PART ONE
FOUNDATIONS OF SLAM

I
Prelude

Luca Carlone, Ayoung Kim, Frank Dellaert, Timothy Barfoot, and Daniel Cremers

This chapter introduces the Simultaneous Localization and Mapping (SLAM) problem, presents the modules that form a typical SLAM system, and explains the role of SLAM in the architecture of an autonomous system. The chapter also provides a short historical perspective of the topic and discusses how the traditional notion of SLAM is evolving to fully leverage new technological trends and opportunities. The ultimate goal of the chapter is to introduce basic terminology and motivations, and to describe the scope and structure of this handbook.

I.1 What is SLAM?

A necessary prerequisite for a robot to operate safely and effectively in an unknown environment is to form an internal representation of its surroundings. These type of representations can be used to support obstacle avoidance, low-level control, planning, and, more generally, the decision-making processes required for the robot to complete the task it has been assigned. The execution of simple tasks (*e.g.,* following a lane, or maintaining a certain distance to an object in front of the robot) may only require tracking entities of interest in the sensor data streams, and complex tasks (*e.g.,* large-scale navigation or mobile manipulation) require building and maintaining a *persistent representation* (a map) of the environment. Such a map describes the presence of obstacles, objects, and other entities of interest, and their relative location with respect to the robot's pose (position and orientation). For instance, the map might be used instruct the robot to reach a location of interest, to grasp a certain object, or to support the exploration of an initially unknown environment. Figure I.1 provides some real-world examples of simultaneous localization and mapping (SLAM) in action.

For a robot operating in an initially unknown environment, the problem of building a map of the environment, while concurrently estimating its pose with respect to that map, is referred to as Simultaneous Localization and Mapping (SLAM). SLAM reduces to *localization* if the map is given, in which case the robot only has to estimate its pose with respect to the map. On the other hand, SLAM reduces to *mapping* if the pose of the robot is already known, for instance when an absolute

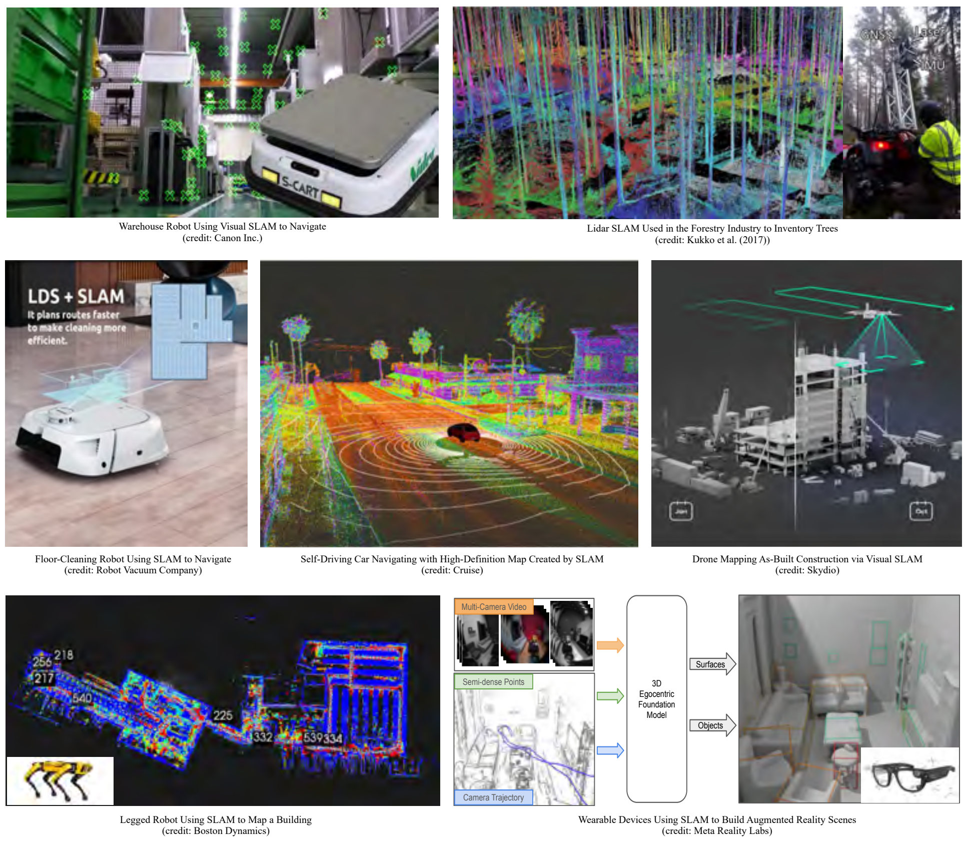
Warehouse Robot Using Visual SLAM to Navigate
(credit: Canon Inc.)
GNSS
Laser
IMU
Lidar SLAM Used in the Forestry Industry to Inventory Trees
(credit: Kukko et al. (2017))
LDS + SLAM
It plans routes faster to make cleaning more efficient.
Floor-Cleaning Robot Using SLAM to Navigate
(credit: Robot Vacuum Company)
Self-Driving Car Navigating with High-Definition Map Created by SLAM
(credit: Cruise)
Drone Mapping As-Built Construction via Visual SLAM
(credit: Skydio)
Legged Robot Using SLAM to Map a Building
(credit: Boston Dynamics)
Multi-Camera Video
Semi-dense Points
Camera Trajectory
3D Egocentric Foundation Model
Surfaces
Objects
Wearable Devices Using SLAM to Build Augmented Reality Scenes
(credit: Meta Reality Labs)

Figure I.1 SLAM is rapidly becoming an enabling technology in a wide array of applications including warehouse robotics, forest inventories [621], floor-cleaning, self-driving cars, drones surveillance, legged-robot mapping, and augmented reality to name only a few.

positioning system is used (e.g., differential GPS or motion capture), in which case the robot only needs to model its surroundings using its sensor data.

The central role of SLAM in robotics research is due to the fact that robot poses are rarely known in practical applications. Differential GPS and motion capture systems are expensive and restricted to small areas, hence being unsuitable for large-scale robot deployments. Consumer-grade GPS is much more broadly available, but its accuracy (with errors typically in the order of meters) and its availability (which is limited to outdoor areas with line-of-sight to satellites) often makes it unsuitable as a source of localization; the consumer-grade GPS —when available— is typically used as an additional source of information for SLAM, rather than a replacement for the localization aspects of SLAM.

Similarly, in many robotics applications, the robot will not typically have access to a prior map, hence it needs to perform SLAM rather than localization. Indeed, in certain applications, building a map is actually the *goal* of the robot deployment; for instance, when robots are used to support disaster response and search-and-rescue operations, they might be deployed to construct a map of the disaster site to help first-responders. In other cases, the map might be stale or not have enough detail. For instance, a domestic robot might have access to the floor plan of the apartment it has to operate in, but such a floor plan may not describe the furniture and objects actually present in the environment, nor the fact that these elements can be rearranged from day to day. In a similar manner, Mars exploration rovers have access to low-resolution satellite maps of the Martian surface, but they still need to perform local mapping to guide obstacle avoidance and motion planning.

The importance of the SLAM problem motivates the large amount of attention this topic has received, both within the research community and from practitioners interested in using SLAM technologies across multiple application domains from robotics, to virtual and augmented reality. At the same time, SLAM remains an exciting area of research, with many open problems and new opportunities.

I.2 Anatomy of a Modern SLAM System

The ultimate goal of SLAM is to infer a map representation and robot poses (*i.e.,* trajectory) from sensor data, including data from *proprioceptive* sensors (*e.g.,* wheel odometry or inertial measurement unit, IMU) and *exteroceptive* sensors (*e.g.,* cameras, light detection and ranging (LiDAR)s, radars). In mathematical terms this can be understood as an *inverse problem*: given a set of measurements, determine a model of the world (the map) and a set of robot poses (trajectory) that could have produced those measurements. There exist two alternative strategies to solve the SLAM problem: indirect and direct methods.

The vast majority of SLAM methods prefers pre-processing the raw sensory data in order to extract "intermediate representations" that are compact and easier to describe mathematically. Instead of using every pixel in an image, these methods extract a few distinctive *2D point features* (or keypoints) and then only model the geometry of how these keypoints depend on the pose of the camera and the geometry of the scene. In contrast, rather than computing an intermediate abstraction, direct methods aim to compute localization and mapping *directly* from the raw sensory data. This categorization is prominent in visual SLAM but is not limited to it as we will see in Chapter 8 and Chapter 9. Both indirect and direct methods have their advantages and shortcomings.

Indirect methods are often faster and more memory efficient. Rather than processing every single pixel of each camera image, for example, they merely process a small subset of keypoints for which the 3D location is determined. As a consequence, real-time capable systems for indirect visual SLAM were already available around the year 2000. To date, indirect methods are the preferred approach for real-time robot vision on platforms with limited compute. Moreover, once the intermediate representation is determined, the subsequent computations are often mathematically simpler, making the resulting inference problems more tractable. In the case of visual SLAM, for example, once a set of corresponding points is identified across a set of images, the resulting problem of localization and mapping amounts to the classical bundle adjustment problem for which a multitude of powerful solvers and approximation methods exist.

In turn, direct methods have the potential to provide superior accuracy because they make use of all available input information. While the processing of all available input information (for example all pixels in each image) is computationally cumbersome and capturing the complex relationship between the quantities of interest (localization and mapping) and the raw input data (e.g. the brightness of each pixel) may create additional non-convexities in the overall loss function, there exist efficient approximation and inference strategies with first real-time capable methods for direct visual SLAM emerging in the 2010s. As we will see in Part II and III, the efficient processing of huge amounts of input data can be facilitated by using graphics processing units (GPUs) to parallelize computations.

In both direct and indirect methods the measurements are used to infer the robot pose and map representation. There is a well-established literature in estimation theory describing how we can infer quantities of interest (in our case, the robot poses and the surrounding map) from observations. This book particularly focuses on estimation theoretic tools —reviewed and tailored to the SLAM problem in Chapter 1 and Chapter 2— that have their foundations in probabilistic inference and that rephrase estimation in terms of solving optimization problems.

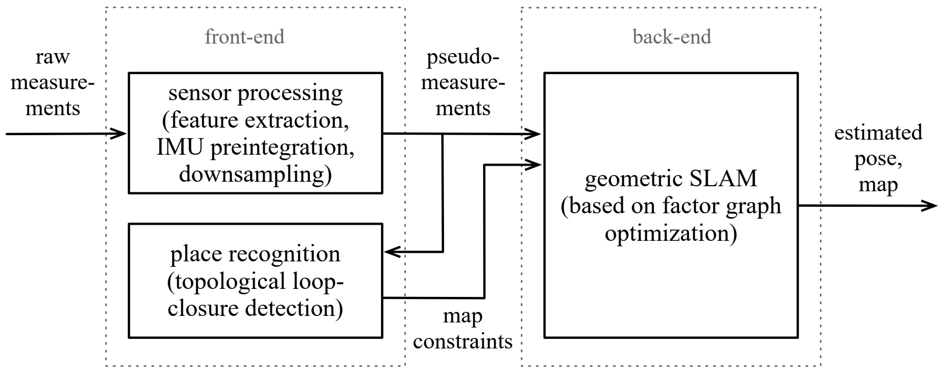
front-end
back-end
raw measure-ments
sensor processing (feature extraction, IMU preintegration, downsampling)
place recognition (topological loop-closure detection)
pseudo-measure-ments
map constraints
geometric SLAM (based on factor graph optimization)
estimated pose, map

Figure I.2 The anatomy of a typical SLAM is made up of a *front-end* (to process rich sensing data into more manageable information and to detect topological loop closures) and a *back-end* (to estimate the robot's pose and a geometric map). The back-end often has a number of helper modules aimed at helping with robustness, computational tractability, and map quality.

Indirect methods produce a natural split in common SLAM architectures (Figure I.2): the raw sensor data is first passed to a set of algorithms (the *SLAM front-end*) in charge of extracting intermediate representations; then such intermediate representations are passed to an estimator (the *SLAM back-end*), that estimates the quantities of interest. The front-end is typically also in charge of building an *initial guess*: this is an initial estimate the back-end can use for iterative optimization, hence mitigating convergence issues due to non-convexity. Let us discuss a few examples to clarify the difference between the SLAM front-end and back-end.

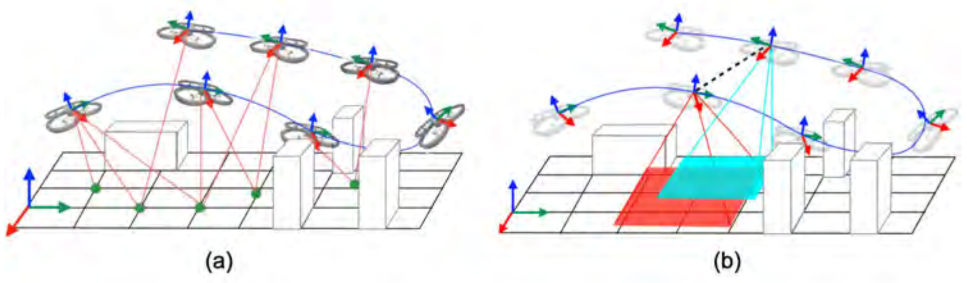

Figure I.3 (a) In landmark-based SLAM models, the front-end produces measurements to 3D landmarks and the back-end estimates the robot trajectory (as a set of poses) and landmark positions. (b) In pose-graph-based SLAM models, the front-end abstracts the raw sensor measurements in terms of odometry and loop closure measurements (these are typically relative pose measurements) and the back-end estimates the overall robot trajectory.

Example I.1 (Visual SLAM: from pixels to landmarks) Visual SLAM uses camera images to estimate the robot trajectory and a sparse 3D point cloud map. The typical front-end of a visual SLAM system extracts 2D keypoints and matches them across frames such that each group (a feature track) corresponds to re-observations of the same 3D point (a landmark) across different camera views. The front-end will also compute rough estimates of the camera poses and 3D landmark positions by using computer vision techniques known as *minimal solvers*.[1] Then, the back-end is in charge of estimating (or refining) the unknown 3D position of the landmarks and the robot poses observing them by solving an optimization problem, known as *bundle adjustment*. This example leads to a landmark-based (of feature-based) SLAM model, visualized in Figure I.3(a). We will discuss visual SLAM at length in Chapter 7.

Example I.2 (Lidar SLAM: from scans to odometry and loop closures) Lidar

[1] A more subtle point is that minimal solvers will also allow pruning away a large portion of outliers, *i.e.*, incorrect detections of a landmark. This makes the job of the back-end easier, while still allowing it to remove any remaining outlier. We discuss outlier rejection and the related problem of data association in Chapter 3.

SLAM uses lidar scans to estimate the robot trajectory and a map. A common front-end for lidar SLAM consists in using scan matching algorithms (*e.g.,* the Iterative Closest Point or ICP) to compute the relative pose between two lidar scans. In particular, the front-end will match scans taken at consecutive time instants to estimate the relative motion of the robot between them (the so called *odometry*) and will also match scans corresponding to multiple visits to the same place (the so called *loop closures*). Odometry and loop closure measurements are then passed to the back-end that optimizes the robot trajectory by solving an optimization problem, known as *pose-graph optimization*. This example leads to a pose-graph-based SLAM model, visualized in Figure I.3(b). We discuss LiDAR SLAM in Chapter 8.

The previous examples showcase three popular examples of "intermediate representations" (or pseudo-measurements) that are produced by the front-end and passed to the back-end (Figure I.2): landmark observations, odometry, and loop closures. In complex SLAM systems, these representations can be used in combination: for instance, in certain visual-SLAM systems one might extract keypoints corresponding to 3D landmarks, and further process them to compute relative poses corresponding to odometry and loop-closures, and finally use a pose-graph-based back-end. The choice of the front-end/back-end split is about selecting a desired trade-off between computation and accuracy. Extracting simpler representations might lead to much faster back-end solvers (*e.g.,* performing pose-graph optimization is typically much faster than doing bundle adjustment); but at the same time abstracting measurements induces approximation in how the measurements are modeled in the back-end, hence leading to small inaccuracies (*e.g.,* bundle adjustment is typically more accurate than pose-graph optimization).

We remark that loop closures are a key aspect of SLAM. If we only use odometry for trajectory estimation, the resulting estimate —obtained by accumulating odometry motion estimates— is bound to drift over time, leading to severe distortion in the trajectory estimate. Revisiting already visited places is crucial to keep the trajectory estimation error bounded and obtain globally consistent maps. We also remark that loop closures are implicitly captured in landmark-based SLAM, where loop closures correspond to new observations of previously seen landmarks.

We conclude this section by observing how SLAM research cuts across multiple disciplines. The SLAM front-end extracts features from raw sensor data, hence touching disciplines ranging from signal processing, geometry, 2D computer vision, and machine learning. The SLAM back-end performs estimation given measurements from the front-end, hence touching estimation theory, optimization, and applied mathematics. This variety of ideas and influences contribute to making SLAM a fascinating and multi-faceted problem.

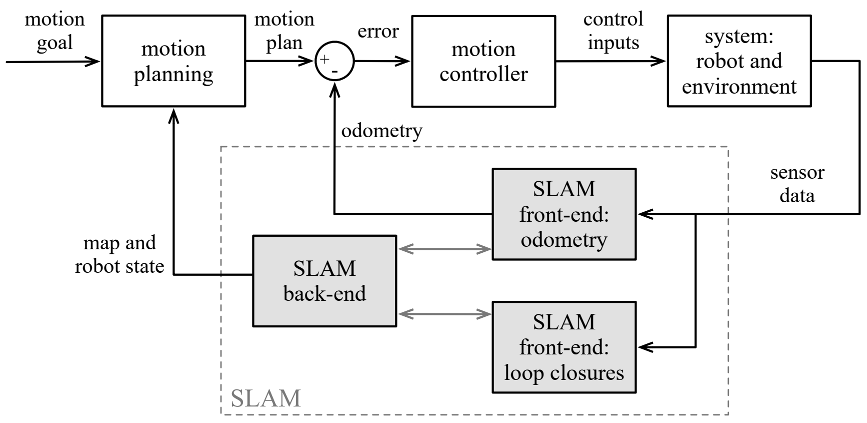

Figure I.4 SLAM plays an important role in the overall autonomy pipeline of a robot that interacts with the world, and provides necessary information for control and motion planning.

I.3 The Role of SLAM in the Autonomy Architecture

The role of SLAM is to serve downstream tasks. For instance, the robot pose estimate can be used to control the robot to follow a desired trajectory, while the map (in combination with the current robot pose) can be used for motion planning (Figure I.4). Here motion planning is used in a broad sense: while SLAM is typically used to build large-scale maps to support navigation tasks, it can also support building local 3D maps to enable manipulation and grasping.

While it would be tempting to think about SLAM as a monolithic system that takes sensor data in input and instantaneously outputs robot poses and map, the actual implementation of these systems and their integration in autonomy architectures is more complicated in practice. This is due to the fact that the robot needs to close different control and decision-making loops with different latency requirements. For instance, with reference to Figure I.4, the robot will need to close low-level control loops over its trajectory (this is the standard feedback control loop at the top-right of the figure), which might require relatively high rates and low-latency to be stable; for instance, a UAV flying at high speed might need the front-end to produce odometry estimates with a latency of a few milliseconds. On the other hand, closing the loop over motion planning (the outer loop in Figure I.4) can accommodate higher latencies, since global planning typically runs at lower rates; hence it might be acceptable for the back-end to provide global trajectory and map estimates with a latency of seconds. For these reasons, a typical implementation of a SLAM system involves multiple processes running in parallel and in a way that slower processes (*e.g.,* global pose and map optimization in the back-end) do not get in the way of faster processes (*e.g.,* odometry estimation). We also observe that the processes involved in a SLAM system have complex interactions (as emphasized by the bi-directional edges in Figure I.4): for instance, while the front-end feeds the odometry to the back-end, the back-end periodically applies global corrections to the odometric trajectory, which is then passed to the motion controller; similarly, while the front-end computes loop closures that are fed to the back-end, the back-end can also inform loop closure detection about plausible or implausible loop closure opportunities.

The problem of visual SLAM is closely related to the problem of Structure from Motion (SfM). While for some researchers both terms are equivalent, others distinguish and argue that visual SLAM systems will typically also integrate additional sensory information (IMUs, wheel odometry, etc) and focus on an online approach where data comes in sequentially whereas in SfM both online and offline versions are conceivable and the input is only images.

Overall, one can distinguish two complementary challenges: There is the *online challenge*, where a robot moves around, where sensory data streams in sequentially and while the SLAM back-end might run at a slower pace, vital estimates such as the localization of the robot must be determined in real-time, often even on embedded hardware with limited compute. These realtime constraints are vital for the robot to properly act in a complex environment, in particular with faster robots such as drones. They often dictate the choice of algorithms and processing steps.

And there is the *offline challenge* where the input data may not exhibit any sequential ordering (say an unordered dataset of images), where computations typically do not require real-time performance and where the compute hardware can be (arbitrarily) large (multiple powerful GPUs), see for example [20]. In such cases, accuracy of the estimated map and trajectories is more important than compute time.

In most applications, however, one will face a mix of these two extreme scenarios where certain quantities need to be determined fast whereas others can be determined offline. In practical applications of SLAM it is of utmost importance to carefully analyze which quantities need to be determined at which frequency and one may come up with an entire hierarchy of different temporal scales at which quantities are being estimated.

I.3.1 Do we really need SLAM for robotics?

From our description above, SLAM feels like an intriguing but very challenging problem, ranging from its complex implementation, to the need of fast runtime on resource-constrained platforms. Therefore, a fair question to ask is whether we can develop complex autonomous robots that do *not* rely on SLAM. We refine this question into three sub-questions.

Q1. Do we need SLAM for any robotics task? We started this section stating that SLAM is designed to support robotics tasks. Then a natural question is whether it is necessary for *any* robotics task. The answer is clearly: no. More reactive tasks, for instance keeping a target in sight can be solved with simpler control strategies (*e.g.,* visual servoing).[2] Similarly, if the robot has to operate over small distances, relying on odometry estimates and local mapping might be acceptable. Moreover, if the environment the robot operates in has some infrastructure for localization, then we may not need to solve SLAM. Nevertheless, SLAM seems an indispensable component for long-term robot operation in unstructured (*i.e.,* infrastructure-free) environments: long-term operation typically requires *memory* (*e.g.,* to go back to previously seen objects or find suitable collision-free paths), and map representations built from SLAM provide such a long-term memory.

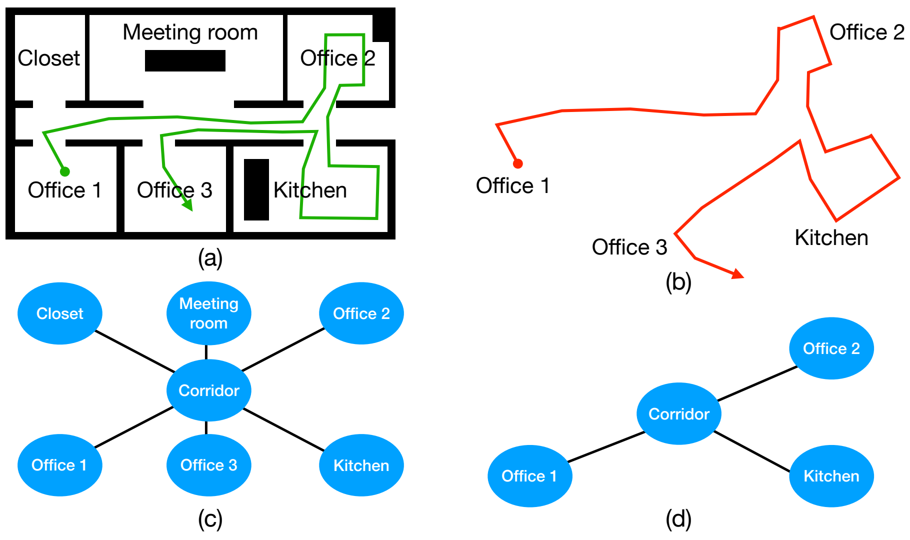

Figure I.5 (a) Our robot visits Office 1 in a building and then —after exploring other areas (including Office 2 and the Kitchen)— it visits Office 3, which is just next door from Office 1. Obstacles are shown in black and ground truth trajectory is shown in green. (b) Odometric estimate of the trajectory, labeled with corresponding room labels. (c) Ground truth topological map of the environment. (d) Estimated topological map in the presence of perceptual aliasing, causing the robot to think that Office 1 and 3 are the same room.

Q2. Do we need globally consistent geometric maps for navigation? A major focus in SLAM is to optimize the trajectory and map representations such that they are metrically accurate (or *globally consistent*) – this is precisely the role of the SLAM back-end. One might ask whether metric accuracy is actually needed. One alternative that comes to mind is to just use odometry to get locally consistent trajectory and map estimates; this circumvents the need for loop closures and

[2] One could argue that while not being strictly necessary for tracking, SLAM and odometry might still be helpful to increase its robustness, *e.g.,* when the target gets out of sight.

back-end optimization. Unfortunately, due to its drift, odometry is unsuitable to support long-term operation: imagine that our robot visits Office 1 in a building and then, after exploring other areas of the building it visits Office 3, which is just next door from Office 1 (see Figure I.5(a)). Using just odometry, the robot might be misled to conclude that Office 1 and Office 3 are quite far from each other (due to the odometry drift), hence being unable to realize there is a short path connecting the two offices (Figure I.5(b)). A slightly more sophisticated alternative is to build a *topological map* instead. A topological map can be thought of as a graph where nodes are places the robot visited and edges represent traversability between the places connected by each edge (Figure I.5(c)). The difference with the *metric* SLAM lens we adopt in this handbook is that nodes and edges in a topological map do not carry metric information (distances, bearing, positions), hence they do not require any optimization: one can simply add edges to a topological map when the robot traveled between two places (odometry) or when a place recognition module recognizes the places to overlap (loop closures). While this seems a perfectly reasonable approach, the main issue is that place recognition techniques are not perfect and, more fundamentally, two different places might look similar (a phenomenon known as *perceptual aliasing*). Therefore, going back to our example above, if Office 1 and Office 3 look very similar, a purely topological approach might be misled to think there is a single office instead (Figure I.5(d)). On the other hand, metric SLAM approaches can use geometric information to conclude that the two offices are indeed two different rooms, by giving the user access to a more powerful set of tools to decide whether place recognition results are correct and if two observations correspond to the same place; we will discuss these tools at length in Chapter 3.

Q3. Do we need maps? SLAM builds a map that can be directly queried, inspected, and visualized. As we will see in Chapter 5, there are many ways to represent a map, including 3D point clouds, voxels, meshes, neural radiance fields, and others. On the other hand, one might take a completely different approach: in order for the robot to execute a task, the robot might be trained to translate raw sensor data directly to actions (*e.g.,* using Reinforcement Learning), hence circumventing the need to build a map. In such an approach, the neural network trained from sensor data to actions will arguably create an internal representation, but such an internal representation cannot be directly queried, inspected, or visualized. While the jury is still out on whether maps are indeed necessary, there is some initial evidence that using maps as an intermediate representation is at least beneficial in completing many visual tasks for robotics [919, 1247]. Moreover, maps have the benefit of being useful across a wide variety of tasks, while a representation that is fully learned in the context of a single task might not be able to support new unseen tasks. Finally, we observe that there are several applications where the *goal* is to have a map that can be inspected. This is the case in search-and-rescue robotics applications where it is desirable to provide a map to help first-responders. Moreover, it is the case for several applications beyond robotics (*e.g.,* real-estate planning and visualization, construction monitoring, virtual and augmented reality), where the goal is for a human to inspect or visualize the map.

I.4 Past, Present, and Future of SLAM, and Scope of this Handbook

The design of algorithms for spatial reasoning has been at the center-stage of robotics and computer vision research since their inception. At the same time, SLAM research keeps evolving and expanding to novel tools and problems.

I.4.1 Short History and Scope of this Handbook

As discussed across the various chapters of this book, SLAM has multiple facets. As a consequence, its history is also multi-faceted with origins that can be traced back across different scientific communities.

Creating maps of the world from observations and measurements is among the oldest challenges in history and leads to the fields of *geodesy* (the science measuring properties of the Earth) and *surveying*. There are many pioneers who contributed to this field. Carl Friedrich Gauss triangulated the Kingdom of Hannover in the years 1821-1825. Sir George Everest served as Surveyor General of India 1830-1843 in the Great Trigonometric Survey, efforts for which he was honored by having the world's largest mountain named after him. In 1856, Carl Maximilian von Bauernfeind published a standard book on "Elements of Surveying" [67]. He subsequently founded the Technical University of Munich in 1868 with a central focus on establishing geodesy as a scientific discipline. André-Louis Cholesky developed the well-known Cholesky matrix decomposition while surveying Crete and North Africa before the First World War.

The problem of visual SLAM is also closely related to the field of photogrammetry and the problems of Structure from Motion in computer vision. Its origins can be traced back to the 19th century. See Chapter 7.

In robotics, the origin of SLAM is typically traced back to the seminal work of Smith and Chessman [1024] and Durrant-Whyte [301], as well as the parallel work by Crowley [241] and Chatila and Laumond [178]. The acronym SLAM was coined in 1995, as part of the survey paper [302]. These early works developed two fundamental insights. The first insight is that to avoid drift in unknown environments, one needs to simultaneously estimate the robot poses and the position of fixed external entities (*e.g.,* landmarks). The second insight is that existing tools from estimation theory, and in particular the celebrated Extended Kalman Filter (EKF), could be used to perform estimation over an extended state describing the robot poses and the landmark positions, leading to a family of *EKF-SLAM* approaches.

EKF-SLAM approaches have been extremely popular but face three main issues in practice. The first is that they are sensitive to outliers and data association errors. These errors may result from failures of place recognition or object detection,

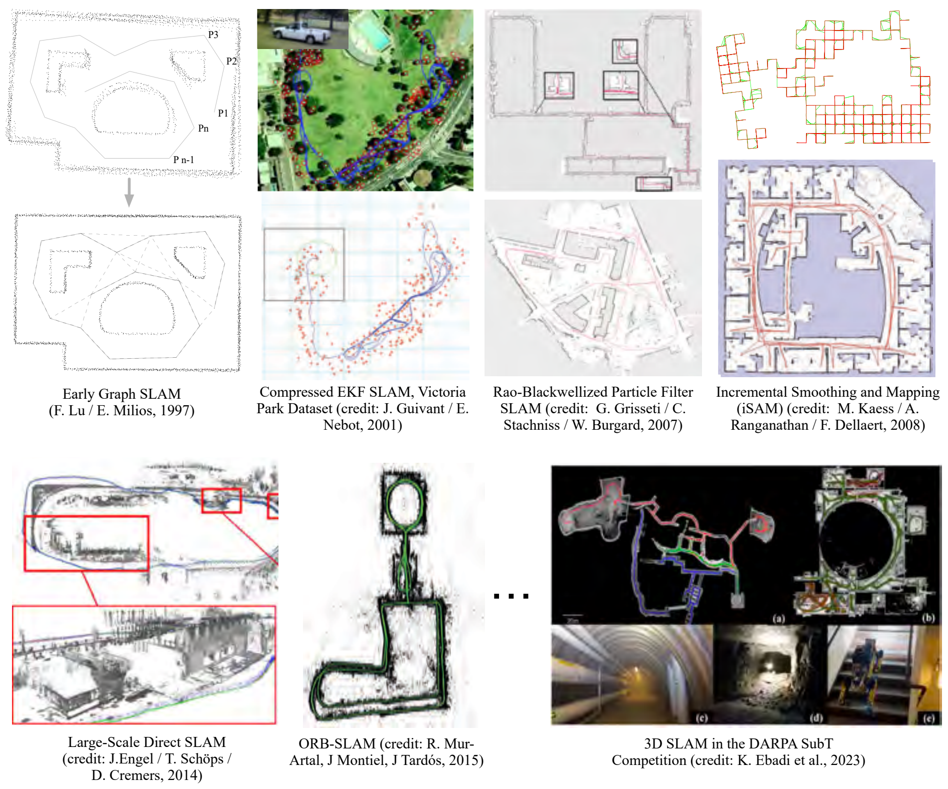
P3
P2
P1
Pn
P n-1
Early Graph SLAM (F. Lu / E. Milios, 1997)
Compressed EKF SLAM, Victoria Park Dataset (credit: J. Guivant / E. Nebot, 2001)
Rao-Blackwellized Particle Filter SLAM (credit: G. Grisseti / C. Stachniss / W. Burgard, 2007)
Incremental Smoothing and Mapping (iSAM) (credit: M. Kaess / A. Ranganathan / F. Dellaert, 2008)
Large-Scale Direct SLAM (credit: J.Engel / T. Schöps / D. Cremers, 2014)
ORB-SLAM (credit: R. Mur-Artal, J Montiel, J Tardós, 2015)
...
(a)
(b)
(c)
(d)
(e)
3D SLAM in the DARPA SubT Competition (credit: K. Ebadi et al., 2023)

Figure I.6 The history of SLAM is filled with numerous advances that have led to modern SLAM systems capable of localizing and mapping robots in challenging real-world environments. This image shows a selection of representative highlights.

where the robot believes it is observing a given object or place, but it is actually observing a different (but possibly similarly looking) one. If these spurious measurements are not properly handled, EKF-SLAM produces grossly incorrect estimates. The second issue is related to the fact that EKF relies on linearization of the equations describing the motion of the robot and sensor observations. In practice, the linearization point is typically built from odometry and when the latter drifts, the linearized system might be a poor approximation of the original nonlinear system. This leads EKF-SLAM to diverge when odometry accumulates substantial drift. The third problem is about computational complexity: a naive implementation of the Kalman Filter leads to a computational complexity that grows quadratically in the number of state variables, due to the need to manipulate a dense covariance matrix. In a landmark-based SLAM problem it is not uncommon to have thousands of landmark, which makes the naive approach prohibitive to run in real-time.

As a response to these issues, in the early 2000s, the community started focusing on *particle-filter-based approaches* [777, 1019, 408], which model the robot trajectory using a set of hypothesis (or *particles*), building on the theory of particle filtering in estimation theory.[3] When used in combination with landmark-based maps, these models allowed using a large number of landmarks (breaking through the quadratic complexity of the EKF); moreover, they allowed to more easily estimate dense map models, such as 2D occupancy grid maps. Also, these approaches did not rely on linearization and were less sensitive to outliers and incorrect data association. However, they still exhibited a trade-off between computation and accuracy: obtaining accurate trajectories and maps requires using many particles (in the thousands) but the more particles, the more computation. In particular, for a finite amount of particles, a particle filter may still diverge when none of the sampled particles are near the real trajectory of the robot (an issue known as *particle* depletion); this issue is exacerbated in 3D problems where one needs many particles to cover potential robot poses.

Between 2005 and 2015, a key insight pushed to the spotlight an alternative approach to SLAM. The insight is that while the covariance matrix appearing in the EKF is dense, its inverse (the so called *Information Matrix*) is very sparse and has a very predictable sparsity pattern when past robot poses are retained in the estimation [321]; this allows designing filtering algorithms that have close-to-linear complexity, as opposed to the quadratic complexity of the EKF. While this insight was initially applied to EKF-like approaches, such as EIF, it also paved the way for *optimization-based approaches.* Optimization-based approaches were first proposed in the early days of SLAM [709], but then disregarded as too slow to be practical. The sparsity structure mentioned above allowed rethinking these optimization methods and making them more scalable and solvable in online fashion [258, 539].[4] This new wave can be interpreted as a shift toward yet another estimation framework: *maximum likelihood* and *maximum a posteriori* estimation. These frameworks rephrase estimation problems in terms of optimization, while describing the structure of the problem in terms of a probabilistic graphical model, or, specifically, a *factor graph.* The resulting factor-graph-based approach to SLAM is still the dominant paradigm today, and has also shaped the way the community thinks about related problems, such as visual and visual-inertial odometry. The optimization lens is a powerful one and allows a much deeper theoretical analysis than previously possible (see Chapter 6). Moreover, it is fairly easy to show that the EKF (with suitable linearization points) can be understood as a single iteration of a nonlinear optimization solver, hence making the optimization lens strictly more powerful than its filtering-based counterpart. Finally, the optimization-based perspective appears more suitable for recent extensions of SLAM (described in the next section and

[3] The resulting algorithms are known with different names in different communities, *e.g.*, Sampling/Importance Re-sampling, Monte-Carlo filter Condensation algorithm, Survival of the fittest algorithm, and others.

[4] More details are in Chapter 1.

Part III of this handbook), where one wants to estimate both continuous variables (describing the scene geometry) and discrete variables (describing semantic aspects of the scene).

This short history review stops at 2015, while the goal of Part III of this handbook is to discuss more modern trends, including those triggered by the “deep learning revolution”, which started around 2012 and slowly permeated to robotics. We also remark that the short history above mostly gravitates around what we called the SLAM back-end (essentially, the estimation engine), while the development of the SLAM front-end traces back to work done across multiple communities, including computer vision, signal processing, and machine learning.

As a result of the considerations mentioned above, this handbook will primarily focus on the factor-graph-based formulation of SLAM. This is a decision about scope and does not detract from the value of ongoing works using other technical tools. For instance, at the time of writing of this handbook, EKF-based tools are still popular for visual-inertial odometry applications (building on the seminal work from Mourikis and Roumeliotis [783]), and novel estimation formulations have been developed, including invariant [63] and equivariant filters [339], as well as alternative formulations based on random finite sets [787].

I.4.2 From SLAM to Spatial AI

SLAM essentially focuses on estimating geometric properties of the environment (and the robot). For instance, the SLAM map carries information about obstacles in the environment, distances and traversable paths between two locations, or geometric coordinates of distinctive landmarks. In this sense, SLAM is useful as a representation for the robot to understand and execute commands such as “robot: go to position $[x, y, z]$”, where $[x, y, z]$ are the coordinates (in the map frame) of a place or object the robot has to reach. However, specifying goals in terms of coordinates is not suitable for non-expert human users and it is definitely not the way we interact or specify goals for humans. Therefore, it would be desirable for the next generation of robots to understand and execute high-level commands specified in natural language, such as “robot: pick up the clothes in the bathroom, and take them to the laundry room”. Parsing these instructions requires the robot to understand both geometry (*e.g.,* where is the bathroom) and semantics (*e.g.,* what is a bathroom or laundry room, which objects are clothes) of the environment.

This realization has recently pushed the research community to think about SLAM as an integrated component of a broader *spatial perception* system, that simultaneously reasons about geometric, semantic, and possibly physical aspects of the scene, in order to build a multi-faceted map representation (a “world model”), that enables the robot to understand and execute complex instructions. The resulting *Spatial AI* algorithms and systems have the potential to increase robot autonomy and have rapidly progressed over the last decade. Intuitively, one can

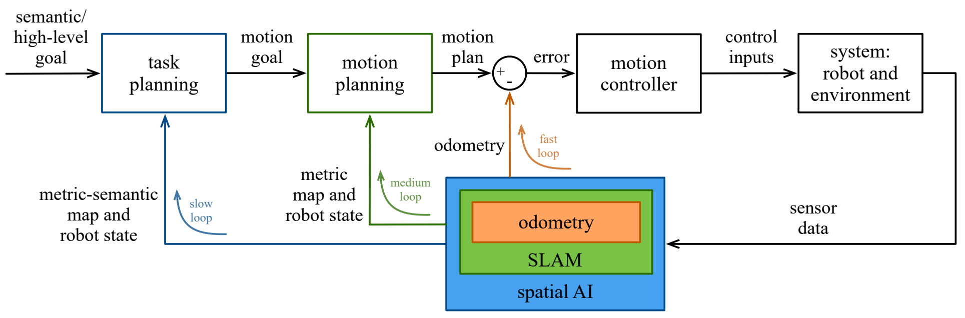

Figure I.7 Spatial AI (or spatial perception) extends the geometric reasoning capabilities of SLAM to also perform semantic and physical reasoning. While the SLAM block is informed by odometry and provides a geometric understanding of the scene, the Spatial AI block is informed by the SLAM results and adds a scene understanding component, spanning semantics, affordances, dynamics, and more. This allows closing the loop over higher-level decision making modules, such as task planning, and allows the user to specify higher-level goals the robot has to achieve.

think that Spatial AI has SLAM as a submodule (to handle the geometric reasoning part), but provides extra semantic reasoning capabilities. This allows closing the loop over task planning, as shown in Figure I.7, where now the robot can take high-level semantic goals instead of coordinates of motion goals. We will discuss Spatial AI at length in Part III of this handbook.

I.5 Handbook Structure

The chapters of this handbook are grouped into three parts.

Part I covers the foundations of SLAM, with particular focus on the estimation-theoretic machinery used in the SLAM back-end and the different types of map representations SLAM can produce. In particular, Chapter 1 introduces the factor-graph formulation of SLAM and reviews how to solve it via iterative nonlinear optimization methods. Then, Chapter 2 takes the indispensable step of extending the formulation to allow the estimation of variables belonging to smooth manifolds, such as rotation and poses. Chapter 3 discusses how to model and mitigate the impact of outliers and incorrect data association in the SLAM back-end. Chapter 4 reviews techniques to make the back-end optimization differentiable, a key step towards interfacing traditional SLAM methods with more recent deep learning architectures. Chapter 5 shifts the focus from the back-end to the question of dense map representations and discusses the most important representations used for SLAM. Finally, Chapter 6 discusses more advanced solvers and theoretical properties of the SLAM back-end.

Part II covers the "state of practice" in SLAM by discussing key approaches and applications of SLAM using different sensing modalities. This part touches on the SLAM front-end design (which is heavily sensor dependent) and exposes what's feasible with modern SLAM algorithms and systems. Chapter 7 reviews the large body of literature on visual SLAM. Chapter 8 and Chapter 9 cover lidar-SLAM and radar-SLAM, respectively. Chapter 10 discusses recent work on SLAM using event-based cameras. Chapter 11 reviews how to model inertial measurements as part of a factor-graph SLAM system and discusses fundamental limits (*e.g.,* observability). Chapter 12 discussed how to model other sources of odometry information, including wheel and legged odometry.

Part III provides a future-looking view of the state of the art and recent trends. In particular, we touch on a variety of topics, ranging from computational architectures, to novel problems and representations, to the role of language and Foundation Models in SLAM. In particular, Chapter 13 reviews recent improvements obtained by introducing deep learning modules in conjunction with differentiable optimization in SLAM. Chapter 14 discusses opportunities and challenges in using novel map presentations, including neural radiance fields (NeRFs) and Gaussian Splatting. Chapter 15 covers recent work on SLAM in highly dynamic and deformable environments, touching on real applications from mapping in crowded environments to surgical robotics. Chapter 16 discusses progress in Spatial AI and metric-semantic map representations. Chapter 17 considers new opportunities arising from the use of Foundation Models (*e.g.,* Large Vision-Language Models) and their role in creating novel map representation for Spatial AI that allow understanding and grounding "open-vocabulary" commands given in natural language. Finally, Chapter 18 focuses on future computational architectures for Spatial AI that could leverage more flexible and distributed computing hardware and better support spatial perception across many robotic platforms.