayscale value of the tracked features in the images of the previous and current epochs are remained the same.

$$I\left(u_k, v_k, t\right) = I\left(u_k + \Delta u_k, v_k + \Delta v_k, t + \Delta t\right) \tag{7.31}$$

By applying the first order of Taylor series expansion, the following equation can be derived. Here, the change of grayscale value is assumed to be linear.

$$I\left(u_k + \Delta u_k, v_k + \Delta v_k, t + \Delta t\right) \approx I\left(u_k, v_k, t\right) + \frac{\partial I}{\partial u_k}\Delta u_k + \frac{\partial I}{\partial v_k}\Delta v_k + \frac{\partial I}{\partial t}\Delta t \tag{7.32}$$

If another assumption that the detected feature did not move in the real world is made (static feature), where the movement of Δu_k and Δv_k are due to camera movement, the following equations can be obtained.

$$\frac{\partial I}{\partial u_k}\Delta u_k + \frac{\partial I}{\partial v_k}\Delta v_k + \frac{\partial I}{\partial t}\Delta t = 0 \tag{7.33}$$

$$\frac{\partial I}{\partial u_k}\frac{\Delta u_k}{\Delta t} + \frac{\partial I}{\partial v_k}\frac{\Delta v_k}{\Delta t} = \frac{\partial I}{\partial t} \tag{7.34}$$

$$\begin{bmatrix} I_{u_k} & I_{v_k} \end{bmatrix}\begin{bmatrix} \dot{u}_k \\ \dot{v}_k \end{bmatrix} = \begin{bmatrix} -I_t \end{bmatrix} \tag{7.35}$$

where $\dot{u}_k$ and $\dot{v}_k$ are the velocity of the tracked feature at the u and v directions, respectively.

Finally, another assumption is made that all the pixels in a searched window in the current epoch of the image are spatially consistent. For example, if a static object is captured inside the window, then this assumption is valid. This assumption means that all the pixels in the search window based on the kth tracked feature are used to calculate the velocity of the pixel of the kth feature tracked. The benefit is to provide more measurements to estimate the velocity of a tracked feature compared with just using a tracked pixel to a tracked pixel of the two epochs. In other words, the estimation of the pixel change of a tracked feature k becomes an overdetermined system. The matrix form below can be obtained as shown in the equation:

$$\begin{bmatrix} I_{u,1} & I_{v,1} \\ I_{u,2} & I_{v,2} \\ \vdots & \vdots \\ I_{u,(w_u \times w_v)} & I_{v,(w_u \times w_v)} \end{bmatrix} \begin{bmatrix} \dot{u}_k \\ \dot{v}_k \end{bmatrix} = - \begin{bmatrix} I_{t,1} \\ I_{t,2} \\ \vdots \\ I_{t,(w_u \times w_v)} \end{bmatrix}$$

$$\mathbf{H}_k \mathbf{x}_k = -\mathbf{x}_k \tag{7.36}$$

$$\hat{\mathbf{x}}_k = \left(\mathbf{H}_k^{\mathrm{T}} \mathbf{H}_k\right)^{-1} \mathbf{H}_k^{\mathrm{T}} \left(-\mathbf{z}_k\right)$$

$$k \in \mathbf{f}_{\text{track}}, \text{ feature tracked}$$

An example of detected and tracked visual features by the methods mentioned above is shown in Figure 7.20. The white lines are the measured 2-D correspondences for the estimation of the camera pose. It should be noted that the accuracy and robustness of feature detections and tracking would greatly determine the performance of visual odometry and SLAM. ORB is a popular and a patent-free method to combine the features as a descriptor to improve the robustness and accuracy of the estimation of the 2-D correspondences [29]. A comparison of the different feature-tracking methods can be found [30].

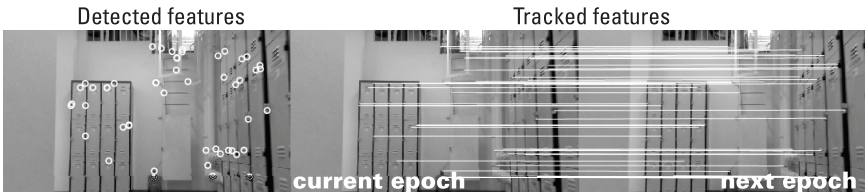

Figure 7.16 Example of feature detection and tracking.

Deep learning networks have been used to detect and track the features [31], which may relax the three assumptions: (1) the constant brightness between two epochs, (2) the track features remaining static in two epochs, and (3) a spatial consistency of the window searched, made in the Lucas-Kanade method.

7.4.1.2 Camera Model

{AU: Edits correct?}

There are several examples of a camera model:

- *Pinhole model for generating an image:* To estimate the pose of the camera, three coordinate systems are involved: the pixel plane (2-D), the imaging plane (2-D), and the camera-body coordinate (3-D). Figure 7.21 illustrates a pinhole imaging model. If there is an object located at $\mathbf{p}^C = [x^C \quad y^C \quad z^C]^{\mathrm{T}}$ in the camera-body coordinate, by this model, its position in imaging plane becomes $[x^I \quad y^I]^{\mathrm{T}}$. According to the triangular similarity, they have a relationship,

$$\frac{z^C}{f} = \frac{x^C}{x^I} = \frac{y^C}{y^I} \tag{7.37}$$

where f denotes the focal length of the camera. As previously mentioned, an image is usually described by pixels, meaning a pixel plane. The position of a pixel $\mathbf{p}^{\text{pixel}} = [u \quad v]^{\mathrm{T}}$ can be obtained by applying scaling (θ_x and θ_y) and translation (c_x and c_y) to the object in the image plane, which is obtained by another scaling with the focal length from the camera-body coordinate, as below.

$$u = \theta_x x^I + c_x = \frac{\theta_x f x^C}{z^C} + c_x = \frac{f_x x^C}{z^C} + c_x \tag{7.38}$$

$$v = \theta_y y^I + c_y = \frac{\theta_y f x^C}{z^C} + c_x = \frac{f_y y^C}{z^C} + c_y \tag{7.39}$$

By reorganizing (7.38) and (7.39) in a matrix form related to the camera-body coordinate,

$$z^C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^C \\ y^C \\ z^C \end{bmatrix} \tag{7.40}$$

$$s\mathbf{p}^{\text{pixel}} = \mathbf{K}\mathbf{p}^{\text{C}}$$

where **K** denotes an intrinsic parameter matrix. f_x and f_y are the focal length in x- and y-axes of the imaging plane incorporating the pixel-image resolution scaling, respectively. $[c_x, c_y]^T$ is the principal point, and $s = z^C$ denotes the depth of the object. It is clear that the object in the physical world became mathematically related to the pixel in an image. Generally speaking, the intrinsic parameter matrix only needs to be calibrated once before the camera is used to apply visual odometry.

- *Distortion:* In visual SLAM, the camera distortion is not negligible. As shown in Figure 7.22, it is caused by both the lens distortion and the assembly angle between the camera sensor and vertical plane. It can divide into radial and tangential distortions (Figure 7.22). These distortions can be corrected by the following model [32],

$$\begin{cases} \hat{x}^I = x^I\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + 2p_1 x^I y^I + p_2(r^2 + 2x^{I\,2})\right) \\ \hat{y}^I = y^I\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + 2p_2 x^I y^I + p_1(r^2 + 2y^{I\,2})\right) \end{cases} \tag{7.41}$$

where, (k_1, k_2, k_3) and (d_1, d_2) are the coefficients of the radial and tangential distortions, respectively. $(x^I \quad y^I)$ is the ideal position on the imaging plane (without distortion), and $\left(\hat{x}^I, \hat{y}^I\right)$ is the real distorted position. With the intrinsic parameters, the distortion parameters can be obtained by camera calibration. Based on this undistorted model, the true projection position $\mathbf{p}^{\text{pixel}} = [u \quad v]$ of a point on the pixel plane is

$$u = f_{x^I}\hat{x}^I + c_{x^I} \tag{7.42}$$

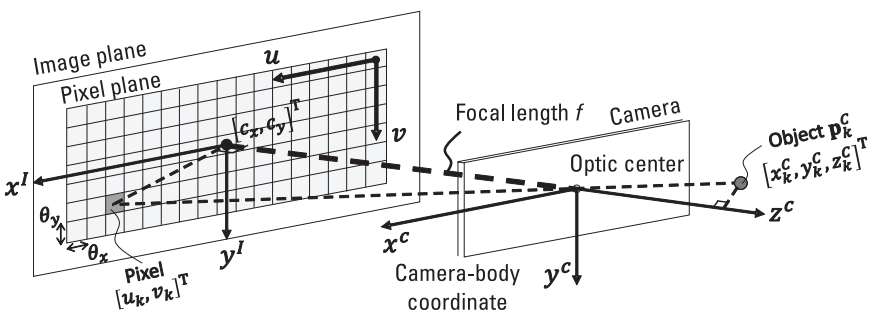

Figure 7.17 The planes and coordinate related a pinhole imaging model.

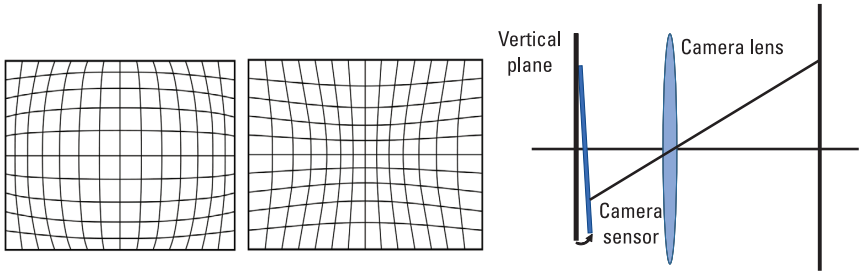

Figure 7.18 Illustration of the camera radial and tangential distortions.

$$y = f_{y^I}\hat{y}^I + c_{y^I} \tag{7.43}$$

7.4.1.3 Visual Odometry

The relationship between the two consecutive camera poses (two epochs) and detected features in the images can be described using Figure 7.23. This is known as an epipolar geometry [33].

The feature points $\mathbf{p}_{k,t}^{\text{pixel}_t}$ and $\mathbf{p}_{k,t+\Delta t}^{\text{pixel}_{t+\Delta t}}$ are a matching pair in the two images, and they correspond to the same 3-D point, $\mathbf{p}_{k,t}^{C_t}$. The camera centers at two adjacent epochs and the 3-D point $\mathbf{p}_{k,t}^{C_t}$ can determine a plane, called the epipolar plane. The line between these two camera centers is the baseline. The intersections of the baseline and the image planes at two adjacent epochs are epipoles, $e_{k,t}$and $e_{k,t+\Delta t}$. The lines $l_{k,t}$ and $l_{k,t+\Delta t}$ between a pixel and the epipole are called the epipolar lines. The pose of the kth detected feature in

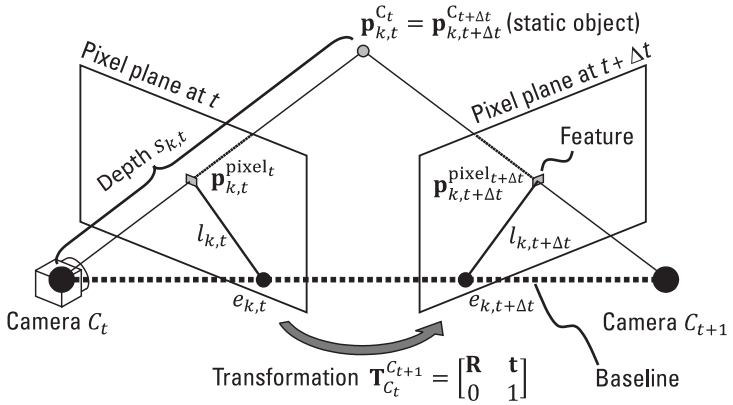

Figure 7.19 Illustration of epipolar geometry with features between adjacent epochs.

the camera-body frame at time $t + \Delta t$ can be obtained from the pose at t by applying a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$ from the camera-body coordinate C_t to $C_{t+\Delta t}$, as follows:

$$\mathbf{p}_{k,t+\Delta t}^{C_{t+\Delta t}} = \mathbf{R}\mathbf{p}_{k,t}^{C_t} + \mathbf{t} \tag{7.44}$$

where the rotation and translation can be described by a transformation matrix as (7.4). In the pixel frame, the detected feature k at time t and $t + \Delta t$ can be expressed as below.

$$\mathbf{p}_{k,t+\Delta t}^{\text{pixel}} = \mathbf{p}_{k,t}^{\text{pixel}} + \begin{bmatrix} \dot{u}_k \\ \dot{v}_k \end{bmatrix} \Delta t,\ k \in \mathbf{f}_{\text{track}},\ \text{feature tracked} \tag{7.45}$$

where $\begin{bmatrix} \dot{u}_k \\ \dot{v}_k \end{bmatrix}$ is the measurement that is obtained from features detection and tracking by (7.36). Then (7.40) is substituted with (7.44).

$$s_{k,t+\Delta t}\mathbf{p}_{k,t+\Delta t}^{\text{pixel}} = \mathbf{K}\left(\mathbf{R}\mathbf{p}_{k,t}^{C_t} + \mathbf{t}\right) \tag{7.46}$$

Note that the depth of the kth feature is changing with time. By using an epipolar constraint, $\mathbf{p}_{k,t}^{C_t} \cdot \left(\mathbf{t} \times \mathbf{p}_{k,t}^{C_t}\right) = 0$ which means that the inner product of the vector, $\mathbf{p}_{k,t}^{C_t}$, and the normal vector of an epipolar plane for the detected feature k, $\left(\mathbf{t} \times \mathbf{p}_{k,t}^{C_t}\right)$, are zero, the following equation can be obtained [34]. The derivation can be found in Appendix 7A.

{AU: Edits correct?}

$$\left(\mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{\text{pixel}}\right)^T \mathbf{t}\hat{}\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_{k,t}^{\text{pixel}} = 0 \tag{7.47}$$

where the operator ^ denotes the skew-symmetric matrix of a vector, as follows:

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix},\ \mathbf{t}\hat{} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{7.48}$$

Then an essential matrix $\mathbf{E}$ is defined as following:

$$\mathbf{E} = \mathbf{t}\hat{}\mathbf{R} = \begin{bmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{bmatrix} \tag{7.49}$$

$$\left[\mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{\text{pixel}}\right]^{\mathrm{T}}\begin{bmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{bmatrix}\left[\mathbf{K}^{-1}\mathbf{p}_{k,t}^{\text{pixel}}\right]=0 \tag{7.50}$$

As can be seen here, by finding eight pairs of matching feature points between two frames [35], the exact solution of the fundamental matrix can be solved. Since the tracked feature might contain random noise, it is common to use more than eight pairs of features to find the elements in the fundamental matrix, meaning that an LS estimation can be used. Since the translation matrix is $\mathbf{t}^{\wedge}$ is a skew-symmetric matrix and $\mathbf{R}$ is an orthogonal matrix, the SVD of the essential matrix $\mathbf{E}$ can decouple $\mathbf{t}^{\wedge}$ and $\mathbf{R}$ as shown in the equations below [36]:

$$\mathbf{E}=\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{T} \tag{7.51}$$

$$\mathbf{t}^{\wedge}=\left[\mathbf{t}_1^{\wedge}\ \ \mathbf{t}_2^{\wedge}\right],\ \mathbf{R}=\left[\mathbf{R}_1\ \ \mathbf{R}_2\right] \tag{7.52}$$

$$\mathbf{t}_1^{\wedge}=\mathbf{U}\mathbf{R}_z(+90^{\circ})\boldsymbol{\Sigma}\mathbf{U}^{\mathrm{T}} \tag{7.53}$$

$$\mathbf{t}_2^{\wedge}=\mathbf{U}\mathbf{R}_z(-90^{\circ})\boldsymbol{\Sigma}\mathbf{U}^{\mathrm{T}} \tag{7.54}$$

$$\mathbf{R}_1=\mathbf{U}\mathbf{R}_z^{\mathrm{T}}(+90^{\circ})\mathbf{V}^{\mathrm{T}} \tag{7.55}$$

$$\mathbf{R}_2=\mathbf{U}\mathbf{R}_z^{\mathrm{T}}(-90^{\circ})\mathbf{V}^{\mathrm{T}} \tag{7.56}$$

$$\mathbf{R}_z(\theta)=\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.57}$$

Finally, the transformation can be obtained, meaning that the visual odometry is achieved. After solving $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{t}_1^{\wedge}$, and $\mathbf{t}_2^{\wedge}$, four combinations of transformation solution $\mathbf{R}\mathbf{p}_{k,t}^{C_t}+\mathbf{t}$ can be obtained. Note that the solution is valid only when the associated depth of $\mathbf{p}^{C}$ is positive in the camera-body frame at time t and $t+\Delta t$. Thus, the unique solution of the transformation matrix can be selected from the above four solutions.

7.4.2 Monocular SLAM

In monocular SLAM, an inherent challenge called scale uncertainty is always present. This problem refers to the real size of the object in the world frame

being unknown. Briefly, if we scale down or up the motion of camera as well as the scene size by a certain multiple, we will capture the same images. Thus, the trajectory estimated by the monocular SLAM will differ from the real trajectory by a scale.

According to the visual odometry, the transformation $\mathbf{T}$ between the two consecutive camera poses can be obtained by (7.51) to (7.57). Then the positions of the features in the 3-D space need to be calculated by the pose estimation results of camera. For the point with the 3-D position $\mathbf{p}_{k,t}^{C_t}$ in Figure 7.23, according to the camera projection model in (7.44), the two projected pixel positions $\mathbf{p}_{k,t}^{I}$, $\mathbf{p}_{k,t+\Delta t}^{I}$ in the two images can be obtained by feature detection, and we have

$$s_{k,t}\mathbf{p}_{k,t}^{I} = \mathbf{K}\mathbf{p}_{k,t}^{C_t} \tag{7.58}$$

$$s_{k,t+\Delta t}\mathbf{p}_{k,t+\Delta t}^{I} = \mathbf{K}\left(\mathbf{R}\mathbf{p}_{k,t}^{C_t} + \mathbf{t}\right) \tag{7.59}$$

By rearranging the equation:

$$\mathbf{K}^{-1}s_{k,t}\mathbf{p}_{k,t}^{I} = \mathbf{p}_{k,t}^{C_t} \tag{7.60}$$

$$\mathbf{K}^{-1}s_{k,t+\Delta t}\mathbf{p}_{k,t+\Delta t}^{I} = \mathbf{R}\mathbf{p}_{k,t}^{C_t} + \mathbf{t} \tag{7.61}$$

By substituting (7.60) into (7.61):

$$s_{k,t+\Delta t}\mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{I} = s_{k,t}\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_{k,t}^{I} + \mathbf{t} \tag{7.62}$$

$$s_{k,t+\Delta t}\mathbf{q}_{k,t+\Delta t}^{C_t} = s_{k,t}\mathbf{R}\mathbf{q}_{k,t}^{C_t} + \mathbf{t} \tag{7.63}$$

where $\mathbf{q}_{k,t+\Delta t}^{C_t} = \mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{I}$ and $\mathbf{q}_{k,t}^{C_t} = \mathbf{K}^{-1}\mathbf{p}_{k,t}^{I}$ are the normalized coordinates of the 3-D point $\mathbf{p}_{k,t}^{C_t}$ with $Z = 1$ in the two camera frames.

In (7.63), there are two unknowns, $s_{t,t+\Delta t}$ and $s_{k,t}$, To solve this, we can use the cross-product with $\mathbf{q}_{k,t+\Delta t}^{C_t}$ on both sides of the equation:

$$\mathbf{q}_{k,t+\Delta t}^{C_t} \times \left(s_{k,t+\Delta t}\mathbf{q}_{k,t+\Delta t}^{C_t}\right) = \mathbf{q}_{k,t+\Delta t}^{C_t} \times \left(s_{k,t}\mathbf{R}\mathbf{q}_{k,t}^{C_t} + \mathbf{t}\right) \tag{7.64}$$

Since any vector crossed with itself equals zero, the left side becomes zero:

$$0 = \mathbf{q}_{k,t+\Delta t}^{C_t} \times \left(s_{k,t+\Delta t}\mathbf{q}_{k,t+\Delta t}^{C_t}\right) \tag{7.65}$$

$$0 = s_{k,t}\left(\mathbf{q}_{k,t+\Delta t}^{C_t} \times \mathbf{R}\mathbf{q}_{k,t}^{C_t}\right) + \mathbf{q}_{k,t+\Delta t}^{C_t} \times \mathbf{t} \tag{7.66}$$

Rearranging to isolate $s_{k,t}$:

$$s_{k,t}\left(\mathbf{q}_{k,t+\Delta t}^{C_t} \times \mathbf{R}\mathbf{q}_{k,t}^{C_t}\right) = -\left(\mathbf{q}_{k,t+\Delta t}^{C_t} \times \mathbf{t}\right) \tag{7.67}$$

In (7.67), $\mathbf{R}$, $\mathbf{t}$ can be solved by (7.51) through (7.57). $\mathbf{q}_{k,t}^{C_t}$ and $\mathbf{q}_{k,t+\Delta t}^{C_t}$ are obtained by their definition. Thus, $s_{k,t}$ can be solved, and then substitute the solved $s_{k,t}$ into $s_{k,t}\mathbf{p}_{k,t}^{I} = \mathbf{K}\mathbf{p}_{k,t}^{C_t}$ to obtain $\mathbf{p}_{k,t}^{C_t}$, meaning the pose of the feature k in the camera-body frame at the time t. However, the solved $s_{k,t}$ is not the real depth of the point $\mathbf{p}_{k,t}^{C_t}$ in the world frame because of the scale uncertainty in the monocular vision. This is because multiplying a nonzero constant in (7.67) will not change the result of $s_{k,t}$.

A very naive SLAM can therefore be achieved using the equations below.

$$\mathbf{p}_k^L = \mathbf{T}_{C_0}^L \prod_t^0 \mathbf{T}_{C_t}^{C_{t-1}} \mathbf{p}_{k,t}^{C_t} \tag{7.68}$$

$$\mathbf{M}^L = \left\{\mathbf{p}_1^L \cdots \mathbf{p}_{N_{\text{feature}}}^L\right\} \tag{7.69}$$

However, its performance is not reliable and the results are prone to drift quickly. The fact that a feature can be detected in multiple epochs is not considered in this naive SLAM. Bundle adjustment is usually applied to tackle this quickly drifting problem.

7.4.2.1 Bundle Adjustment for Multi-Epoch SLAM

Finally, a batch optimization is applied to achieve a bundle adjustment of the map. Here, instead of using the time epoch to index the image, an image index i is used.

$$\left\{\mathbf{R}_i, \mathbf{t}_i, \mathbf{p}_k^{C_i}\right\} = \underset{\mathbf{R}_i, \mathbf{t}_i, \mathbf{p}_k^{C_i}}{\arg\min} \sum_{i \in \text{Img}_l} \sum_{k \in \text{Feature}} \left\| \mathbf{p}_{i,k}^{\text{pixel}_i} - \pi\left(\mathbf{R}_i \mathbf{p}_k^{C_i} + \mathbf{t}_i\right) \right\|^2 \tag{7.70}$$

{AU: This does not appear in (7.70); please check and reword if necessary}

where Img_l are the set of co-visible images and Feature_k are all the feature points can be seen in these images I_l. $\pi(\ )$ is the projection function from the camera-body frame to the pixel frame, meaning (7.40), $\mathbf{p}^{C_i}$ is the 3-D point of the kth feature in the camera-body frame when the ith image is taken, $\mathbf{p}_k^{C_i}$ is the 3-D point of the kth feature in the camera-body frame when the ith image is taken, and $\mathbf{p}_{i,k}^{\text{pixel}_i}$ is the detected feature point in image I_i corresponding

with the point $\mathbf{p}_k^{C_i}$. By reprojecting feature k to image i, considering the rotation matrix $\mathbf{R}_i$ and the translation vector $\mathbf{t}_i$, the predicted pixel position can be found. Then the transformation matrix that can minimize the Euclidean distance between the predicted and measured pixel position of the feature is the optimized relative positioning. The map with the consideration of bundle adjustment can be finally obtained using (7.68). Note that the $\mathbf{R}_i$, $\mathbf{t}_i$ estimated using the bundle adjustment are much more reliable than that calculated using (7.51) to (7.57). The rough estimation by (7.51) to (7.57) is usually only used as an initial guess for (7.70).

7.4.3 RGB-D Camera

A popular RGB-D camera (such as Microsoft Kinect) is to integrate an RGB camera with an infrared (IR) sensor as shown in Figure 7.X. The IR sensor is a radiation-sensitive optoelectronic component, which can measure the distance of an object. It uses an IR LED to send an IR light and a photodiode to sense the reflected light. Similar to a LiDAR, the two-way TOF can therefore be used to measure the distance (or usually called "depth" in the image processing field) between the camera and an object. Due to the integration of an RGB camera and an IR sensor, each pixel also includes its depth information. The principle of integration is similar to Section 7.2.4. Since the depths are provided by an RGB-"D" camera, its visual odometry and SLAM are very similar to that of the LiDAR ones mentioned in the previous section. The RGB part is similar to that of monocular camera. The idea of ICP, NDT, and LOAM can be directly applied. However, it is usually more challenging compared to that using LiDAR. It is because:

{AU: Which figure do you mean here?}

1. Higher noise in the depth measurement is generated by the IR sensor compared with that of a LiDAR. One of the popular solutions to improve the performance of the SLAM using the RGB-D camera is to combine the pixel feature with the depth information in the SLAM. It is a PCD SLAM that integrates with a monocular camera-based SLAM. Effective integration depends on accurate knowledge of the information matrices of both systems. An adaptive approach is preferred, but it is also one of the challenges for this integration.
2. There is a narrower field of view in a camera compared to a mechanical LiDAR, similar to a solid-state LiDAR. The narrower field of view usually causes a decrease of the observability of the model, meaning a larger positioning error or a higher chance of obtaining a local minimum solution. A common approach of dealing with the lower

{AU: Edits correct?}

observability problem is to provide a better initial guess to the solution for the nonlinear optimization problem. However, this is a chicken-and-egg problem. For SLAM, an integration with IMU is an efficient approach to serve this purpose because IMU can measure the propagation of the agent's pose.

7.4.4 Challenges of a Visual SLAM

For monocular SLAM, the biggest bottleneck is the inability to determine the true scale of the scene, which makes the SLAM system relying only on a single camera severely limited in practical applications. Stereo SLAM can compute the depth information, but its accuracy is greatly dependent on the resolution of cameras and the baseline length between cameras, and the calculation of depth of each pixel is computationally time-consuming and unreliable. The integration of camera and IMU can effectively solve the problem of scale [10]. The RGB-D camera can avoid the large computation required for depth acquisition, but the range is usually small, and they are noisy and weak against interference. Therefore, it is commonly used indoors rather than in outdoor environments.

For the monocular camera-based SLAM, feature tracking plays an important role in determining the performance of data association for the motion estimation. The objective of feature tracking is to find the correct feature matching between consecutive series of images. The following are the challenges for rich and stable feature detection and checking:

- *Illumination change:* Both the optical flow-based feature-tracking method (Lucas-Kanade method) and the descriptor-based ORB descriptor rely on the image brightness constancy. Specifically, the optical flow-based feature-tracking method relies heavily on the assumption of brightness constancy to find that the feature corresponds between two consecutive images. However, the assumption of the image brightness constancy in practical scene may not always be satisfied due to lighting conditions and environment structures.
- *Imaging blur:* The image blur caused by a large motion or a sharp turn (large accelerations) will degrade the quality of features. In other words, the image blur leads to incorrect feature detection and tracking.
- *Dynamic objects:* The dynamic objects violate the assumption that the surrounding features are static for the SLAM. Specifically, dynamic objects violate the assumption of the optical-flow tracking algorithm in that the pixels around the key points have same motion.

{AU: Edits correct?}

- *Low-textured environment:* If the environment captured is a white wall without any corner points to be detected, then no features can be detected and tracked.
- *Environment with repetitive features:* If the multiple images taken at different locations are similar, then the optic flow will mistakenly give a near-zero velocity. For example, the corridor of hotel rooms is a typical scenario that visual SLAM will drift by very quickly.

7.4.5 Integration of a Camera with LiDAR SLAM

If the expected LiDAR SLAM accuracy is higher than that of a visual SLAM, it is popular to map the pixels of an image into the accumulated LiDAR PCD map. This requires very precise calibrations on both the intrinsic parameters of the camera and the extrinsic parameters between the LiDAR and the camera body coordinate systems. The time synchronization between the camera and LiDAR is also required to be carefully handled. Assuming that the calibrations are done precisely, the pixel (which the RGB values) of an image can be transformed into a global coordinate system using the LiDAR SLAM. The idea is to give an RGB color to the PCD map generated by LiDAR SLAM as described in Figure 7.24.

Once the pixels are transformed into the LiDAR frame, the RGB value of the pixel closest to a PCD point is used to color that PCD point. Note that the field of view of the camera and LiDAR may differ. To ensure accurate integration, only PCD points and pixels within the overlapping field of view are used for data fusion.

7.5 Roles of IMU in LiDAR SLAMs

Most of the SLAM methods are challenged by a practical aspect, the computational loads. The size of the model means the size of a matrix used in the linear algebra calculation. As the size of the matrix increases, the computational loads of calculating the inverse, the Jacobian matrix, will increase

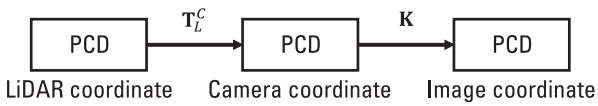

Figure 7.20 The flowchart of transforming pixels to a LiDAR frame.

drastically [37]. Thus, a small-sized model that can maintain its observability is preferred. In a real system, a threshold on the numbers of the features used is usually predetermined to avoid the excessive computational load. The selection of the features used can be done using RANSAC [38]. However, there are cases in which the observability is compromised after the selection of the features. As mentioned in Section 7.2.2, IMU is a good candidate to provide a good initial guess to mitigate the challenge of the small model observability. IMUs measure the 3-D accelerations and the 3-D rotational rates, which can describe the propagation of the agent's pose in a short time. Here, how short it depends on the grade of the IMU used. For example, a MEMS-level IMU used in a smartphone can survive for a few seconds. In addition, if the SLAMs are calculated whenever the LiDAR PCD and images are received, the computational load could be too large for some IPIN applications. The rough output frequencies of LiDAR and the image could be 10 Hz and 15 Hz, respectively, meaning that SLAM calculation needs 10 or more times in a second, which may not be very meaningful if the movement of the agent is not fast. At this time, IMU can also be used to predict the motion of the agent, and the prediction can be used as an initial guess before the calculation of the odometry and SLAM. This can effectively reduce the output rate of the SLAM while maintaining accuracy. It is important to note that the update rate of the IMU is very high, which can reach 400 Hz. If the INS mechanization is used 400 Hz of the update rate, the computational load is high as well. Thus, a pre-integration method is popularly used to solve this problem.

{AU: Spell out all acronyms at first mention}

For the LiDAR and INS integration, we recommend LIO-SAM [8], and its open-source code is available at https://github.com/TixiaoShan/LIO-SAM. For the visual odometry and INS integration, we recommend VINS [10], and its open-source code is available at https://github.com/HKUST- Aerial-Robotics/VINS-Mono.

7.6 Conclusions

This chapter has provided a comprehensive and systematic overview of indoor SLAM technologies, emphasizing their mathematical foundations, algorithmic advancements, and practical implementations. The chapter traces the historical development of SLAM, highlighting its evolution from classical filter-based approaches to modern optimization-based methods. Key milestones such as EKF-SLAM, FastSLAM, GraphSLAM, and recent robust perception frameworks have been discussed, illustrating how each era built upon the strengths of its predecessors while addressing their limitations.

The general mathematical model of SLAM has been detailed, outlining the core principles of feature extraction, transformation estimation, and batch optimization. This foundational framework has supported in-depth discussions on LiDAR SLAM, visual SLAM, and the integration of IMUs. LiDAR SLAM has been explored through methods such as ICP, NDT, and LOAM, each offering distinct advantages in terms of accuracy, computational efficiency, and robustness. Visual SLAM has been examined in the context of monocular and RGB-D cameras, with a focus on feature detection, tracking, and the challenges posed by scale uncertainty and dynamic environments. The critical role of IMUs in enhancing SLAM performance has been emphasized, particularly in providing initial guesses for pose estimation and mitigating computational burdens.

The integration of LiDAR, cameras, and IMUs has been demonstrated as a powerful approach for achieving robust and accurate indoor mapping and localization. By leveraging the complementary strengths of these sensors, LiDAR's precise depth measurements, cameras' rich visual information, and IMUs' high-frequency motion estimates, modern SLAM systems can operate effectively in complex, dynamic environments. Comparative examples of LiDAR SLAM algorithms have further illustrated the practical implications of these technologies, demonstrating their potential for real-world applications.

Looking ahead, the next chapter will build upon the concepts introduced in this chapter by exploring advanced sensor fusion techniques. The integration of additional sensors and data sources to further enhance the reliability, accuracy, and adaptability of SLAM systems will be investigated. This will include discussions on adaptive fusion strategies and the handling of challenging scenarios such as dynamic objects and resource-constrained environments. The insights gained from this chapter help readers to grasp the fundamental concepts and methodologies underlying indoor SLAM technologies. These insights lay the groundwork for understanding the sophisticated sensor fusion frameworks that are currently being explored in the advancement of SLAM systems. By building on this foundation, readers will be better equipped to comprehend the complex sensor fusion techniques discussed in the next chapter, which aim to enhance the reliability, accuracy, and adaptability of SLAM systems for applications in indoor navigation and other relevant fields.

References

[1] Smith, R. C., and P. Cheeseman, "Estimating Uncertain Spatial Relationships In Robotics," *Autonomous Robot Vehicles*, 1990, pp. 167–193.

[2] Montemerlo, M., et al., "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002, pp. 593–598.

[3] Dellaert, F., "Square Root SAM: Simultaneous Localization and Mapping Via Square Root Information Smoothing," *The International Journal of Robotics Research*, Vol. 25, 2006, pp. 1181–1203.

[4] Thrun, S., and M. Montemerlo, "The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures," *The International Journal of Robotics Research*, Vol. 25, 2006, pp. 403–429.

[5] Mur-Artal, R., J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular Slam System," *IEEE Transactions on Robotics*, Vol. 31, No. 5, 2015, pp. 1147–1163.

[6] Engel, J., T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," *European Conference on Computer Vision*, 2014, pp. 834–849.

[7] Zhang, J., and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-Time," *Robotics: Science and Systems*, Vol. 2, 2014.

[8] Shan, T., et al., "LIO-SAM: Tightly-Coupled Lidar Inertial Odometry Via Smoothing and Mapping," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5135–5142.

[9] Klein, G., and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234.

[10] 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.

[11] Salas-Moreno, R. F., et al., "SLAM++: Simultaneous Localisation and Mapping at the Level of Objects," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1352–1359.

[12] Tateno, K., et al., "CNN-SLAM: Real-Time Dense Monocular Slam with Learned Depth Prediction," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6243–6252.

[13] Bescos, B., et al., "DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes," *IEEE Robotics and Automation Letters*, Vol. 3, 2018, pp. 4076–4083.

[14] Mur-Artal, R., and J. D. Tardós, "Orb-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, Vol. 33, No. 5, 2017, pp. 1255–1262.

[15] Rebecq, H., T. Horstschaefer, and D. Scaramuzza, "Real-Time Visual-Inertial Odometry for Event Cameras Using Keyframe-Based Nonlinear Optimization," *British Machine Vision Conference (BMVC)*, 2017.

[16] Huang, Z., J. Rajasegaran, and A. Kanazawa, "Point-Slam: Dense Neural Point Cloud-Based Slam," *Proceedings Computer Vision–ECCV 2022: 17th European Conference*, Tel Aviv, Israel, October 23–27, 2022, pp. 597–614.

[17] Besl, P. J., and N. D. McKay, "Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, No. 2, 1992, pp. 239–256.

[18] Rusu, R. B., and S. Cousins, "3D Is Here: Point Cloud Library (PCL)," *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1–4.

[19] Biber, P., and W. Straßer, "The Normal Distributions Transform: A New Approach to Laser Scan Matching," *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Vol. 3, 2003, pp. 2743–2748.

[20] Magnusson, M., A. Lilienthal, and T. Duckett, "Evaluation of 3D Registration Reliability and Speed—A Comparison of ICP and NDT," *2007 IEEE International Conference on Robotics and Automation*, 2009, pp. 3907–3912.

[21] Pang, J., et al., "3D NDT Transformation Estimation with Ground Segmentation Data," *IEEE Access*, Vol. 6, 2018, pp. 26838–26849.

[22] Segal, A., D. Haehnel, and S. Thrun, "Generalized-ICP," *Robotics: Science and Systems*, Vol. 2, 2009, p. 435.

[23] Ye, H., Y. Chen, and M. Liu, "Tightly Coupled 3D Lidar Inertial Odometry and Mapping," *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3144–3150.

[24] Wen, P., et al., "Performance Evaluation of a Lidar Inertial Odometry and Mapping (LIOM) Method," *2018 IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2018)*, 2018, pp. 7598–7601.

[25] Huang, P., et al., "Coarse-to-Fine Semantic Localization with HD Map for Autonomous Driving," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 9664–9670.

[26] Wang, H., C. Wang, and L. Xie, "F-LOAM: Fast Lidar Odometry and Mapping," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 4390–4396.

[27] Shi, J., and C. Tomasi, "Good Features to Track," *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[28] Lucas, B. D., et al., "An Iterative Image Registration Technique with an Application to Stereo Vision," *IJCAI*, Vol. 81, 1981, pp. 674–679.

[29] Rublee, E., et al., "Orb: An Efficient Alternative to Sift or Surf," *2011 International IEEE Conference on Computer Vision*, 2011, pp. 2564–2571.

[30] Bradski, G., and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, Sebastapol, CA: O'Reilly Media, 2008.

[31] He, K., et al., "Deep Residual Learning for Image Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[32] Zhang, Z., "A Flexible New Technique for Camera Calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, 2000, pp. 1330–1334.

[33] Zhang, Z., "Determining Motion from 3D Line Segment Matches: A Comparative Study," *Image and Vision Computing*, Vol. 16, 1998, pp. 549–565.

[34] Gao, W., et al., *Lectures on Visual Perception*, New York: Springer, 2017.

[35] Hartley, R. I., "In Defense of the Eight-Point Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 6, 1997, pp. 580–593.

[36] Hartley, R., and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge, U.K.: Cambridge University Press, 2004.

Appendix 7A: Derivation of Epipolar Constraint

From (7.46), the transformation can be represented by

$$s_{k,t+\Delta t}\mathbf{p}^{\text{pixel}}_{k,t+\Delta t} = \mathbf{K}\left(\mathbf{R}\mathbf{p}^{C_t}_{k,t} + \mathbf{t}\right) \tag{7A.1}$$

$$s_{k,t+\Delta t}\mathbf{p}^{\text{pixel}}_{k,t+\Delta t} = \mathbf{K}\mathbf{R}\mathbf{p}^{C_t}_{k,t} + \mathbf{K}\mathbf{t} \tag{7A.2}$$

Then the inverse transformation is applied from the camera-body coordinate to the pixel plane:

$$s_{k,t+\Delta t}\mathbf{p}^{\text{pixel}}_{k,t+\Delta t} = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}s_{k,t}\mathbf{p}^{\text{pixel}}_{k,t} + \mathbf{K}\mathbf{t} \tag{7A.3}$$

$$\mathbf{K}^{-1}s_{k,t+\Delta t}\mathbf{p}^{\text{pixel}}_{k,t+\Delta t} = \mathbf{R}\mathbf{K}^{-1}s_{k,t}\mathbf{p}^{\text{pixel}}_{k,t} + \mathbf{t} \tag{7A.4}$$

After that, we can use the property that for any vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a} \times \boldsymbol{b} = \boldsymbol{a}^{\wedge}\boldsymbol{b}$, where $\boldsymbol{a}^{\wedge}$ is the skew-symmetric matrix of $\boldsymbol{a}$.

Multiplying both sides by $\boldsymbol{t}^{\wedge}$:

$$\mathbf{t}^{\wedge}\mathbf{K}^{-1}s_{k,t+\Delta t}\mathbf{p}^{\text{pixel}}_{k,t+\Delta t} = \mathbf{t}^{\wedge}\mathbf{R}\mathbf{K}^{-1}s_{k,t}\mathbf{p}^{\text{pixel}}_{k,t} + \mathbf{t}^{\wedge}\mathbf{t} \tag{7A.5}$$

Since $\boldsymbol{t}^{\wedge}\boldsymbol{t} = 0$ (the cross-product of a vector with itself is zero), we have:

$$\mathbf{t}^{\wedge}\mathbf{K}^{-1}s_{k,t+\Delta t}\mathbf{p}^{\text{pixel}}_{k,t+\Delta t} = \mathbf{t}^{\wedge}\mathbf{R}\mathbf{K}^{-1}s_{k,t}\mathbf{p}^{\text{pixel}}_{k,t} \tag{7A.6}$$

Taking the dot product with $\mathbf{K}^{-1}\mathbf{p}^{\text{pixel}}_{k,t}$ on both sides:

$$\left(\mathbf{K}^{-1}\mathbf{p}^{\text{pixel}}_{k,t}\right)^{T}\mathbf{t}^{\wedge}\mathbf{K}^{-1}s_{k,t+\Delta t}\mathbf{p}^{\text{pixel}}_{k,t+\Delta t} = \left(\mathbf{K}^{-1}\mathbf{p}^{\text{pixel}}_{k,t}\right)^{T}\mathbf{t}^{\wedge}\mathbf{R}\mathbf{K}^{-1}s_{k,t}\mathbf{p}^{\text{pixel}}_{k,t} \tag{7A.7}$$

The left side is zero because $\left(\mathbf{K}^{-1}\mathbf{p}_{k,t}^{\text{pixel}}\right)^T \mathbf{t}\hat{}\mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{\text{pixel}}$ is the dot product of a vector with its own cross-product, which is always zero.

Therefore:

$$0 = \left(\mathbf{K}^{-1}\mathbf{p}_{k,t}^{\text{pixel}}\right)^T \mathbf{t}\hat{}\mathbf{R}\mathbf{K}^{-1}s_{k,t}\mathbf{p}_{k,t}^{\text{pixel}} \tag{7A.8}$$

Since $s_{k,t}$ is a scalar, we can rewrite:

$$0 = s_{k,t}\left(\mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{\text{pixel}}\right)^T \mathbf{t}\hat{}\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_{k,t}^{\text{pixel}} \tag{7A.9}$$

For this equation to be true for any nonzero $s_{k,t}$, we must have:

$$\left(\mathbf{K}^{-1}\mathbf{p}_{k,t+\Delta t}^{\text{pixel}}\right)^T \mathbf{t}\hat{}\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_{k,t}^{\text{pixel}} = 0 \tag{7A.10}$$

This is the epipolar constraint equation.