16

Metric-Semantic SLAM

Arash Asgharivaskasi, Kevin Doherty, Jens Behley, Nathan Hughes, Yun Chang, John Leonard, Henrik I. Christensen, Luca Carlone, and Nikolay Atanasov

This chapter discusses how geometric map representations produced by SLAM methods can be augmented into more semantically rich representations to enable a broader set of downstream tasks, while also enhancing the SLAM performance SLAM plays a critical role in mobile robot autonomy, by enabling robots to localize themselves and maintain a consistent map representation of a priori unknown environments. As we discussed in Chapter I, another equally important role of SLAM is to support robot task and motion planning by providing information about task and motion goals and constraints. Dense map representations (as the ones we reviewed in Chapter 5) offer efficient ways to encode safety constraints for collision checking in robot navigation and manipulation applications. Examples include motion planning and trajectory optimization supported by occupancy [100, 176], signed distance function [828, 1272], and mesh [895, 118] representations. However, as one considers encoding more complex robot tasks, including semantic goals and requirements (*e.g.,* "navigate to the laptop in the office" or "avoid entering the meeting room if there are people inside"), geometric information about the environment alone may be insufficient. In the context of robot navigation, information about the semantics of the environment, *e.g.,* whether a portion of the map is a road, sidewalk, or vegetation, can be utilized to specify different traversability costs [917, 672]. More generally, map representations generated by SLAM techniques can support robot task specification and planning by providing semantic information about objects, places, and their relations. Such information is not only useful for downstream tasks, but is often useful for the SLAM system itself: semantically meaningful features are more uniquely identifiable and viewpoint invariant, leading to improvements in data association and loop closure [114, 718]. Similarly, semantic information can help handle dynamic entities in the scene [948], which are typically less informative for motion estimation.

Being able to construct semantically rich map representations requires extracting additional information from the visual observations beyond image-gradient-based keypoints. Advances in deep learning allow obtaining object detections, object keypoints, semantic edges, semantic segmentation, instance segmentation, panoptic segmentation, or other semantic information from the visual observations. *Object*

detection is a computer vision task that identifies and localizes objects within the images, typically via bounding box annotations. *Semantic keypoints* [860] or *semantic edges* [1252] can be identified as mid-level parts of objects (e.g., doors, wheels, windshield of a car). *Semantic segmentation* [452, 939] goes beyond object detection by assigning a class label to every pixel in an image. *Instance segmentation* also identifies and separates individual objects, even if they belong to the same category, by creating pixel-by-pixel masks around each object instance. *Panoptic segmentation* [588] combines semantic and instance segmentation to achieve a comprehensive understanding of an image. This chapter describes different extensions of traditional SLAM to utilize semantic information from the visual front-end and to generalize the map optimized by the back-end to accumulate and fuse this information in 3D spatial metric-semantic representation. In particular, Section 16.2 discusses how to augment landmark-based SLAM with information about the landmark semantics, Section 16.3 describes extensions of the dense representations in Chapter 5 to incorporate semantic information, and Section 16.4 discusses hierarchical map representations, that capture semantics at different levels of abstraction. As usual, we close the chapter with a discussion about recent trends in Section 16.5. We remark that this chapter focuses on "closed-set" semantics, *i.e.,* the case where semantic information is restricted to a relatively small (*e.g.,* 100-1000 labels) and predefined dictionary of semantic concepts (*e.g.,* "chair", "table", "office"). We postpone discussing how to incorporate open-set semantics (*i.e.,* language embeddings) into SLAM to Chapter 17.

16.1 From Traditional SLAM to Metric-Semantic SLAM

There are multiple ways to include semantic information in a map representation. For instance, one can include sparse semantic information by adding semantics to the landmarks in landmark-based maps, embed semantic labels in dense surface-based maps, or even assign semantics to entire regions of free space (*e.g.,* rooms in indoors, or more general regions, such as a parking lot, in outdoors). In the case of sparse maps, this involves a generalization of the usual 3D point landmarks to represent objects with their categories, and possibly also include their poses and shapes. The associated sensor measurement models also need to be extended from low-level visual keypoints and features to high-level detections such as bounding boxes [926], segmentation masks [452], or object parts [861]. This, in turn, leads to *novel formulations of probabilistic factors* and novel factor-graph optimization techniques that capture both metric and semantic information. In the case of dense maps, semantic information can be captured by extending occupancy models from a binary representation of free and occupied space to a *multi-class* representation of semantic categories. More generally, semantic features extracted by computer vision models at the front-end can be grounded to points, surfels, mesh faces, or voxels maintained in dense maps. Besides a transformation of semantic observa-

tions to 3D space, this requires the formulation of novel observation models for semantic information and sequential probabilistic inference techniques for semantic map information updates. As we will see in Section 16.4, hierarchical mapping approaches often combine sparse and dense metric-semantic maps into a single unified representation, describing semantics at multiple levels of abstraction, from dense surface-based maps, to objects and regions.

16.2 Sparse Metric-Semantic Representations

Consider a landmark-based SLAM problem in which the landmarks represent physical objects in the environment. We refer to this setting as sparse metric-semantic SLAM because the map consists of object landmarks with metric properties, such as position, orientation, and shape, and semantic properties, such as object category.

As discussed in Chapter I, landmark-based SLAM is often formulated as a nonlinear least-squares problem, where the landmark observations induce squared terms $r_i(\boldsymbol{x}_i)^2$ in the objective function, and residual errors have the following form:

$$r_i(\boldsymbol{x}_i) = \|\boldsymbol{z}_i - \boldsymbol{h}_i(\boldsymbol{T}_i, \boldsymbol{\ell}_i)\|_{\boldsymbol{\Sigma}_i}, \tag{16.1}$$

with measurement $\boldsymbol{z}_i$ and state variable $\boldsymbol{x}_i = (\boldsymbol{T}_i, \boldsymbol{\ell}_i)$ containing robot pose $\boldsymbol{T}_i$ and landmark state $\boldsymbol{\ell}_i$. In visual SLAM (Chapter 7), the landmarks $\boldsymbol{\ell}_i$ are most commonly represented as 3D points, and the associated measurements $\boldsymbol{z}_i$ are 2D pixel coordinates of the landmark projection to the image plane. In sparse metric-semantic SLAM, the landmarks are generalized to model objects and their properties. Measurements and residual errors are also generalized to model object detections from computer vision algorithms and measure error with respect to projections of objects from 3D space to the 2D image plane. Object representations, object measurements, and object residual error definitions are discussed next.

16.2.1 Object Representation and Factor Graph Modeling

We begin by extending the notion of landmark from a 3D point to a rigid-body object. An object landmark is characterized by not only its position but also its orientation, scale, shape, and semantic category.

Definition 16.1 An *object landmark* is a tuple $\boldsymbol{\ell} = (\sigma, \boldsymbol{q}, \lambda, \boldsymbol{s})$ including a semantic category $\sigma \in \mathbb{N}$ (*e.g.,* "car", "chair", "table"), pose $\boldsymbol{q} \in \mathrm{SE}(3)$, scale $\lambda \in \mathbb{R}_{>0}$, and shape $\boldsymbol{s} \in \mathbb{R}^d$.

The pose $\boldsymbol{q}$ of an object landmark $\boldsymbol{\ell}$ specifies the position and orientation of the object local coordinate frame with respect to the global coordinate frame of the map. In addition to a rigid-body transformation, the definition of an object local coordinate frame requires scaling to allow the presence of objects from the same

category and shape but of different sizes (*e.g.,* a car vs. a toy car). We present possible descriptions of object shape next and illustrate object landmark representations in several examples. Similar to dense surface modeling in SLAM (Chapter 5), the shape $\boldsymbol{s}$ of an object can be represented using an *explicit surface model* (*e.g.,* points, elementary geometric shapes, mesh) or an *implicit surface model* (*e.g.,* occupancy function, signed distance function field). We present commonly used object shape representations in object SLAM below.

16.2.1.1 Object Shape Representation as Semantic Landmarks

The shape of an object from a given category (Definition 16.1) can be modeled as a collection $\{\boldsymbol{s}_j\}_j$ of sparse 3D points $\boldsymbol{s}_j \in \mathbb{R}^3$ called *semantic landmarks*. While this shape model may be viewed simply as a point cloud, it is necessary for SLAM backend optimization to relate the 3D semantic landmarks to 2D visual observations and measure residual error. Instead of a generic point cloud, the semantic landmarks $\boldsymbol{s}$ are defined as keypoints located in a 3D Computer Aided Design (CAD) object model using a deformable (or active) shape model [239]:

$$\boldsymbol{s}_j = \boldsymbol{b}_{j,0} + \sum_k c_k \boldsymbol{b}_{j,k}, \tag{16.2}$$

where $\boldsymbol{b}_{j,0}$ are semantic landmarks from an average category-level shape model and $\boldsymbol{b}_{j,k}$ are several modes of shape variability obtained by PCA with associated shape deformation coefficients $c_k \in \mathbb{R}_{\geq 0}$ as described in [861]. For example, the average shape of a car object may be represented by semantic landmarks $\boldsymbol{b}_{j,0}$ but, to allow shape variations in car instances of different makes and models, we allow shape deformation along the principal modes of variability $\boldsymbol{b}_{j,k}$ with coefficients c_k. Hence, the shape of an observed object landmark may be estimated by optimizing $\{\boldsymbol{s}_j\}_j$ directly or by optimizing the shape deformation coefficients c_k [1012]. Contrary to optimizing independent points, the parametrization (16.2) enforces the semantic landmarks to describe plausible objects shapes.

We refer to the projection of a semantic landmark $\boldsymbol{s}_j$ to the image plane of a camera sensor as a *semantic keypoint* $\boldsymbol{z}_j \in \mathbb{R}^2$. Semantic keypoints can be detected in camera images using convolutional neural network models trained using supervised learning to provide heatmaps consisting of 2D Gaussians centered at each keypoint [861, 1295]. This is illustrated in Fig. 16.1.

The residual error between an object landmark $\boldsymbol{\ell}_i$, with pose $\boldsymbol{q}_i$, scale λ_i, and semantic landmark shape $\{\boldsymbol{s}_{i,j}\}_j$, and corresponding semantic keypoint detections $\{\boldsymbol{z}_{i,j}\}_j$ obtained from robot pose $\boldsymbol{T}_i$ is measured using the camera perspective projection model:

$$r_i(\boldsymbol{x}_i)^2 = \sum_j \|\boldsymbol{z}_{i,j} - \boldsymbol{\pi}(\boldsymbol{q}_i^{-1}\boldsymbol{T}_i, \lambda_i \boldsymbol{s}_{i,j})\|^2_{\boldsymbol{\Sigma}_i}, \tag{16.3}$$

where $\boldsymbol{\pi}(\boldsymbol{T}, \boldsymbol{s}_j)$ denotes perspective projection of 3D point $\boldsymbol{s}_j$ to 2D pixel coordinates in the image plane of a camera with pose $\boldsymbol{T}$. Each semantic landmark $\boldsymbol{s}_{i,j}$ is

(a) (b) (c)

Figure 16.1 Semantic keypoints extracted from an RGB measurement. (a) An outdoor environment with two cars and a robot equipped with a camera. (b) RGB measurement in the camera frame. (c) Semantic keypoints detected in the RGB image.

first scaled by λ_i, then transformed from the object coordinate frame to the global coordinate frame via $\boldsymbol{q}_i$, and finally to the camera coordinate frame via $\boldsymbol{T}_i$. The predicted pixel coordinates of the object semantic landmarks are compared to the semantic keypoint detections $\boldsymbol{z}_{i,j}$ by the residual error in (16.3).

An example of object SLAM using semantic landmarks as the object shape representation applied to the real-world KITTI dataset [381] is shown in Fig. 16.2. The approach is an extension of that in [114] to utilize multiple semantic landmarks per object instead of just the object centroid.

16.2.1.2 Object Shape Representation as Quadric Surface

Another commonly used explicit model of object shape is based on quadric surfaces [811, 827, 1001]. In this case, we model the shape of an object landmark as an ellipsoid $\mathcal{E}_{\boldsymbol{s}} \subset \mathbb{R}^3$ with semi-axis lengths $\boldsymbol{s} \in \mathbb{R}^3$ defined as:

$$\mathcal{E}_{\boldsymbol{s}} = \{\boldsymbol{x} \in \mathbb{R}^3 \mid \boldsymbol{x}^\top \operatorname{diag}(\boldsymbol{s})^{-2} \boldsymbol{x} \leq 1\}. \tag{16.4}$$

An ellipsoid is an example of a quadric surface [811] defined by matrix $\boldsymbol{Q}_{\boldsymbol{s}} \in \mathbb{R}^{4\times 4}$ obtained by rewriting the inequality in (16.4) in homogeneous coordinates:

$$\mathcal{E}_{\boldsymbol{s}} = \left\{ \boldsymbol{x} \in \mathbb{R}^3 \;\middle|\; \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}^\top \underbrace{\begin{bmatrix} \operatorname{diag}(\boldsymbol{s})^{-2} & \boldsymbol{0} \\ \boldsymbol{0}^\top & -1 \end{bmatrix}}_{\boldsymbol{Q}_{\boldsymbol{s}}} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \leq 0 \right\}. \tag{16.5}$$

An ellipsoid representation of object shape is attractive because its projection to the image of a camera can be obtained analytically and, in turn, can be compared to bounding-box object detections using a residual error term (see illustration in Fig. 16.3). The relationship between a 3D ellipsoid and its perspective projection to a 2D conic can be obtained analytically using a dual quadric representation. The

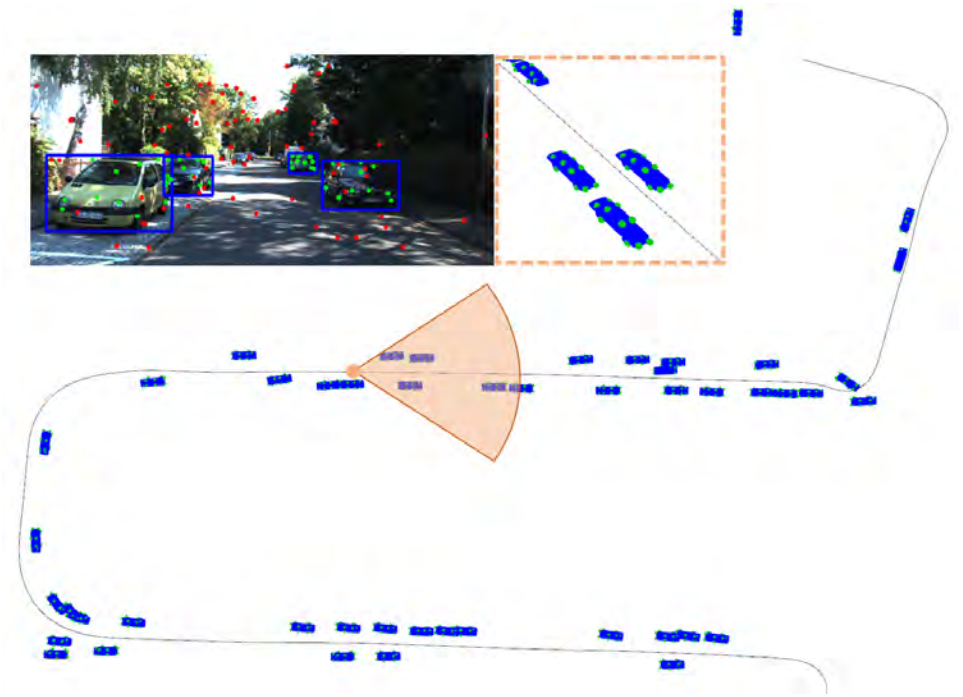

Figure 16.2 Object detections (top left blue bounding boxes), semantic keypoints (top left green 2D points), and regular visual keypoints (top left red 2D points) extracted from RGB images in the KITTI dataset [381] are used to estimate car semantic landmarks (green 3D points, *e.g.,* wheels, front lights), car poses (visualized as blue meshes), and the sensor trajectory (black curve).

dual of an ellipsoid $\mathcal{E}_{\boldsymbol{s}}$ is the set of all hyperplanes tangent to it:

$$\mathcal{E}_{\boldsymbol{s}}^* = \{\boldsymbol{y} \in \mathbb{R}^3 \mid \boldsymbol{y}^\top \operatorname{diag}(\boldsymbol{s})^2 \, \boldsymbol{y} \leq 1\} = \left\{\boldsymbol{y} \in \mathbb{R}^3 \mid \begin{bmatrix}\boldsymbol{y}\\ 1\end{bmatrix}^\top \boldsymbol{Q}_{\boldsymbol{s}}^* \begin{bmatrix}\boldsymbol{y}\\ 1\end{bmatrix} \leq 0\right\}, \tag{16.6}$$

where $\boldsymbol{Q}_{\boldsymbol{s}}^*$ is the adjugate of $\boldsymbol{Q}_{\boldsymbol{s}}$. For an ellipsoid, $\boldsymbol{Q}_{\boldsymbol{s}}$ is invertible and $\boldsymbol{Q}_{\boldsymbol{s}}^* = \operatorname{adj}(\boldsymbol{Q}_{\boldsymbol{s}}) = \det(\boldsymbol{Q}_{\boldsymbol{s}})\boldsymbol{Q}_{\boldsymbol{s}}^{-1}$ can be simplified to $\boldsymbol{Q}_{\boldsymbol{s}}^{-1}$ due to the scale invariance of the dual quadric definition in (16.6).

Consider an object landmark $\boldsymbol{\ell}$ with pose $\boldsymbol{q}$, scale λ, and shape $\boldsymbol{s}$ describing a quadric surface $\boldsymbol{Q}_{\boldsymbol{s}}$. The object surface is scaled and transformed from the local object coordinate frame to the global coordinate frame as follows:

$$\boldsymbol{Q}_{\boldsymbol{\ell}}^* \propto \boldsymbol{q} \begin{bmatrix}\lambda \boldsymbol{I}_3 & \boldsymbol{0}\\ \boldsymbol{0}^\top & 1\end{bmatrix} \boldsymbol{Q}_{\boldsymbol{s}}^* \begin{bmatrix}\lambda \boldsymbol{I}_3 & \boldsymbol{0}\\ \boldsymbol{0}^\top & 1\end{bmatrix}^\top \boldsymbol{q}^\top. \tag{16.7}$$

The perspective projection of the dual quadric $\boldsymbol{Q}_{\boldsymbol{\ell}}^*$ from the global coordinate frame to the image plane of a camera with pose $\boldsymbol{T} \in \mathrm{SE}(3)$ is a dual conic $\boldsymbol{C}_{\boldsymbol{z}}^* \in \mathbb{R}^{3\times 3}$

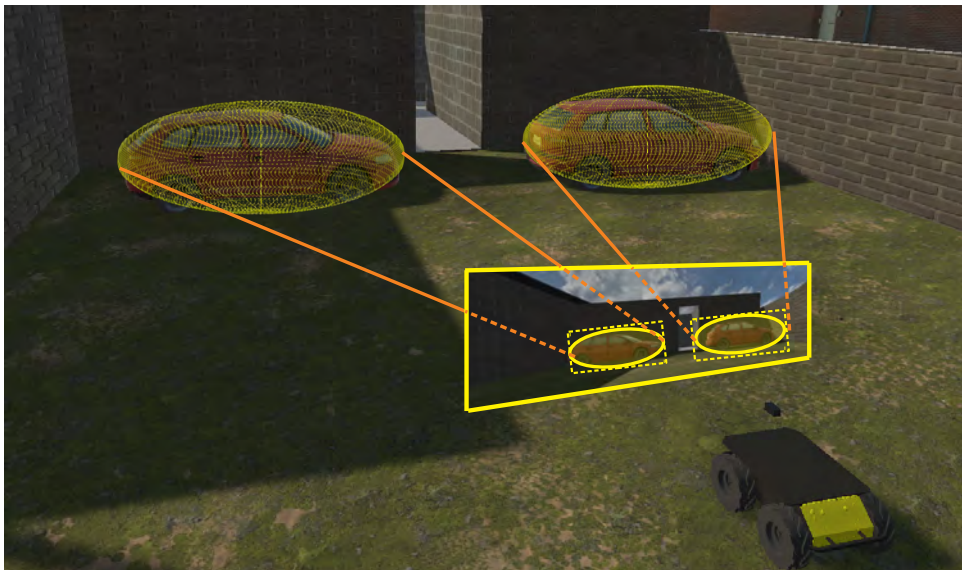

Figure 16.3 Ellipsoid representation of detected objects. The 2D projection of the 3D ellipsoids onto the robot camera frame is shown, in addition to the detection bounding boxes.

defined by:

$$\boldsymbol{C}_{\boldsymbol{x}}^{*} \propto \frac{1}{\beta} \boldsymbol{P} \boldsymbol{T}^{-1} \boldsymbol{Q}_{\boldsymbol{\ell}}^{*} \boldsymbol{T}^{-\top} \boldsymbol{P}^{\top}, \tag{16.8}$$

where $\boldsymbol{P} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{3\times 4}$ is a projection matrix the drops the last coordinate and β captures the depth scaling of the camera. The subscript $\boldsymbol{x}$ emphasizes that $\boldsymbol{C}_{\boldsymbol{x}}^{*}$ is determined by the state variable $\boldsymbol{x} = (\boldsymbol{T}, \boldsymbol{\ell})$, including both the camera pose $\boldsymbol{T}$ and the object landmark $\boldsymbol{\ell}$.

An object bounding-box detection $\boldsymbol{z} = \begin{bmatrix} u & v & w & h \end{bmatrix}^{\top} \in \mathbb{R}^4$ consists of the normalized pixel coordinates (u, v) of the bottom-left bounding-box corner (after application of the inverse camera intrinsics matrix) and the width and height (w, h) of the bounding box. The dual conic representation of an axis-aligned ellipse fit inside the bounding box $\boldsymbol{z}$ with center $(c_u, c_v) = (u + w/2, v + h/2) \in \mathbb{R}^2$ is:

$$\boldsymbol{C}_{\boldsymbol{z}}^{*} \propto \begin{bmatrix} 1 & 0 & c_u \\ 0 & 1 & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (w/2)^2 & 0 & 0 \\ 0 & (h/2)^2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & c_u \\ 0 & 1 & c_v \\ 0 & 0 & 1 \end{bmatrix}^{\top} \tag{16.9}$$

Thus, the residual error between an object landmark $\boldsymbol{\ell}_i$, with pose $\boldsymbol{q}_i$, scale λ_i, and quadric shape $\boldsymbol{Q}_{\boldsymbol{s}_i}$, and corresponding bounding box detection $\boldsymbol{z}_j$ obtained from camera pose $\boldsymbol{T}_i$ can be measured by the discrepancy between the conic $\boldsymbol{C}_{\boldsymbol{z}_j}^{*}$ in (16.9) obtained from the bounding box and the conic $\boldsymbol{C}_{\boldsymbol{x}_i}^{*}$ in (16.8) obtained by

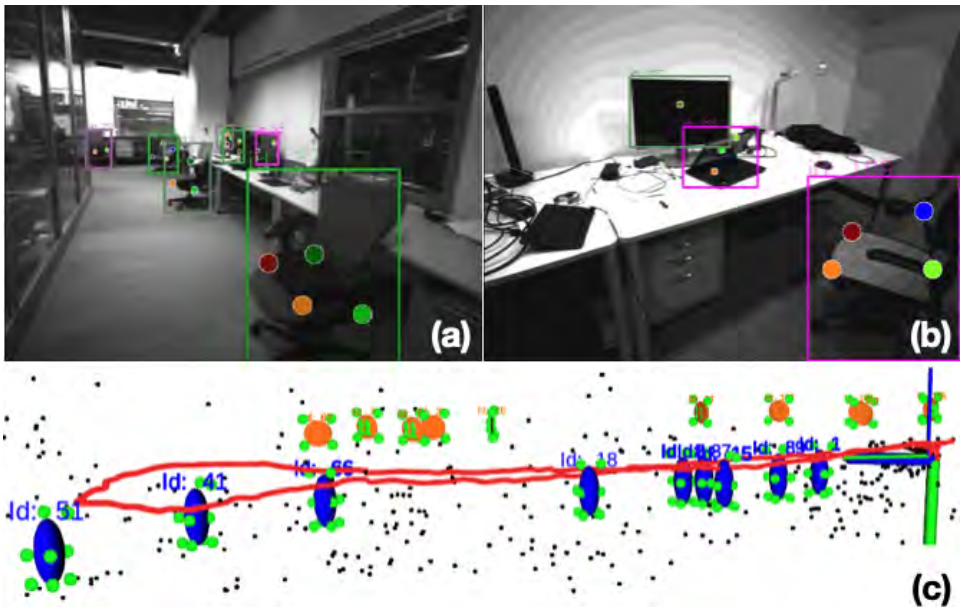

Figure 16.4 Localization and object mapping in an indoor scene with chairs and monitors. Bounding-box and semantic-keypoint detections are shown in (a) and (b). The estimated sensor trajectory (red curve), geometric landmarks (black dots), semantic landmarks (green dots), and object ellipsoids (blue for chairs, orange for monitors) obtained by OrcVIO [1000] are shown in (c).

projecting the quadric shape to the image plane:

$$r_i(\boldsymbol{x}_i) = \|\boldsymbol{C}^*_{\boldsymbol{z}_j} - \boldsymbol{C}^*_{\boldsymbol{x}_i}\|_{\boldsymbol{\Sigma}_i}, \tag{16.10}$$

To avoid defining a matrix norm and exploit the fact that conics are defined by symmetric matrices, we can vectorize $\boldsymbol{C}^*_{\boldsymbol{z}_j} - \frac{1}{\beta_i}\boldsymbol{P}\boldsymbol{T}_i^{-1}\boldsymbol{Q}^*_{\boldsymbol{\ell}_i}\boldsymbol{T}_i^{-\top}\boldsymbol{P}^\top$. In detail, define $\boldsymbol{b}_j = \text{vech}(\boldsymbol{C}^*_{\boldsymbol{z}_j})$ as the half-vectorization of the symmetric matrix $\boldsymbol{C}^*_{\boldsymbol{z}_j}$ and similarly define $\boldsymbol{v}(\boldsymbol{T}_i, \boldsymbol{\ell}_i) = \text{vech}(\boldsymbol{T}_i^{-1}\boldsymbol{Q}^*_{\boldsymbol{\ell}_i}\boldsymbol{T}_i^{-\top})$. Let $\boldsymbol{A} = \boldsymbol{D}(\boldsymbol{P} \otimes \boldsymbol{P})\boldsymbol{E}$, where $\otimes$ is the Kronecker product and matrices $\boldsymbol{D} \in \mathbb{R}^{6\times 9}$ and $\boldsymbol{E} \in \mathbb{R}^{16\times 10}$ are such that $\text{vech}(\boldsymbol{Q}) = \boldsymbol{D}\,\text{vec}(\boldsymbol{Q})$ and $\text{vec}(\boldsymbol{Q}) = \boldsymbol{E}\,\text{vech}(\boldsymbol{Q})$. Thus, we can rewrite the residual error in (16.10) as:

$$r_i(\boldsymbol{x}_i) = \|\boldsymbol{b}_j - \tfrac{1}{\beta_i}\boldsymbol{A}\boldsymbol{v}(\boldsymbol{T}_i, \boldsymbol{\ell}_i)\|_{\boldsymbol{\Sigma}_i}. \tag{16.11}$$

The reader my refer to [957] for details and to [811, 827] for alternative residual error formulations.

An example of object SLAM with the OrcVIO algorithm [1000], combining semantic landmarks and ellipsoids as the object shape representation, applied to real-world data is shown in Fig. 16.4.

16.2.1.3 Object Shape Representation as Mesh

A more expressive object shape representation can be obtained by using a triangular mesh with vertices $\mathcal{V} = \{\boldsymbol{s}_j\}_j$ with 3D coordinates $\boldsymbol{s}_j \in \mathbb{R}^3$ and faces $\mathcal{F} \subset \mathcal{V} \times \mathcal{V} \times \mathcal{V}$. To estimate the parameters of an object landmark with mesh shape, we associate a rendering of the mesh with a *segmentation mask* $\boldsymbol{z}_i \subset \mathbb{R}^2$ in a camera image obtained from detecting and segmenting [452, 926, 1125] the object. This is illustrated in Fig. 16.5.

To optimize the mesh shape of an object in SLAM, we need to define a differentiable residual error. The key challenge is to project and rasterize the mesh in the image plane using differentiable rendering. Given a mesh with vertices $\mathcal{V}$ and faces $\mathcal{F}$, a rasterization function, $\boldsymbol{\rho}(\mathcal{V}, \mathcal{F})$, can be defined by projecting the mesh vertices to the image plane and drawing only the front-most face at each pixel when multiple faces are present [740]. Kato et al. [549] and Liu et al. [687] were among the first works to obtain an approximate gradient for the rasterization function $\boldsymbol{\rho}$ with respect to the mesh vertices. The reader is invited to refer to [550] for a survey on differentiable rendering.

The residual error between an object landmark $\boldsymbol{\ell}_i$, with pose $\boldsymbol{q}_i$, scale λ_i, and mesh shape $(\mathcal{V}, \mathcal{F})$ and corresponding segmentation mask $\boldsymbol{z}_i$ obtained from robot pose $\boldsymbol{T}_i$ is measured using the reciprocal of the intersection-over-union between the mesh image-plane projection and the segmentation mask:

$$r_i(\boldsymbol{x}_i) = \frac{\boldsymbol{\rho}(\{\boldsymbol{\pi}(\boldsymbol{q}_i^{-1}\boldsymbol{T}_i, \lambda_i \boldsymbol{s}_{i,j})\}_j, \mathcal{F}) \cup \boldsymbol{z}_i}{\boldsymbol{\rho}(\{\boldsymbol{\pi}(\boldsymbol{q}_i^{-1}\boldsymbol{T}_i, \lambda_i \boldsymbol{s}_{i,j})\}_j, \mathcal{F}) \cap \boldsymbol{z}_i}, \tag{16.12}$$

where $\boldsymbol{\pi}(\boldsymbol{q}_i^{-1}\boldsymbol{T}_i, \lambda_i \boldsymbol{s}_{i,j})$ is the perspective projection of the 3D mesh vertices $\boldsymbol{s}_{i,j}$ to the image plane as in (16.3) and $\boldsymbol{\rho}$ is a differentiable mesh rasterization function. Similar to the semantic landmark representation above, the mesh vertex coordinates $\{\boldsymbol{s}_j\}_j$ can be optimized directly or indirectly by defining an average category-level shape and optimizing the deformation coefficients of the principle components of shape variability. Other residual error function r_i such as the ℓ_2 loss or the binary cross entropy between the rendered object mask $\boldsymbol{\rho}(\{\boldsymbol{\pi}(\boldsymbol{q}_i^{-1}\boldsymbol{T}_i, \lambda_i \boldsymbol{s}_{i,j})\}_j, \mathcal{F})$ and the segmentation mask $\boldsymbol{z}_i$ may be used [710].

Differentiable mesh rendering has been used for object SLAM in [331], for human pose estimation in [862], and for robot pose estimation in [710]. An example of object SLAM from [331] using meshes as the object shape representation applied to the real-world KITTI dataset [381] is shown in Fig. 16.6. A similar example from [1159], generating differentiable segmentation masks from a deformable shape model (16.2) of a voxel grid storing signed distance values is shown in Fig. 16.7.

16.2.1.4 Implicit Object Shape Representations

The semantic landmarks, quadric surface, and mesh representations of object shape discussed above are examples of explicit shape models because they represent the

(a) (b)

Figure 16.5 Mesh representation of the shape of detected objects. (a) Bounding boxes and segmentation masks of detected objects in the camera view, highlighted in blue. (b) Result of minimizing the residual error between the segmentation masks and the projections of the object meshes.

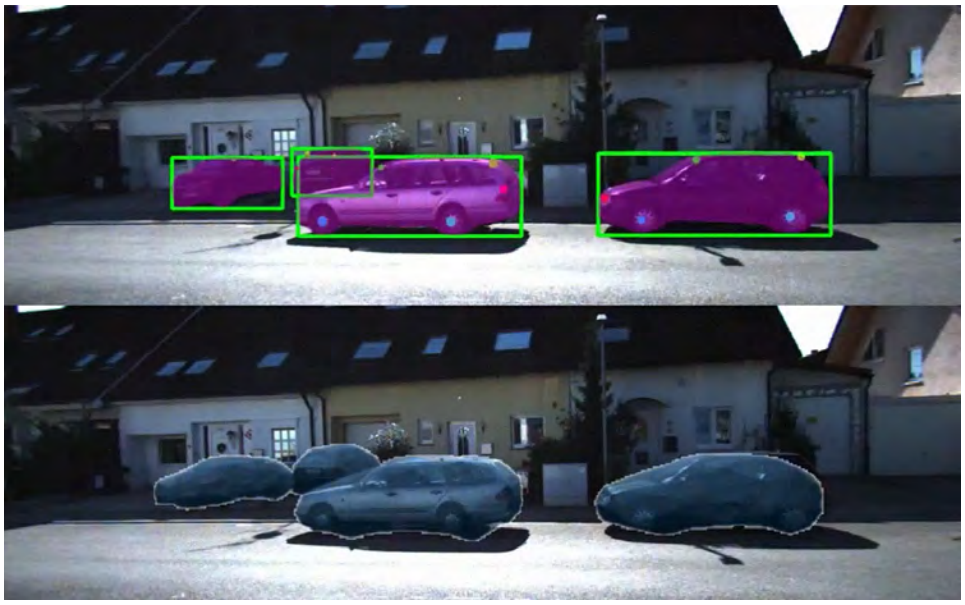

Figure 16.6 Qualitative results of 3D object mesh shape and pose estimation using differentiable segmentation masks [331] on the KITTI odometry dataset [381]. The method takes bounding boxes (green), segmentation masks (magenta) and semantic keypoints (multiple colors) as input and optimizes object poses and mesh shapes using the intersection-over-union residual in (16.12).

object surface geometry directly. Next, we discuss implicit shape representations, which model an object as a spatial function and a particular level set of this function represents the object's surface. Widely used implicit shape representations include occupancy functions [766], SDFs [853], and NeRFs [1273].

Consider an object landmark with shape represented as a set $\mathcal{S} \subset \mathbb{R}^3$. The *occupancy function* $f_{\mathcal{S}}(\boldsymbol{x})$ of set $\mathcal{S}$ is a binary function indicating whether a point

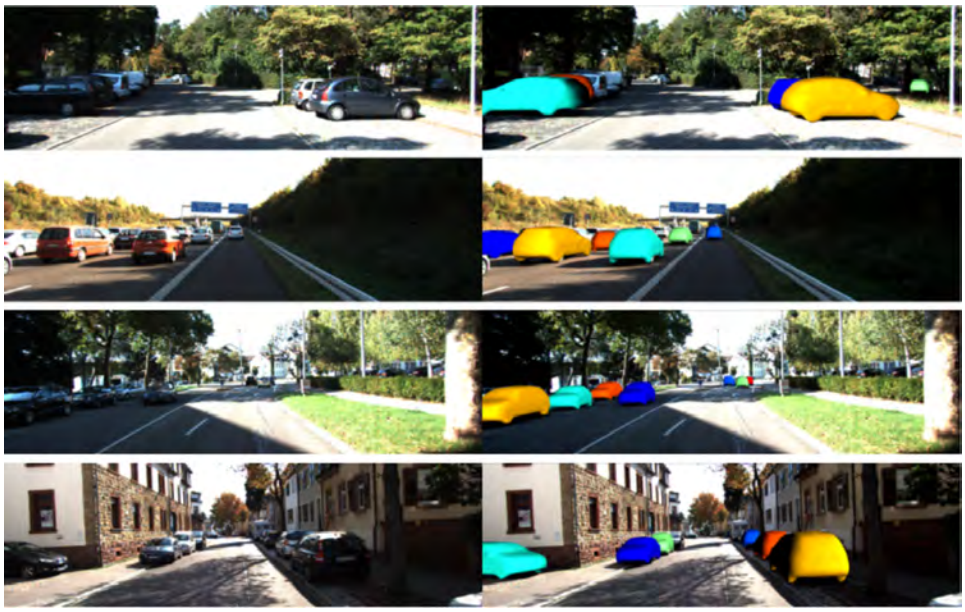

Figure 16.7 Qualitative results of 3D object shape and pose estimation using differentiable segmentation masks obtained from deformable signed distance shape model [1159] on the KITTI Stereo 2015 benchmark [763]. The method takes a stereo image with segmentation masks, initial object pose and learned object mean shape as input. The object is projected to the images and the consistencies between the projections and the segmentation masks are measured by silhouette alignment and photometric consistency residuals to optimize the object pose and shape.

$\boldsymbol{x} \in \mathbb{R}^3$ is inside $\mathcal{S}$:

$$f_{\mathcal{S}}(\boldsymbol{x}) = \begin{cases} -1 & \text{if } \boldsymbol{x} \in \mathcal{S}, \\ 1 & \text{if } \boldsymbol{x} \notin \mathcal{S}. \end{cases} \tag{16.13}$$

The *signed distance function* $d_{\mathcal{S}}(\boldsymbol{x})$ of set $\mathcal{S}$ is a real-valued function measuring the signed distance from a point $\boldsymbol{x} \in \mathbb{R}^3$ to the boundary of $\mathcal{S}$:

$$d_{\mathcal{S}}(\boldsymbol{x}) = \begin{cases} -\inf_{\boldsymbol{y} \in \partial\mathcal{S}} \|\boldsymbol{x} - \boldsymbol{y}\|, & \text{if } \boldsymbol{x} \in \mathcal{S}, \\ \inf_{\boldsymbol{y} \in \partial\mathcal{S}} \|\boldsymbol{x} - \boldsymbol{y}\|, & \text{if } \boldsymbol{x} \notin \mathcal{S}. \end{cases} \tag{16.14}$$

The SDF definition is illustrated in Fig. 16.8.

Given the definitions above, reconstructing an implicit model of object shape from sensor measurements can be viewed as a regression problem to estimate the object's occupancy function $f_{\mathcal{S}}(\boldsymbol{x})$ or SDF $d_{\mathcal{S}}(\boldsymbol{x})$. We focus our discussion on estimating SDF shape but a similar approach can be used to obtain occupancy models instead. To approximate $d_{\mathcal{S}}(\boldsymbol{x})$, we introduce a neural network $d_\theta(\boldsymbol{x}, \boldsymbol{s})$ with parameters θ and latent feature vector $\boldsymbol{s} \in \mathbb{R}^d$ (referred to as *shape code*) that captures the specific

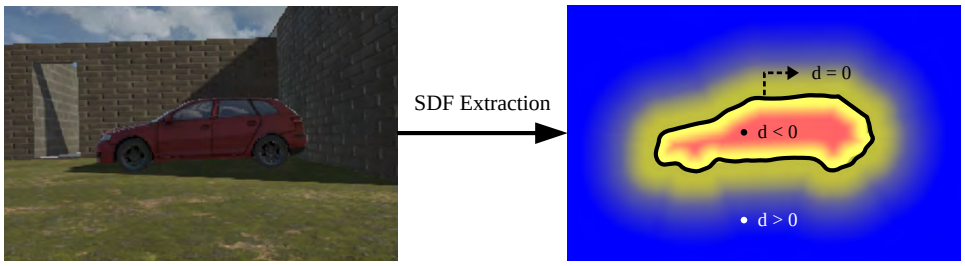

Figure 16.8 SDF of an object landmark. By definition, the object surface is at zero distance, while the distances inside and outside of the object are negative and positive, respectively.

shape of $\mathcal{S}$. In other words, we can think of d_θ as a neural network decoder that estimates the signed distance from a query point $\boldsymbol{x}$ to the surface of an object with shape code $\boldsymbol{s}$. Varying $\boldsymbol{s}$ allows us to consider variations of object shapes with the same neural network decoder, which is useful for category-level shape representation [853, 1002].

To estimate the SDF an object landmark $\boldsymbol{\ell}_i$, we consider a distance sensor, such as LiDAR or depth camera, that provides data $\{\boldsymbol{y}_{i,j}, \boldsymbol{z}_{i,j}\}_j$ from robot pose $\boldsymbol{T}_i$ consisting of point measurements $\boldsymbol{y}_{i,j}$ near the surface of object $\boldsymbol{\ell}_i$ and distance measurements $\boldsymbol{z}_{i,j}$ to the object surface. Then, the residual error between an object landmark $\boldsymbol{\ell}_i$, with pose $\boldsymbol{q}_i$, scale λ_i, and shape code $\boldsymbol{s}_i$, and corresponding observation $\{\boldsymbol{y}_{i,j}, \boldsymbol{z}_{i,j}\}_j$ from robot pose $\boldsymbol{T}_i$ can be measured by comparing the SDF prediction to the measured distance:

$$r_i(\boldsymbol{x}_i)^2 = \sum_j \left| \boldsymbol{z}_{i,j} - d_\theta \left(\lambda_i^{-1} \boldsymbol{q}_i \boldsymbol{T}_i^{-1} \boldsymbol{y}_{i,j}, \boldsymbol{s}_i \right) \right|^2_{\sigma_i}, \tag{16.15}$$

where $\lambda_i^{-1} \boldsymbol{q}_i \boldsymbol{T}_i^{-1} \boldsymbol{y}_{i,j}$ transforms the point $\boldsymbol{y}_{i,j}$, first from the robot frame to the global frame, then from the global frame to the object frame, and finally scales it by λ_i^{-1} to obtain a query point in the canonical object frame [1152]. Assuming that the SDF decoder is already trained, the residual in (16.15) can be minimized to simultaneously estimate the robot pose $\boldsymbol{T}_i$ and the pose $\boldsymbol{q}_i$, scale λ_i, and shape code $\boldsymbol{s}_i$ of the object landmark $\boldsymbol{\ell}_i$.

SDF regression methods often introduce other residual terms, *e.g.,* relating normals at the object surface to the gradient of d_θ via an Eikonal equation [412] or encouraging positive/negative values far away from the surface [840].

We distinguish between the training phase, where we optimize the parameters θ of the SDF decoder of an object category using offline data, and the testing phase, where we optimize the pose $\boldsymbol{q}_i$, scale λ_i, and shape code $\boldsymbol{s}_i$ of a previously unseen object instance from a category with pretrained decoder using online data. In training, one usually assumes that the sensor pose $\boldsymbol{T}_i$ and landmark pose $\boldsymbol{q}_i$ and

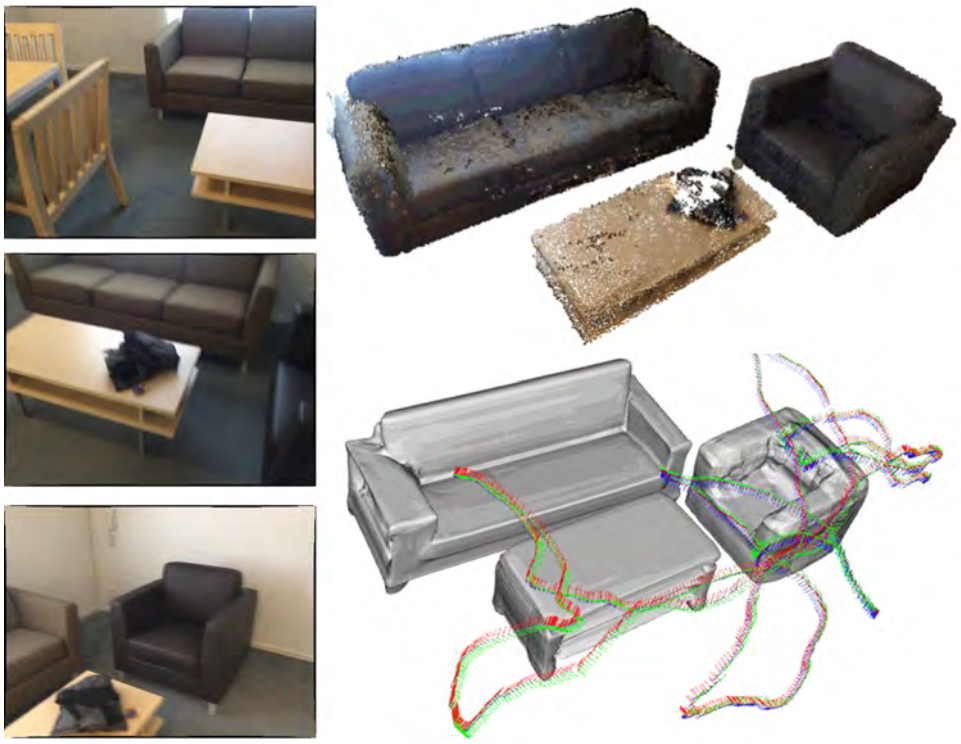

Figure 16.9 Qualitative results of 3D object pose and implicit SDF shape reconstruction on the ScanNet dataset [245] (scene 0087) using ELLIPSDF [1002]. The figure shows RGB images from the scene (left), a ground-truth colored point cloud reconstruction (top right), and reconstructed object meshes using SDF models decoded from the object shape codes and optimized SIM(3) poses (bottom right).

scale λ_i are known, and the residual in (16.15) is minimized with respect to θ and $\boldsymbol{s}_i$. The decoder parameters θ are usually the same for a whole object category, while the shape code $\boldsymbol{s}_i$ is optimized separately for each object instance. In testing, one usually assumes that the decoder parameters θ are known, and the residual in (16.15) is minimized with respect to the state $\boldsymbol{x}_i$, consisting of $\boldsymbol{T}_i$ and $\boldsymbol{q}_i$, λ_i, and $\boldsymbol{s}_i$.

An example of object SLAM with the ELLIPSDF algorithm [1002], using an implicit representation of both coarse (ellipsoid) and fine (SDF) object shape, applied to the real-world ScanNet dataset [245] is shown in Fig. 16.9.

16.2.2 Hybrid Solvers for Sparse Metric-Semantic SLAM

The representation of object-based or *sparse* metric-semantic SLAM combines *discrete* information in the form of semantic object *classes* with *continuous* information, like the position, orientation, and shape of an object. Consequently, solving metric-semantic SLAM problems often requires reasoning jointly about these states which are often coupled in intricate ways. For example: the semantic class of an object might inform us about the shape of that object (*e.g.,* as in [999]) and vice versa. Discrete variables enter the sparse metric-semantic SLAM problems in other ways, too, such as in the case where objects of interest have some discrete symmetries [713]. When using a learning-based front-end system for object detection and pose estimation, like a neural network, occasionally the front end will provide multiple discrete pose hypotheses we would like to track (see, e.g. [352]). Crucially, much like in the case of landmark-based SLAM with purely geometric features, *data association* (see Chapter 3) is a key challenge when developing systems for sparse metric-semantic SLAM.

The introduction of discrete states into the SLAM problem greatly increases its practical difficulty. When formulated as an optimization problem, metric-semantic SLAM in its most general form is nonlinear and nonconvex, just as in purely metric SLAM, but also combinatorial, involving search over a state space whose size can be exponentially large in the number of discrete states to be estimated.

Given a model of object detections and classifications as the output of a (noisy) sensor, the problem of jointly estimating the latent semantic class and geometry of landmarks in the environment can be posed in terms of MAP inference (*cf.* Chapter 1):

$$\hat{X}, \hat{L}, \hat{D} = \arg \max_{X,L,D} p(X, L, D \mid Z), \tag{16.16}$$

where Z denotes the full set of measurements (including semantic measurements); X the set of robot poses; L the set of environmental landmarks, which typically consist of some geometric information (*e.g.,* position, orientation, size, and shape) coupled with a discrete semantic label from a known, fixed set of classes; and D the set of associations between measurements in Z and landmarks in L. A key observation is that discrete-valued categorical information about objects can be naturally combined with the already discrete inference problem of data association: the knowledge of an object's category can help distinguish it in clutter from other objects. This formulation unifies discrete models of semantic category, geometric estimation, and data association; however, in addition to being nonconvex and high-dimensional (as in the standard SLAM formulation), it now also involves combinatorial optimization. Moreover, in committing to the use of semantics for data association, one must cope with the errors of learned perception models.

In general, the MAP inference problem in (16.18) is computationally intractable [600, Section 13.1.1]. Indeed, even the purely continuous estimation problems arising

in robot perception are typically NP-hard (see Chapter 6 for a broader discussion). Despite this, smooth (local) optimization methods often perform quite well on such problems, both in their computational efficiency (owing to the fact that gradient computations are typically inexpensive) and quality of solutions when a good initialization can be supplied. However, even if we assume the ability to efficiently solve continuous estimation problems, the introduction of discrete variables complicates matters considerably: in the worst-case, solving for the joint MAP estimate globally requires that for each assignment to the discrete states we solve a continuous optimization subproblem, and discrete state spaces grow *exponentially* in the number of discrete variables under consideration. Consequently, efficient approximate solutions are needed.

It will be useful to consider the representation of eq. (16.16) in terms of its factorization as a *hybrid factor graph*:

$$\begin{aligned} p(\Theta, D \mid Z) &\propto \prod_k \phi_k(\mathcal{V}_k), \\ \mathcal{V}_k &\triangleq \{v_i \in \mathcal{V} \mid (\phi_k, v_i) \in \mathcal{E}\}, \end{aligned} \tag{16.17}$$

where we grouped continuous variables Θ and discrete variables D, and each factor ϕ_k is in correspondence with either a measurement likelihood of the form $p(z_k \mid \mathcal{V}_k)$ or a prior $p(\mathcal{V}_k)$. In (16.17), $\mathcal{V}_k$ is the set of (continuous or discrete) variables involved in factor ϕ_k. In particular, the posterior $p(\Theta, D \mid Z)$ can be decomposed into factors ϕ_k of three possible types: *discrete* factors $\phi_k(D_k)$ where $D_k \subseteq D$, *continuous* factors $\phi_k(\Theta_k)$, $\Theta_k \subseteq \Theta$, and *discrete-continuous* factors $\phi_k(\Theta_k, D_k)$. In turn, the maximum *a posteriori* inference problem (from (16.16)) can be posed as follows:

$$\begin{aligned} \Theta^*, D^* &= \arg\max_{\Theta, D} p(\Theta, D \mid Z) \\ &= \arg\max_{\Theta, D} \prod_k \phi_k(\mathcal{V}_k) \\ &= \arg\min_{\Theta, D} \underbrace{\sum_k -\log \phi_k(\mathcal{V}_k)}_{\triangleq \mathcal{L}(\Theta, D)} . \end{aligned} \tag{16.18}$$

That is to say, we can maximize the posterior probability $p(\Theta, D \mid Z)$ by minimizing the negative log posterior, which in turn decomposes as a summation.

It is common to consider hybrid estimation problems that can be represented in terms of nonlinear least-squares problems, which will allow the use of the tools developed in Chapter 1, like iSAM2 [540], for the estimation of continuous states of interest. In particular, we can consider discrete-continuous factors $\phi_k(\Theta_k, D_k)$ admitting a description as:

$$\begin{aligned} -\log \phi_k(\Theta_k, D_k) &= \|r_k(\Theta_k, D_k)\|_2^2, \\ \Theta_k \subseteq \Theta,\ D_k &\subseteq D, \end{aligned} \tag{16.19}$$

where r_k is typically nonlinear in Θ and first-order differentiable with respect to Θ, and the equality may be up to a constant independent of Θ and D. Likewise, we consider factors involving only continuous variables admitting an analogous representation.

With this formulation, we can leverage the conditional independence structure of the factor graph model to develop an efficient local inference method. First, note that if we fix any assignment to the discrete states, the only variables remaining are continuous and approximate inference can be performed efficiently using smooth optimization techniques. In this sense, if we happened to know the assignment to the discrete variables, continuous optimization becomes "easy." On the other hand, if we fix an estimate for the continuous variables, we are left with an optimization problem defined over a discrete factor graph which can be solved to global optimality using max-product variable elimination. The latter still requires exploration of *exponentially many* discrete states in the worst case, but also opens the door to well-established heuristics to compute approximate estimates.

In particular, consider a partition of the discrete states into mutually exclusive subsets $D_j \subseteq D$ which are *conditionally independent* given the continuous states:

$$p(D \mid \Theta, Z) \propto \prod_j p(D_j \mid \Theta, Z). \tag{16.20}$$

It is straightforward to verify from the mutual exclusivity of each set D_j that the problem of optimizing the conditional in (16.20) then breaks up into subproblems involving each D_j:

$$\max_D p(D \mid \Theta, Z) \propto \prod_j \left[\max_{D_j} p(D_j \mid \Theta, Z) \right]. \tag{16.21}$$

Critically, we have exchanged computation of the maximum of the product with the product of each maximum computed *independently*. In cases where the discrete states decompose into particularly small subsets ($|D_j| \ll |D|$), inference may be carried out efficiently.

Many hybrid optimization problems in robotics admit factorizations of the form in (16.20). For example, point-cloud registration (with discrete data association variables), robust pose-graph optimization (with switch variables for outlier rejection), and metric-semantic SLAM (with data association variables and discrete object classes) can all be formulated in a way that admits this relatively "friendly" decomposition (see, e.g. [284, 626]).

We can make use of this idea in a few ways. For example, DC-SAM [284] makes use of alternating minimization by first, fixing an initial iterate $\Theta^{(i)}$. Then, DC-SAM attempts to solve the following subproblems:

$$D^{(i+1)} = \arg\min_D \mathcal{L}(\Theta^{(i)}, D) \tag{16.22a}$$

$$\Theta^{(i+1)} = \arg\min_{\Theta} \mathcal{L}(\Theta, D^{(i+1)}). \tag{16.22b}$$

We may then repeat (16.22a) and (16.22b) until the relative decrease in $\mathcal{L}(\Theta, D)$ is sufficiently small or we have reached a maximum desired number of iterations. In practice, each subproblem need not be solved to optimality. Rather, the usual techniques for optimization are used for the continuous subproblem in eq. (16.22b). Moreover, it is possible to generalize this two-stage block coordinate descent strategy by taking steps with respect to smaller groups of variables, trading off the per-step complexity with the number of optimization steps taken.

Bowman et al. [114] adopted a similar strategy for sparse metric-semantic SLAM. They performed joint optimization via expectation maximization (EM): First, we fix the robot poses and landmark locations to compute data association *probabilities* and landmark classes (the *E-step*). Next, we fix the data association probabilities and landmark classes and optimize the robot poses and landmark locations, with measurements weighted by the respective probabilities of their landmark correspondence (the *M-step*). This approach assigns "soft" associations to objects, gradually converging to a locally optimal solution of (16.16). It has also been shown that the probabilities in the E-step can be exactly recovered using a matrix permanent computation. By approximating the matrix permanent, we can in turn approximate the data association probabilities needed in the M-step more efficiently [47].

Multi-hypothesis methods aim to solve the optimization in eq. (16.18) by explicitly searching over possible assignments to the discrete states in D. Since the size of the search space is exponentially large in the number of discrete states, these methods use heuristics to prune the search space in order to remain computationally tractable. General approaches like MH-iSAM2 [487] and iMHS [526] have been applied to hybrid estimation problems like robust pose-graph optimization and contact estimation for legged robots, but have not yet been applied to sparse metric-semantic SLAM. Bernreiter et al. [81] present a method tailored to metric-semantic mapping and localization that makes use of a multi-hypothesis representation.

An alternative approach to the combinatorial inference problem of Equation (16.16) is to reframe the optimization over discrete variables as one over only continuous-valued variables. In early work to this end, Sünderhauf et al. [1066] optimized probabilities of semantic labels, which are defined over the $(K-1)$-dimensional unit simplex for a K-class semantic labeling problem. This approach is similar to the strategies used in Chapter 3 for outlier rejection, but with the extension to multiple labels. Similarly, there are methods that attempt to approximate the full posterior in (16.16) with all of the discrete states marginalized out, resulting in a mixture. Since this posterior distribution is generally non-Gaussian, approximation methods are used in practice (see, *e.g.,* [345, 498] for non-Gaussian solvers targeted toward SLAM applications, and [282, 48] for applications to metric-semantic SLAM).

16.3 Dense Metric-Semantic Representations

In this section, we discuss dense metric-semantic representations, which extend the representations discussed in Chapter 5 to include semantic information. We start with point and surfel map representations that utilize semantic information in Section 16.3.1. Next, we discuss how voxel map representations, such as the widely used OctoMap [486], can be extended to capture semantic information in a probabilistically consistent way in Section 16.3.2. Finally, in Section 16.3.3, we discuss mesh representations which can be less expensive to store and easier to deform in response to loop closures in comparison to voxel maps.

16.3.1 Point-based and Surfel-based Metric-Semantic SLAM

In this section, we focus on metric-semantic point cloud and surfel map representations. The availability of large-scale densely annotated LiDAR datasets [75, 143, 76, 338] made it possible to learn semantic segmentation [774] and panoptic segmentation [773] directly on point clouds. As such models provide point-wise semantics and potentially also instance information, several LiDAR SLAM approaches [192, 659, 529] exploited this semantic information directly to build metric-semantic map representations. As we already mentioned earlier in this chapter, capturing semantic information allows identifying dynamic objects and improving data association, either by exploiting semantics in finding correspondences for odometry [659, 529] or building a semantic scene descriptor for loop closure detection [601, 529].

A key challenge addressed by these methods is how to perform spatial integration of the potentially conflicting per-frame semantic segmentation of LiDAR point clouds into a unified map representation. To this end, one of the first LiDAR-based dense SLAM approach that uses a semantic segmentation, called SuMa++ [192], enriched the surfel-based representation with a per-surfel semantic classes and confidence score of the surfel semantics. Using the semantic confidence together with the measurement uncertainty, Chen *et al.* [192] update the surfel confidence score such that conflicting semantic labels will lead to reduced surfel confidences. The confidence of a surfel is ultimately used to filter out low-confidence surfels that are either caused by measurement uncertainty or semantic uncertainty.

By integrating the semantics into the surfel-based map representation, the approach is able to account for conflicting semantic labels and equipped with this capability, the semantic map representation can more effectively filter dynamic objects. Here, Chen *et al.* [192] showed that simply removing all potentially moving objects just based on the semantics is inferior to an approach that accounts only for conflicting semantic labels between the map and the current semantic interpretation of the scene. Chen *et al.* [192] attribute this to the prevalence of vehicles in urban environments, which also can be often used for pose estimation. Thus, re-

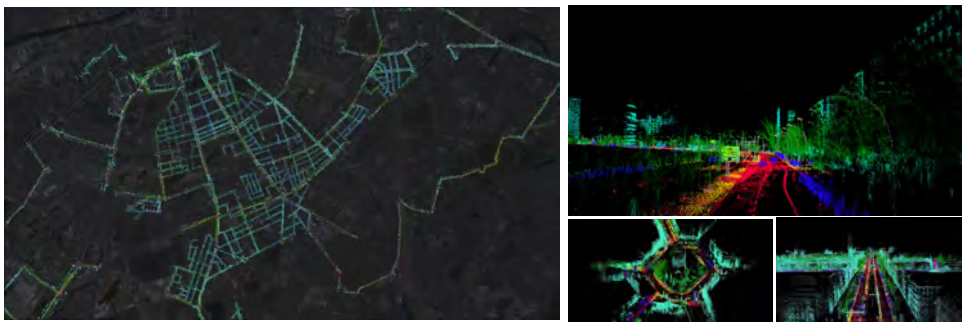

Figure 16.10 Large-scale semantic map of Berlin covering 8000 km of roads, generated by a fleet of vehicles using the method in [201] (left). Zoomed-in sections of the semantic map show fine-grained 3D reconstruction details (right).

moving simply all potentially moving objects will also remove valuable information from the map representation that could be used for ICP alignment.

Follow-up approaches, like SA-LOAM [659] and SELVO [529], integrate semantics in the pose estimation by matching only semantically compatible features [1258], but also exploit semantics in the process of determining loop closures [601].

Building on LOAM [1264], Li et al. [659] use a point-based representation to store features for surface-like and corner-like structures in distinct semantic maps. For loop closures, the approach of this work uses a semantic graph generated from a point-wise semantic segmentation to match scenes using the semantics.

Similarly, the approach of Jiang et al. [529] also only accounts for correspondences of matching semantic classes, but extend the well-known ScanContext descriptor to account for the semantic in the matching process of scene descriptors.

An example of a large-scale semantically annotated point-cloud map obtained using stereo camera images only is shown in Fig. 16.10. The approach [201] uses dense 2D semantic labels from the stereo camera images and combines a direct sparse visual odometry front-end with a global optimization back-end.

16.3.2 Voxel-based Metric-Semantic SLAM

Dense environment models utilizing voxels have been widely used as alternatives to point cloud mapping [103, 1069, 1128, 355]. Let $\mathcal{E}$ denote the environment in which a robot is navigating. As we have seen in Chapter 5, a voxel map $\boldsymbol{m}$ is a partitioning of $\mathcal{E}$ using a grid of non-overlapping cubes (*i.e.,* voxels), where each voxel stores a set of environment attributes, such as occupancy, temperature, etc., that are assumed to be constant within the corresponding physical 3D space. This section discusses how to augment voxel-based representations with semantic information, in addition to occupancy or distance information.

The first step towards extending voxel-based representations to metric-semantic

mapping is the inclusion of semantics into robot observations. Let the environment $\mathcal{E}$ be divided into a collection of disjoint sets $\mathcal{E}_k \subset \mathbb{R}^3$, each associated with a semantic category $k \in \mathcal{K} := \{0, \dots, K\}$, with $\mathcal{E}_0$ denoting the free space and each $\mathcal{E}_k$ for $k > 0$ represents a different object class such as building, cars, and terrain. Consider a robot equipped with a sensor that provides information about the distance to and semantic categories of surrounding objects along a set of rays $\{\eta_b\}$, where b is the ray index, $\eta_b \subset \mathbb{R}^3$ with $\|\eta_b\| = r_{\max}$, and $r_{\max}$ is the maximum sensing range. Note that $\{\eta_b\}$ is just a collection of unit vectors that represent the rays of sensor such as an RGBD camera or a LiDAR. We can associate a semantically annotated point measurement with each ray by processing camera [772] or LiDAR [774] measurements with a semantic segmentation algorithm. More concretely, for a robot with orientation $\boldsymbol{R}_t \in \mathrm{SO}(3)$ and position $\boldsymbol{p}_t \in \mathbb{R}^3$ at time t, a *range-category observation* is defined as a collection $\mathcal{Z}_t = \{\boldsymbol{z}_{t,b}\}_b$ of range and category measurements $\boldsymbol{z} := (r_{t,b}, y_{t,b}) \in \mathbb{R}_{\geq 0} \times \mathcal{K}$ along the sensor rays $\boldsymbol{R}_t \eta_b + \boldsymbol{p}_t$. Fig. 16.11 shows an example where each pixel in an RGB image corresponds to a ray η_b, while its corresponding values in the semantic segmentation and range images are the category $y_{t,b}$ and range $r_{t,b}$, respectively. The goal of voxel-based metric-semantic mapping is to incrementally construct a multi-class map $\boldsymbol{m}$ of the environment $\mathcal{E}$ based on streaming range-category observations. The map $\boldsymbol{m}$ is modeled as a grid of cells $i \in \mathcal{I} := \{1, \dots, N\}$, each labeled with a category $m_i \in \mathcal{K}$.

Consider the PDF $p(\mathcal{Z}_t | \boldsymbol{m}, \boldsymbol{R}_t, \boldsymbol{p}_t)$ that models noise in sensor observations. This observation model allows integrating the measurements into a probabilistic map representation using Bayesian updates. Let $p_t(\boldsymbol{m}) := p(\boldsymbol{m} | \mathcal{Z}_{1:t}, \boldsymbol{R}_{1:t}, \boldsymbol{p}_{1:t})$ be the probability mass function (PMF) of the map $\boldsymbol{m}$ given the robot trajectory $(\boldsymbol{R}_{1:t}, \boldsymbol{p}_{1:t})$ and observations $\mathcal{Z}_{1:t}$ up to time t. Given a new observation $\mathcal{Z}_{t+1}$ obtained from robot pose $(\boldsymbol{R}_{t+1}, \boldsymbol{p}_{t+1})$, the Bayesian update of the map is:

$$p_{t+1}(\boldsymbol{m}) \propto p(\mathcal{Z}_{t+1} | \boldsymbol{m}, \boldsymbol{R}_{t+1}, \boldsymbol{p}_{t+1}) p_t(\boldsymbol{m}). \tag{16.23}$$

For brevity, we omit the dependence of the map distribution and the observation model on the robot pose throughout the rest of this analysis.

The work by Asgharivaskasi and Atanasov [44] presents an online voxel-based semantic mapping technique by generalizing the log-odds occupancy mapping algorithm [1092, Ch. 9] to multi-class maps. In particular, an explicit derivation for the Bayesian update in (16.23) is obtained using a multinomial logit model to represent the map PMF $p_t(\boldsymbol{m})$ where each cell m_i of the map stores the probability of object classes in $\mathcal{K}$. To ensure linear model complexity with respect to the map size N, a factorized PMF is maintained over the map cells:

$$p_t(\boldsymbol{m}) = \prod_{i=1}^{N} p_t(m_i).$$

The authors in [44] introduce *semantic log-odds* as a vector that represents the

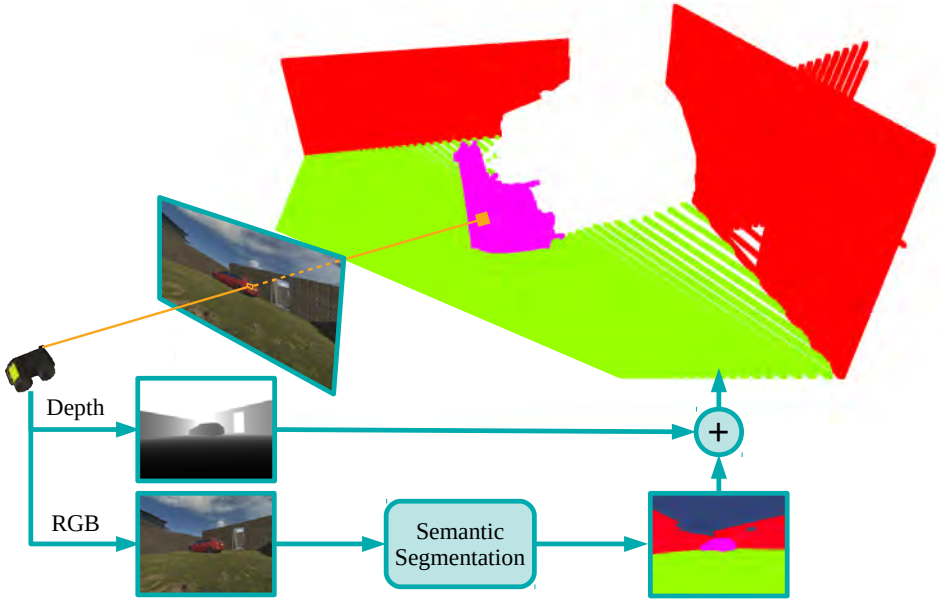

Figure 16.11 Creation process of a range-category observation. The robot is equipped with an RGB-D sensor, where the RGB part of the sensor measurement is fed to a semantic segmentation algorithm. Given the camera intrinsics, the semantic segmentation and depth images are combined to form a semantically-annotated 3D point cloud.

individual cell PMFs $p_t(m_i)$ over the semantic categories $\mathcal{K}$:

$$\boldsymbol{h}_{t,i} := \begin{bmatrix} \log \frac{p_t(m_i=0)}{p_t(m_i=0)} & \dots & \log \frac{p_t(m_i=K)}{p_t(m_i=0)} \end{bmatrix}^\top \in \mathbb{R}^{K+1},$$

where the free-class likelihood $p_t(m_i = 0)$ is used as a pivot. Given the log-odds vector $\boldsymbol{h}_{t,i}$, PMF of cell m_i may be recovered using the softmax function $\sigma : \mathbb{R}^{K+1} \mapsto \mathbb{R}^{K+1}$:

$$p_t(m_i = k) = \sigma_{k+1}(\boldsymbol{h}_{t,i}) := \frac{\boldsymbol{e}_{k+1}^\top \exp(\boldsymbol{h}_{t,i})}{\mathbf{1}^\top \exp(\boldsymbol{h}_{t,i})},$$

where $\boldsymbol{e}_k$ is the standard basis vector with k^{th} element equal to 1 and 0 elsewhere, $\mathbf{1}$ is the vector with all elements equal to 1, and $\exp(\cdot)$ is applied elementwise to the vector $\boldsymbol{h}_{t,i}$. The Bayesian update in (16.23) for $\boldsymbol{h}_{t,i}$ can then be obtained in terms of the range-category observation model, evaluated at a new measurement set $\mathcal{Z}_{t+1}$:

$$\boldsymbol{h}_{t+1,i} = \boldsymbol{h}_{t,i} + \sum_{\boldsymbol{z} \in \mathcal{Z}_{t+1}} \boldsymbol{l}_i(\boldsymbol{z}), \tag{16.24}$$

where $\boldsymbol{l}_i(\boldsymbol{z})$ is the observation model log-odds:

$$\boldsymbol{l}_i(\boldsymbol{z}) := \begin{bmatrix} \log \frac{p(\boldsymbol{z}|m_i=0)}{p(\boldsymbol{z}|m_i=0)} & \dots & \log \frac{p(\boldsymbol{z}|m_i=K)}{p(\boldsymbol{z}|m_i=0)} \end{bmatrix}^\top. \tag{16.25}$$

To complete the Bayesian multi-class mapping equation in (16.24) we need a particular observation model. When a sensor measurement is generated, the sensor ray continues to travel until it hits an obstacle of category $\mathcal{K} - \{0\}$ or reaches the maximum sensing range $r_{\max}$. The labeled range measurement $\boldsymbol{z} = (r, y)$ obtained from position $\boldsymbol{p}$ with orientation $\boldsymbol{R}$ indicates that map cell m_i is occupied if the measurement end point $\boldsymbol{p} + \frac{r}{r_{\max}}\boldsymbol{R}\boldsymbol{\eta}$ lies in the cell. If m_i lies along the sensor ray but does not contain the end point, it is observed as free. Finally, if m_i is not intersected by the sensor ray, no information is provided about its occupancy. A consistent observation model should hence reflect such properties in its definition, either through maually tuning [44] or learning form data [1163]. For example, the work in [44] parameterizes the observation model log-odds vector in (16.25) as a piecewise constant function along the sensor ray:

$$\boldsymbol{l}_i((r, y)) := \begin{cases} \boldsymbol{\phi}^+ + \boldsymbol{E}_{y+1}\boldsymbol{\psi}^+, & r \text{ indicates } m_i \text{ is occupied,} \\ \boldsymbol{\phi}^-, & r \text{ indicates } m_i \text{ is free,} \\ 0, & \text{otherwise,} \end{cases} \tag{16.26}$$

where $\boldsymbol{E}_k := \boldsymbol{e}_k\boldsymbol{e}_k^\top$ and $\boldsymbol{\psi}^+, \boldsymbol{\phi}^-, \boldsymbol{\phi}^+ \in \mathbb{R}^{K+1}$ are parameter vectors, whose first elements are 0 to ensure that $\boldsymbol{l}_i(\boldsymbol{z})$ is a valid semantic log-odds vector. Fig. 16.12 illustrates the proposed Bayesian multi-class mapping method using the observation model of (16.26).

It is important to note that utilizing a regular-grid discretization to represent $\mathcal{E}$ has prohibitive storage and computation requirements. Large continuous portions of many real environments are unoccupied, suggesting that adaptive discretization is significantly more efficient. A popular method to improve the memory efficiency of voxel maps is proposed by Hornung *et al.* [486], called *octomap*, in which the octree data structure is utilized to combine voxels with equal occupancy probability. An octree is a hierarchical data structure containing nodes that represent a section of the physical environment. Each node has either 0 or 8 children, where the latter corresponds to the 8 octants of the Euclidean 3D coordinate system. Thus, the children of a parent node form an eight-way octant partition of the space associated with the parent node. An octree node that does not have any children is called a *leaf* node, which provides the highest-resolution visualization of an octree map. In [44], the authors extend the probabilistic 3D mapping technique of octomap to metric-semantic representations through introduction of multi-class octree. A multi-class octree is a type of octree data structure where each node stores a categorical probability distribution over the set of object classes $\mathcal{K}$ in the form of a semantic log-odds vector $\boldsymbol{h}_{i,t}$. Fig. 16.13 shows an example of a multi-class octree data structure.

In order to register a new observation, a ray-casting procedure over an octree

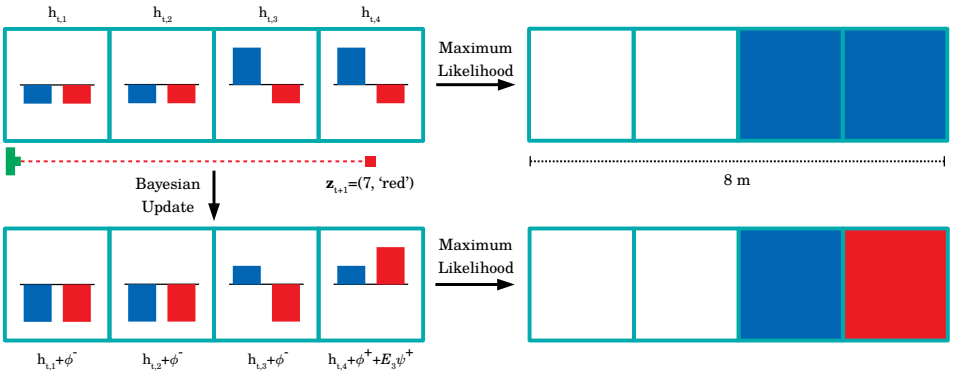

Figure 16.12 Illustration of the Bayesian multi-class mapping using a range-category sensor measurement, where the object classes are defined as $\mathcal{K} = \{\text{'free'}, \text{'blue'}, \text{'red'}\}$. The top-left shows the semantic log-odds vector at time t for each map cell along an observation ray, shown as the dotted red line. The top-left represents a maximum likelihood estimation of the map at time t, based on semantic log-odds vector. The range-category observation $\boldsymbol{z}_{t+1}$ contain a range measurement of 7 meters and classification of 'red' at the point of incidence. The bottom row shows the updated semantic log-odds vectors alongside the maximum likelihood category estimations for each cell at time $t+1$.

(e.g. [434] or [23]) is performed to find the observed leaf nodes. Then, for each observed leaf node, if the observation demands an update, the leaf node is recursively expanded to the smallest resolution and the semantic log-odds of the downstream nodes are updated using (16.24). To obtain a compressed octree, it is necessary to define a rule for information fusion from child nodes towards parent nodes. Depending on the application, different information fusion strategies may be implemented. For example, a conservative strategy would assign the semantic log-odds of the child node with the highest occupancy probability to the parent node. Alternatively, one can simply assign the average semantic log-odds vector of the child nodes to their parent node, which is equivalent to the Bayesian fusion of information in the original probability space. The benefit of an octree representation is the ability to combine similar cells (leaf nodes) into a large cell (inner node). This is called *pruning* the octree. Every time after an observation is integrated to the map, the tree nodes are checked in a bottom-up manner to identify pruning opportunities. If the children of an inner node are all leaf nodes and have equal semantic log-odds, the children are pruned and the inner node is converted into a leaf node with the same semantic log-odds as its children. This helps to compress the majority of the free cells into a few large cells, while the occupied cells usually do not undergo pruning since only their surfaces are observed by the sensor and their inside remains an unexplored region. Fig. 16.14 displays the update and pruning procedure for a multi-class octree.

Implementations of multi-class octree mapping, such as the one in [44], consider

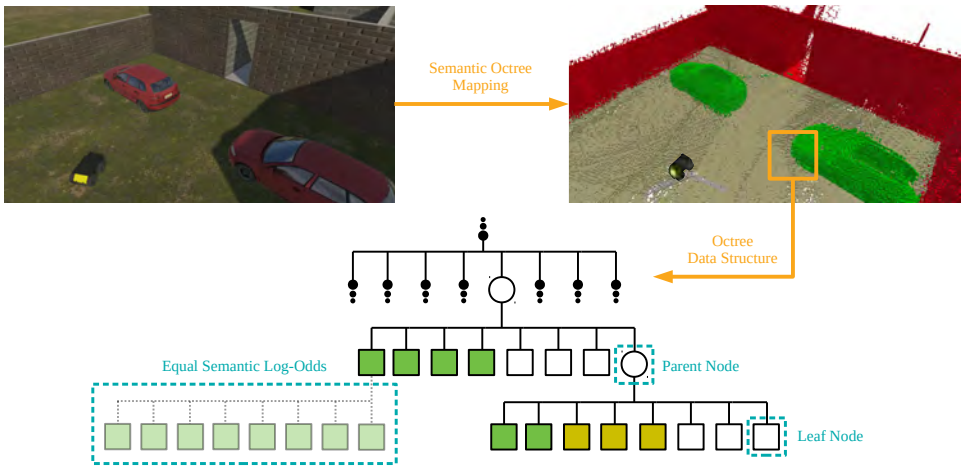

Figure 16.13 An instance of multi-class octree mapping. The top-left shows the environment in which a robot with range-category sensor performs mapping. The top-right shows the resulting multi-class octree map, where each class is shown with a distinct color (free class is transparent). The bottom depicts the octree data structure corresponding to a section of the map. The circle and square nodes represent *parent* and *leaf* nodes, respectively. Each leaf node either belongs to the smallest resolution of the octree, or contains children with identical semantic log-odds vectors.

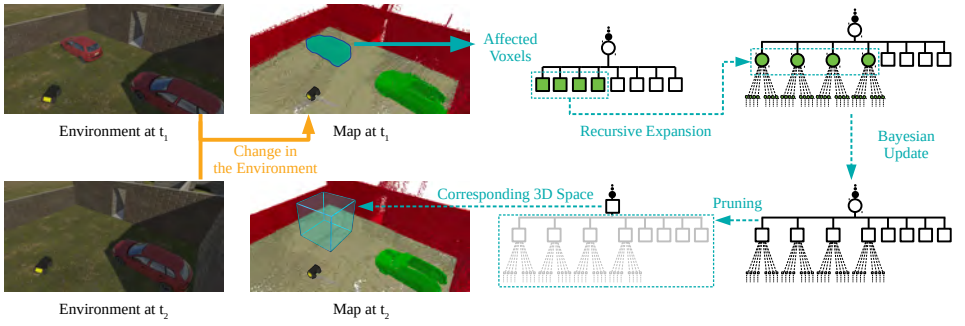

Figure 16.14 Multi-class octree map update. A change in the environment between $t = t_1$ and $t = t_2$ triggers a map update. Through ray-casting, the voxels highlighted in cyan need to be updated. For this aim, the corresponding nodes in the octree are recursively expanded, and updated via (16.24). Simultaneously, pruning opportunities are found for the children of each parent node. The pruning process leads to smaller, but physically larger, octree nodes, shown as the cyan cube for the map at $t = t_2$.

additional design choices in order to enable real-time performance. Foe example, due to sensor noise, it is unlikely that cells belonging to the same class (*e.g.,* free or occupied by the same obstacle) attain identical semantic log-odds. Maximum

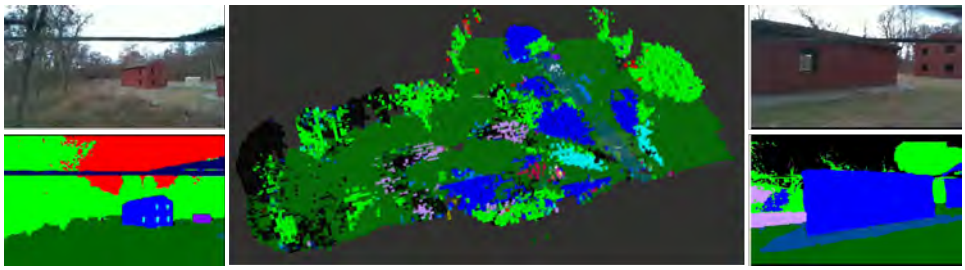

Figure 16.15 Qualitative results from a real-world SLAM experiment in which two quadrotor robots construct a multi-class octree map using the ROAM algorithm [45]. RGB and semantically segmented images from RGBD cameras onboard the two robots are shown on the left and right. The middle plot shows the resulting multi-class octree map.

and minimum limits for the elements of the semantic log-odds are used so that each cell arrives at a stable state as its semantic log-odds entries reach the limits. Stable cells are more likely to share the same multi-class probability distribution, consequently increasing the chance of octree pruning. However, thresholding causes loss of information near $p_t(m_i = k) = 1$, $k \in \mathcal{K}$ which can be controlled by the maximum and minimum limits. Furthermore, the space and computational complexity of multi-class map updates increases linearly with respect to the number of class labels K, while the probability of pruning reduces exponentially as K grows. Hence, one may store the semantic log-odds for only the $\bar{K}$ most likely class labels, and lump the rest of the labels into a single *others* variable. This encourages more frequent pruning and significantly reduces the computation complexity during map updates in cases with many semantic categories.

Several robots can construct a multi-class octree map collaboratively. Riemannian Optimization for Active Mapping (ROAM) [45] formulates an optimization problem over a graph with node variables representing octree maps of different robots and a consensus constraint requiring the robots to achieve agreement on their local maps. ROAM is based on distributed Riemannian optimization relying only only on one-hop communication to achieve consensus and optimality guarantees. An example of multi-class octree mapping with the ROAM algorithm [1002] applied to real-world data is shown in Fig. 16.15.

16.3.3 Mesh-based Metric-Semantic SLAM

While voxel-based maps provide an effective representation to probabilistically fuse information over time, they have two downsides. First, they tend to be expensive to store (an issue that can be partially mitigated using octree data structures). Second, they cannot be easily edited in response to loop closures: loop closures entail large deformations in the robot trajectory estimate and the corresponding map,

and voxel-based maps are expensive to update in response to these deformations. Updating dense volumetric representations typically involves de-integrating and re-integrating previous observations, which is computationally expensive (*e.g.,* [246]).

In this section we discuss mesh-based representations as a way to circumvent both issues. Meshes are collections of polygons (typically triangles) and their storage complexity scales with the complexity of the scene being represented. Rather than being directly built from sensor data, it is fairly common to build a voxel-based representation first, and then compute a mesh from it using standard algorithms (*e.g.,* marching cubes). Advanced approaches, such as [948], build on ideas used in geometric SLAM [1180], and only build a voxel-based map in a spatial window around the robot, while the voxel-based representation is incrementally converted into a mesh as voxels leave this active window. More interestingly, the mesh representation enables to efficiently deform the map in response to loop closures, hence providing a more computationally effective way to correct the map compared to voxel de-integration and re-integration. We now review how to deform mesh-based representations, an approach called Pose-Graph and Mesh Optimization (PGMO) [948].

PGMO builds on the notion of *embedded deformation graph*, which was first introduced in the context of shape manipulation in computer graphics [1054]. The deformation graph is a collection of *control nodes*, obtained by downsampling the origin mesh. Each control node is attached a local coordinate frame and the overall graph can be deformed by solving an optimization problem that enforces local rigidity of the graph edges. Then, after solving this optimization that computes the deformation of the control nodes, the rest of the mesh is interpolated back, using the newly found control nodes' configurations. We formalize the approach below.

For dense 3D metric-semantic SLAM, a deformation graph can be created from the pose graph of robot poses and a downsampled version of the dense mesh. The pose vertices of the deformation graph are defined as the nodes of the robot pose graph with associated transformation $\boldsymbol{X}_i = [\boldsymbol{R}_i^X, \boldsymbol{t}_i^X]$, and are connected according to the connectivity of the pose graph. The mesh vertices of the deformation graph are defined as the vertices of the downsampled mesh with associated transformation $\boldsymbol{M}_k = [\boldsymbol{R}_k^M, \boldsymbol{t}_k^M]$ for mesh vertex k, and are connected according to the edges of the simplified mesh. A pose vertex i is connected to the mesh vertex k if the mesh vertex k is visible to the sensor associated to pose i. Fig. 16.16 shows the components of the deformation graph.

In the undeformed state, $\boldsymbol{X}_i$ is the unoptimized pose for node i, and $\boldsymbol{R}_k^M = \mathbf{I}_3$ and $\boldsymbol{t}_k^M = \boldsymbol{g}_k$, where $\boldsymbol{g}_k$ is the original world frame position of vertex k. Intuitively, these transformations describe the local deformations on the mesh: $\boldsymbol{R}_k^M$ is the local rotation centered at vertex k while $\boldsymbol{t}_k^M - \boldsymbol{g}_k$ is the local translation. Once deformation is triggered (*e.g.,* loop closure detected), the deformation graph is optimized as

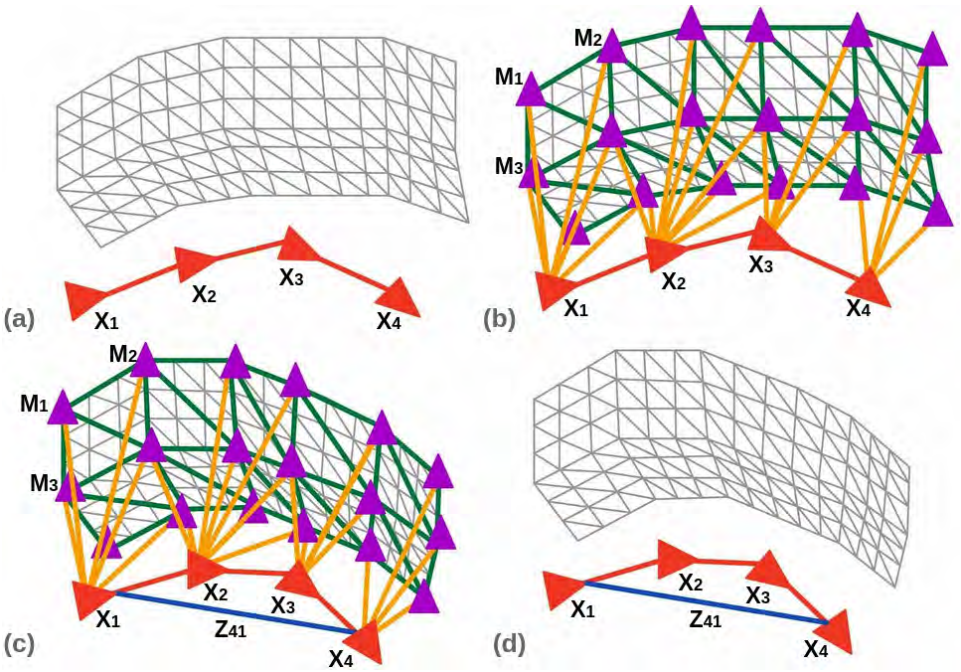

Figure 16.16 PGMO deformation. (a) Un-deformed pose graph and mesh. (b) Creating a deformation graph. Pose vertices are shown in red and mesh vertices are shown in purple. The green edges are the edges describing the connectivity of the simplified mesh. The yellow edges are the edges connecting the pose vertices to the mesh vertices based on visibility from the sensor. (c) A loop closure triggers the pose graph and mesh optimization. $\boldsymbol{X}_i$ and $\boldsymbol{M}_i$ are updated based on the optimization results and the dense mesh deformed. (d) The resulting deformed mesh and pose graph.

follows,

$$\begin{aligned}\underset{\substack{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_n\in\mathrm{SE}(3)\\ \boldsymbol{M}_1,\ldots,\boldsymbol{M}_m\in\mathrm{SE}(3)}}{\arg\min} & \sum_{\boldsymbol{Z}_{ij}} \left\|\boldsymbol{X}_i^{-1}\boldsymbol{X}_j - \boldsymbol{Z}_{ij}\right\|^2_{\boldsymbol{\Omega}_{ij}} \\ & + \sum_{k=0}^{m} \sum_{l\in\mathcal{N}^M(k)} \left\|\boldsymbol{R}_k^M(\boldsymbol{g}_l - \boldsymbol{g}_k) + \boldsymbol{t}_k^M - \boldsymbol{t}_l^M\right\|^2_{\boldsymbol{\Omega}_{kl}} \\ & + \sum_{i=0}^{n} \sum_{l\in\mathcal{N}^M(i)} \left\|\boldsymbol{R}_i^X\tilde{\boldsymbol{g}}_{il} + \boldsymbol{t}_i^X - \boldsymbol{t}_l^M\right\|^2_{\boldsymbol{\Omega}_{il}}, \end{aligned} \tag{16.27}$$

where $\mathcal{N}^M(i)$ indicates the neighboring mesh vertices to a vertex i in the deformation graph, $\boldsymbol{g}_i$ denotes the non-deformed (initial) position of mesh or pose vertex i in the deformation graph, and $\tilde{\boldsymbol{g}}_{il}$ denotes the non-deformed position of vertex l in the coordinate frame of the odometric pose of node i.

Notice that the first term of (16.27) follows from generic pose graph optimization, the second term is enforces local rigidity between mesh vertices by preserving the edge connecting two mesh vertices, and the third term enforces the local rigidity between a pose vertex i and a mesh vertex l. Note that $\|\cdot\|^2_{\boldsymbol{\Omega}}$ indicates the (squared) weighted Frobenius norm

$$\|\boldsymbol{X}\|^2_{\boldsymbol{\Omega}} = \operatorname{tr}\left(\boldsymbol{X}^T\boldsymbol{\Omega}\boldsymbol{X}\right). \tag{16.28}$$

Following [948], we can show (16.27) can be formulated as a pose graph optimization problem. Defining $\tilde{\boldsymbol{R}}_i$ as the initial rotation of pose vertex i in the deformation graph, we can introduce $\boldsymbol{G}_{ij}$ and $\bar{\boldsymbol{G}}_{ij}$ such that

$$\begin{aligned}\boldsymbol{G}_{ij} &= \begin{bmatrix}\boldsymbol{I}_3 & \boldsymbol{g}_j - \boldsymbol{g}_i \\ 0 & 1\end{bmatrix} \\ \bar{\boldsymbol{G}}_{ij} &= \begin{bmatrix}\boldsymbol{I}_3 & \tilde{\boldsymbol{R}}_i^{\ -1}(\boldsymbol{g}_j - \boldsymbol{g}_i) \\ 0 & 1\end{bmatrix}.\end{aligned} \tag{16.29}$$

We then can rewrite (16.27) as

$$\begin{aligned}\underset{\substack{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_n\in\mathrm{SE}(3)\\ \boldsymbol{M}_1,\ldots,\boldsymbol{M}_m\in\mathrm{SE}(3)}}{\arg\min} &\sum_{\boldsymbol{Z}_{ij}\in\mathcal{Z}} \left\|\boldsymbol{X}_i^{-1}\boldsymbol{X}_j - \boldsymbol{Z}_{ij}\right\|^2_{\boldsymbol{\Omega}_{\boldsymbol{Z}_{ij}}} + \\ &\sum_{\boldsymbol{G}_{ij}\in\mathcal{G}} \left\|\boldsymbol{M}_i^{-1}\boldsymbol{M}_j - \boldsymbol{G}_{ij}\right\|^2_{\boldsymbol{\Omega}_{\boldsymbol{G}_{ij}}} + \\ &\sum_{\bar{\boldsymbol{G}}_{ij}\in\bar{\mathcal{G}}} \left\|\boldsymbol{X}_i^{-1}\boldsymbol{M}_j - \bar{\boldsymbol{G}}_{ij}\right\|^2_{\boldsymbol{\Omega}_{\bar{\boldsymbol{G}}_{ij}}},\end{aligned} \tag{16.30}$$

where $\mathcal{Z}$ is the set of all pose graph edges (the red and blue in Fig. 16.16), $\mathcal{G}$ is the set edges from the simplified mesh (the green in Fig. 16.16), and $\bar{\mathcal{G}}$ is the set of all the edges connecting a pose vertex to a mesh vertex (the yellow in Fig. 16.16). We only optimize over translation in the second and third term, and as such $\boldsymbol{\Omega}_{\boldsymbol{G}_{ij}}$ and $\boldsymbol{\Omega}_{\bar{\boldsymbol{G}}_{ij}}$ correspond to information matrices where the rotation part is set to zero. Taking it one step further and observing that the terms are all based on the edges in the deformation graph, we can define $\boldsymbol{T}_i$ as the transformation of pose or mesh vertex i and $\boldsymbol{E}_{ij}$ is the transformation that corresponds to an edge in the deformation graph that is of the form $\boldsymbol{Z}_{ij}$, $\boldsymbol{G}_{ij}$, $\bar{\boldsymbol{G}}_{ij}$ depending on the type of edge. We are then left with the classic pose graph optimization problem,

$$\underset{\boldsymbol{T}_1,\ldots,\boldsymbol{T}_{n+m}\in\mathrm{SE}(3)}{\arg\min} \sum_{\boldsymbol{E}_{ij}} \left\|\boldsymbol{T}_i^{-1}\boldsymbol{T}_j - \boldsymbol{E}_{ij}\right\|^2_{\boldsymbol{\Omega}_{ij}}. \tag{16.31}$$

After the optimization, the position of each mesh vertex is updated from the affine transformations of the m nearest control nodes in the deformation graph as

$$\widetilde{\boldsymbol{v}}_i = \sum_{j=1}^{m} w_j(\boldsymbol{v}_i)[\boldsymbol{R}_j^M(\boldsymbol{v}_i - \boldsymbol{g}_j) + \boldsymbol{t}_j^M], \tag{16.32}$$

where $\boldsymbol{v}_i$ indicates the original position and $\widetilde{\boldsymbol{v}}_i$ the new deformed position of the i-th mesh vertex. The weights w_j assigned to each control node are defined as

$$w_j(\boldsymbol{v}_i) = \frac{\bar{w_j}(\boldsymbol{v}_i)}{\sum_{k=1}^{m} \bar{w_k}(\boldsymbol{v}_i)}$$
$$\bar{w_j}(\boldsymbol{v}_i) = \left(1 - \frac{\|\boldsymbol{v}_i - \boldsymbol{g}_j\|}{d_{\max}}\right)^2 \tag{16.33}$$

where $d_{\max}$ is the distance to furthest of the m control nodes from the mesh vertex.

16.4 Hierarchical Metric-Semantic Representations and 3D Scene Graphs

Up to this point, we have discussed sparse representations in Section 16.2 and dense representations in Section 16.3 for encoding semantic information into the maps that our robots build. Both representations have different advantages; as an example, sparse representations present a convenient interface for manipulating objects while dense volumetric representations excel at encoding information about free-space and traversability. This section presents hierarchical representations that combine dense and sparse maps into a unified model. In particular, we first discuss how choices of spatial representations and semantic categories impact the memory required by a robot, then observe that organizing information hierarchically leads to memory and computational savings, and then provide a more concrete example of a metric-semantic hierarchical map representations, namely *3D scene graphs*.

16.4.1 Hierarchical Representations and Symbol Grounding

So far, we have primarily considered semantic features that are closed-set semantic categories provided by some learned model, based on an input camera image or LiDAR scan. These features are groundings of *symbols*, representations of concepts that have specific meanings for a human, in the robot's sensor data.[1] High-level instructions issued by humans, such as "go and pick up the chair", intrinsically involve symbols. For the robot to correctly execute the instruction "go and pick up the chair", the robot needs to ground the symbol "chair" to the physical location it is situated in. This problem of embedding symbols into a map is commonly known as the *symbol grounding* problem [438].

In principle, we could design the perception system of our robot to ground symbols directly in posed sensor data. For instance, a robot with a camera could map image pixels to appropriate symbols (*e.g.,* "chair", "furniture", "kitchen", "apartment").[2] However, grounding symbols directly into sensor data is not scalable as

[1] *e.g.,* the word "chair" is a symbol in the sense that as humans we understand what a chair is.
[2] Dense 2D semantic segmentation networks are examples of this strategy.

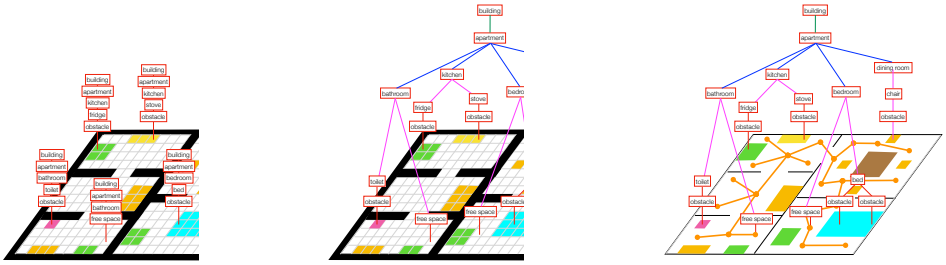

Figure 16.17 Groundings of symbols of interest to various sub-symbolic representations.

sensor data (*e.g.,* images) is collected at high rates and is relatively expensive to store. This is neither convenient nor efficient when grounding symbols for long-term operation. Instead, we need intermediate (or *sub-symbolic*) representations that compress the raw sensor data into a more compact format, which can be used to ground symbols. In this light, the previous sections of this chapter have been concerned with choices of geometric (*i.e.,* sub-symbolic) representations and algorithms for grounding semantic information into these sub-symbolic representations.

Hierarchy and Memory. The most naive strategy for grounding symbols in a sub-symbolic representation is to simply store whether or not a specific symbol (out of the L symbols of interest in the scene) is present in any given voxel. This is subtly different than the voxel-based semantic map discussed in Section 16.3 as we may not be willing to make the assumption that symbols are disjoint, *i.e.,* multiple symbols may be grounded to a single voxel (*e.g.,* we might say that a voxel belong to a chair, but it is also part of a kitchen). At a resolution δ, such a voxel grid representing a scene with volume V has a memory requirement of

$$m = \mathcal{O}\left(L \cdot V/\delta^3\right). \tag{16.34}$$

This is shown in the leftmost subfigure of Fig. 16.17: here we simply have a voxel-based map, where each voxel can store up to L semantic attributes.

Often the symbols that we are interested in display some sort of hierarchy. For indoor environments, we may be interested in the presence of objects and rooms (among other categories of concepts such as floors). Instead of densely and redundantly storing these symbols in every voxel, they can be rearranged according to their hierarchy. Intuitively, we can organize the representation into a graph, where a nodes can represent objects, rooms, or buildings, and edges represent inclusion: the children nodes of a building are the rooms contained in that building, while the children nodes of a room are the objects contained within. This is pictured in the middle subfigure of Fig. 16.17. This hierarchical organization already reduces memory requirement to

$$m = \mathcal{O}\left(V/\delta^3 + N_{\text{objects}} + N_{\text{rooms}} + \cdots + N_{\text{buildings}}\right). \tag{16.35}$$

Importantly, this memory compression is lossless; the original dense exhaustive voxel grid representation can easily be recovered from the hierarchical representation [503].

As seen previously in this chapter, having a dense and high-resolution voxel grid of an entire scene is often impractical. If instead of using standard voxel-based representations, we use more memory-efficient sub-symbolic representations, such as an octree, the memory requirement becomes

$$m = \mathcal{O}\left(N_{\text{sub-sym}} + N_{\text{objects}} + N_{\text{rooms}} + \cdots + N_{\text{buildings}}\right) \tag{16.36}$$

where $N_{\text{sub-sym}}$ is often much smaller than V/δ^3. This is often a "lossy" compression, and the original voxel-based sub-symbolic representation may not be perfectly recoverable even if the symbolic portion of the representation is [503]. This is shown in the rightmost subfigure of Fig. 16.17

Hierarchy and Inference. In addition to being memory efficient, the hierarchical representation that we have outlined can also have important computational properties that make it more advantageous than a strictly "flat" representation. To explore this, we first need to formalize the notion of hierarchical representation by introducing the definition of hierarchical graph.

Definition 16.2 (Hierarchical Graph) A graph $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$ is said to be hierarchical if there exists a partition of the vertices $\mathcal{V}$ into l layers (*i.e.,* $\mathcal{V} = \cup_{i \in 1:l} \mathcal{V}_i$) satisfying the following properties

1 **Single Parent.** Given a child node $v \in \mathcal{V}_i$ and a parent node $u \in \mathcal{V}_{i+1}$ that share an edge $(v, u) \in \mathcal{E}$, there is no other edge (v, u') where $u' \in \mathcal{V}_{i+1}$.
2 **Locality.** Every interlayer edge must be between consecutive layers, or formally no edge $(u, v) \in \mathcal{E}$ where $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ exists such that $|i - j| > 1$.
3 **Disjoint Children.** For every node $u, v \in \mathcal{V}_{i+1}$, we have that the children $\mathcal{C}(u)$ of u and $\mathcal{C}(v)$ v are disjoint, *i.e.,* there is no edge connecting any of the children. The set of children of a node $s \in \mathcal{V}_{i+1}$ is defined as $\mathcal{C}(s) \triangleq \{t : t \in \mathcal{V}_i \wedge (s, t) \in \mathcal{E}\}$.

Note that this definition uses an assumed ordering of the layers, where 1 is the lowest layer and l is the highest. This definition implies that for any node $v \in \mathcal{V}_i$ in the graph, the tree of descendants v is disjoint from the tree of descendants of *any* other node in $\mathcal{V}_i$. As shown by Hughes et al. [504], this property allows for the existence of a hierarchical tree decomposition and a corresponding bound of the treewidth of any hierarchical graph. The resulting bound is given by

$$\max_{v \in \mathcal{V}} tw[\mathcal{G}[\mathcal{C}(v)]] + 1, \tag{16.37}$$

where $\mathcal{G}[\mathcal{C}(v)]$ is the subgraph of $\mathcal{G}$ formed by the children of v, and $tw[\mathcal{G}]$ is the treewidth of the graph $\mathcal{G}$.[3] Treewidth is an important property that influences the

[3] The bounds given in [504] have an extra term for the treewidth of the top layer l, but we assume a $\mathcal{V}_l$ is comprised of a single node for simplicity.

Figure 16.18 Example 3D scene graph produced by [504].

computational complexity of inference of the graph. The treewidth is a well-known measure of complexity for many problems on graphs [99, 256, 328, 411]. In particular, while inference is NP-hard in general for inference on graphical models [238], proving that a graph has small treewidth opens the door to efficient, polynomial-time inference algorithms. The bound provided in (16.37) therefore allows for efficient inference over hierarchical graphs if the treewidth of descendant subgraphs remains small; this appears to hold in practice for a broad class of hierarchical graphs called 3D scene graphs [504].

We conclude this section by observing that while hierarchical representations have recently attracted a large amount of attention, hierarchical maps have been pervasive in robotics since its inception [620, 178, 1094]. Early work focused on 2D maps and investigated the use of hierarchical maps to resolve the divide between metric and topological representations [961, 363, 1261, 221, 72]. These works preceded the "deep learning revolution" and could not leverage the rich semantics currently accessible via deep neural networks.

16.4.2 3D Scene Graphs

The definition of a 3D scene graph varies slightly depending on the community of interest. The concept of 3D scene graph as a hierarchical map representation was first introduced by Armeni et al. [43] as a hierarchical representation of semantic and spatial information, including objects and rooms. Kim et al. [585] independently introduced 3D scene graphs as a graph of objects and their relationships, extending 2D scene graphs. Wald et al. [1145] constructed similar 3D scene graph of objects and their relationships from dense scene reconstructions, eventually as part of a online SLAM system [1195], adding an implicit hierarchy to [585]. Rosinol et al. [948] proposed a 3D scene graph structure that also included dynamic entities (such as humans) in addition to objects and a representation of free-space (referred to

as *places*). Hughes et al. [504] constructed the first online system to produce fully hierarchical scene graphs that included objects, places, and rooms; the proposed system was able to correct the 3D scene graph to be globally consistent. An example of a 3D scene graph produced by [504] is shown in Fig. 16.18.

Formally, we define a 3D scene graph as an graph $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$ where each node $v \in \mathcal{V}$ is associated with a corresponding node feature or properties x_v and each edge $e \in \mathcal{E}$ is associated with a corresponding edge feature or properties x_e. Every 3D scene graph spatially grounds symbols, and as such, every node feature x_v is assumed to contain positional information.[4] Often, it makes sense to treat the scene graph $\mathcal{G}$ as a layered graph such that $\mathcal{V}$ is partitioned into l layers, *i.e.,* $\mathcal{V} = \cup_{i \in 1:l} \mathcal{V}_i$, where 3D scene graphs of objects and their relationships such as [585, 1145, 1195] trivially have at least one layer. Approaches frequently construct 3D scene graphs from dense metric [1145, 1195] or metric-semantic [43, 948, 504] representations by clustering or segmenting the representation to obtain objects, creating a sparse, compressed grounding of object symbols that is often more computationally tractable to work with than the underlying dense representation. Additional layers, such as the places or rooms in [504], are sparse representations of the free-space.

Inter-layer edges (edges between nodes in different layers) are often used in conjunction with room and place layers to encode spatial containment (*e.g.,* an edge between a room and a specific object denotes that the object is in the room), which allows for the construction of hierarchical graphs matching and efficient inference [504]. Edges may encode other spatial relationships such as *on top of* or *nearby* (often between objects) [1195], but many 2D scene graph relationship categories (*e.g., left of*, *behind*) are view-dependent and are ambiguous in 3D contexts. Often these relationships are directed (*e.g.,* an edge encoding a relationship of *on top of* between a book and a table has a different meaning depending on the direction of the edge).

Factor Graphs and 3D Scene Graphs. As we discussed in Section 16.3.3, it is important to efficiently correct the map representation in response to loop closures. Hughes et al. [504] propose an approach to jointly optimize the 3D scene graph and the underlying dense representation that extends the PGMO approach outlined in Section 16.3.3. In addition to the pose graph and mesh control points, the layer of places is included in (16.27) and jointly optimized, providing a structural prior in addition to the factors between the mesh control nodes. The solution to this optimization is then used to interpolate and refine the other layers rather than reconstruct them from scratch. Other approaches such as [68] propose additional factors between the different layers of the 3D scene graph that allow them to optimize the *entire* 3D scene graph to correct for drift without requiring additional

[4] Different domains may make different modeling choices, *i.e.,* for 3D scene graphs of indoor environments positions in $\mathbb{R}^3$ make sense, whereas for 3D scene graphs targeted at autonomous driving such as [405], global position coordinates may be more useful.

refinement steps afterwards. This however means that the optimization problem can no longer be formulated as a pose graph optimization problem.

16.5 New Trends

Frontiers of 3D Scene Graphs. 3D scene graphs have been adopted for planning [24, 912, 920], manipulation [527], map compression [172], prediction [401, 700], loop closure detection [504], and localization [537], among other problems. At the same time, there is a growing body of work extending this representations in several directions. First, extending 3D scene graphs to unstructured and outdoor scenes is still an active research area, since the layers in the scene graph are harder to define and build in generic outdoor environments [80, 1042]. A related problem is that the layers of the scene graph are typically hard-coded by a human user, while ideally we would like for the robot to automatically reorganize its hierarchical representation depending on the task it is assigned; early works focusing on more fluid task-driven 3D scene graphs include [731, 174]. A third direction consists in extending scene graphs beyond closed-set semantics, by incorporating language (or language embeddings) into 3D scene graphs [1179, 415, 731, 174]; we will discuss open-set semantics mapping approaches at length in the next chapter. A forth direction focuses on extending 3D scene graphs with additional semantics information, including motion, dynamics, and affordances [1263]. Finally, while there are established ways to perform uncertainty quantification in traditional SLAM problems (*cf.* Chapter 6), performing uncertainty quantification in hierarchical representations mixing discrete and continuous variables is still a largely unexplored problem.

Multi-Modal Implicit Maps, Task Planning, and Action Models. An emerging trend is to generalize metric-semantic mapping beyond geometric, photometric, and semantic representations by encoding features from large vision-language models (VLMs). Recent approaches encode high-dimensional, language-grounded features that enable robots to reason about object affordances and robot tasks. Embedding language features, such as CLIP embeddings [906], allows to locate objects using an open vocabulary and opens up potentially new avenues of building robotic systems by lifting the closed-world assumption. In particular equipping neural scene representation, such as NeRF and Gaussian Splatting, with language embeddings attacted recently considerable interest [563]. For instance, Online Language Splatting [551] builds dense, language-aware scene representations in real time by fusing CLIP features into Gaussian splats, enabling downstream query and grounding tasks. We discuss many of these approaches in the next chapter.

Turning the information encoded in a semantic map into action [835, 625] is another recent development that is also enabled by encoding semantics directly into the map representation by using lanuage-grounded embeddings. ATLAS Navigator [835] actively constructs spatial maps embedded with language features to facilitate efficient navigation to object-centric goals described in natural lan-

guage. M3 [1308] introduces a persistent 3D-spatial multimodal memory that stores VLM-aligned features and supports retrieval-based reasoning for long-horizon tasks. Moving beyond mapping, LUMOS [804] integrates world modeling with language-conditioned imitation learning, allowing robots to simulate, imagine, and plan in VLM-grounded latent spaces. Finally, GNFactor [1260] demonstrates how generalizable neural feature fields can bridge multi-task robot learning with real-world generalization by encoding rich perceptual priors that inform control. These works illustrate a shift toward actionable representations, where spatial memory, language, and multimodal perception are tightly coupled to support task planning and skill execution.

However, often these maps are build as a post-processing step and an open challenge is to build such representations incrementally [551, 664], where aggregating ambiguous information from multiple varying viewpoints in a consistent and instance-aware way remains a formidable research challenge. Also in this case, coupling such representations with uncertainty seems like a potential way forward, which would allow modeling ambiguities in a principled way.

Multi-Robot Semantic Understanding. While in this chapter we have mostly focused on single-robot metric-semantic SLAM approaches, the recent literature has also investigated extensions of these representations to multi-robot teams, where multiple robots explore an unknown environment and have to build a unified metric-semantic map representation. This setting is challenging since now the information is collected in a distributed way (hence creating communication bottlenecks when sharing data among robots). Moreover, the amount of collected data and the size of the resulting problems become unmanageable by a single robot and either require a powerful central server or a distributed solution where the workload is shared across multiple robots. Finally, in multi-robot SLAM problems one no longer has a reliable initial guess for the poses of all robots (intuitively, all robots might use a different coordinate frame), hence creating additional challenge in terms of robustness and outlier rejection (*cf.* Chapter 3 and Chapter 6). Kimera-Multi [1099] is the first approach to distributed multi-robot metric-semantic SLAM, while [1096] provides a detailed experimental analysis discussing tradeoffs between centralized and distributed SLAM approaches. Finally, recent work has extended 3D scene graph construction to multi-robot teams [173, 333].