Autonomous Robots (2013)
Preprint, final version available at DOI 10.1007/s10514-012-9321-0

OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees

Armin Hornung · Kai M. Wurm · Maren Bennewitz · Cyrill Stachniss · Wolfram Burgard

Received: 30 April 2012 / Accepted: 31 December 2012

Abstract Three-dimensional models provide a volumetric representation of space which is important for a variety of robotic applications including flying robots and robots that are equipped with manipulators. In this paper, we present an open-source framework to generate volumetric 3D environment models. Our mapping approach is based on octrees and uses probabilistic occupancy estimation. It explicitly represents not only occupied space, but also free and unknown areas. Furthermore, we propose an octree map compression method that keeps the 3D models compact. Our framework is available as an open-source C++ library and has already been successfully applied in several robotics projects. We present a series of experimental results carried out with real robots and on publicly available real-world datasets. The results demonstrate that our approach is able to update the representation efficiently and models the data consistently while keeping the memory requirement at a minimum.

Keywords 3D · Probabilistic · Mapping · Navigation

1 Introduction

Several robotic applications require a 3D model of the environment. These include airborne, underwater, outdoor, or extra-terrestrial missions. However, 3D models are also relevant for many domestic scenarios, for example, for mobile manipulation and also navigation tasks.

This work has been supported by the German Research Foundation (DFG) under contract number SFB/TR-8 and by the European Commission under grant agreement numbers FP7-248258-First-MM and FP7-600890-ROVINA.

A. Hornung · K.M. Wurm · M. Bennewitz · C. Stachniss · W. Burgard
Department of Computer Science, University of Freiburg,
Georges-Koehler-Allee 74, 79110 Freiburg, Germany
E-mail: {hornunga,wurm,maren,stachnis,burgard}@informatik.uni-freiburg.de

Although 3D mapping is an integral component of many robotic systems, there exist few readily available, reliable, and efficient implementations. The lack of such implementations leads to the re-creation of basic software components and, thus, can be seen as a bottleneck in robotics research. We therefore believe that the development of an open-source 3D mapping framework will greatly facilitate the development of robotic systems that require a three-dimensional geometric representation of the environment.

Most robotics applications require a probabilistic representation, modeling of free, occupied, and unmapped areas, and additionally efficiency with respect to runtime and memory usage. We will now discuss these three requirements in detail.

- **Probabilistic representation:** To create 3D maps, mobile robots sense the environment by taking 3D range measurements. Such measurements are afflicted with uncertainty: Typically, the error in the range measurements is in the order of centimeters. But there may also be seemingly random measurements that are caused by reflections or dynamic obstacles. When the task is to create an accurate model of the environment from such noisy measurements, the underlying uncertainty has to be taken into account probabilistically. Multiple uncertain measurements can then be fused into a robust estimate of the true state of the environment. Another important aspect is that probabilistic sensor fusion allows for the integration of data from multiple sensors and of multiple robots.
- **Modeling of unmapped areas:** In autonomous navigation tasks, a robot can plan collision-free paths only for those areas that have been covered by sensor measurements and detected to be free. Unmapped areas, in contrast, need to be avoided and for this reason the map has to represent such areas. Furthermore, the knowledge

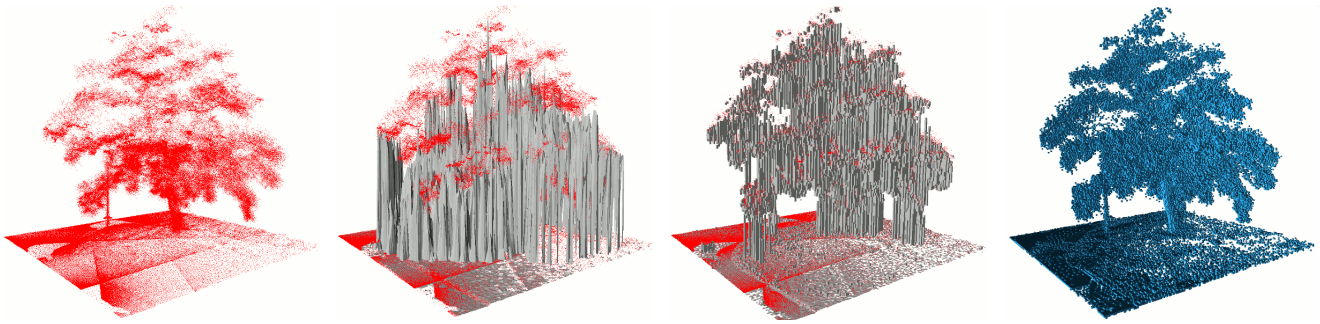

Fig. 1 3D representations of a tree scanned with a laser range sensor (from left to right): Point cloud, elevation map, multi-level surface map, and our volumetric (voxel) representation. Please note that our volumetric representation explicitly models free space but that for clarity only occupied volumes are visualized.

about unmapped areas is essential during exploration. When maps are created autonomously, the robot has to plan its actions so that measurements are taken in previously unmapped areas.

– **Efficiency:** The map is a central component of any autonomous system because it is used during action planning and execution. For this reason, the map needs to be efficient with respect to access times but also with respect to memory consumption. From a practical point of view, memory consumption is often the major bottleneck in 3D mapping systems. Therefore, it is important that the model is compact in memory so that large environments can be mapped, a robot can keep the model in its main memory, and it can be transmitted efficiently between multiple robots.

Several approaches have been proposed to model 3D environments in robotics. As an illustration, we compare our approach to three common mapping approaches – a visualization of the results is given in Fig. 1. In the example, 3D measurements are represented using point clouds, elevation maps (Hebert et al., 1989), multi-level surface maps (Triebel et al., 2006), and in a volumetric way using our framework. None of the previous approaches fulfill all of the requirements we set out above. Point clouds store large amounts of measurement points and hence are not memory-efficient. They furthermore do not allow to differentiate between obstacle-free and unmapped areas and provide no means of fusing multiple measurements probabilistically. Elevation maps and multi-level surface maps are efficient but do not represent unmapped areas either. Most importantly, these approaches cannot represent arbitrary 3D environments, such as the branches of the tree in the example.

In this work we present OctoMap, an integrated framework based on octrees for the representation of three-dimensional environments. In our framework, we combine the advantages of previous approaches to 3D environment modeling in order to meet the requirements discussed above. A central property of our approach is that it allows for efficient and probabilistic updates of occupied and free space while keeping the memory consumption at a minimum. Occupied space is obtained by the end points of a distance sensor such as a laser range finder, while free space corresponds to the observed area between the sensor and the end point. As a key contribution of our approach, we introduce a compression method that reduces the memory requirement by locally combining coherent map volumes, both in the mapped free areas and the occupied space. We implemented our approach and thoroughly evaluated it using various publicly available real-world robotics datasets of both indoor and outdoor environments.

Our open source implementation is freely available in form of a self-contained C++ library. It was released under the BSD-license and can be obtained from http://octomap.github.com. The library supports several platforms, such as Linux, Mac OS, and Windows. It has been integrated into the Robot Operating System (ROS) and can be used in other software frameworks in a straightforward way. Since its first introduction in 2010 (Wurm et al., 2010), the OctoMap framework was constantly improved and used in an increasing number of robotics research projects.

This paper is organized as follows. After providing a detailed discussion of related work in the area of 3D data structures and mapping approaches in the next section, we present our OctoMap framework in Sect. 3. Implementation details are given in Sect. 4, followed by an evaluation of the proposed framework in Sect. 5. Finally, case studies on how OctoMap has been used in various areas of robotics demonstrate the versatility and ease of integration in Sect. 6.

2 Related Work

Three-dimensional models of the environment are a key prerequisite for many robotic systems and consequently they have been the subject of research for more than two decades.

A popular approach to modeling environments in 3D is to use a grid of cubic volumes of equal size, called voxels, to discretize the mapped area. Roth-Tabak and Jain (1989) as well as Moravec (1996) presented early works using such a

representation. A major drawback of rigid grids is their large memory requirement. The grid map needs to be initialized so that it is at least as big as the bounding box of the mapped area, regardless of the actual distribution of map cells in the volume. In large-scale outdoor scenarios or when there is the need for fine resolutions, memory consumption can become prohibitive. Furthermore, the extent of the mapped area needs to be known beforehand or costly copy operations need to be performed every time the map area is expanded.

A discretization of the environment can be avoided by storing 3D range measurements directly. The occupied space in the environment is then modeled by the 3D point clouds returned by range sensors such as laser range finders or stereo cameras. This point cloud approach has been used in several 3D SLAM systems such as those presented by Cole and Newman (2006) as well as in the SLAM approach of Nüchter et al. (2007). The drawbacks of this method are that neither free space nor unknown areas are modeled and that sensor noise and dynamic objects cannot be dealt with directly. Thus, point clouds are only suitable for high precision sensors in static environments and when unknown areas do not need to be represented. Furthermore, the memory consumption of this representation increases with the number of measurements which is problematic as there is no upper bound.

If certain assumptions about the mapped area can be made, 2.5D maps are sufficient to model the environment. Typically, a 2D grid is used to store the measured height for each cell. In its most basic form, this results in an elevation map where the map stores exactly one value per cell (Hebert et al., 1989). One approach in which such maps have been demonstrated to be sufficient is the outdoor terrain navigation method described by Hadsell et al. (2009). Whenever there is a single surface that the robot uses for navigation, an elevation map is sufficient to model the environment, since overhanging obstacles that are higher than the vehicle, such as trees, bridges or underpasses, can be safely ignored. The strict assumption of a single surface can be relaxed by allowing multiple surfaces per cell (Triebel et al., 2006; Pfaff et al., 2007), or by using classes of cells which correspond to different types of structures (Gutmann et al., 2008). A general drawback of most 2.5D maps is that they do not represent the environment in a volumetric way but discretize it in the vertical dimension based on the robot's height. While this is sufficient for path planning and navigation with a fixed robot shape, the map does not represent the actual environment, e.g. for localization.

To overcome this problem, a related approach was proposed by Ryde and Hu (2010). The approach stores a list of occupied voxels for each cell in a 2D grid. Although this representation is volumetric, it does not differentiate between free and unknown volumes. Dryanovski et al. (2010) store lists of occupied and free voxels for each 2D cell in their Multi-Volume Occupancy Grid approach. In contrast to our approach, however, the map extent needs to be known beforehand, map updates are more computationally involved, and there is no multi-resolution capability. Another potential problem is that subsequent map updates cannot subdivide existing volumes, leading to an incorrect model of the environment. Similarly, Douillard et al. (2010) combine a coarse elevation map for background structures with object voxel maps at a higher resolution. In contrast to our work, this approach focuses on 3D segmentation of single measurements and does not integrate several measurements into a model of the environment.

In robotic mapping, octrees avoid one of the main shortcomings of fixed grid structures: They delay the initialization of map volumes until measurements need to be integrated. In this way, the extent of the mapped environment does not need to be known beforehand and the map only contains volumes that have been measured. If inner nodes of a tree are updated properly, the tree can also be used as a multi-resolution representation since it can be cut at any level to obtain a coarser subdivision. The use of octrees for mapping was originally proposed by Meagher (1982). Early works mainly focused on modeling a Boolean property such as occupancy (Wilhelms and Van Gelder, 1992). Payeur et al. (1997) used octrees to adapt occupancy grid mapping from 2D to 3D and thereby introduced a probabilistic way of modeling occupied and free space. A similar approach was used by Fournier et al. (2007) and Pathak et al. (2007). In contrast to the approach presented in this paper, however, the authors did not explicitly address the issues of map compression or bounded confidence in the map.

An octree-based 3D map representation was also proposed by Fairfield et al. (2007). Their map structure called Deferred Reference Counting Octree is designed to allow for efficient map updates, especially in the context of particle filter SLAM. To achieve map compactness, a lossy maximum-likelihood compression is performed periodically. Compared to the compression technique used in our approach, this discards the probability information for future updates. Furthermore, the problem of overconfident maps and multi-resolution queries are not addressed.

As a data structure, octrees are applied in a variety of applications, most notably in the area of computer graphics for efficient rendering (Botsch et al., 2002; Surmann et al., 2003; Laine and Karras, 2010) and in the field of photogrammetry to store and address large point clouds (Girardeau-Montaut et al., 2005; Elseberg et al., 2011). Another popular use case is the compression of static point clouds (Schnabel and Klein, 2006) or point cloud streams (Kammerl et al., 2012). While our framework is general enough to also store raw point clouds, its main purpose is to integrate these point clouds into a memory-efficient, volumetric occupancy map, since point clouds as environment representation in robotics

have a number of disadvantages as detailed at the beginning of this section.

Yguel et al. (2007b) presented a 3D map based on the Haar wavelet data structure. This representation is also multi-resolution and probabilistic. However, the authors did not evaluate applications to 3D modeling in-depth. In their evaluation, unknown areas are not modeled and only a single simulated 3D dataset is used. Whether this map structure is as memory-efficient as octrees is hard to assess without a publicly available implementation.

Surface representations such as the 3D Normal Distribution Transform (Magnusson et al., 2007) or Surfels (Habbecke and Kobbelt, 2007) were recently used for 3D path planning (Stoyanov et al., 2010) and object modeling (Weise et al., 2009; Krainin et al., 2011). Similarly, an accurate real-time 3D SLAM system based on a low-cost depth camera and GPU processing was proposed by Newcombe et al. (2011) to reconstruct dense surfaces in indoor scenes. Recently, this work has been extended to work in larger indoor environments (Whelan et al., 2012). However, surface representations are unable to distinguish between free and unknown space, may require large memory particularly outdoors, and are often based on strong assumptions about the corresponding environment. In mobile manipulation scenarios, for example, being able to differentiate free from unknown space is essential for safe navigation.

Finally, to the best of our knowledge, no open source implementation of a 3D occupancy mapping framework meeting the requirements outlined in the introduction is freely available.

3 OctoMap Mapping Framework

The approach proposed in this paper uses a tree-based representation to offer maximum flexibility with regard to the mapped area and resolution. It performs a probabilistic occupancy estimation to ensure updatability and to cope with sensor noise. Furthermore, compression methods ensure the compactness of the resulting models.

3.1 Octrees

An octree is a hierarchical data structure for spatial subdivision in 3D (Meagher, 1982; Wilhelms and Van Gelder, 1992). Each node in an octree represents the space contained in a cubic volume, usually called a voxel. This volume is recursively subdivided into eight sub-volumes until a given minimum voxel size is reached, as illustrated in Fig. 2. The minimum voxel size determines the resolution of the octree. Since an octree is a hierarchical data structure, the tree can be cut at any level to obtain a coarser subdivision if the inner nodes are maintained accordingly. An example of an octree map queried for occupied voxels at several resolutions is shown in Fig. 3.

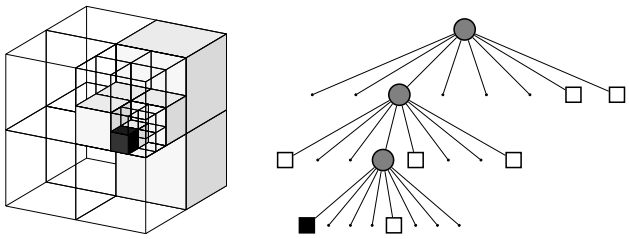

Fig. 2 Example of an octree storing free (shaded white) and occupied (black) cells. The volumetric model is shown on the left and the corresponding tree representation on the right.

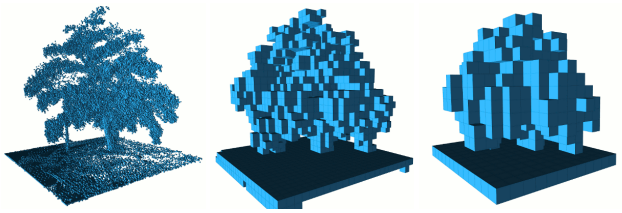

Fig. 3 By limiting the depth of a query, multiple resolutions of the same map can be obtained at any time. Occupied voxels are displayed in resolutions 0.08 m, 0.64 , and 1.28 m.

In its most basic form, octrees can be used to model a Boolean property. In the context of robotic mapping, this is usually the occupancy of a volume. If a certain volume is measured as occupied, the corresponding node in the octree is initialized. Any uninitialized node could be free or unknown in this Boolean setting. To resolve this ambiguity, we explicitly represent free volumes in the tree. These are created in the area between the sensor and the measured end point, e.g., along a ray determined with raycasting. Areas that are not initialized implicitly model unknown space. An illustration of an octree containing free and occupied nodes from real laser sensor data can be seen in Fig. 4. Using Boolean occupancy states or discrete labels allows for compact representations of the octree: If all children of a node have the same state (occupied or free) they can be pruned. This leads to a substantial reduction in the number of nodes that need to be maintained in the tree.

In robotic systems, one typically has to cope with sensor noise and temporarily or permanently changing environments. In such cases, a discrete occupancy label will not be sufficient. Instead, occupancy has to be modeled probabilistically, for instance by applying occupancy grid mapping (Moravec and Elfes, 1985). However, such a probabilistic model lacks the possibility of lossless compression by pruning.

The approach presented in this paper offers means of combining the compactness of octrees that use discrete labels with the updatability and flexibility of probabilistic modeling as we will discuss in Sect. 3.4.

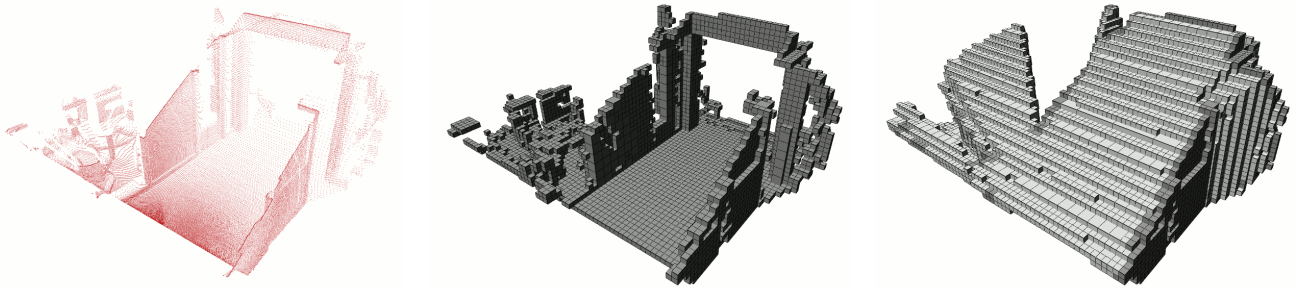

Fig. 4 An octree map generated from example data. *Left:* Point cloud recorded in a corridor with a tilting laser range finder. *Center:* Octree generated from the data, showing occupied voxels only. *Right:* Visualization of the octree showing occupied voxels (dark) and free voxels (white). The free areas are obtained by clearing the space on a ray from the sensor origin to each end point. Lossless pruning results in leaf nodes of different sizes, mostly visible in the free areas on the right.

In terms of data access complexity, octrees require an overhead compared to a fixed-size 3D grid due to the tree structure. A single, random query on a tree data structure containing n nodes with a tree depth of d can be performed with a complexity of $\mathcal{O}(d) = \mathcal{O}(\log n)$. Traversing the complete tree in a depth-first manner requires a complexity of $\mathcal{O}(n)$. Note that, in practice, our octree is limited to a fixed maximum depth $d_{\max}$. This results in a random node lookup complexity of $\mathcal{O}(d_{\max})$ with $d_{\max}$ being constant. Therefore, for a fixed depth $d_{\max}$, the overhead compared to a corresponding 3D grid is constant. Note that in all our experiments a maximum depth of 16 was used, which is sufficient to cover a cube with a vollume of $(655.36\,\mathrm{m})^3$ at 1 cm resolution. The exact timings for this setting are provided in Sect. 5.5.

3.2 Probabilistic Sensor Fusion

In our approach, sensor readings are integrated using occupancy grid mapping as introduced by Moravec and Elfes (1985). The probability $P(n \mid z_{1:t})$ of a leaf node n to be occupied given the sensor measurements $z_{1:t}$ is estimated according to

$$P(n \mid z_{1:t}) = \left[1 + \frac{1-P(n \mid z_t)}{P(n \mid z_t)} \frac{1-P(n \mid z_{1:t-1})}{P(n \mid z_{1:t-1})} \frac{P(n)}{1-P(n)}\right]^{-1} \quad (1)$$

This update formula depends on the current measurement z_t, a prior probability $P(n)$, and the previous estimate $P(n \mid z_{1:t-1})$. The term $P(n \mid z_t)$ denotes the probability of voxel n to be occupied given the measurement z_t. This value is specific to the sensor that generated z_t. We provide details on the sensor model used throughout our experiments in Sect. 5.1.

The common assumption of a uniform prior probability leads to $P(n) = 0.5$ and by using the log-odds notation, Eq. (1) can be rewritten as

$$\mathrm{L}(n \mid z_{1:t}) = \mathrm{L}(n \mid z_{1:t-1}) + \mathrm{L}(n \mid z_t), \quad (2)$$

with

$$\mathrm{L}(n) = \log\left[\frac{P(n)}{1-P(n)}\right]. \quad (3)$$

This formulation of the update rule allows for faster updates since multiplications are replaced by additions. In case of pre-computed sensor models, the logarithms do not have to be computed during the update step. Note that log-odds values can be converted into probabilities and vice versa and we therefore store this value for each voxel instead of the occupancy probability. It is worth noting that for certain configurations of the sensor model that are symmetric, i.e., nodes being updated as hits have the same weight as the ones updated as misses, this probability update has the same effect as counting hits and misses similar to (Kelly et al., 2006).

When a 3D map is used for navigation, a threshold on the occupancy probability $P(n \mid z_{1:t})$ is often applied. A voxel is considered to be occupied when the threshold is reached and is assumed to be free otherwise, thereby defining two discrete states. From Eq. (2) it is evident that to change the state of a voxel we need to integrate as many observations as have been integrated to define its current state. In other words, if a voxel was observed free for k times, then it has to be observed occupied at least k times before it is considered occupied according to the threshold (assuming that free and occupied measurements are equally likely in the sensor model). While this property is desirable in static environments, a mobile robot is often faced with temporary or permanent changes in the environment and the map has to adapt to these changes quickly. To ensure this adaptability, Yguel et al. (2007a) proposed a clamping update policy that defines an upper and lower bound on the occupancy estimate. Instead of using Eq. (2) directly, occupancy estimates

are updated according to

$$\mathrm{L}(n \mid z_{1:t}) = \max(\min(\mathrm{L}(n \mid z_{1:t-1}) + \mathrm{L}(n \mid z_t), l_{\max}), l_{\min}), \quad (4)$$

where $l_{\min}$ and $l_{\max}$ denote the lower and upper bound on the log-odds value. Intuitively, this modified update formula limits the number of updates that are needed to change the state of a voxel. Applying the clamping update policy in our approach leads to two advantages: we ensure that the confidence in the map remains bounded and as a consequence the model can adapt to changes in the environment quickly. Furthermore, we are able to compress neighboring voxels with pruning (see Sect. 3.4). As we will discuss in Sect. 5.4, this leads to a considerable reduction in the number of voxels that have to be maintained. The compression achieved with clamping is no longer completely lossless in terms of the full probabilities, since information close to zero and one is lost. In between the clamping thresholds, however, full probabilities are preserved.

3.3 Multi-Resolution Queries

When measurements are integrated into our map structure, probabilistic updates are performed only for the leaf nodes in the octree. But since an octree is a hierarchical data structure, we can make use of the inner nodes in the tree to enable multi-resolution queries. Observe that we yield a coarser subdivision of the 3D space when the tree is traversed only up to a given depth that is not the depth of the leaf nodes. Each inner node spans the volume that its eight children occupy, so to determine the occupancy probability of an inner node, we have to aggregate the probabilities of its children. Several strategies could be pursued to determine the occupancy probability of a node n given its eight sub-volumes n_i (Kraetzschmar et al., 2004). Depending on the application at hand, either the average occupancy

$$\bar{l}(n) = \frac{1}{8} \sum_{i=1}^{8} \mathrm{L}(n_i) \quad (5)$$

or the maximum occupancy

$$\hat{l}(n) = \max_i \mathrm{L}(n_i) \quad (6)$$

can be used, where $\mathrm{L}(n)$ returns the current log-odds occupancy value of a node n. Using the maximum child occupancy to update inner nodes can be regarded a conservative strategy which is well suited for robot navigation. By assuming that a volume is occupied if any part of it has been measured occupied, collision-free paths can be planned and for this reason the maximum occupancy update is used in our system. Note that in an even more conservative setting, $\mathrm{L}(n)$ can be defined to return a positive occupancy probability for unknown cells as well. An example of an octree queried for occupied voxels at several resolutions is shown in Fig. 3.

3.4 Octree Map Compression

In Sect. 3.1, we explained how tree pruning can reduce the amount of redundant information in octrees with discrete occupancy states in which a voxel can be either occupied or free. The same technique can also be applied in maps that use probabilistic occupancy estimates to model occupied and free space. In general, however, one cannot expect the occupancy probability of neighboring nodes to be identical, even if both voxel are occupied by the same physical obstacle. Sensor noise and discretization errors can lead to different probabilities and therefore interfere with compression schemes that rely on identical node information. A possible solution to this problem is to apply a threshold on the voxel probability, for example 0.5, and in this way generate a discrete state estimation as suggested by Fairfield et al. (2007). With that approach, however, individual probability estimates cannot be recovered after the tree has been pruned.

In our approach, we achieve map compression by applying the clamping update policy given in Eq. (4). Whenever the log-odds value of a voxel reaches either the lower bound $l_{\min}$ or the upper bound $l_{\max}$, we consider the node as stable in our approach. Intuitively, stable nodes have been measured free or occupied with high confidence. In a static environment, all voxels will converge to a stable state after a sufficient number of measurements have been integrated. With the parameters chosen in our experiments, for example, five agreeing measurements are sufficient to render an unknown voxel into a stable voxel. If all children of an inner tree node are stable leaf nodes with the same occupancy state, then the children can be pruned. Should future measurements be integrated that contradict the state of the corresponding inner node, then its children are regenerated and updated accordingly. Applying this compression only leads to a loss of information close to $P(n) = 0$ and $P(n) = 1$ while preserving the probabilities in between. In our experiments, combining octree pruning and clamping leads to a compression improvement of up to 44%.

In many robotic navigation tasks such as obstacle avoidance or localization, only the maximum likelihood map containing either free or occupied nodes is sufficient. In these cases, a lossy compression based on the occupancy threshold, as suggested by Fairfield et al. (2007), can be performed. For this compression, all nodes are converted to their maximum likelihood (clamped) probabilities, followed by tree pruning. This yields an even greater compression and less memory requirements.

Fig. 5 Detail of a volumetric indoor OctoMap containing color information. The complete map covers an area of 7.3 m × 7.9 m × 4.6 m at 2 cm resolution.

3.5 Extensions

3.5.1 Maps with Rich Information

Octree nodes can be extended to store additional data to enrich the map representation. Voxels could, for example, store terrain information, environmental data such as the temperature, or color information. Each additional voxel property requires a method that allows several measurements to be fused. As an example, we extended our mapping framework to store the average color of each voxel. This creates visualizations for the user and enables a color-based classification of the environment or appearance-based robot localization from virtual views (similar to (Einhorn et al., 2011; Mason et al., 2011)). It can also be used as a starting point to create colored, high-resolution surface meshes (Hoppe et al., 1992). Figure 5 shows an octree map that was created by integrating colored point clouds recorded with a hand-held Microsoft Kinect sensor. The data is available in the sequence called freiburg1_360 of the RGBD-dataset (Sturm et al., 2012) and was aligned using RGB-D SLAM (Endres et al., 2012).

3.5.2 Octree Hierarchies

We developed an extension to our mapping approach that exploits hierarchical dependencies in the environment (Wurm et al., 2011). This extension maintains a collection of submaps in a tree-structure, where each node represents a subspace of the environment. The subdivision applied in our system is based on a user-defined segmentation of the input and on a given spatial relation that expresses the relation between segments.

Figure 6 gives an illustration of a hierarchy that is based on the assumption that objects are located on top of supporting planes. In this application, we first estimated supporting planes in the input. Objects on top of these supporting planes were then segmented in the input data and modeled in individual volumetric submaps. As a result, the table is a submap that is on top of the floor and several household objects are in turn represented as submaps on top of the table.

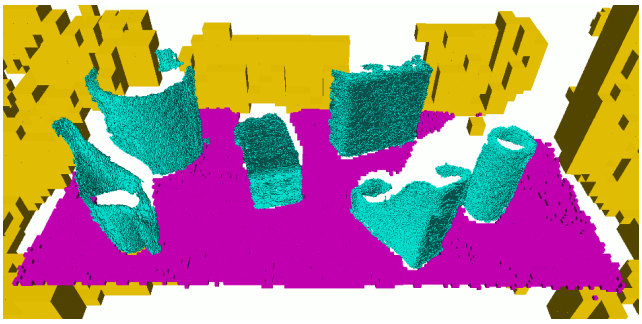

Fig. 6 Hierarchical octree model of a tabletop scene. Background (yellow), table (magenta), and objects (cyan) are represented by individual octree maps of different resolutions.

Compared to a single, monolithic map of the environment, our hierarchical approach exhibits a number of advantages: First, each submap is maintained independently and mapping parameters such as the resolution can be adapted for each submap. Second, submaps can be manipulated independently. For example, one of the submaps representing an individual object can be moved while the rest remains static. Third, hierarchical dependencies of submaps can be encoded in the hierarchy. For example, all objects on a table can be associated to this table and if the table is moved then the objects are moved along with it.

The approach has been evaluated in the context of tabletop manipulation. Objects on a table were mapped at very fine resolutions while the table and background structures were mapped at lower resolutions. This approach led to models that were about an order of magnitude more compact than a single map that represents the complete scene.

4 Implementation Details

4.1 Memory-Efficient Node Implementation

In a straight-forward octree implementation, each node in the tree stores in addition to the data payload the coordinate of its center location, its voxel size, and pointers to its children. This, however, can lead to a substantial memory overhead. Since the node location and its voxel size can be reconstructed while traversing the octree, we do not explicitly store this information in the nodes to reduce the memory overhead.

In general, octree nodes need to maintain an ordered list of their children. This can be directly achieved by using eight pointers per node. If sparse data are modeled, the memory requirement of those pointers (8×4 byte $= 32$ byte on a

32 bit architecture) will lead to a significant memory overhead (Wilhelms and Van Gelder, 1992). We overcome that by using one child pointer per node that points to an array of eight pointers (Fig. 7, left). This array is only allocated if the node indeed has children and is not allocated for leaf nodes. Thus, any leaf node in the octree only stores the mapping data itself (e.g., the occupancy probability) and one (null) pointer. Inner nodes additionally store eight pointers to their children. In the robotics-related datasets used in our evaluation, 80% – 85% of the octree nodes are leafs. In our experiments, the above-mentioned implementation saves 60% – 65% of memory compared to allocating 8 pointers for each node.

To store a per-voxel occupancy probability, a single float value (usually 4 byte) is sufficient to represent the log-odds value. This results in a node size of 40 byte for inner nodes and 8 byte for leafs on a 32-bit architecture. Note that most compilers align member data in memory for runtime efficiency, that is, the data of a node is padded to be multiples of one word large (4 byte on a 32-bit architecture). 64-bit architectures can address large amounts of memory at the cost of pointers and words having twice the size. On such architectures, the memory size of inner nodes increases to 80 byte and the size of leaf nodes to 16 byte. Note that the actual size of the data structure (76 byte for inner nodes and 12 byte for leaf nodes) is again padded to multiples of the word size (8 byte on a 64-bit architecture) by most compilers.

In our approach, the octree is homogeneous by design, that is, all nodes have the same structure and store occupancy. While inner nodes could potentially save 8 byte by omitting occupancy information, maintaining it according to Eq. (5) or (6) enables multi-resolution queries, where tree traversal is stopped at a fixed depth.

Virtual inheritance between classes allows dynamic dispatch during run-time, at the cost of one extra pointer to the virtual function table (vtable) for each object instance. To minimize the memory footprint, we avoided this overhead in the octree node implementation. We apply direct inheritance and casts for the nodes, and use virtual inheritance only in the octree classes. This method results in an overhead of the size of only one pointer per octree map.

4.2 Octree Types

The most common octree and node types in our framework are summarized in Fig. 8 as a UML diagram. The basic octree functionality is implemented in *OcTreeBase*, and the basic node functionality is implemented in *OcTreeDataNode*. *OcTreeDataNode* is templated over data that is stored in the node while *OcTreeBase* is templated over the node type. *OccupancyOcTreeBase* adds occupancy mapping functionality to the tree implementation, such as scan insertions and

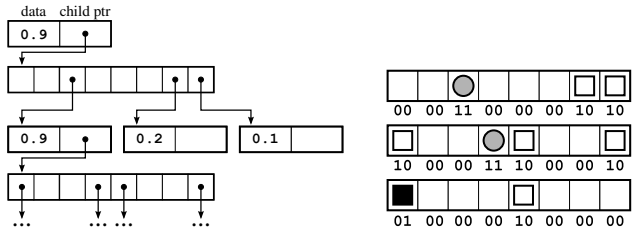

Fig. 7 *Left*: The first nodes of the octree example from Fig. 2 in memory connected by pointers. Data is stored as one float denoting occupancy. *Right*: The complete tree from Fig. 2 as compact serialized bitstream. All maximum-likelihood occupancy information can be stored serially in only six bytes, using two bits for each of a node's eight child labels (00: unknown; 01: occupied; 10: free; 11: inner node with child next in the stream).

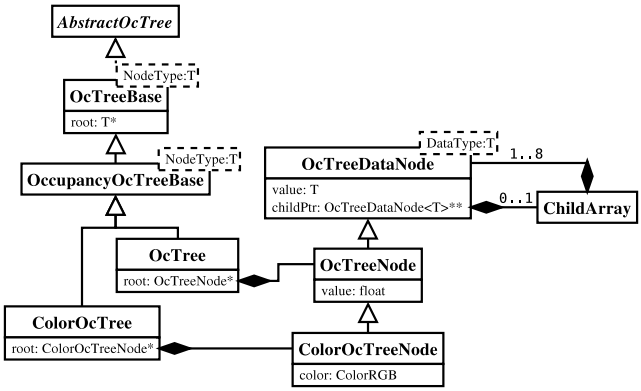

Fig. 8 UML diagram of the most common octree and node classes.

ray casting. The main occupancy octree class *OcTree* derives from *OccupancyOcTreeBase* using *OcTreeNode* for its nodes. This structure allows for flexible extensions of our framework at different levels, e.g., to extend nodes with custom data or to add new functionality to the octree. One example is the implementation of *ColorOcTree* that uses *ColorOcTreeNodes* (illustrated in Fig. 5). These nodes store color in addition to an occupancy estimate, as introduced in Sect. 3.5.1.

The maximum tree depth is limited to 16 levels in our current implementation. This enables fast tree traversals by using computable voxel addresses. However, the depth limit also poses a limit on the maximum spatial extent of the octree. At a resolution of 1 cm, for example, the map can cover a maximum of $2^{16} \times 0.01\,\text{m} = 655.36\,\text{m}$ in each dimension. While this is sufficient for most indoor applications, the implementation can directly be extended to 32 depth levels, allowing to cover $2^{32} \times 0.01\,\text{m} = 42\,949\,672.96\,\text{m}$ at a resolution of 1 cm.

4.3 Map File Generation

Many robotic applications require maps to be stored in files. This includes cases where a map is generated during a setup phase and is later used by mobile robots for path planning

and localization. Another scenario is a multi-robot system where maps are exchanged between robots. In either case, a compact serialized representation is required to minimize the consumption of disk space or communication bandwidth.

The most compact files can be generated whenever a maximum likelihood estimate of the map is sufficient for the task at hand. In this case the per-node probabilities are discarded. As motivated above, volumes in which no information has been recorded can be of special interest in robotic systems, for example, during exploration. For this reason, we explicitly differentiate between free and unknown areas and encode nodes as either occupied, free, unknown, or as inner nodes in our map files. Using these labels, octree maps can be recursively encoded as a compact bit stream. Each node is represented only by the eight labels of its children. Beginning at the root node, each child that is not a leaf is recursively added to the bit stream. Leaf nodes do not have to be added since they can be reconstructed from their label during the decoding process. Figure 7 (right) illustrates the bit-stream encoding. Each row represents one node with the upper row corresponding to the root node. The last row only contains leafs so no further nodes are added.

In this maximum likelihood representation, each node occupies 16 bits of memory, 2 bits per child, resulting in a compact map file. In our experiments, file sizes never exceeded 15 MB even for large outdoor environments at a fine resolution (see Sect. 5.4 for details).

There exist applications in which all information in a map needs to be stored and maintained. This includes cases in which hard disk space is used as a secondary memory and maps are temporarily saved to disk until they need to be accessed again. Another use case is the storage of additional node data such as color or terrain information which would be lost in a maximum likelihood encoding. In these cases, we encode nodes by storing their data (occupancy and additional data) and eight bits per node which specify whether a child node exists. This, however, results in considerably larger files as we will show in the experiments.

Note that, analog to the octree representation in memory, the serialized stream does not contain any actual 3D coordinates. To reconstruct a map, only the location of the root node needs to be known. All other spatial relationships between the nodes are implicitly stored in the encoding.

4.4 Our OctoMap Implementation

OctoMap is available as a self-contained C++ library. It is released under the BSD-license and can be obtained from http://octomap.github.com. The source code is thoroughly documented and the library uses CMake to support several platforms (Linux and Mac OS X with GCC, Windows with MinGW or Visual Studio). Within the Robot Operating System (ROS), OctoMap is available as a pre-compiled Debian package, e.g., for the Ubuntu distribution[1]. Further ROS integration is available in the packages octomap_ros and octomap_msgs.

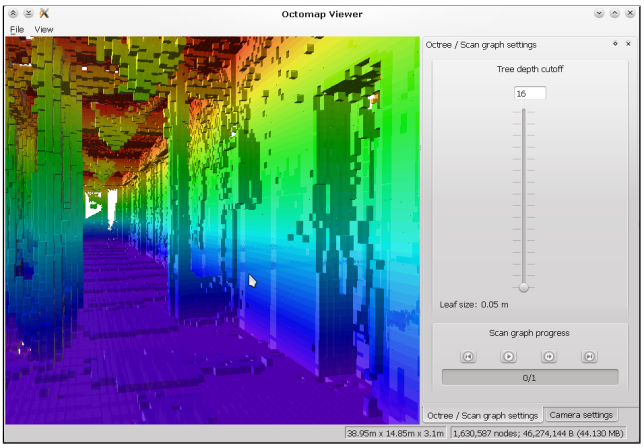

Fig. 9 The OctoMap visualization application octovis

OctoMap can be easily integrated into any other framework by compiling and linking against it with the help of pkg-config, or with the find_package mechanism in the CMake build system.

An OpenGL-based 3D visualization application is available along with the library to view stored octree files and to incrementally build up maps from range data, which eases troubleshooting and map data inspection (see Fig. 9). It also offers basic editing functionality.

4.4.1 Integrating Sensor Measurements

Individual range measurements are integrated using ray-casting by calling the method *insertRay(·)* of the occupancy octree class *OcTree*. This updates the end point of the measurement as occupied while all other voxels along a ray to the sensor origin are updated as free.

Point clouds, e.g., from 3D laser scans or stereo cameras are integrated using *insertScan(·)*. This batch operation has been optimized to be more efficient than tracing each single ray from the origin.

Finally, a single node in the octree can be updated with a point measurement by calling *updateNode(·)*.

4.4.2 Accessing Data

Individual octree nodes can be accessed by searching for their coordinate. For efficient batch queries, our implementation provides iterators to traverse the octree analogous to a standard C++ container class. With these iterators, all nodes,

[1] http://www.ros.org/wiki/octomap

leaf nodes, or leaf nodes in a certain bounding box can be queried or they can be filtered according to further criteria.

Ray intersection queries, i.e., casting a ray from an origin into a given direction until it hits an occupied volume, are an important use-case for a 3D map in robotics. This kind of query is used for visibility checks or to localize with range sensors. Thus, we provide this functionality in the *castRay*(·) method.

5 Evaluation

The approach presented in this paper has been evaluated using several real world datasets as well as simulated ones. The experiments are designed to verify that the proposed representation is meeting the requirements formulated in the introduction. More specifically, we demonstrate that our approach is able to adequately model various types of environments and that it is an updatable and flexible map structure that can be compactly stored.

For evaluation, we used the current implementation of OctoMap 1.5.1[2]. The evaluated datasets are available online[3] and can be converted from 3D point clouds into octree maps with the tool `graph2tree`, which also prints all necessary statistics.

5.1 Sensor Model for Laser Range Data

OctoMap can be used with any kind of distance sensor, as long as an inverse sensor model is available. Since our real-world datasets were mostly acquired with laser range finders, we employ a beam-based inverse sensor model which assumes that endpoints of a measurement correspond to obstacle surfaces and that the line of sight between sensor origin end endpoint does not contain any obstacles. The occupancy probability of all volumes is initialized to the uniform prior of $P(n) = 0.5$. To efficiently determine the map cells which need to be updated, a ray-casting operation is performed that determines voxels along a beam from the sensor origin to the measured endpoint. For efficiency, we use a 3D variant of the Bresenham algorithm to approximate the beam (Amanatides and Woo, 1987). Volumes along the beam are updated as described in Sect. 3.2 using the following inverse sensor model:

$$\mathrm{L}(n \mid z_t) = \begin{cases} l_{\mathrm{occ}} & \text{if beam is reflected within volume} \\ l_{\mathrm{free}} & \text{if beam traversed volume} \end{cases} \quad (7)$$

Throughout our experiments, we used log-odds values of $l_{\mathrm{occ}} = 0.85$ and $l_{\mathrm{free}} = -0.4$, corresponding to probabilities of 0.7 and 0.4 for occupied and free volumes, respectively.

[2] https://github.com/OctoMap/octomap/archive/v1.5.3.tar.gz
[3] http://ais.informatik.uni-freiburg.de/projects/datasets/octomap

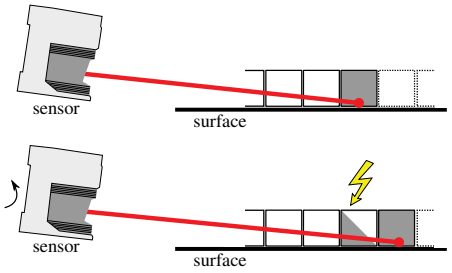

Fig. 10 A laser scanner sweeps over a flat surface at a shallow angle by rotating. A cell measured occupied in the first scan (top) is updated as free in the following scan (bottom) after the sensor rotated. Occupied cells are visualized as gray boxes, free cells are visualized in white.

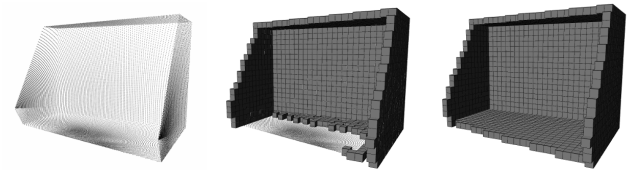

Fig. 11 A simulated noise-free 3D laser scan (left) is integrated into our 3D map structure. Sensor sweeps at shallow angles lead to undesired discretization effects (center). By updating each volume at most once, the map correctly represents the environment (right). For clarity, only occupied cells are shown.

The clamping thresholds are set to $l_{\mathrm{min}} = -2$ and $l_{\mathrm{max}} = 3.5$, corresponding to the probabilities of 0.12 and 0.97. We experimentally determined these values to work best for our use case of mapping mostly static environments with laser range finders, while still preserving map updatability for occasional changes. By adapting these changeable thresholds, a stronger compression can be achieved. As we will evaluate in Sect. 5.6, there is a trade-off between map confidence and compression.

Discretization effects of the ray-casting operation can lead to undesired results when using a sweeping laser range finder. During a sensor sweep over flat surfaces at shallow angles, volumes measured occupied in one 2D scan may be marked as free in the ray-casting of following scans. This effect is illustrated in Fig. 10. Such undesired updates usually creates holes in the modeled surface, as shown in the example in Fig. 11. To overcome this problem, we treat a collection of scan lines in a sensor sweep from the same location as single 3D point cloud in our mapping approach. Since measurements of laser scanners usually result from reflections at obstacle surfaces, we ensure that the voxels corresponding to endpoints are updated as occupied. More precisely, whenever a voxel is updated as occupied according to Eq. (7), it is not updated as free in the same measurement update of the map. By updating the map in this way, the described effect can be prevented and the environment is represented accurately, as can be seen in Fig. 11 (right).

5.2 3D Models from Real Sensor Data

In this experiment, we demonstrate the ability of our approach to model real-world environments. A variety of different datasets has been used. Note that the free space was explicitly modeled in the experiments but is not shown in the figures for clarity.

The indoor dataset called FR-079 corridor was recorded using a Pioneer2 AT platform equipped with a SICK LMS laser range finder on a pan-tilt unit. We reduced odometry errors by applying a 3D scan matching approach. The robot traversed the corridor of building 079 at the Freiburg campus three times, resulting in 66 3D scans with 6 million end points in total. When processing this dataset, we limited the maximum range of the laser beams to 10 m. This removes stray measurements outside of the building which were observed through windows. Figure 12 shows the resulting map.

A fairly large outdoor dataset was recorded at the computer science campus in Freiburg[4]. It consists of 81 dense 3D scans covering an area of 292 m × 167 m along a trajectory of 723 m. This dataset contains a total of 20 million end points. In a further experiment, we used laser range data of the New College data set (Smith et al., 2009) (Epoch C, 14 million end points in total). This data was recorded in a large-scale outdoor environment with two fixed laser scanners sweeping to the left and right side of the robot as it advances. For this dataset, an optimized estimate of the robot's trajectory generated by visual odometry was used (Sibley et al., 2009). The resulting outdoor maps are shown in Fig. 13.

Finally, we integrated data of the freiburg1_360 RGBD-dataset into our map representation with a total of 210 million end points from the Microsoft Kinect sensor (see Sect. 3.5.1). The final map, visualized in Fig. 5, represents an office environment at a resolution of 2 cm. In this map, we additionally stored per-voxel color information.

5.3 Map Accuracy

This experiment demonstrates how accurate a 3D map represents the data that was used to build that map. Note that this particular evaluation is independent of the underlying octree structure since our mapping approach is able to model the same data as a 3D grid. We measure the accuracy as the percentage of correctly mapped cells in all 3D scans. A 3D map cell counts as correctly mapped, if it has the same maximum-likelihood state (free or occupied) in the map and the evaluated 3D scan. The scan is hereby treated as if it were inserted into the already-built map, i.e., endpoints must be occupied and all cells along a ray between the sensor and the endpoint must be free. As a second measure, we cross-validate the map by skipping each 5th scan when building the map, and using these skipped scans to evaluate the percentage of correctly mapped cells.

[4] Courtesy of B. Steder, available at http://ais.informatik.uni-freiburg.de/projects/datasets/fr360/

Map dataset	Accuracy	Cross-validation
FR-079 corridor (5 cm)	97.27%	96.00%
Freiburg campus (10 cm)	97.89%	95.80%
New College (Ep. C) (10 cm)	98.79%	98.46%

Table 1 Map accuracy and cross-validation as percentage of correctly mapped cells between evaluated 3D scans and the built map. For the accuracy, we used all scans for map construction and evaluation. For cross-validation, we used 80% of all scans to build the map, and the remaining 20% for evaluation.

The results in Table 1 show that our mapping approach accurately represents the environment. The remaining error is most likely due to sensor noise, discretization effects, or a not completely perfect scan alignment. The cross-validation results only lose little accuracy, which demonstrates that the probabilistic sensor model yields realistic and predictive results.

5.4 Memory Consumption

In this experiment, we evaluate the memory consumption of our approach. Several datasets were processed at various tree resolutions. We analyzed the memory usage of our representation with and without performing octree compression, as well as the maximum-likelihood compression that converts each node to be either completely free or occupied. For comparison, we also determined the amount of memory that would be required by an optimally aligned 3D grid of minimal size that is initialized linearly in memory. According to Sect. 4.1, the memory consumption of occupancy stored in an octree on a 32-bit architecture is given by

$$\text{mem}_{\text{tree}} = n_{\text{inner}} \times 40\,\text{B} + n_{\text{leafs}} \times 8\,\text{B}\,, \tag{8}$$

where n_{inner} is the number of inner nodes and n_{leafs} the number of leaf nodes. The size of the minimal 3D grid storing the same information (one float for the occupancy probability) is given by

$$\text{mem}_{\text{grid}} = \frac{x \times y \times z}{r^3}\,4\,\text{B}\,, \tag{9}$$

where x, y, z is the size of the map's minimal bounding box in each dimension and r is the map resolution.

We furthermore wrote each map to disk using the full probabilistic model and the compressed binary format described in Sect. 4.3, and evaluated the resulting file sizes.

The memory usage for exemplary resolutions is given in Table 2. It can be seen that high compression ratios can be achieved especially in large outdoor environments. Here,

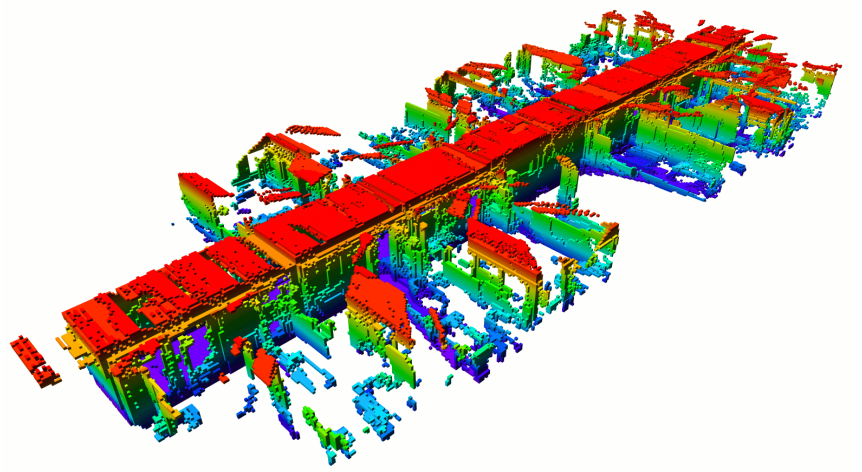

Fig. 12 3D map of the FR-079 corridor dataset, as seen from the top. The structure of the adjacent rooms has been partially observed through the glass doors (size of the scene: 43.7 m × 18.2 m × 3.3 m).

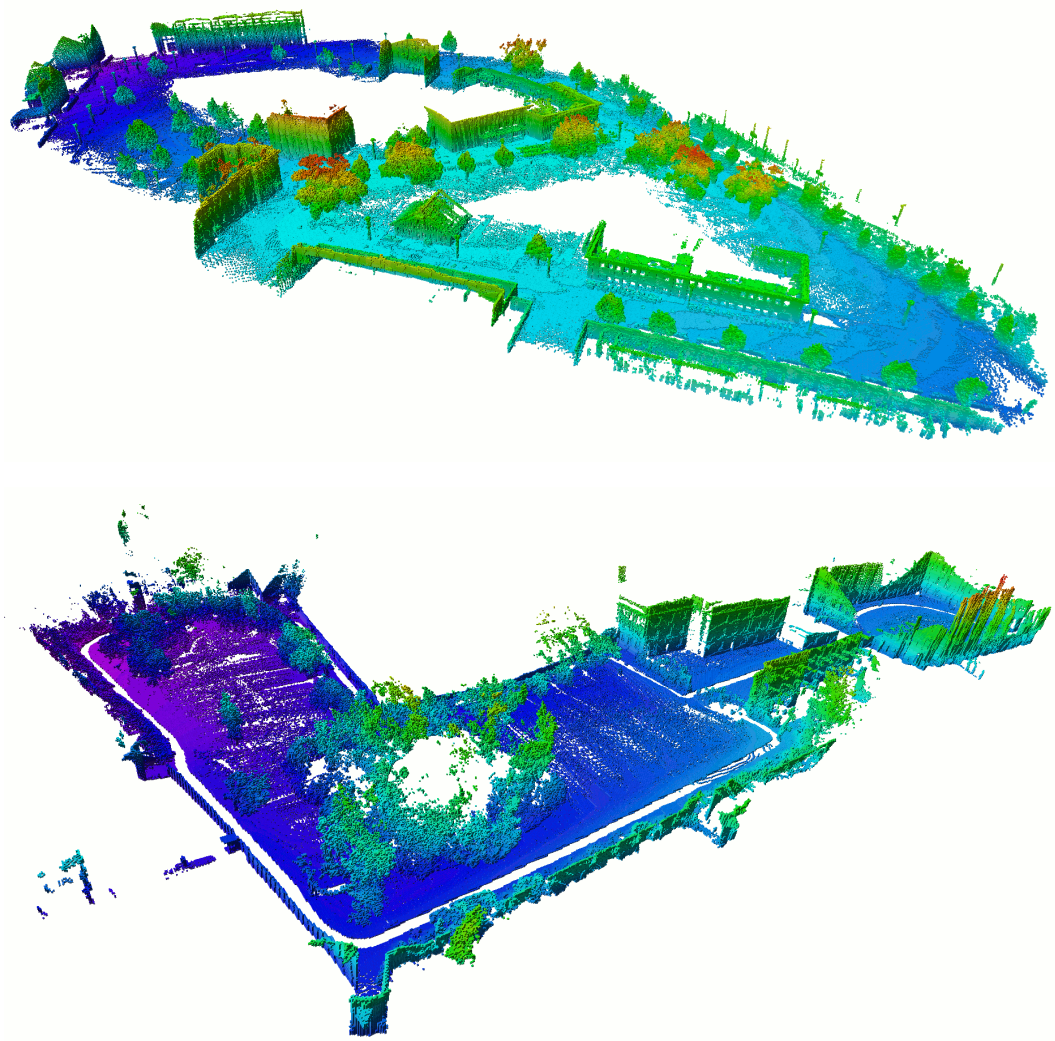

Fig. 13 Resulting octree maps of two outdoor environments at 0.2 m resolution. For clarity, only occupied volumes are shown with height visualized by a color (gray scale) coding. Top: Freiburg campus dataset (size of the scene: 292 m × 167 m × 28 m), bottom: New College dataset (size of the scene: 250 m × 161 m × 33 m).

				Memory w. octree compression [MB]			File size [MB]	
Map dataset	Mapped area [m^3]	Res. [cm]	Mem. 3D grid [MB]	None	Pruned	Max. likelih.	Full	Lossy
FR-079 corridor	$43.7 \times 18.2 \times 3.3$	5	78.88	73.55	41.62	24.72	15.76	0.67
		10	10.01	10.87	7.22	5.02	2.70	0.14
Freiburg campus	$292 \times 167 \times 28$	10	5162.90	1257.57	990.66	504.76	379.70	13.82
		20	648.52	187.93	130.24	74.12	49.68	2.00
		80	10.58	4.55	4.12	3.09	1.53	0.08
New College (Ep. C)	$250 \times 161 \times 33$	10	5058.76	607.92	395.42	230.33	148.75	6.40
		20	633.64	91.33	50.57	35.95	18.65	0.99
		80	10.13	2.34	1.79	1.69	0.63	0.05
freiburg1_360 (RGBD)	$7.9 \times 7.3 \times 4.6$	2	252.99*	159.97*	45.52*	20.05	21.59*	0.52
		5	16.19*	11.24*	4.55*	2.52	2.11*	0.07

Table 2 Memory consumption of different octree compression types compared to full 3D occupancy maps (called 3D grid) on a 32-bit architecture. Octree compression in memory is achieved by merging identical children into the parent node (called Pruned). A more efficient but more lossy compression in memory is achieved by converting each node to its maximum-likelihood value (completely free or occupied) followed by pruning the complete tree. A maximum-likelihood tree containing only free and occupied nodes can then be serialized to a compact binary file format (called Lossy file). (*): Voxels contain the full color information from the RGBD dataset.

pruning will merge considerable amounts of free space volumes and areas of unknown space don't use any memory. Note that a 3D grid of the outdoor data sets with a resolution of 10 cm would not even fit into the addressable main memory of a 32-bit machine. On the other hand, our map structure is also able to model fine-graded indoor environments with moderate memory requirements. In very confined spaces, an optimally aligned 3D grid may take less memory than an uncompressed mapping octree. However, this effect is diminished as soon as compression techniques are used.

The evolution of memory consumption over time is shown in Fig. 14. Memory usage grows when the robot explores new areas (scans 1–22 and 39–44 in FR-079 corridor, scans 1–50 and 65–81 in Freiburg campus). In the remaining time, previously mapped areas were revisited where memory usage remained nearly constant or even decreased due to pruning.

As expected, memory usage increases exponentially with the tree resolution. This effect can be seen in Fig. 15, where we used a logarithmic scaling in the plot.

Table 2 gives the file sizes of the serialized binary maximum likelihood map (denoted as "Lossy") and the full probabilistic model ("Full"). Note that map files can be compressed even further by using standard file compression methods. Even maps of the fairly large outdoor datasets Freiburg campus and New College result in file sizes of less than 14 MB.

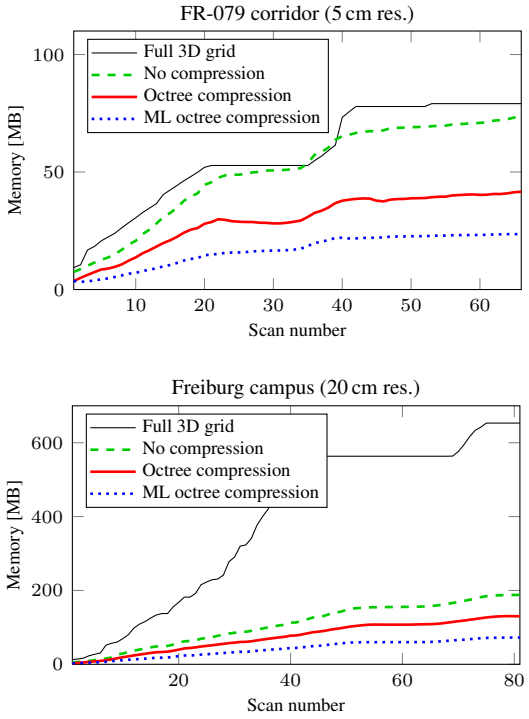

Fig. 14 Memory usage while mapping the two data sets FR-079 corridor and Freiburg campus.

5.5 Runtimes

In the following experiments, we analyzed the time required to integrate and access data in our framework. All runtimes were evaluated on a single core of a standard desktop CPU (Intel Core i7-2600, 3.4 GHz) for various map data sets.

5.5.1 Map Generation

First, we analyzed the time required to generate maps by integrating range data. This time depends on the map resolution and the length of the beams that are integrated. We processed the FR-079 corridor and Freiburg campus datasets both with the full laser range (up to 50 m) and with a limited

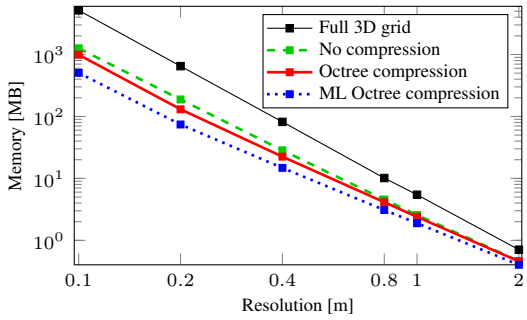

Fig. 15 Effect of resolution on memory usage of the Freiburg campus dataset. Note that a logarithmic scaling is used.

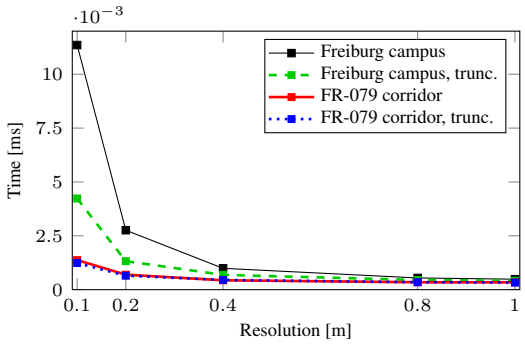

Fig. 16 Average time to update an octree map by inserting one data point for the datasets Freiburg campus and FR-079 corridor. The truncated versions insert rays up to a maximum sensor range of 10 m only.

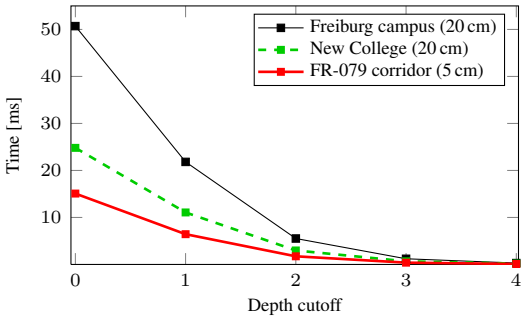

Fig. 17 Time to traverse all octree leaf nodes in several maps. By limiting the depth of the query (called depth cutoff) a coarser map is traversed.

maximum range of 10 m for several resolutions. The average insert times for one beam are given in Fig. 16.

In our experiments, 3D scans usually consisted of about 90 000 – 250 000 valid measurements. Typically, such a scan could be integrated into the map in less than a second. This demonstrates that our current implementation can cope even with the demanding data of RGBD-cameras that output up to 300 000 points at fast frame rates, albeit at shorter ranges.

With long measurement beams and large outdoor areas as in Freiburg campus dataset, a speedup can be obtained by limiting the map update range. Indoors, however, where only few sensor beams reach far, there is no noticeable speedup by limiting the sensor range.

5.5.2 Map Queries

We evaluated the time to traverse all leaf nodes (free or occupied) in an existing map using iterators (see Sect. 4.4.2). The depth of a query can be limited during run time which in our data structure is equivalent to a map query in a coarser map. This allows for more efficient tree traversals in those cases when a coarser resolution is sufficient.

Figure 17 shows the time to traverse several maps to their maximum tree depth of 16 corresponding to the full map resolution (depth cutoff=0). The plot furthermore gives the times to query all leaf nodes when the query depth is limited. Each increment in the depth cutoff doubles the edge length of the smallest voxels and speeds up the traversal by a factor of about two. It can be seen that map traversals are efficient. Even at full map resolution, the large map of the Freiburg campus containing 1 087 014 occupied and 3 377 882 free leaf nodes can be traversed within 51 ms.

5.6 Clamping parameters

Finally, we analyzed the impact of the clamping thresholds on map accuracy and compression. Since these thresholds provide a lower and upper bound for the occupancy probability, information close to $P = 0$ and $P = 1$ is lost compared to the full map with no clamping. A clamped map represents an approximation of the full map, thus we use the Kullback-Leibler divergence (KLD) summed over the complete map as measure. Since occupancy is a binary random variable with the discrete states free and occupied, the KLD of a clamped map M_c from the full map M_f can be computed by summing over all map nodes n:

$$\mathrm{KLD}(M_f, M_c) = \sum_n \left(\ln\left(\frac{P(n)}{Q(n)}\right) P(n) + \ln\left(\frac{1-P(n)}{1-Q(n)}\right)(1-P(n)) \right), \tag{10}$$

where $P(n)$ is the occupancy probability of node n in M_f, and $Q(n)$ in M_c.

The results for a series of occupancy ranges from $[0:1]$ (no clamping, lossless) to $[0.4:0.6]$ (strong clamping, most loss) and different maps can be seen in Fig. 18. The values for our chosen default threshold $[0.12:0.97]$ are shown as thin horizontal lines, dashed blue for the memory consumption and red for the KLD. This clamping range was chosen primarily to work best in the context of laser-based mapping and occasional changes in the environment, such as people moving through the scans or doors closing. As can be seen, a stronger compression can be achieved with higher clamping, at the cost of losing map confidence. In the most degenerated case, one sensor update can be enough to mark a voxel as completely

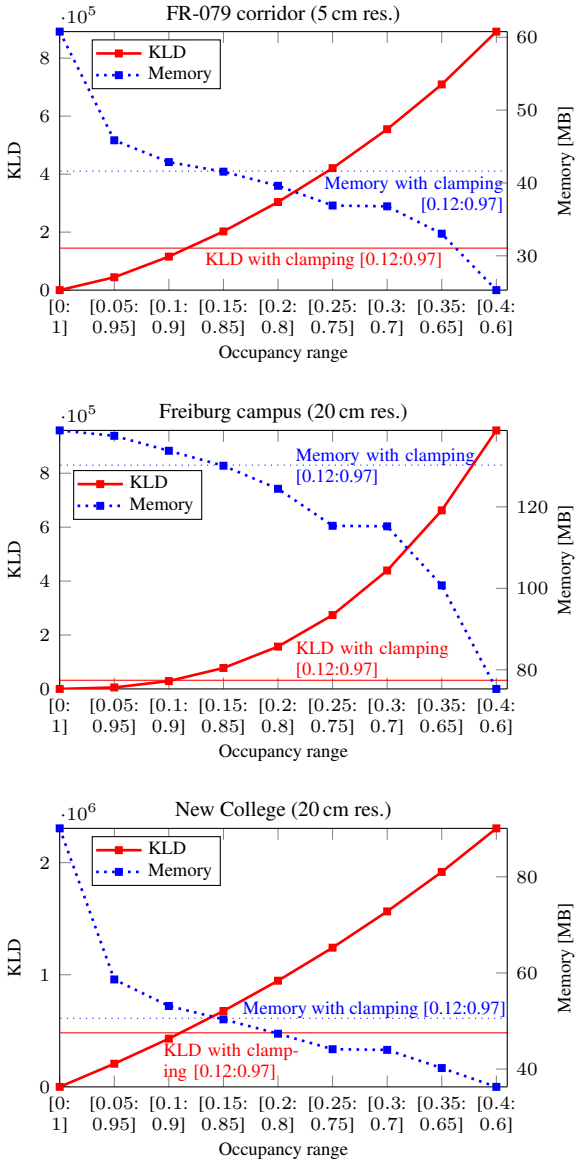

Fig. 18 Effect of different clamping ranges on map compression and accuracy in our three datasets. A higher clamping, resulting in a smaller occupancy range, increases the efficiency of the octree compression (memory consumption, dashed blue). The Kullback-Leibler divergence (KLD, red) measures the information loss between the unclamped map with full probabilities in [0:1] and a clamped representation. Our default clamping range [0.12:0.97] is shown for comparison by horizontal lines in blue (dashed) for memory consumption and red for the KLD.

free or occupied, losing any ability to filter noise with a probabilistic update. Note that, while clamping is beneficial for map compression, even with no clamping the lossless compressed maps are smaller than a 3D grid (cf. Table 2).

6 Case Studies

Since its first introduction in 2010 (Wurm et al., 2010), the OctoMap framework received a considerable interest and has been used in several applications. These include 6D localization (Hornung et al., 2010), autonomous navigation with air vehicles (Heng et al., 2011; Müller et al., 2011), autonomous navigation with humanoid robots (Oßwald et al., 2012; Maier et al., 2012), 3D exploration (Shade and Newman, 2011; Dornhege and Kleiner, 2011), 3D SLAM (Hertzberg et al., 2011), 3D arm navigation (Ciocarlie et al., 2010), semantic mapping (Blodow et al., 2011), and navigation in cluttered environments (Hornung et al., 2012).

In the following, we will describe some of these use cases in more detail in order to demonstrate the versatility and ease of integration of the OctoMap library.

6.1 Localization in 3D

In our previous work (Hornung et al., 2010), we developed a localization method based on OctoMap as 3D environment model. In this approach, the 6D torso pose of a humanoid robot in a complex indoor environment is tracked with Monte Carlo localization based on 2D laser range measurements, as well as IMU and joint encoder data. For the particle filter observation model, we first used the endpoint model and later an optimized ray-casting method in combination with visual observations for local refinement (Oßwald et al., 2012). The resulting localization is highly accurate and even enables the humanoid to climb spiral staircases. Our implementation is available open-source[5] and uses the ray-casting functionality in OctoMap (see Sect. 4.4.2). This enables the re-use for other robot localization systems.

6.2 Tabletop Manipulation

The ROS collider package[6] builds a collision map based on 3D point clouds. Sensor data from several sources, such as a tilting laser and a stereo camera, are fused using OctoMap. Octree nodes were extended to store a time stamp attribute that allows to gradually clear out nodes in dynamically changing environments. This new collision map enables the ROS arm navigation and grasping pipeline (Ciocarlie et al., 2010) to dynamically react to changes and to cope with sensor noise. In contrast to the previous fixed-size voxel grid, the new implementation allows for an initially unbounded workspace, the integration of data from multiple sensors, and it is more memory-efficient.

[5] http://www.ros.org/wiki/humanoid_localization
[6] http://www.ros.org/wiki/collider

6.3 Navigation in Cluttered Environments

OctoMap was furthermore used to create a navigation module for mobile manipulation. In this project, a PR2 robot picked up large objects from one table with two arms and carried it to another table through narrow passages (Hornung et al., 2012). The system integrates 3D sensor data in OctoMap. The resulting 3D occupancy map is then used to perform collision checks based on the robot's full kinematic configuration and the attached objects. Multi-layered projected 2D maps and an anytime planner using motion primitives allow for planning in almost real time with bounded sub-optimality. The navigation system and incremental mapping framework based on OctoMap are both available open-source in ROS[7].

7 Conclusion

In this paper, we presented OctoMap, an open source framework for three-dimensional mapping. Our approach uses an efficient data structure based on octrees that enables a compact memory representation and multi-resolution map queries. Using probabilistic occupancy estimation, our approach is able to represent volumetric 3D models that include free and unknown areas. The proposed approach uses a bounded per-volume confidence that allows for a lossless compression scheme and leads to substantially reduced memory usage. We evaluated our approach with various real-world data sets. The results demonstrate that our approach is able to model the environment in an accurate way and, at the same time, minimizes memory requirements.

OctoMap can easily be integrated into robotic systems and has already been successfully applied in a variety of robotic projects. The implementation is available as BSD-licensed C++ source code. Data sets are available online to verify our experimental results and to compare against them.

Acknowledgements The authors would like to thank J. Müller, S. Oßwald, R.B. Rusu, R. Schmitt, and C. Sprunk for the fruitful discussions and their contributions to the OctoMap library.

References

Amanatides J, Woo A (1987) A fast voxel traversal algorithm for ray tracing. In: Proceedings of Eurographics, Amsterdam, The Netherlands

Blodow N, Goron L, Marton Z, Pangercic D, Ruhr T, Tenorth M, Beetz M (2011) Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Botsch M, Wiratanaya A, Kobbelt L (2002) Efficient high quality rendering of point sampled geometry. In: EGRW '02: Proc. of the 13th Eurographics workshop on Rendering, pp 53–64

Ciocarlie M, Hsiao K, Jones EG, Chitta S, Rusu RB, Sucan IA (2010) Towards reliable grasping and manipulation in household environments. In: Intl. Symposium on Experimental Robotics (ISER)

Cole D, Newman P (2006) Using laser range data for 3D SLAM in outdoor environments. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Dornhege C, Kleiner A (2011) A frontier-void-based approach for autonomous exploration in 3D. In: IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)

Douillard B, Underwood J, Melkumyan N, Singh S, Vasudevan S, Brunner C, Quadros A (2010) Hybrid elevation maps: 3D surface models for segmentation. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Dryanovski I, Morris W, Xiao J (2010) Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Einhorn E, Schröter C, Groß HM (2011) Attention-driven monocular scene reconstruction for obstacle detection, robot navigation and map building. Robotics & Autonomous Systems 59(5):296–309

Elseberg J, Borrmann D, Nüchter A (2011) Efficient processing of large 3d point clouds. In: Proc. of the XXIII Int. Symp. on Information, Communication and Automation Technologies (ICAT '11)

Endres F, Hess J, Engelhard N, Sturm J, Cremers D, Burgard W (2012) An evaluation of the RGB-D SLAM system. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Fairfield N, Kantor G, Wettergreen D (2007) Real-time SLAM with octree evidence grids for exploration in underwater tunnels. Journal of Field Robotics

Fournier J, Ricard B, Laurendeau D (2007) Mapping and exploration of complex environments using persistent 3D model. In: Computer and Robot Vision, 2007. Fourth Canadian Conf. on, pp 403–410

Girardeau-Montaut D, Roux M, Marc R, Thibault G (2005) Change detection on points cloud data acquired with a ground laser scanner. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 36:30–35

Gutmann JS, Fukuchi M, Fujita M (2008) 3D perception and environment map generation for humanoid robot navigation. Int J Rob Res 27(10):1117–1134

Habbecke M, Kobbelt L (2007) A surface-growing approach to multi-view stereo reconstruction. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)

Hadsell R, Bagnell JA, Hebert M (2009) Accurate rough terrain estimation with space-carving kernels. In: Proc. of Robotics: Science and Systems (RSS)

Hebert M, Caillas C, Krotkov E, Kweon IS, Kanade T (1989) Terrain mapping for a roving planetary explorer. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Heng L, Meier L, Tanskanen P, Fraundorfer F, Pollefeys M (2011) Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Hertzberg C, Wagner R, Birbach O, Hammer T, Frese U (2011) Experiences in building a visual slam system from open source components. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Hoppe H, DeRose T, Duchamp T, Mcdonald J, Stuetzle W (1992) Surface reconstruction from unorganized points. SIGGRAPH Computer Graphics 26(2):71–78

Hornung A, Wurm KM, Bennewitz M (2010) Humanoid robot localization in complex indoor environments. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

[7] http://www.ros.org/wiki/3d_navigation and http://www.ros.org/wiki/octomap_server

Hornung A, Phillips M, Jones EG, Bennewitz M, Likhachev M, Chitta S (2012) Navigation in three-dimensional cluttered environments for mobile manipulation. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Kammerl J, Blodow N, Rusu RB, Gedikli S, Beetz M, Steinbach EG (2012) Real-time compression of point cloud streams. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Kelly A, Stentz A, Amidi O, Bode M, Bradley DM, Diaz-Calderon A, Happold M, Herman H, Mandelbaum R, Pilarski T, Rander P, Thayer S, Vallidis N, Warner R (2006) Toward reliable off road autonomous vehicles operating in challenging environments. J of Robotics Research 25(5-6):449–483

Kraetzschmar G, Gassull G, Uhl K (2004) Probabilistic quadtrees for variable-resolution mapping of large environments. In: Ribeiro MI, Victor SJ (eds) Proc. of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal

Krainin M, Henry P, Ren X, Fox D (2011) Manipulator and object tracking for in-hand 3d object modeling. J of Robotics Research 30(11):1311–1327

Laine S, Karras T (2010) Efficient sparse voxel octrees. In: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games

Magnusson M, Duckett T, Lilienthal AJ (2007) Scan registration for autonomous mining vehicles using 3D-NDT. Journal of Field Robotics 24(10):803–827

Maier D, Hornung A, Bennewitz M (2012) Real-time navigation in 3d environments based on depth camera data. In: Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)

Mason J, Ricco S, Parr R (2011) Textured occupancy grids for monocular localization without features. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Meagher D (1982) Geometric modeling using octree encoding. Computer Graphics and Image Processing 19(2):129–147

Moravec H (1996) Robot spatial perception by stereoscopic vision and 3D evidence grids. Tech. Rep. CMU-RI-TR-96-34, Robotics Institute, Pittsburgh, PA

Moravec H, Elfes A (1985) High resolution maps from wide angle sonar. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), St. Louis, MO, USA, pp 116–121

Müller J, Kohler N, Burgard W (2011) Autonomous miniature blimp navigation with online motion planning and re-planning. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Newcombe R, Izadi S, Hilliges O, Molyneaux D, Kim D, Davison A, Kohli P, Shotton J, Hodges S, Fitzgibbon A (2011) KinectFusion: Real-time dense surface mapping and tracking. In: Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on, IEEE, pp 127–136

Nüchter A, Lingemann K, Hertzberg J, Surmann H (2007) 6D SLAM—3D mapping outdoor environments: Research articles. J Field Robot 24(8-9):699–722

Oßwald S, Hornung A, Bennewitz M (2012) Improved proposals for highly accurate localization using range and vision data. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Pathak K, Birk A, Poppinga J, Schwertfeger S (2007) 3D forward sensor modeling and application to occupancy grid based sensor fusion. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Payeur P, Hebert P, Laurendeau D, Gosselin C (1997) Probabilistic octree modeling of a 3-d dynamic environment. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Pfaff P, Triebel R, Stachniss C, Lamon P, Burgard W, Siegwart R (2007) Towards mapping of cities. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), Rome, Italy

Roth-Tabak Y, Jain R (1989) Building an environment model using depth information. Computer 22(6):85–90

Ryde J, Hu H (2010) 3D mapping with multi-resolution occupied voxel lists. Autonomous Robots 28(2):169–185

Schnabel R, Klein R (2006) Octree-based point-cloud compression. In: Symposium on Point-Based Graphics 2006, Eurographics

Shade R, Newman P (2011) Choosing where to go: Complete 3D exploration with stereo. In: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)

Sibley G, Mei C, Reid I, Newman P (2009) Adaptive relative bundle adjustment. In: Proc. of Robotics: Science and Systems (RSS)

Smith M, Baldwin I, Churchill W, Paul R, Newman P (2009) The new college vision and laser data set. International Journal for Robotics Research (IJRR) 28(5):595–599, DOI http://dx.doi.org/10.1177/0278364909103911

Stoyanov T, Magnusson M, Andreasson H, Lilienthal AJ (2010) Path planning in 3d environments using the normal distributions transform. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Sturm J, Engelhard N, Endres F, Burgard W, Cremers D (2012) A benchmark for the evaluation of RGB-D slam systems. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), http://cvpr.in.tum.de/data/datasets/rgbd-dataset/download

Surmann H, Nüchter A, Hertzberg J (2003) An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. Robotics and Autonomous Systems 45(3):181–198

Triebel R, Pfaff P, Burgard W (2006) Multi-level surface maps for outdoor terrain mapping and loop closing. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)

Weise T, Wismer T, Leibe B, Van Gool L (2009) In-hand scanning with online loop closure. In: ICCV Workshops

Whelan T, Kaess M, Fallon M, Johannsson H, Leonard J, McDonald J (2012) Kintinuous: Spatially extended KinectFusion. Tech. rep., URL http://hdl.handle.net/1721.1/71756

Wilhelms J, Van Gelder A (1992) Octrees for faster isosurface generation. ACM Trans Graph 11(3):201–227

Wurm KM, Hornung A, Bennewitz M, Stachniss C, Burgard W (2010) OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In: Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation

Wurm KM, Hennes D, Holz D, Rusu RB, Stachniss C, Konolige K, Burgard W (2011) Hierarchies of octrees for efficient 3d mapping. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), San Francisco, CA, USA

Yguel M, Aycard O, Laugier C (2007a) Update policy of dense maps: Efficient algorithms and sparse representation. In: Field and Service Robotics, Results of the Int. Conf., FSR 2007, vol 42, pp 23–33

Yguel M, Keat CTM, Braillon C, Laugier C, Aycard O (2007b) Dense mapping for range sensors: Efficient algorithms and sparse representations. In: Proceedings of Robotics: Science and Systems