Article

The International Journal of Robotics Research
2018, Vol. 37(6) 566–584

DOI: 10.1177/0278364918771476
journals.sagepub.com/home/ijr

Towards real-time 3D continuous occupancy mapping using Hilbert maps

Vitor Guizilini and Fabio Ramos

Abstract

The ability to model the surrounding space and determine which areas are occupied is of key importance in many robotic applications, ranging from grasping and manipulation to path planning and obstacle avoidance. Occupancy modeling is often hindered by several factors, such as: real-time constraints, that require quick updates and access to estimates; quality of available data, that may contain gaps and partial occlusions; and memory requirements, especially for large-scale environments. In this work we propose a novel framework that elegantly addresses all these issues, by producing an efficient non-stationary continuous occupancy function that can be efficiently queried at arbitrary resolutions. Furthermore, we introduce techniques that allow the learning of individual features for different areas of the input space, that are better able to model its contained information and promote a higher-level understanding of the observed scene. Experimental tests were conducted on both simulated and real large-scale datasets, showing how the proposed framework rivals current state-of-the-art techniques in terms of computational speed while achieving a substantial decrease (of orders of magnitude) in memory requirements and demonstrating better interpolative powers, that are able to smooth out sparse and noisy information.

Keywords

Occupancy mapping, scene reconstruction, 3D modeling, kernel methods, reproducing kernel Hilbert space

1. Introduction

In this day and age, obtaining information from the surrounding environment is no longer an issue in mobile robotics as standard sensors are able to output million of points in a fraction of a second. Stereo and RGBD cameras produce per-pixel scale-aware dense point clouds that include color information, and 3D laser range sensors produce snapshots of nearby structures at sub-degree resolution and sub-centimeter accuracy. The challenge now is to process and store all this raw information in a useful and efficient manner that can be exploited by both humans and machines. Furthermore, a successful model should be able to extract patterns that describe the same environment in a much more compact way, thus removing redundancy and introducing semantically meaningful representations.

One key application of this collected information is estimating which areas of the surrounding space are empty and therefore can be safely traversed by a moving vehicle, and which are occupied and would result in a collision. This ability lies at the foundation for many other research areas in robotics, ranging from grasping and object manipulation to obstacle avoidance and autonomous navigation. A model that represents the occupancy state of different areas in the input space is known as an *occupancy map* (Thrun et al., 2005) (Figure 1), and other desired properties of such a model include:

1. **probabilistic reasoning**, taking into account imprecisions in sensor measurements;
2. **spatial relationships**, use available information to improve its estimates in unobserved areas;
3. **incremental learning**, allow updates during navigation, as more information is collected by the sensors;
4. **update and query efficiency**, real-time access to model, both to incorporate and retrieve information.

Initial attempts at producing occupancy maps would simply discretize the input space (Elfes, 1989; Moravec, 1996), maintaining a grid of equally sized cells that store the occupancy state of that particular area, either as a binary value or a probability distribution. This approach, however, does not take into account spatial relationships (property 2), since as a simplification technique each cell is treated independently from each other. It is also very memory-intensive,

School of Information Technologies, The University of Sydney, Australia

Corresponding author:
Vitor Campanholo Guizilini, School of Information Technologies, University of Sydney, Building J12, Sydney, NSW 2006, Australia.
Email: vitor.guizilini@sydney.edu.au

particularly when dealing with finer resolutions and higher dimensions, which severely limits its applicability to large-scale and detailed datasets (property 4). Owing to that, over the years substantial work has been done to improve this initial model and circumvent some of its limitations under particular circumstances, such as extension to 3D environments (Nüchter et al., 2007), construction of hybrid elevation maps (Douillard et al., 2010), and use of visual textures to simulate environment features (Mason et al., 2013).

Nowadays, the state-of-the-art technique in 3D grid-based occupancy mapping is arguably OctoMaps, initially proposed by Hornung et al. (2013). Rather than using equally sized cells, the OctoMaps framework employs a well-known tree-like structure (an *octree*) to recursively divide the space into smaller areas. Branches with similar classification are merged or pruned, to control memory requirements and access time (property 4), and new information is incorporated by simply adding internal nodes (property 3). However, each node is still treated independently, and its occupancy state is not reflected in the surrounding space (property 2), which leads to gaps and uninformed regions, particularly when dealing with sparse data.

An alternative to grid-based occupancy maps is the use of kernel methods (Hofmann et al., 2008), in which a continuous function is learned based on available information that maps input space points directly into output occupancy probability states. The resulting model can be queried at arbitrary resolutions and naturally captures spatial relationships between data points, thus offering a formal methodology to reason over the presence of occlusions and data gaps. Callaghan and Ramos (2012) modeled this continuous function as a Gaussian process (Rasmussen and Williams, 2005), a non-parametric Bayesian regression technique that uses a set of functions to learn relationships between observed points, and is then able to extrapolate this information and estimate the state of unobserved areas of the input space.

The resulting framework, called Gaussian process occupancy maps (GPOM), and its corresponding extensions over the past few years, such as dealing with dynamic environments (Callaghan and Ramos, 2015), is currently considered the state-of-the-art in kernel-based occupancy mapping. Dragiev et al. (2011) proposed a novel kernel with spline regularization for the modeling of implicit surfaces, and in Hadsell et al. (2010) the issue of uneven sampling was addressed for the task of 3D terrain modeling. All these approaches, however, do not scale to larger datasets (property 4), since the computational complexity of the Gaussian process framework increases cubically with the number of training points. The use of overlapping local approximations (Soohwan and Jonghyuk, 2013) is able to amortize this increase in computational complexity, but these solutions are circumstantial and do not address the underlying problem of efficient data processing for large-scale modeling, especially in 3D environments.

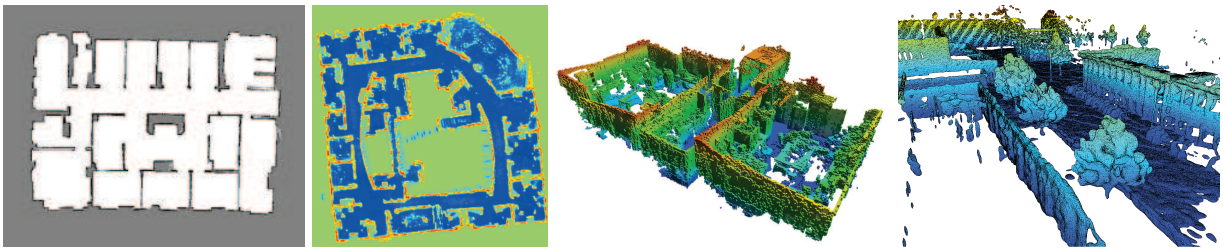
In Ramos and Ott (2015), a novel continuous occupancy technique was introduced, called Hilbert maps, in which real-world complexity is represented linearly by operating on a high-dimensional feature vector, that projects observations into a reproducing kernel Hilbert space (RKHS) (Schölkopf and Smola, 2001). In this high-dimensional space, classification can be performed using simple linear techniques, resulting in a continuous occupancy function (property 2) that can be efficiently updated and queried at arbitrary resolutions (properties 3 and 4). Since its introduction, substantial work has been done to further develop the Hilbert maps framework, such as: introduction of local length scales (Guizilini and Ramos, 2016) to achieve non-stationarity in the input space; automatic kernel selection (Guizilini and Ramos, 2017), to improve the modeling of different structures; overlapping local maps (Doherty et al., 2016), as a way to fuse individual scans into a single global map; and extension to dynamic environments (Senanayake et al., 2016) to address temporal dependencies between scans.

This paper improves on the foundations of the Hilbert maps framework itself, and proposes novel training and inference methodologies that drastically increase computational speed, thus allowing its efficient use in large-scale 3D occupancy mapping applications. The resulting framework, entitled *efficient Hilbert maps* (EHM) rivals current state-of-the-art occupancy mapping techniques, both in terms of computational speed and accuracy, while providing robust results in the presence of sparse and noisy data. Furthermore, it benefits from all the above mentioned extensions, and therefore can be considered the baseline for development of new Hilbert maps applications. In summary, the main contributions of this paper are as follows.

- **Unsupervised feature learning.** A novel clustering technique that automatically determines the number of features necessary to properly describe the environment, alongside their corresponding positions and length-scales.
- **Incremental training.** Novel training strategies that produce optimal weights for the occupancy classifier in a fraction of previously achieved times. We also show how new observations can be efficiently incorporated into the current model, to account for online learning and changing environments.
- **Efficient grid reconstruction.** A novel methodology for the 3D reconstruction of the current Hilbert Maps model at arbitrary resolutions, thus allowing the quick rendering of observed structures.

We then proceed to describe some of its extensions that serve as the basis for the proposed training and inference methodologies

The remainder of this paper is organized as follows: Section 2 provides an overview on the Hilbert Maps framework, as introduced in Ramos and Ott (2015) and later extended in Ramos and Ott (2016). Section 3 introduces the proposed



(a) 2D occupancy grid (Thrun et al., 2005)

(b) 2D Gaussian process occupancy map (Jadidi et al., 2017)

(c) 3D octomap (Hornung et al., 2013)

(d) 3D Hilbert map (Guizilini and Ramos, 2016)

Fig. 1. Examples of occupancy maps produced using different techniques. In (a) each cell stores independent occupancy probabilities, whereas in (b) a continuous function is generated, including occupancy mean and variance estimates. In (c) an octree is used to hierarchically store data points at varying resolution levels, and (d) uses a feature vector to project data points into a high-dimensional space for efficient linear classification.

training and inference methodologies, alongside several extensions in feature generation and kernel placement, that allow the efficient deployment of the Hilbert maps framework. Experiments conducted in large-scale simulated and real environments are presented and discussed in Section 4. Finally, Section 5 concludes the paper and proposes directions for future work in the area of 3D occupancy mapping, using the methodologies described here.

2. Hilbert maps overview

The Hilbert maps framework, first introduced in Ramos and Ott (2015), proposes the modeling of real-world complexity by projecting spatial coordinates $\mathbf{x}$ into a high-dimensional feature representation known as a *Hilbert space* (Sansone, 2012). Furthermore, if point evaluation in this space is a continuous linear functional, i.e. if $\|f - g\|$ is small for functions f and g, then $|f(x) - g(x)|$ is also small for all x, then it becomes a RKHS (Schölkopf et al., 2015). This projection is performed using a feature vector $\Phi(\mathbf{x})$, and the dot product of these feature vectors can be used to approximate popular kernels found in the literature for nonlinear classification (Rasmussen and Williams, 2005). By operating only in terms of kernel evaluations we never have to explicitly perform calculations in this high-dimensional (and potentially infinite) feature space.

Furthermore, it can be shown (Komarek, 2004) that linear separators are almost always adequate to separate between classes in sufficiently high-dimensional spaces. Therefore, in the RKHS a simple classifier can be used to deal even with highly nonlinear behaviors, such as occupancy mapping functions. Here we employ the *logistic regression* classifier (Freedman, 2005), owing to its computational speed and direct extension to online learning. Assuming a training dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathcal{R}^3$ is a point in the 3D space and $y = \{-1, +1\}$ is a binary classification variable that describes the occupancy state of $\mathbf{x}$, the occupancy probability for a query point $\mathbf{x}_*$ is defined as

$$p(y_* = 1|\Phi(\mathbf{x}_*), \mathbf{w}) = \frac{1}{1 + \exp\left(\mathbf{w}^{\mathrm{T}}\Phi(\mathbf{x}_*)\right)} \tag{1}$$

and, conversely, the non-occupancy probability for $\mathbf{x}_*$ is $p(y_* = -1|\Phi(\mathbf{x}_*), \mathbf{w}) = 1 - p(y_* = 1|\Phi(\mathbf{x}_*), \mathbf{w})$. The vector $\mathbf{w}$ represents the weight parameters that describe the discriminative model $p(y|\mathbf{x}, \mathbf{w})$. To estimate the optimal weight parameters $\bar{\mathbf{x}}$ we minimize the regularized negative log-likelihood (RNLL) function:

$$RNLL(\mathbf{w}) = \sum_{i=1}^{N}\left(1 + \exp\left(-y_i\mathbf{w}^{\mathrm{T}}\Phi(\mathbf{x}_i)\right)\right) + R(\mathbf{w}) \tag{2}$$

where $R(\mathbf{w})$ is a regularization function, used to prevent overfitting and promote sparseness in $\mathbf{w}$. A particularly useful property of Equation (2) is its suitability for *stochastic gradient descent* (SGD) optimization (Bottou, 2010), in which each training point is considered individually (or in mini-batches) and contributes with one small step towards a local minimum, given by

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta_t \frac{\delta}{\delta \mathbf{w}} RNNL(\mathbf{w}) \tag{3}$$

where $\eta > 0$ is the learning rate, usually a constant or asymptotically decaying with the number of iterations. For a more complete overview of different SGD techniques and how they affect convergence speed and accuracy, we refer the reader to Ruder (2016). Note that this technique is naturally suited to online learning, since new information can be incorporated into the current model by incrementally performing the stochastic update step given by Equation (3). It also eliminates the need to store all training information simultaneously, which might be unfeasible in large-scale datasets, and instead requires only mini-batches that can be discarded after processing.

2.1. Feature selection

The key insight in the Hilbert maps framework is that its discriminative model is not applied directly to the inputs $\mathbf{x}$, but rather on a high-dimensional vector $\Phi(.)$ calculated directly from $\mathbf{x}$. These feature vectors approximate a Hilbert space by their dot product, such that $\Phi(\mathbf{x}_i)\,\Phi(\mathbf{x}_j) \approx k(\mathbf{x}_i, \mathbf{x}_j)$.

Different kernels have various properties (i.e. smoothness/sharpness, stationarity/non-stationarity) that are better suited to certain applications, and the choice of which one should be used is crucial in the development of kernel-based methods. Arguably the most common kernel used in machine learning tasks is the *radial basis function* (RBF), that can be defined as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)\right) \tag{4}$$

In Ramos and Ott (2015) three different feature vectors that approximate the RBF kernel are described. The first is based on *random Fourier* features (Rahimi and Recht, 2008), that can be approximate arbitrary kernels and has the generic form

$$\Phi(\mathbf{x}) = \frac{1}{\sqrt{M}}\left[e^{-i\mathbf{s}_1\cdot\mathbf{x}}, \ldots, e^{-i\mathbf{s}_M\cdot\mathbf{x}}\right] \tag{5}$$

where M is the dimensionality of $\Phi(.)$ and $\mathbf{s}$ are samples obtained from the spectral density $S(\mathbf{s})$ of k. For the particular case of the RBF kernel, we have $e^{i\mathbf{s}\cdot\mathbf{x}} = \cos(\mathbf{s}\cdot\mathbf{x}) - i\sin(\mathbf{s}\cdot\mathbf{x}) = \cos(\mathbf{s}\cdot\mathbf{x} + b)$, where $S(\mathbf{s}) \sim \mathcal{N}(0, 2\sigma^{-2}I)$ and $b \sim [-\pi, +\pi]$ is introduced to rotate the projection about the real axis by a random amount (the imaginary axis must be zero for real kernels). The random Fourier feature vector can then be defined as

$$\Phi_{RF}(\mathbf{x}) = \frac{1}{\sqrt{M}}\left[\cos(\mathbf{s}_1\mathbf{x} + b_1), \ldots, \cos(\mathbf{s}_M\mathbf{x} + b_M)\right]^{\mathrm{T}} \tag{6}$$

The second is based on the *Nyström* approximation (Williams and Seeger, 2000), in which a kernel matrix $K_{NN} = \left[k(\mathbf{x}_i, \mathbf{x}_j)\right]_{N\times N}$ is approximated by projecting it into a set of M inducing points, denoted as $\{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_M\}$. Then, we can write $K_{NN} \approx K_{NM}\hat{K}_{MM}^{\dagger}K_{NM}^{\mathrm{T}}$, where $K_{NM} = \left[k(\mathbf{x}_i, \hat{\mathbf{x}}_j)\right]_{N\times M}$ is a kernel matrix between training and inducing points, $\hat{K}_{MM} = \left[k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)\right]_{M\times M}$ is a kernel matrix between inducing points and $K^{\dagger}$ is the pseudo-inverse of K. This approximation can be factorized and put into a feature vector of the form

$$\Phi_{NS}(\mathbf{x}) = \hat{D}^{-1/2}\hat{V}^{\mathrm{T}}\left(k(\mathbf{x}, \hat{\mathbf{x}}_1), \ldots, k(\mathbf{x}, \hat{\mathbf{x}}_M)\right)^{\mathrm{T}} \tag{7}$$

where $\hat{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_r)$ are the first r non-negative eigenvalues of $\hat{K}_{MM}$ in ascending order and $\hat{V} = (\mathbf{v}, \ldots, \mathbf{v}_r)$ are the corresponding eigenvectors. Note that, while the *random Fourier* approximation is dataset independent (i.e. it can be precomputed solely based on the choice of kernel), the *Nyström* approximation depends on the particular training dataset being used for each application.

Third, a sparse approximation is proposed, aiming to produce a set of features that can more easily and efficiently be optimized using SGD. It is based on the sparse kernel introduced in Melkumyan and Ramos (2009), that vanishes to exactly zero when $r = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\,\Sigma^{-1}(\mathbf{x}_i - \mathbf{x}_j)} \geq 1$, where Σ is a 3×3 symmetric positive-definite length-scale matrix. It also approximates the smoothness of the RBF kernel, being four times differentiable. This sparse kernel has the form:

$$k_{SP}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \frac{2+\cos(2\pi r)}{3}(1-r) + \frac{1}{2\pi}\sin(2\pi r) & \text{if } r < 1 \\ 0 & \text{if } r \geq 1 \end{cases} \tag{8}$$

The sparse feature vector is defined by simply concatenating sparse kernels in each dimensionality:

$$\phi_{SP}(\mathbf{x}) = \begin{bmatrix} k_{SP}(\mathbf{x}, \hat{\mathbf{x}}_1) \\ k_{SP}(\mathbf{x}, \hat{\mathbf{x}}_2) \\ \vdots \\ k_{SP}(\mathbf{x}, \hat{\mathbf{x}}_M) \end{bmatrix} \tag{9}$$

where, similar to the Nyström feature, $\{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_M\}$ is a set of inducing points that represents the center of each kernel (the authors propose uniform sampling from the observed point cloud or placement on an equally sized grid). Note that, during calculations, kernels with $r \geq 1$ can be safely ignored, since their value will be exactly zero in that dimension of the feature vector.

3. Efficient Hilbert maps

Here we introduce and discuss techniques that significantly improve the performance of the Hilbert maps framework, both in terms of computational speed and memory requirements. The objective is to achieve occupancy mapping under real-time constraints, in which a model of observed structures is generated incrementally as new information about the environment is collected, and can be queried as necessary for tasks such as path planning, active search, etc.

In the next sections we address several key aspects in achieving this improvement in performance, namely: *localized length-scales*, that promote non-stationarity in the input space as a way to improve modeling results with fewer RKHS dimensions; *automatic kernel selection*, in which different kernels are used to model various areas of the input space; *efficient feature placement*, that introduces a novel way to automatically determine the number and location of the inducing points that will be used to produce the feature vector; *batch training and inference*, which improves upon the optimization and querying aspects of the classifier used by the Hilbert maps framework to produce occupancy mapping estimates; and *incremental reconstruction*, an efficient way to produce the isometric surface that represents transitions between occupied and unoccupied areas of the input space.

3.1. Localized length-scales

Owing to its sparsity, the kernel introduced in Equation (8) is the most computationally efficient out of the three

approximations presented in Ramos and Ott (2015). However, a few questions still remain: where should the inducing points be placed, and how many are enough for an accurate representation of the observed structures? Random sampling might lead to redundant kernels, that describe very similar portions of the input space, whereas placement on an equally sized grid might produce kernels in areas where there are no observations, and areas with highly detailed structures are underrepresented.

Owing to this, Guizilini and Ramos (2016) proposed a novel methodology to generate inducing points for sparse kernel placement, in which the input point cloud is clustered and the clusters centers are used as inducing points. Furthermore, the covariance matrices calculated from the points belonging to each cluster (i.e. closest to its center) can be used to generate non-stationarity in the form of automatic relevance determination (ARD) (Wipf and Nagarajan, 2008), where different length-scales are attributed to each kernel, to better represent the structures in that particular area of the input space. For the 3D space, the mean $\boldsymbol{\mu}$ and covariance matrix Σ for each cluster $\mathcal{M}^m = \{\boldsymbol{\mu}, \Sigma\}^m$, represented by the subset N^m of the input point cloud, can be calculated as

$$\boldsymbol{\mu}^m = \{\bar{x}^m, \bar{y}^m, \bar{z}^m\} = \frac{1}{N^m} \sum_{i=0}^{N^m} \mathbf{x}_i^m \tag{10}$$

$$\Sigma^m = \frac{1}{N^m - 1} \sum_{i=0}^{N^m} \begin{bmatrix} (\Delta x_i^m)^2 & \Delta y_i^m \Delta x_i^m & \Delta z_i^m \Delta x_i^m \\ \Delta x_i^m \Delta y_i^m & (\Delta y_i^m)^2 & \Delta z_i^m \Delta y_i^m \\ \Delta x_i^m \Delta z_i^m & \Delta y_i^m \Delta z_i^m & (\Delta z_i^m)^2 \end{bmatrix} \tag{11}$$

where $\Delta \mathbf{x}_i^m = \mathbf{x}_i^m - \boldsymbol{\mu}^m$. Within this methodology, the length-scale of each kernel is tied to the variance of its corresponding cluster, for each input dimension. Larger variances will produce a slower change in occupancy state for that specific dimension, and vice versa, which in turn promotes a better representation of structures for that particular area of the input space. The *localized length-scale* feature vector is obtained by concatenating sparse kernels calculated based on the statistical information of each cluster:

$$\Phi_{LLS}(\mathbf{x}, \mathcal{M}) = \begin{bmatrix} k_{SP}(\mathbf{x}, \mathcal{M}_1) \\ k_{SP}(\mathbf{x}, \mathcal{M}_2) \\ \vdots \\ k_{SP}(\mathbf{x}, \mathcal{M}_M) \end{bmatrix} \tag{12}$$

where k_{SP} is given by Equation (8), with weighted Euclidean distance between points $r = \sqrt{(\mathbf{x}_i - \boldsymbol{\mu}_m)^{\mathrm{T}} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)}$. The search for points within this distance threshold can be done efficiently by maintaining a *kd*-tree (Muja and Lowe, 2009), and only the k nearest neighbors are used to generate the feature vector (the others dimensions are set to zero). Furthermore, this search for nearest neighbors can be extended to an anisotropic space using the work proposed in Pereira and Andreazza (2010), where the covariance matrix of each cluster is also taken into consideration during distance calculations. It has been shown (Guizilini and Ramos, 2017) that this search in the anisotropic space is particularly beneficial in areas with low-density of clusters.

A 2D example of an occupancy map produced by the Hilbert maps framework is shown in Figure 2, where we can see the benefits of introducing localized length-scales to the sparse feature vector formulation. This is due to the use of a stationary kernel to model non-stationary behavior, such as the environment produced by the observed point cloud. Different areas of the input space are shaped differently (e.g. vertical and horizontal walls, sharp corners, etc.), and the kernel used to describe these areas should be able to adapt accordingly.

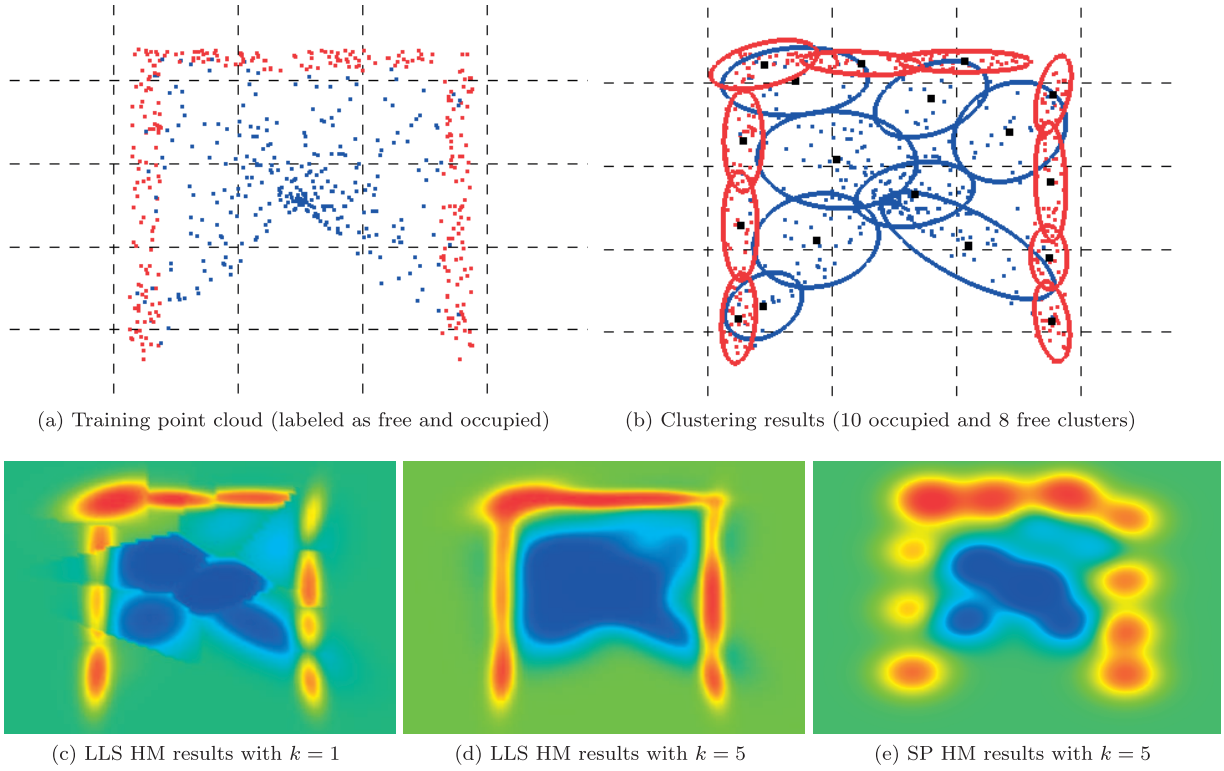
In addition, by introducing more descriptive kernels we can achieve similar reconstruction results with fewer clusters, and by extension fewer RKHS dimensions. Decreasing the number of RKHS dimensions translates into fewer kernel calculations during feature vector generation, thus improving computational speed, and it also produces a smaller memory footprint, since the same dataset can be described using fewer features. Naturally, there is a trade-off between quality and efficiency when the number of RKHS dimensions is considered, however by introducing more descriptive kernels we can improve efficiency without compromising quality.

3.2. Automatic kernel selection

The local length-scale methodology described in the previous section is able to improve occupancy mapping results because it provides a more complex kernel, from which feature vectors that are better equipped to represent observed structures can be constructed. While the standard SP feature vector from Equation (9) is only capable of modeling radially symmetric objects, the local length-scale feature vector from Equation (12) can "stretch" its spatial dimensions to simulate different parameters of thickness, that are unique to each area of the input space. These parameters should be learned in an unsupervised manner, based on information extracted directly from the observed point cloud (i.e. from the statistical information contained in each cluster).

In the work of Guizilini and Ramos (2017) it was shown that, in fact, entirely different kernels can be used for each dimension of a sparse feature vector, and that this diversity further increases the modeling capabilities of the Hilbert maps framework. As a test case, the planar surface kernel (PS) was proposed, that is designed to specifically model structures with one spatial dimension (thickness) much smaller than all the others, acting as the normal vector. Since planar surfaces are found in most environments (e.g. walls, ground, etc.), this new kernel can be readily applied to nearly every occupancy mapping scenario. It is defined as

$$k_{PS}(\mathbf{x}, \mathcal{M}^m) = \begin{cases} 1 & \text{if } d_k < \lambda_k \mid k = \{1, 2, 3\} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$



(a) Training point cloud (labeled as free and occupied)

(b) Clustering results (10 occupied and 8 free clusters)

(c) LLS HM results with $k = 1$

(d) LLS HM results with $k = 5$

(e) SP HM results with $k = 5$

Fig. 2. Two-dimensional example of occupancy mapping using the Hilbert maps framework. The training points in (a) are clustered, and mean and covariance values are extracted from each cluster, as shown in (b). The nearest k neighbors are then selected as inducing points to generate feature vectors during training and inference. In (c) only one nearest neighbor is used, resulting in sudden transitions between one cluster and another that explain the jagged lines. In (d) five nearest neighbors are taken into consideration, which produces a more smooth transition between different areas of the input space. Lastly, (e) shows the effects of removing the use of localized length-scales during the feature vector generation process. Each cluster is now unable to adapt to the distribution of its surrounding points, which results in the same stationary shape being repeated for the entire input space.

where $d = U^m(\mathbf{x} - \boldsymbol{\mu}^m)$, with $U^m = \{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\}$ being the eigenvectors of Σ^m, and λ_k are the corresponding eigenvalues for U^m. This transformation is necessary to align the distance vector in relation to the eigenvectors, so they can be properly compared with the eigenvalues. Essentially, the PS kernel defines as occupied the areas within its aligned eigenvectors, scaled by their corresponding eigenvalues. To better deal with uncertainty boundaries, these eigenvalues can be scaled such that $\lambda'_k = 2\sqrt{\lambda_k}$, so the transition between occupied and unoccupied states takes place within two standard deviations.

Furthermore, due to the repetitive nature of planar surfaces, it is possible that multiple clusters are used to represent the same structure (see Figure 3(b)). Owing to this, to decrease the feature vector dimensions necessary to properly describe the surrounding environment, a second clustering step is proposed, that acts on the clusters themselves. It is performed using a distance-based search method with one threshold for the maximum spatial separation Δ_d on the localized length-scale space and another for the maximum angular deviation Δ_θ between normal vectors, here represented as the eigenvectors $\mathbf{u}'$ corresponding to the smallest eigenvalue of each cluster. These two calculations are performed as follows:

$$d_d(\mathcal{M}_i, \mathcal{M}_j) = \sqrt{(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)\,\Sigma^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^{\mathrm{T}}} \tag{14}$$

$$d_\theta(\mathcal{M}_i, \mathcal{M}_j) = \cos^{-1}(\mathbf{u}'_i \cdot \mathbf{u}'_j) \tag{15}$$

where $\Sigma = (\sqrt{\Sigma_i} + \sqrt{\Sigma_j})^2$ is the weighted length-scale between two clusters. This metric was selected due to its natural way of providing a separation threshold, that is independent on the covariance values themselves. A value of 2, for example, indicates that the 95% certainty boundaries (two standard deviations) are in close proximity. Because it is faster to compute, the angular deviation is used as an initial filter, and the spatial separation on the length-scale space is calculated only if necessary (the square root of covariance matrices can be precomputed for efficiency).

When two clusters are close enough given these two thresholds, their points are merged together to produce a new cluster, with mean and covariance values calculated based on this new subset of $N^i + N^j$ points . The same process is repeated iteratively, until there are no more changes in the number of clusters. The effectiveness of this second clustering step can be seen in Figures 3(b)–(d), where the

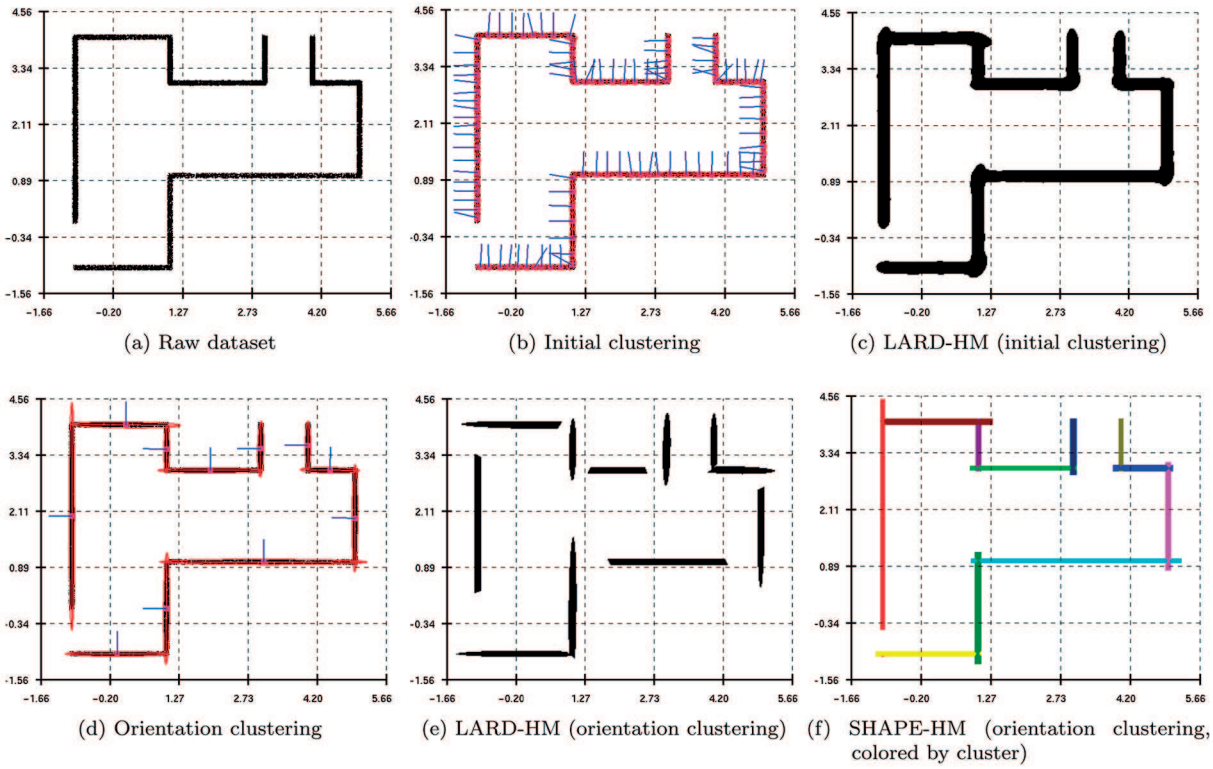


Fig. 3. Two-dimensional example of orientation clustering. Pink dots represent cluster centers, red ellipses indicate covariance matrices within two standard deviations, and blue lines depict normal vectors.

100 original clusters were reduced to only 11 without compromising occupancy mapping results (Figure 3(f)). Note that clusters not classified as planar surfaces can still be modeled using the local length-scale kernel, which produces a more semantically meaningful representation of observed structures, since now they can be categorized with different labels.

3.3. *Efficient feature placement*

Whereas the original work of Ramos and Ott (2015) uses a grid-like structure to place inducing points on the input space, to generate the sparse feature vector, in Guizilini and Ramos (2016) it was proposed the use of clustering techniques. The main benefit of this approach is that it does not produce inducing points in empty areas, thus decreasing the resulting feature vector size. In addition, it can deal with varying density in observations and different levels of detail, by placing more kernels where necessary to produce a more accurate representation of occupancy states.

Despite its simplicity, the k-means algorithm (Lloyd, 1982) still remains the most widely used technique for unsupervised clustering. It takes as input the dataset X to be clustered, and the initial k cluster positions $\mathcal{C}$ (originally obtained by randomly sampling from X), and returns as output the optimized cluster positions $\mathcal{C}'$ that minimize the potential function (Equation (16)). This is done iteratively, by alternating between an *assignment* step, in which each data point is assigned to its closest cluster center; and an *update* step, in which the cluster centers are updated based on this new set of assigned data points,

$$\phi = \sum_{\mathbf{x} \in X} \min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2 \tag{16}$$

Since its introduction, several extensions have been proposed to circumvent some of the limitations present in the original algorithm, such as: (1) initial cluster positioning; (2) scalability to larger datasets; and (3) the number k of clusters. The *k-means++*, for example, was introduced in Arthur and Vassilvitskii (2007) as a way to select initial cluster centers for further optimization, thus avoiding local minima. The *mini-batch k-means* (Sculley, 2010) addresses the scalability issue by subdividing the data points into subsets and performing optimization using stochastic gradient descent, while the *k-means‖* (Kucukyilmaz, 2014) explores parallelism to improve efficiency. The optimal number of clusters can be automatically determined using multiple-hypotheses techniques, such as those introduced in Pelleg and Moore (2000) and Hamerly and Elkan (2004). Recently, the *k-MC*^{2} approximate clustering technique was proposed (Bachem et al., 2016), that completely eliminates dependency in the number of data points and instead has a computational complexity of k^2d, where d is the dimensionality of the input space.

Here we build upon these algorithms and propose a novel technique for clustering initialization that addresses

Algorithm 1 Quick-Means initialization algorithm

Require: point cloud X with N points
radial inner r_i and outer r_o thresholds
minimum number of points per cluster k
distance function $d(.,.)$
Ensure: clusters $\mathcal{C}$

1: $t = N$ % Number of available points
2: $\mathbf{v}_i = \{0, 1, 2, \ldots, N\}$ % Aux. index vector
3: $\mathbf{v}_j = \{0, 1, 2, \ldots, N\}$ % Aux. index vector
4: $\mathbf{v}_b = \mathbf{0}_N$ % Aux. boolean vector of size N
5: $\mathcal{C} \leftarrow \{\}$ % Empty cluster set
6: **while** $t > 0$ **do**
7: $\quad \mathbf{x}_* \leftarrow X[\mathbf{v}_i[randint(0, t)]]$ % Sample available point
8: $\quad \mathcal{M} \leftarrow \mathbf{x} \ : \ d(\mathbf{x}_*, \mathbf{x}) < r_o \ , \forall \mathbf{x} \in X[\mathbf{v}_i[0, 1, \ldots, t]]$
9: $\quad$ **if** $|\mathcal{M}| \geq k$ **then** % If there are enough points
10: $\quad\quad \mathcal{C} \leftarrow \mathcal{M}$ % Add new cluster to set
11: $\quad$ **end if**
12: $\quad$ **for** $\mathbf{m} \in \mathcal{M}$ **do** % Remove points from available set
13: $\quad\quad i \leftarrow$ index of $\mathbf{m}$ in X % Get point index
14: $\quad\quad$ % If available and close enough
15: $\quad\quad$ **if** $\mathbf{v}_b[i] == 0$ **and** $d(\mathbf{m}, \mathbf{x}_*) < r_i$ **then**
16: $\quad\quad\quad \mathbf{v}_b[i] = 1$ % Remove from availability pool
17: $\quad\quad\quad j \leftarrow \mathbf{v}_j[i]$ % Store aux. index j of i
18: $\quad\quad\quad \mathbf{v}_i[j] = \mathbf{v}_i[\text{--}t]$ % Switch aux. index i
19: $\quad\quad\quad \mathbf{v}_j[t] = \mathbf{v}_j[\mathbf{v}_i[j]] = j$ % Replace aux. index j
20: $\quad\quad$ **end if**
21: $\quad$ **end for**
22: **end while**

Algorithm 2 Batch Feature Training algorithm

Require: Hilbert Map $\mathcal{H}$ to be updated
point cloud X and occupancy states $\mathbf{y}$
Ensure: updated Hilbert Map $\mathcal{H}$

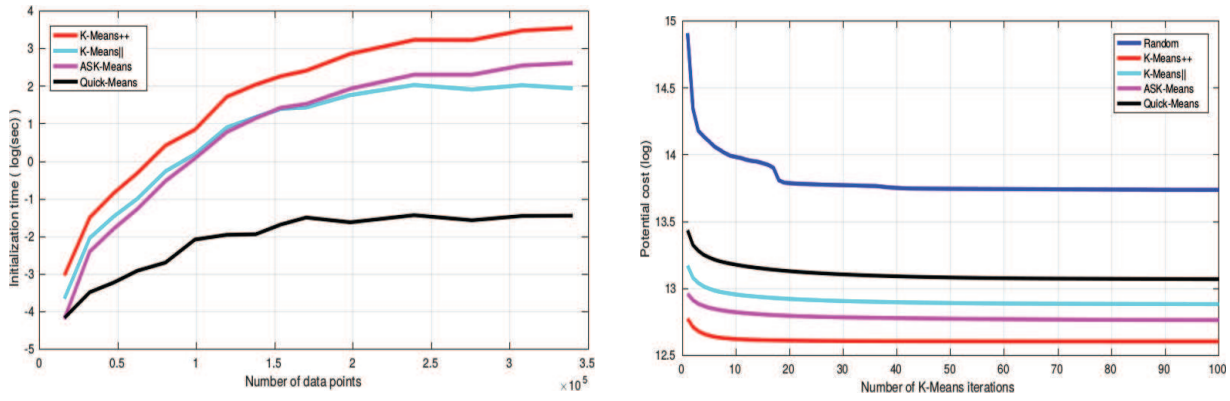
1: $\mathcal{C} \leftarrow quick\_means(X, r_i, r_o, k, d(.,.))$
2: $add = \{\}$ % Indices to add
3: $merge = \{\}$ % Indices to merge
4: **for** $\mathcal{C}_i \in \mathcal{C}$ **do**
5: $\quad$ **for** $\mathcal{M}_j \in \mathcal{M}$ **do**
6: $\quad\quad \boldsymbol{\mu}_{ij} = \boldsymbol{\mu}_i^{\mathcal{C}} - \boldsymbol{\mu}_j^{\mathcal{M}}$ % Calculate relative mean
7: $\quad\quad \Sigma_{ij} = (\sqrt{\Sigma_i^{\mathcal{C}}} + \sqrt{\Sigma_j^{\mathcal{M}}})^2$ % Calculate relative variance
8: $\quad\quad$ **if** $\sqrt{\boldsymbol{\mu}_{ij} \Sigma_{ij}^{-1} \boldsymbol{\mu}_{ij}^{\mathrm{T}}} < 1.0$ **then** % If close enough
9: $\quad\quad\quad merge \leftarrow (i, j)$ % New cluster to merge
10: $\quad\quad$ **else**
11: $\quad\quad\quad add \leftarrow i$ % New cluster to add
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **end for**
15: **for** $i \in add$ **do** % Add new cluster
16: $\quad \mathcal{M} \leftarrow \{\mathcal{C}_i, 0\}$
17: **end for**
18: **for** $(i, j) \in merge$ **do** % Merge two clusters
19: $\quad \boldsymbol{\mu}_j \leftarrow (\boldsymbol{\mu}_i^{\mathcal{C}} + \boldsymbol{\mu}_j^{\mathcal{M}})/2$
20: $\quad \Sigma_j \leftarrow (\Sigma_i^{\mathcal{C}} + \Sigma_j^{\mathcal{M}})/2$
21: **end for**
22: **for** $\mathcal{C}_i \in \mathcal{C}$ **do** % Update weights
23: $\quad \mathbf{k} \leftarrow nearest\_neighbors(\mathcal{C}_i, \mathcal{M})$
24: $\quad$ **for** $(\mathbf{x}, y) \in \mathcal{C}_i$ **do**
25: $\quad\quad \Phi_{LSS}(\mathbf{x}) = \{k_{SP}(\mathbf{x}, \mathcal{M}_k)\}_{k \in \mathbf{k}}$ % New feature vector
26: $\quad\quad \mathbf{w} \text{ -= } \eta \frac{\partial}{\partial \mathbf{w}} RNLL(\mathbf{w}, \Phi_{LSS}(\mathbf{x}), y)$ % Stochastic step
27: $\quad$ **end for**
28: **end for**

all the above mentioned limitations, called *quick-means*. The key insight is that, for the particular case of producing inducing points for feature vector generation, we are not concerned with optimal cluster placement, but rather with regularly spaced clusters. If two clusters are too far apart, the observed structures might become too complex for each kernel to correctly model, and if they are too close it might create unnecessary computational costs (in addition, each cluster might not have enough information to produce a statistically meaningful kernel). The quick-means algorithms exploits this trade-off in distance between clusters to automatically select the optimal k for any given dataset, while scaling to very large datasets because it does not require explicit distance calculations between points and cluster centers produce a probability distribution for sampling.

Pseudo-code for the proposed clustering initialization technique can be found in Algorithm 1. It receives as input the point cloud X to be clustered, two distance threshold radii r_i and r_o, a threshold cluster size k and a distance function between any two points in the input space $d(.,.)$. The use of two threshold radii, inner and outer, is important to allow overlapping between clusters, as a way to increase feature complexity and model structures from different perspectives. In lines 2–4 three auxiliary vectors are produced: v_i, to store the indexes of available points; v_j, to store the cluster number for each point; and v_b, to determine which points should be searched for new cluster centers.

Afterwards, starting from an empty cluster set $\mathcal{C}$, we iterate while there are still available points to be considered. Initially, a random sample $\mathbf{x}_*$ is obtained from the list of available points (line 7), and a new cluster $\mathcal{M}$ is produced from its neighbors within r_o. If this new cluster has at least k points, it is incorporated into $\mathcal{C}$ (lines 9 and 10). Then, the distance from each point in $\mathcal{M}$ to $\mathbf{x}_*$ is calculated, and if it is smaller than r_i this point can no longer be considered either as a seed or part of a new cluster. This can be efficiently done by updating the indexes stored in v_i (line 16), so sampling is still performed by randomly selecting an integer from 0 to the current number of available points t. When $t = 0$, the algorithm ends and returns the full cluster set $\mathcal{C}$, that can then be further optimized by the standard *k-means* algorithm.

Figure 4 shows comparisons between the proposed quick-means algorithm and other state-of-the-art clustering initialization techniques, both in terms of (a) computational speed during the initialization process and (b) potential function during *k-means* optimization, for each iteration. As we can see, quick-means is able to calculate initial clusters orders of magnitude faster than other techniques, without significantly compromising accuracy. In fact, empirical



(a) Initialization time (random values are negligible).

(b) Potential cost (Equation (16)) during k-means optimization.

Fig. 4. Comparison between different clustering initialization techniques: random, *k-means++*, *k-means‖*, ASK-Means (Guizilini and Ramos, 2017), and the proposed quick-means algorithm (average of 50 runs with different random seeds).

Table 1. Average processing times over 10 runs (milliseconds) for the LLS-HM framework, on the *Virtual* dataset.

Task	Total time	Individual time
Clustering	471 ± 8	5.23×10^{-3} / point
Calc. length-scale	23 ± 4	9.21×10^{-3} / cluster
Calc. train features	116 ± 12	1.29×10^{-3} / feature
SGD training	5 ± 1	5.56×10^{-5} / feature
Calc. grid features	873 ± 121	1.37×10^{-3} / feature
Grid inference	17 ± 4	2.66×10^{-5} / feature

evidence shows that, for occupancy mapping applications within the proposed Hilbert maps framework, the output of quick-means can be used directly without noticeable degradation in results, which further improves the algorithm's applicability to real-time tasks.

3.4. Hierarchical batch feature calculation

As shown in Guizilini and Ramos (2016), and further explored in the experiments conducted in Section 4 (see Table 1), the feature calculation process also consumes a significant portion of the total computational time, in addition to the clustering process addressed previously. This is due to the necessity of obtaining the k nearest neighbors of each input point, to determine which inducing points (and, by extension, which dimensions) are relevant when producing the corresponding feature vector. Although this process can be sped up significantly by using efficient data structures such as *kd*-trees (Muja and Lowe, 2009), in relation to the naive approach of calculating kernels for all dimensions, the sheer number of new training and querying points produced at each iteration, combined with the introduction of search in the anisotropic space (Pereira and Andreazza, 2010), still pose a challenge for real-time applications of the Hilbert maps framework.

Owing to this, here we propose a novel methodology that uses batches of input training and query data, rather than individual points, and performs nearest neighbors calculations directly on these batches. In addition, we employ a hierarchical clustering (HC) process that is scalable to large-scale datasets and naturally allows the incorporation of new information, by merging overlapping clusters and creating new ones if necessary. The basis for this novel methodology is the natural assumption that points belonging to the same cluster will have similar nearest neighbors, and therefore can share this information without the need for redundant calculations. In addition, by increasing the number of nearest neighbors used to produce the local length-scale feature vector (see Equation (12)) we can further minimize the impact of individual variations between each point in a cluster. This new feature calculation process can then either be applied during training, to efficiently optimize the parameters of the internal classifier using stochastic gradient descent on batches of available data, rather than individual points, or during inference, to quickly produce occupancy estimates for unobserved areas of the input space.

Pseudo-code for the proposed batch feature calculation technique, applied for the training step, can be found in Algorithm 2. It receives as input a Hilbert map $\mathcal{H}$ to be updated and a point cloud X with respective occupancy states $\mathbf{y}$. Initially, this point cloud is clustered using the quick-means technique (see Algorithm 1) to produce the set $\mathcal{C} = \{\boldsymbol{\mu}, \Sigma\}_{i=1}^{N}$ (line 1). These clusters are then compared in proximity with the current set $\mathcal{M} = \{\boldsymbol{\mu}, \Sigma\}_{i=1}^{M}$ from $\mathcal{H}$, using the anisotropic distance metric introduced in Equation (14). Here we use a threshold of one standard deviation in said anisotropic space to determine whether a cluster has been re-observed, as shown in line 8.

If there are no clusters within this vicinity, $\mathcal{C}_i$ is added to $\mathcal{M}$ (line 16), with an initial weight parameter of 0 for the logistic regression classifier, indicating no prior knowledge of its occupancy state. Otherwise, the statistical information of $\mathcal{C}_i$ is merged to its nearest neighbor (lines 19 and 20), producing an updated cluster. Figure 5 depicts this process of cluster addition and merging, as new point clouds

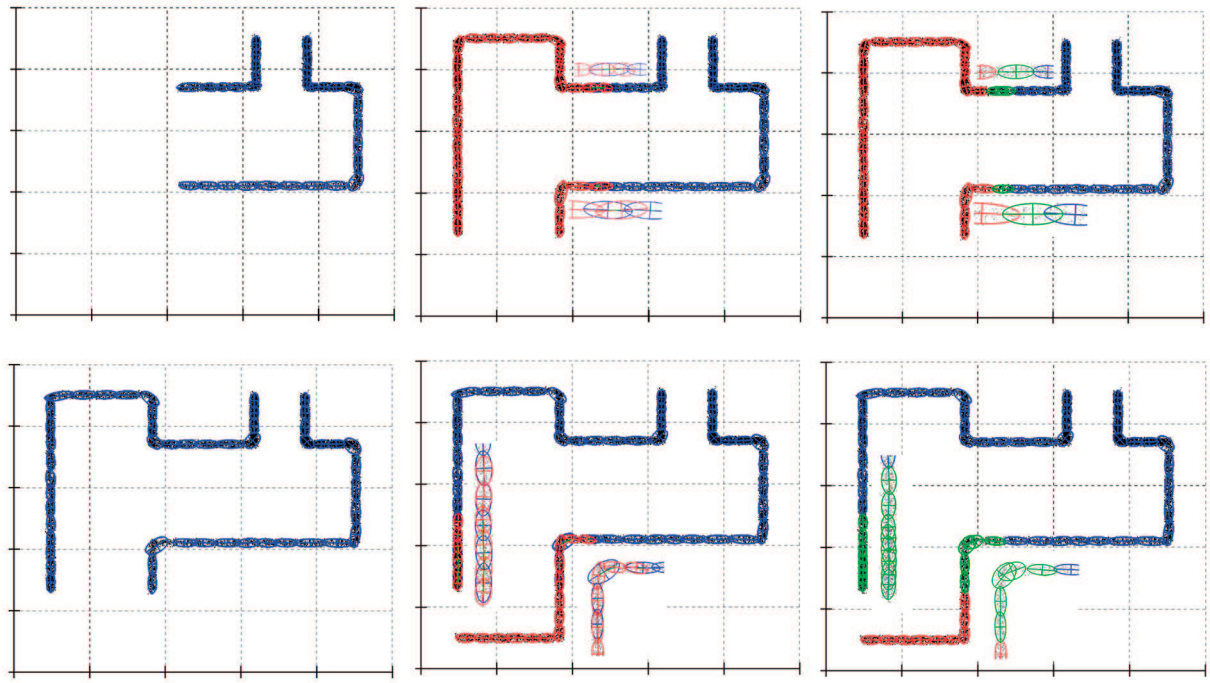


Fig. 5. Two-dimensional example of point cloud incorporation into the occupancy mapping model. In each row, we start with the current clustered point cloud (blue), and introduce a second clustered point cloud (red), that represents new information to be incorporated. Overlapping clusters are merged (green) to produce a better representation of boundaries between point clouds, and other ones are maintained and used during the feature generation process.

are collected. Note that this merging process reinforces the current state of the observed cluster, producing better-defined occupancy estimates, since new information is used to update its weight parameter.

Finally, the feature vector is calculated for each point in X (line 25), using batches of nearest neighbors calculated for each cluster (line 23). Each feature vector $\Phi_{LSS}(\mathbf{x})$, alongside its corresponding occupancy state y, is then used to update the weight parameters of the logistic regression classifier via stochastic gradient descent (line 26), so the trained model better reflects the observed structures in the surrounding environment. A similar process can be applied to the inference step, by clustering the query points and calculating each feature vector according to its cluster's nearest neighbors, before estimating the occupancy state according to Equation (1). Pseudo-code for the proposed batch feature calculation technique, applied to the inference step, can be found in Algorithm 3. In essence, it replicates lines 1 (clustering) and 22–28 of Algorithm 2 (nearest neighbor search and feature vector calculation), but instead of optimizing the weight parameters based on Equation (3) it estimates occupancy states based on Equation (1).

However, as the explored environment grows, so does the number of clusters necessary to accurately represent observed structures, which in turn increases the computational cost of nearest neighbors search. The use of *kd*-trees and the proposed batch feature calculation technique significantly decreases the computational time required for this step, but this is still an unbounded increase that will eventually hinder real-time applications of the Hilbert maps framework. One way to address this issue is by employing local cluster maps, that can be queried independently from each other, and therefore have a computational cost that does not increase as more clusters are globally added.

This technique is known as HC (Rajalingam and Ranjini, 2011), and can be broadly divided into two categories: *agglomerative*, where each observation starts in its own cluster and is merged as it moves up the hierarchy; and *divisive*, where all observations start in one cluster and are split as they move down the hierarchy. In the general case, agglomerative clustering has complexity $\mathcal{O}(n^2 \log(n))$ and divisive clustering with exhaustive search has complexity

Algorithm 3 Batch Feature Inference algorithm

Require: Hilbert Map $\mathcal{H}$ and inference points X
Ensure: occupancy probability vector $\mathbf{p}$
1: $\mathcal{C} \leftarrow quick\_means(X, r_i, r_o, k, d(.,.))$
2: **for** $\mathcal{C}_i \in \mathcal{C}$ **do** % For each cluster
3: $\quad \mathbf{k} \leftarrow nearest\_neighbors(\mathcal{C}_i, \mathcal{M})$
4: $\quad$ **for** $(\mathbf{x}, y) \in \mathcal{C}_i$ **do** % For each point
5: $\quad\quad \Phi_{LSS}(\mathbf{x}) = \{k_{SP}(\mathbf{x}, \mathcal{M}_k)\}_{k \in \mathbf{k}}$ % Feature vector
6: $\quad\quad p(\mathbf{x}) \leftarrow \frac{1}{1 + \exp(\mathbf{w}^{\mathrm{T}} \Phi_{LSS}(\mathbf{x}))}$ % Occupancy state
7: $\quad$ **end for**
8: **end for**

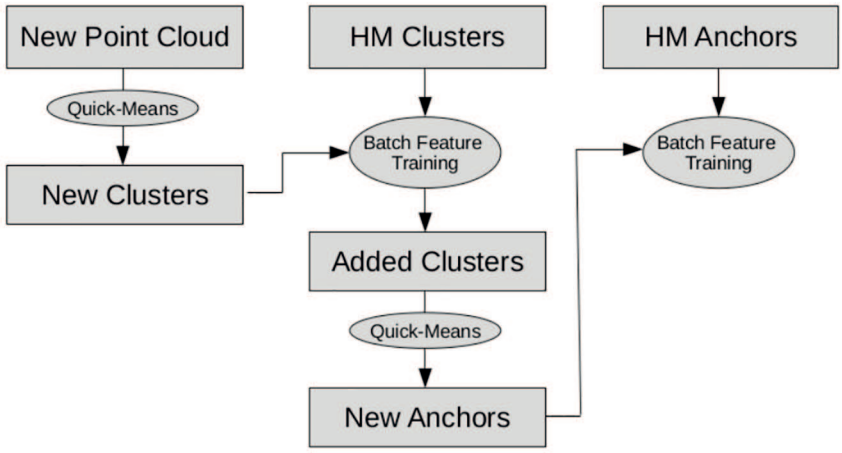


Fig. 6. Diagram of the proposed multi-layered hierarchy clustering technique.

$\mathcal{O}(2^n)$, that can be brought down to $\mathcal{O}(n^2)$ for certain special cases, such as the single and complete linkage (Defays, 1977).

For the particular application of occupancy mapping from an incremental introduction of point clouds, there are structural patterns that can be used to significantly decrease this computational complexity. For example, we can assume a hierarchy on the point clouds themselves, in which each one contributes to a single cluster, that is then subdivided to produce the local clusters used to generate feature vectors. This approach is particularly attractive if each point cloud is spatially bounded, i.e. it has a limited range, such as a laser scanner. Other sensors, such as cameras, are unbounded, meaning that they can produce distance estimates at any range depending on how structures in the environment are observed. To address these situations, here we propose a multi-layered hybrid approach, composed of two stages:

1. **divisive**, in which a point clouds starts as one single structure that is subdivided into multiple local clusters (see Section 3.3);
2. **agglomerative**, in which local clusters are merged to produce super-clusters (here referred to as *anchors*), representing different areas of the input space.

A diagram of the proposed multi-layered hierarchy clustering technique can be found in Figure 6. It takes as input a new point cloud, that is clustered and incorporated into the current Hilbert maps model according to the batch feature training methodology introduced in Algorithm 2. Afterwards, clusters added to the current model are further clustered to produce a set of new anchors (using larger inner r_i and outer r_o radii thresholds). This set is also incorporated into the Hilbert maps framework, to produce a second layer of inducing points that can be used to produce feature vectors, albeit in a lower cluster resolution due to the difference in quick-means parameters.

This process can be repeated as many times as necessary, introducing new layers composed of increasingly fewer clusters that represent larger portions of the environment. The top layer can either be directly used to estimate occupancy states, at a lower cluster resolution, or iteratively searched down for nearest neighbors in previous layers, which are then used to produce occupancy estimates. In each layer, the k nearest neighbors are calculated amongst their anchors, and then for each one the k nearest neighbors are calculated amongst its anchors, and so forth. Although this iterative search creates extra computational cost, experimental tests show that, for datasets that span a sufficiently large volumetric space, it can lead to significant gains in performance.

3.5. Incremental grid reconstruction

As is the case with most generalized interpolative models in statistics, by performing calculations in the RKHS, based on feature vectors that approximate kernel functions, the Hilbert Maps framework is capable of producing occupancy estimates for any point in the input space, at arbitrary resolutions. This is particularly useful because it does not require the generation of a grid to store occupancy estimates, which can be prohibitive for very large datasets that cover a significant volumetric space, especially at finer resolutions. However, in some situations an occupancy grid is necessary, such as scene reconstruction using polygonal meshes obtained from a discrete scalar field (Lorensen and Cline, 1987). This is an intuitive way of presenting results from an occupancy model, that is both understandable by humans and easily exploited by computers, for tasks such as path planning and obstacle avoidance.

Owing to this, here we propose a novel technique for the incremental generation of occupancy grids that significantly outperforms the naive approach of individually estimating the occupancy state of each point in the grid, even when using the batch feature inference methodology described previously (Algorithm 3). We assume a fixed-resolution equally sized grid, and maintain local grids representing the

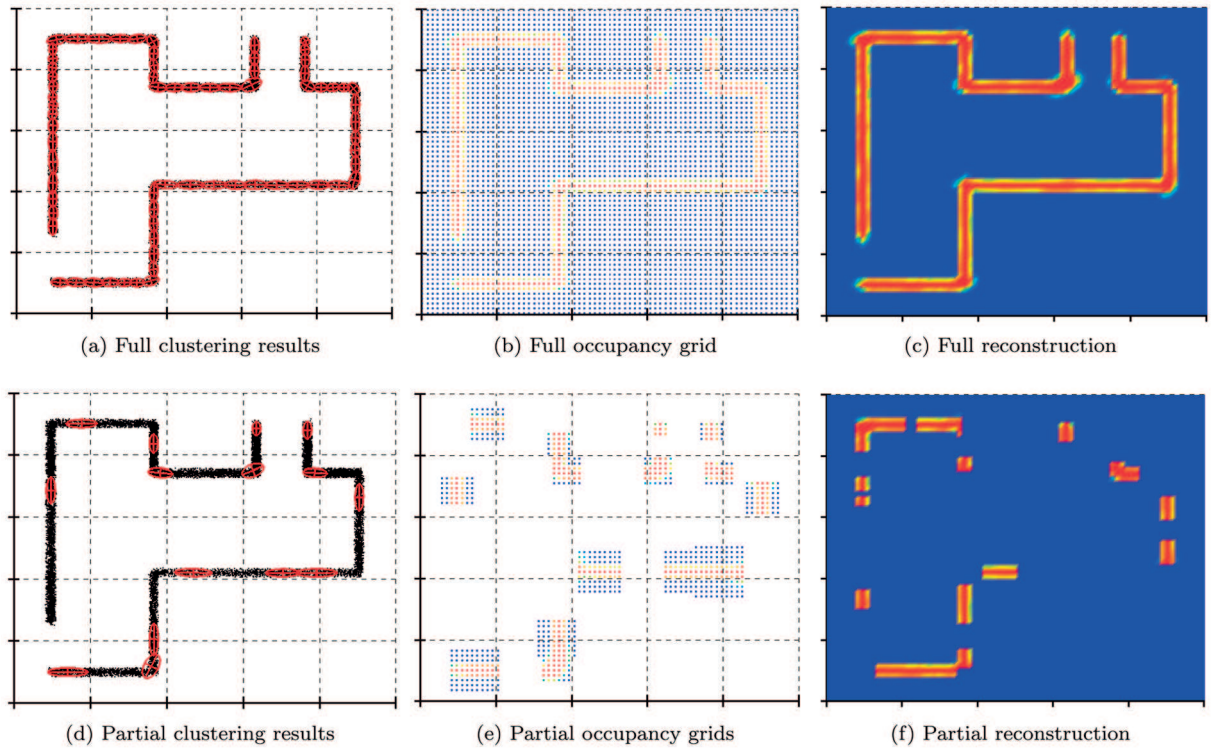


Fig. 7. Two-dimensional example of the proposed incremental grid reconstruction technique. In the top row all clusters are taken into consideration (a), each one producing its own local grid that is combined to produce the global grid (b), and in (c) the resulting discrete mesh surface is depicted. In the bottom row, only a few clusters are considered (d), so the contribution of each local grid can be better viewed (e), and the resulting sparse reconstruction can be found in (f).

contribution of each cluster to the calculation of occupancy estimates in different portions of the input space. These local grids are updated every time a corresponding weight parameter is modified during the training process, resulting in an incremental technique that is always up to date with the current Hilbert maps model, and can be efficiently combined to produce a global grid representation.

We define the dimension of each local grid according to the eigenvalues λ_d of its corresponding cluster, projected to the global coordinate system in terms of both translation and orientation. Similarly to Section 3.2 these eigenvalues are scaled so that $\lambda'_d = 3\sqrt{\lambda_d}$, so that distances of up to 3 standard deviations are still considered relevant for calculations. A local grid G_m stores kernel evaluations $k_{SP}(\mathbf{x}_g, \mathcal{M}_m)$ for that particular inducing point, in relation to each of its spatial coordinates $\mathbf{x}_g \in G_m$. The occupancy state of a spatial coordinate on the global grid is obtained by first combining these kernel evaluations to produce the feature vector $\Phi(\mathbf{x}_g)$ and then evaluate Equation (1). An example of the proposed technique for a 2D dataset can be found in Figure 7, first for a complete reconstruction of the observed environment (top row) and afterwards for a partial reconstruction, where only some local grids are used (bottom row), for a better understanding of the underlying process.

Combining kernel evaluations can be done efficiently by maintaining indexes for each local grid, so they can be quickly positioned within the global grid. Assuming starting coordinates of $(0, 0)$ for the global grid, in the 2D space, and starting coordinates of (i, j) for any given local grid, the coordinates (u, v) for a point in the local grid will be $(u + i, v + j)$ in the global grid (note that local grids are already constructed in alignment with global coordinates, so their points overlap spatially). A quick inspection of Equation (1) shows that, for the logistic regression classifier, the row-weight vector $\mathbf{w}$ is multiplied by the column-feature vector $\Phi(.)$ to produce a single scalar $s = \mathbf{w}^{\mathrm{T}}\Phi(\mathbf{x})$. This scalar can be iteratively calculated by multiplying each kernel evaluation with its corresponding cluster weight parameter as they are added to the global grid:

$$s_g = \sum_m w_m \cdot k(\mathbf{x}_g, \mathcal{M}_m) \tag{17}$$

If a local grid does not include a certain point in the global grid, the kernel evaluation is implicitly assumed to be zero and does not contribute to the estimation of that particular occupancy state. Therefore, this process eliminates the need of nearest neighbors search, with the dimensions of each local grid now dictating which kernels should be taken into consideration when producing feature vectors. Once the values of s_g are obtained for all points of the global grid, the corresponding occupancy state estimates can be

directly calculated by applying the logistic function:

$$p(y_g = 1|\Phi(\mathbf{x}_g), \mathbf{w}) = \frac{1}{1 + \exp(s_g)} \tag{18}$$

Furthermore, within this incremental methodology each dimension of a feature vector can be updated independently, as the corresponding cluster parameters (i.e. weight values and length-scales) are modified with the addition of new information. The introduction of new clusters can also be done efficiently by producing a corresponding local grid, that contains all coordinates in which that new cluster is relevant for the production of feature vectors. Different grid resolutions can be maintained in parallel, sharing the same Hilbert maps model, and queried independently to produce varying levels of details according to the application at hand.

4. Experimental results

In this section we perform extensive experimental tests with the proposed EHM framework, to validate its accuracy and performance under different circumstances in comparison with other state-of-the-art 3D occupancy mapping techniques. Three different datasets were considered, one representing a small-scale simulated indoors environment, another composed of a single 3D laser scan of an outdoors environment, and a third large-scale dataset composed of a series of 3D laser scans obtained in different overlapping areas of an urban environment. Note that each model is generated from scratch by incrementally incorporating new pointclouds as they become available, there is no prior training using other observed scenes. Unless noted otherwise (i.e. for tests with varying sparsity levels), all available points are used in the optimization process, to update the current model's parameters and ensure a better occupancy classification performance.

The first dataset, entitled *Virtual*, is composed of around 90,000 points (Figure 8(a)) and was obtained in a simulated environment of an empty room. Although this is a simple dataset from a modeling perspective, it serves to validate the proposed algorithm in comparison with the various implementations of the Hilbert maps framework found in current literature, and depicts some of the limitations present in other occupancy mapping techniques. In particular, results obtained using OctoMaps (Hornung et al., 2013) in this dataset can be found in Figure 8(c), where we see the various occupied voxels colored by height. In Figure 8(b) the original point cloud was clustered with an average cluster distance of 0.05 m to produce around 2,000 clusters, that were used in three different versions of the Hilbert maps framework to produce an occupancy map of the observed environment.

In Figure 8(d) the original Hilbert maps framework found in Ramos and Ott (2015) was used, in which sparse kernels are used with a single length-scale, determined by the average cluster distance. It is clear that, within this resolution, such sparse kernel is incapable of correctly modeling the environment, since it is limited to a single shape and cannot adapt to different structures. In particular, it is unable to extrapolate the information found in observed areas to unknown portions of the input space. In Figure 8(e) the LLS-HM framework found in Guizilini and Ramos (2016) is used, in which each cluster contain its own length-scale value, calculated based on the statistical information its own subset of points. We see that this framework produces a much smoother occupancy map, in which random gaps in the observed point cloud are correctly labeled as occupied in the resulting continuous mapping function.

Finally, Figure 8(f) depicts the results of the proposed EHM framework in the *Virtual* dataset, where we can qualitatively see that they are virtually identical to the LLS-HM results. This is to be expected, since the proposed framework does not improve upon the accuracy of LLS-HM, but rather on its processing speed and computational efficiency, as shown in Tables 1 and 2. They break down the computational time of different tasks performed by the Hilbert Maps framework, the first one for the LLS-HM implementation and the second one for the proposed EHM implementation. These numbers show that the extensions introduced in this paper indeed promote a substantial increase in performance, particularly for the following tasks: clustering (Section 3.3), of around 92%; feature calculation (Section 3.4), of around 86% and grid inference (Section 3.5), of around 58%.

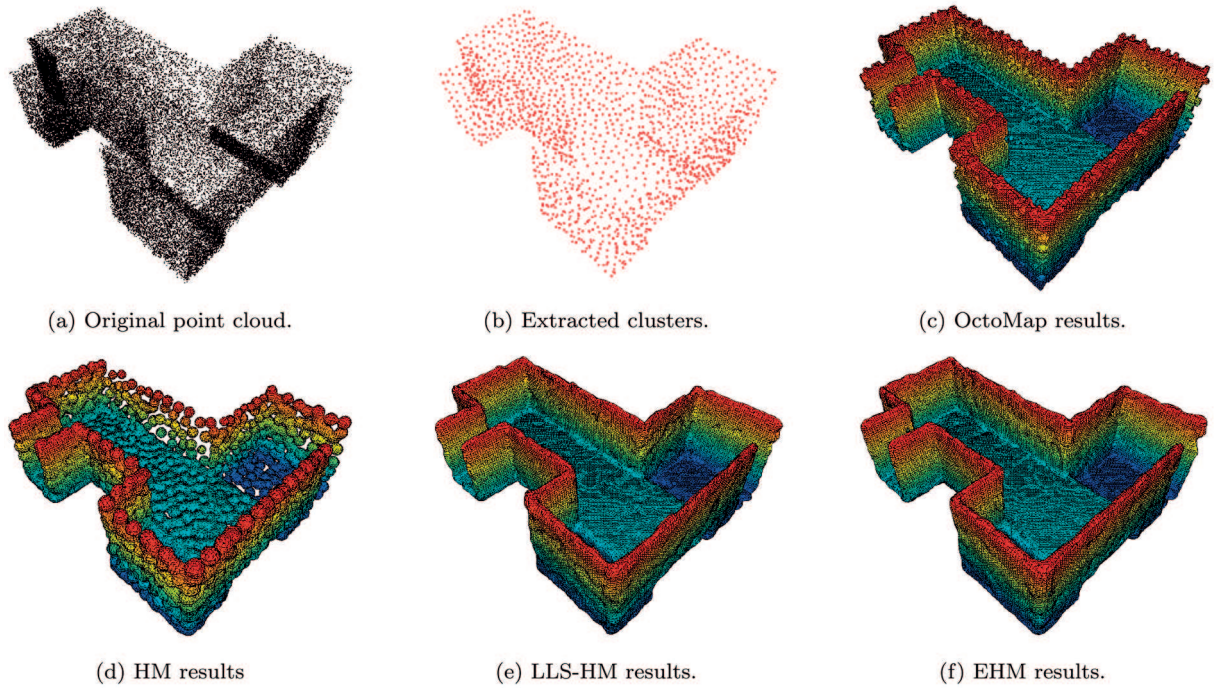
Given these results, we calculate that the average processing time for a point cloud of size $\sim$ 90,000 is $37+22+17+3 = 79$ ms, which allows operation at around 12 Hz. Inference on the current model would increase this processing time, however the number of queries per iteration in real applications is usually much smaller and does not require

Table 2. Average processing times over 10 runs (milliseconds) for the EHM framework, on the *Virtual* dataset.

Task	Total time	Individual time
Clustering	37 ± 4	4.11×10^{-4} / point
Calc. length-scale	22 ± 3	8.79×10^{-3} / cluster
Calc. train features	17 ± 4	1.89×10^{-4} / feature
SGD training	3 ± 1	3.33×10^{-5} / feature
Calc. grid features	113 ± 19	1.76×10^{-4} / feature
Grid inference	7 ± 2	1.12×10^{-5} / feature

Table 3. Average processing times over 10 runs (milliseconds) for different methods of 3D occupancy mapping, on the *Virtual* dataset.

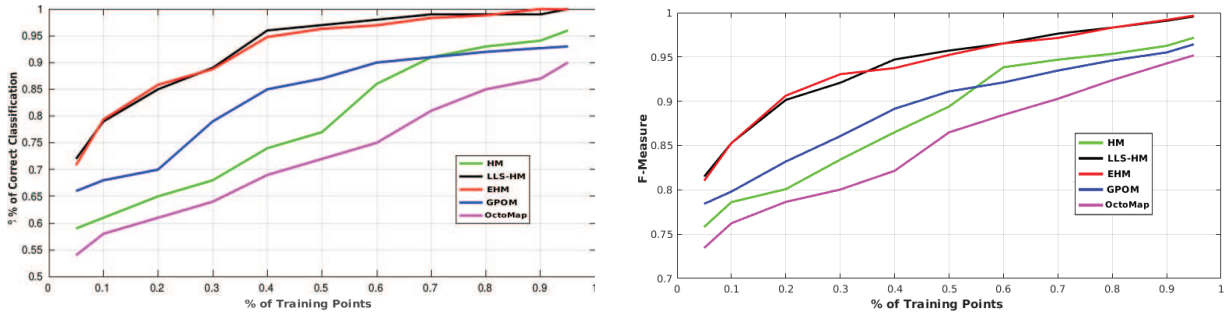
Method	Training time	Query time
GPOM	7127 ± 529	$73{,}638 \pm 4{,}154$
OctoMap	107 ± 22	22 ± 5
HM	$3{,}722 \pm 237$	$23{,}714 \pm 3{,}424$
LLS-HM	615 ± 41	890 ± 55
EHM	79 ± 9	120 ± 16



(a) Original point cloud. (b) Extracted clusters. (c) OctoMap results.

(d) HM results (e) LLS-HM results. (f) EHM results.

Fig. 8. Three-dimensional occupancy mapping results on the virtual dataset using different techniques (colored by height).



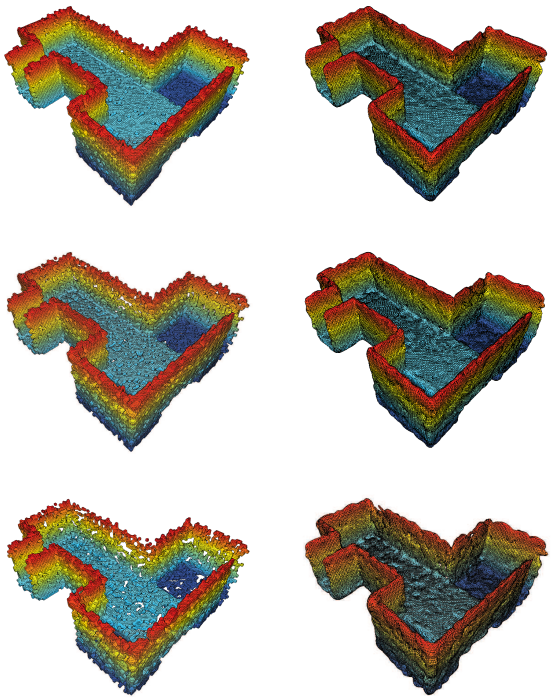
(a) Percentage of correct classifications (0.5 threshold) (b) F_1-measure score

Fig. 9. Occupancy mapping classification results for different techniques, with varying sparsity levels.

the reconstruction of the entire observed environment, only of small portions that are relevant for tasks such as obstacle avoidance and trajectory following.[1] Average processing times for different 3D occupancy mapping techniques can be found in Table 3, including OctoMaps and GPOMs (as introduced in Callaghan and Ramos (2012)). As we can see, the GP framework already struggles to process this small-scale dataset in a timely manner, which is reflected in its slow training and query times.

The original Hilbert maps framework, that places inducing points in an equally spaced grid covering the entire observed environment, has the second slowest training and query times, due mostly to the sparse feature calculation process, that requires kernel calculations for each inducing point. The introduction of automatic kernel placement in the LLS-HM framework, and efficient search for nearest neighbors using *kd*-trees, produces a significant increase in computational speed, that is further enhanced in the EHM framework proposed in this paper. In fact, the EHM framework rivals OctoMaps in terms of training speed and is on average faster, by a margin of around 24%.

In addition, EHM produces an occupancy model that is much more robust to the presence of sparse data, here simulated by randomly removing a percentage of points from the original point cloud and using only the remaining ones to produce the occupancy model. The removed points served to validate the final trained model, and results for different occupancy mapping techniques are depicted in Figure 9, both in terms of the percentage of correctly classified points (with a threshold of 0.5) and F_1-measure scores. As expected, the LLS-HM and EHM frameworks have very similar results, and the original HM framework degrades quickly as sparsity increases, since the sparse kernel without localized length-scales is not able to model different shapes in thee environment (see Figure 8(d)). At higher sparsity levels GPOM is able to produce better results than



(a) OctoMaps (b) EHM

Fig. 10. Effects of increasing sparsity in occupancy mapping results, for OctoMaps and EHM. From top to bottom, sparsity has been respectively set to 80%, 60%, and 40%.

Hilbert maps, and OctoMaps remains consistently in last place, with a roughly linear decrease as sparsity increases. Figure 10 provides a visual comparison of the effects of an increasingly sparse training dataset in the final occupancy mapping results, for the OctoMaps and EHM frameworks.

The second dataset considered in this paper, entitled *LiDAR*, is composed of roughly 550,000 points covering an area of $50 \times 50 \times 15$ m^3, obtained using a RIEGL LiDAR laser sensor. This sensor was used to produce a snapshot of an outdoors environment, including the 3D position of nearby structures and corresponding returned intensity values. Occupancy mapping results obtained from this dataset, based on OctoMaps and EHM, are depicted in Figure 11, including an overview of the scene and different zoomed-in areas. Interestingly, the point cloud becomes sparser as it moves away from the center, due to a decrease in sensor resolution, which is apparent in the OctoMaps occupancy model. The EHM framework, on the other hand, is able to smooth out this information and correctly estimate the occupancy state of sparser areas, thus producing a more consistent representation of the observed environment.

For an average cluster distance of 0.05 m, roughly 12,000 clusters were produced to model the *LiDAR* dataset point cloud, and the reconstruction grid contained around 344 million points. The corresponding training and query times for both frameworks are presented in Table 4, alongside F_1-measure values (Rijsbergen, 1979) for different levels of sparsity.[2]. From this table we see that, for a point cloud of size $\sim$ 550,000, OctoMaps outperforms EHM in terms of training and querying times, however the proposed framework still achieves higher F_1-measure values for all sparsity levels considered. Note that these F_1-measure values do not take into consideration the naturally sparse areas of the input point cloud, which we can visually see in Figure 11 that the EHM framework is able to consistently reconstruct the original shape of partially observed objects.

Table 4. Average processing times over 10 runs (milliseconds) and F_1-measure values for different percentages of training points, for the *LiDAR* dataset using the OctoMaps and EHM frameworks.

	OctoMaps	EHM
Training time	345 ± 18	526 ± 48
Query time	279 ± 23	$4{,}121 \pm 508$
F_1-measure (80%)	0.783	0.933 ± 0.026
F_1-measure (60%)	0.716	0.872 ± 0.017
F_1-measure (40%)	0.654	0.793 ± 0.032

Table 5. Average processing times over 10 runs (milliseconds) and F_1-measure values for different percentages of training points, for the *Freiburg* dataset using the single-batch (SB) and incremental (INC) EHM frameworks.

	SB-EHM	INC-EHM
Training time	$3{,}452 \pm 104$	$2{,}822 \pm 91$
Query time	$7{,}228 \pm 201$	$7{,}839 \pm 274$
F_1-measure (80%)	0.913 ± 0.025	0.921 ± 0.018
F_1-measure (60%)	0.859 ± 0.019	0.848 ± 0.027
F_1-measure (40%)	0.742 ± 0.013	0.753 ± 0.008

Finally, the third dataset considered is the *Freiburg* dataset, freely available to download at http://ais.informatik.uni-freiburg.de/projects/datasets/ fr360/. It is composed of 77 scans obtained using a SICK LMS laser range scanner mounted on a pan–tilt unit. Roughly 25 s were necessary to capture each point cloud, with an average distance of about 10 m between scans and 150,000–200,000 points per scan. Figure 12 depicts the incremental occupancy maps produced using the EHM framework for the first 30 scans, both incrementally (each scan is added sequentially) and in a single batch (all scans are trained simultaneously).

As expected, the results are visually nearly identical, and Table 5 provides numerical comparisons that corroborate this similarity for both approaches. Interestingly, the incremental EHM framework is able to produce the final occupancy model significantly faster ($\sim$ 22%), since it performs the training steps only in subsets of the full dataset, thus eliminating the need of processing all available information simultaneously. On the other hand, inference speed (assuming an average cluster distance of 0.10 m, which produces around 112,000 clusters in total) is higher for the single-batch EHM framework, since each kernel only has to be calculated for the final grid reconstruction, rather than being continuously updated as more information is incorporated.

Given these average processing times, new scans can be incorporated into the EHM occupancy model at around 10

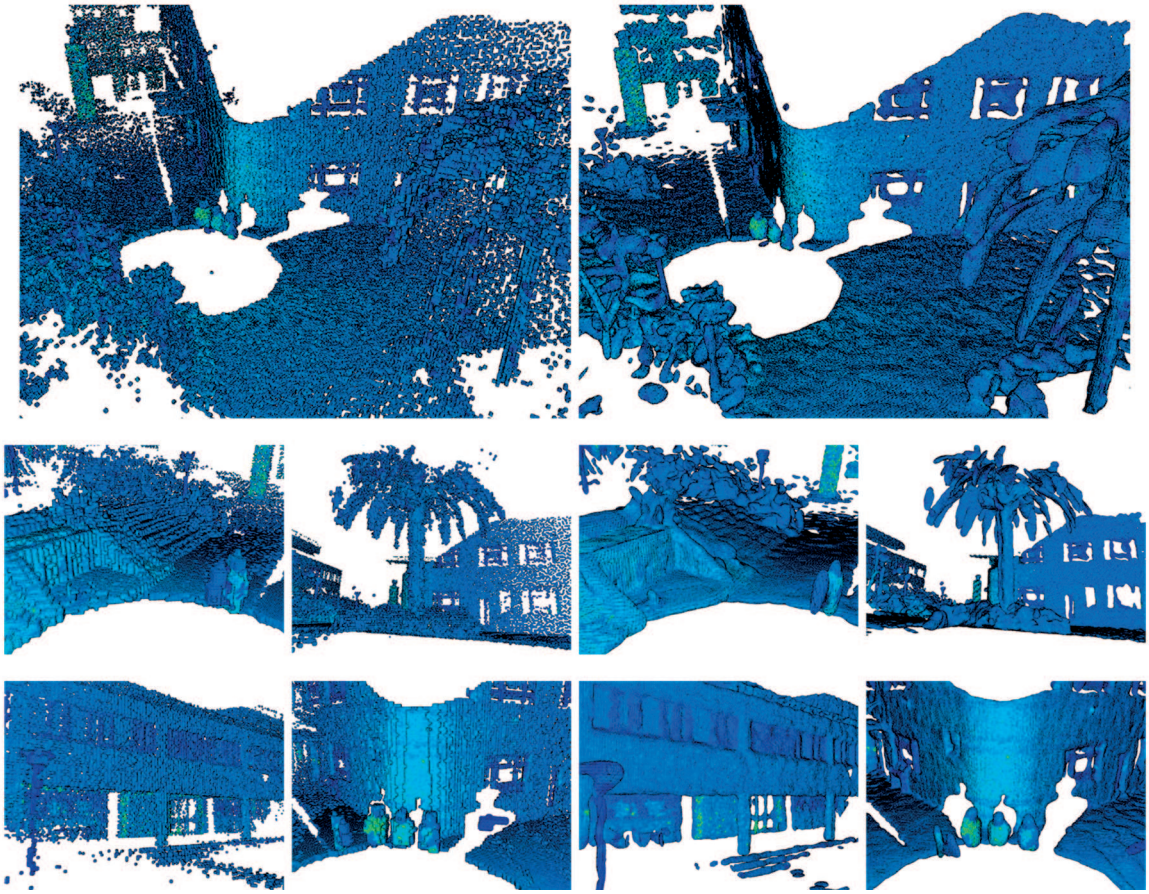


Fig. 11. Three-dimensional occupancy mapping results on the *LiDAR* dataset, using OctoMaps (left) and EHM (right), colored by returned laser intensity values.

Hz. Furthermore, empirical tests show that the processing time of incorporating a new scan is roughly constant, indicating that the extra costs introduced by the use of hierarchical batch features (Section 3.4) and incremental grid reconstruction (Section 3.5), as the occupancy model increases in complexity, are negligible in relation to the core components of the Hilbert maps framework (i.e. clustering and feature vector calculation). In fact, the introduction of a two-layer hierarchy clustering technique (see Figure 6) produced a speed gain of around 10% for batch feature calculation times, in relation to a single-layer composed of extracted clusters from all point clouds. Our hypothesis is that these gains in speed become more apparent as the observed environment occupies a larger volumetric space, since clusters from different areas can be separated more accurately, thus improving search times for nearest neighbors.

Another attractive property of the proposed framework is its smaller memory footprint, since new points do not have to be stored after being used to update the occupancy model, only the generated cluster set (and even so, only clusters that are considered new and were not merged with those already stored). On the three datasets considered here, the number of generated clusters was on average 0.018% the total number of input points, which translates into a decrease of roughly two orders of magnitude. As a trade-off between quality and performance, using larger average cluster distance values would produce even fewer clusters, with an inverse cubic ratio (i.e. doubling the average cluster distance would decrease the number of clusters by a factor of eight). In addition, the use of more descriptive and specialized clusters, that are capable of modeling certain types of structures more accurately, would further decrease this value, producing more compact and efficient occupancy models. Owing to the use of nearest neighbors for feature vector calculation, decreasing the number of clusters has a $\log(n)$ impact on training and querying computational times.

5. Conclusion

This paper has introduced a novel technique EHM for the incremental generation of 3D occupancy maps, based on the Hilbert maps framework. Several extensions are proposed, based on known limitations found in previous iterations of such framework, and the result is an efficient methodology that rivals current state-of-the-art occupancy mapping techniques in terms of computational speed, while providing

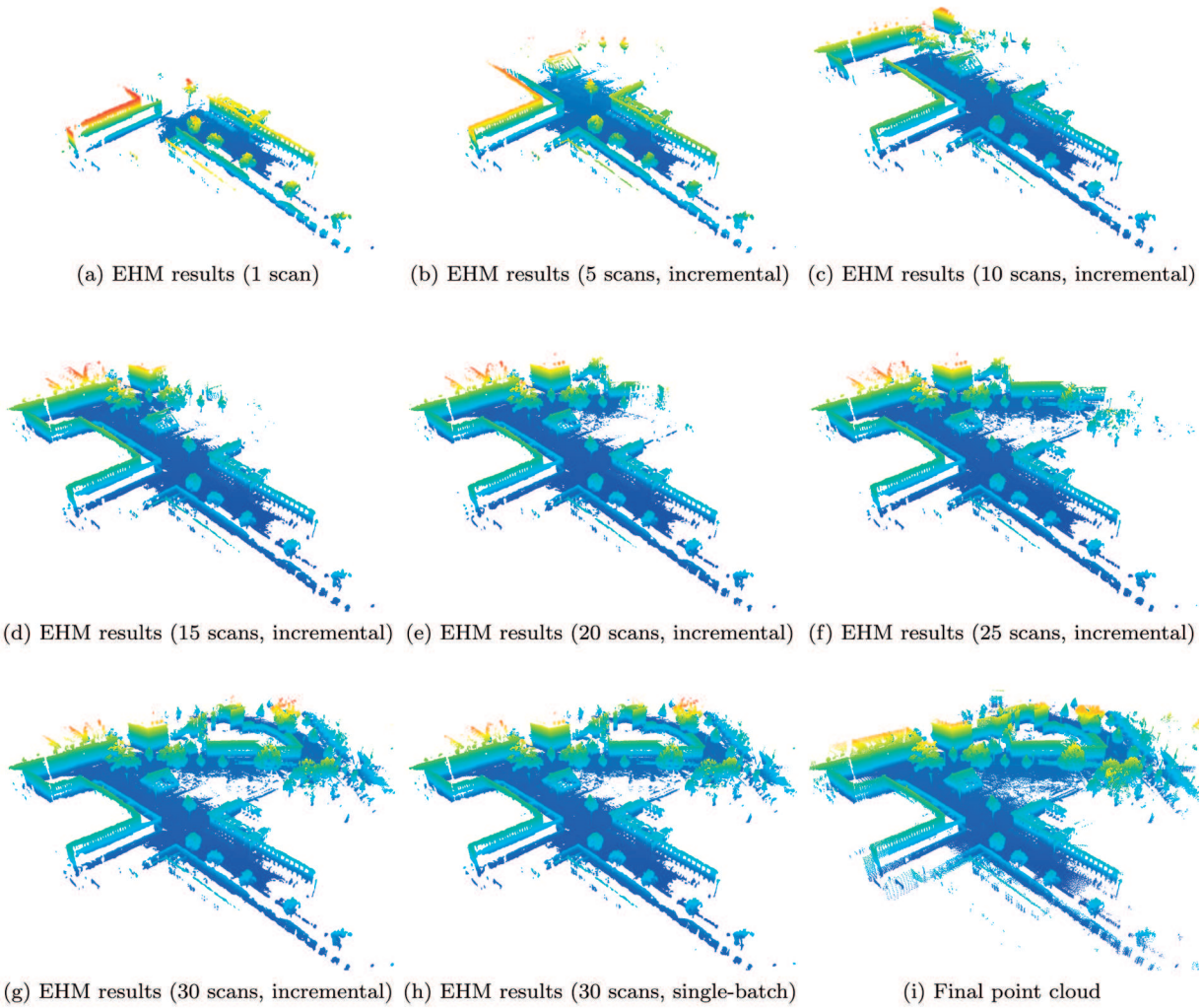


Fig. 12. Three-dimensional occupancy mapping results on the *Freiburg* dataset using EHM, as more scans are incorporated into an initially empty model (colored by height). In the bottom row, (g) depicts the final grid reconstruction results obtained incrementally; (h) depicts he final grid reconstruction results obtained when all point clouds are trained in a single batch; and (i) is the final point cloud, for comparison purposes.

results that are more robust in the presence of sparse data and have significantly better interpolative powers. Experimental tests have been conducted using both virtual and large-scale real datasets, depicting outdoors unstructured environments, with results that testify to the claim that the proposed methodology is a viable alternative to the task of large-scale 3D scene reconstruction and modeling of observed structures. Future work will focus on applying the proposed methodology to different tasks that traditionally require occupancy models, such as grasping, obstacle avoidance and autonomous navigation in unknown environments, developing further extensions that are beneficial to such scenarios. An extension to dynamic environments is also planned, so the occupancy model can adapt to gradual changes in the observed structures and use this information to predict its state in the near future. In addition, the authors aim to introduce the Hilbert maps framework to regression problems, by modifying the training and inference methodologies, thus allowing the use of EHM in a multitude of other applications.

Funding

This research was supported by funding from the Faculty of Engineering and Information Technologies, The University of Sydney, under the Faculty Research Cluster Program.

Notes

1. Note that the proposed incremental grid reconstruction technique can be used to generate estimates for subsets of the full observed grid, so the performance increase is still applicable.
2. Since EHM is non-deterministic, due to the clustering process, confidence intervals are also provided for the F_1-measure values. Other occupancy mapping techniques were not considered because they do not scale to datasets of such size.

References

Arthur D and Vassilvitskii S (2007) *k*-means++: The advantages of careful seeding. In: *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1027–1035.

Bachem O, Lucic M, Hassani S and Krause A (2016) Approximate *k*-means++ in sublinear time. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: *Proceedings of the International Conference on Computational Statistics (COMPSTAT)*, pp. 177–186.

Callaghan S and Ramos F (2012) Gaussian process occupancy maps. *The International Journal of Robotics Research* 31(1): 42–62.

Callaghan S and Ramos F (2015) Gaussian process occupancy maps for dynamic environments. In: *Experimental Robotics* (*Springer Tracts in Advanced Robotics*, vol. 109). New York: Springer, pp. 791–805.

Defays D (1977) An efficient algorithm for a complete-link method. *The Computer Journal* 20(4): 364–366.

Doherty K, Wang J and Englot B (2016) Probabilistic map fusion for fast, incremental occupancy mapping with 3D Hilbert maps. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Douillard B, Underwood J, Melkumyan N, Singh S, Vasudevan S and Quadros A (2010) Hybrid elevation maps: 3D surface models for segmentation. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 1532–1538.

Dragiev S, Toussaint M and Gienger M (2011) Gaussian process implicit surfaces for shape estimation and grasping. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2845–2850.

Elfes A (1989) *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD Thesis, Carnegie Mellon University, Pittsburgh PA, USA.

Freedman D (2005) *Statistical Models: Theory and Practice*. Cambridge: Cambridge University Press.

Guizilini V and Ramos F (2016) Large-scale 3D scene reconstruction with Hilbert maps. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Guizilini V and Ramos F (2017) Unsupervised feature learning for 3D scene reconstruction with occupancy maps. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Hadsell R, Bagnell J, Huber D and Hebert M (2010) Space-carving kernels for accurate rough terrain estimation. *The International Journal of Robotics Research* 29(8): 981–996.

Hamerly G and Elkan C (2004) Learning the k in k-means. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 16, pp. 281–288.

Hofmann T, Schölkopf B and Smola A (2008) Kernel methods in machine learning. *Annals of Statistics* 36(3): 1171–1220.

Hornung A, Wurm K, Bennewitz M, Stachniss C and Burgard W (2013) Octomap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots* 34(3): 189–206.

Jadidi M, Miro J and Dissanayake G (2017) Warped Gaussian processes occupancy mapping with uncertain inputs. *IEEE Robotics and Automation Letters* 2(2): 680–687.

Komarek P (2004) Logistic regression for data mining and high-dimensional classification. Technical report, Carnegie Mellon University.

Kucukyilmaz T (2014) Parallel *k*-means algorithm for shared memory multiprocessors. *Journal of Computer and Communications* 2: 15–23.

Lloyd S (1982) Least-squares quantization in PCM. *IEEE Transactions on Information Theory* 28: 129–136.

Lorensen W and Cline H (1987) Marching cubes: A high resolution 3D surface construction algorithm. In: *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, vol. 14, pp. 163–169.

Mason J, Ricco S and Parr R (2013) Texture occupancy grids for monocular localization without features. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Melkumyan A and Ramos F (2009) A sparse covariance function for exact Gaussian process inference in large datasets. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1936–1942.

Moravec H (1996) Robot spatial perception by stereoscopic vision and 3D evidence grids. Technical report, Carnegie Mellon University: Robotics Institute.

Muja M and Lowe D (2009) Fast approximate nearest neighbours with automatic algorithm configuration. In: *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*, vol. 4, pp. 331–340.

Nüchter A, Lingemann K, Hertzberg J and Surmann H (2007) 6D SLAM - 3D mapping outdoor environments. *Journal of Field Robotics* 24(8): 699–722.

Pelleg D and Moore A (2000) *X*-means: Extending *k*-means with efficient estimation of the number of clusters. In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 727–734.

Pereira E and Andreazza C (2010) Anisotropic *k*-nearest neighbor search using covariance quadtree. *Mecanica Computacional (Computational Geometry)* 24: 60.

Rahimi A and Recht B (2008) Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 1313–1320.

Rajalingam N and Ranjini K (2011) Hierarchical clustering algorithm - a comparative study. *International Journal of Computer Applications* 19(3): 42–46.

Ramos F and Ott L (2015) Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. In: *Proceedings of Robotics: Science and Systems (RSS)*.

Ramos F and Ott L (2016) Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. *The International Journal of Robotics Research* 35(14): 1717–1730.

Rasmussen C and Williams C (2005) *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press.

Rijsbergen V (1979) *Information Retrieval* (2nd edn). London: Butterworth.

Ruder S (2016) An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747v1*.

Sansone G (2012) *Orthogonal Functions: Revised English Version* (*Dover Books on Mathematics*). New York: Dover.

Schölkopf B, Muandet K, Fukumizu K, Harmeling S and Peters J (2015) Computing functions of random variables via reproducing kernel Hilbert space representations. *Statistics and Computing* 25(4): 755–766.

Schölkopf B and Smola A (2001) *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press.

Sculley D (2010) Web-scale k-means clustering. In: *Proceedings of the International Conference on World Wide Web (WWW)*, vol. 19, pp. 1177–1178.

Senanayake R, Ott L, Callaghan S and Ramos F (2016) Spatio-temporal Hilbert maps for continuous occupancy representation in dynamic environments. In: *Advances in Neural Information Processing Systems (NIPS)*.

Soohwan K and Jonghyuk K (2013) Continuous occupancy maps using overlapping local Gaussian processes. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 4709–4714.

Thrun S, Burgard W and Fox D (2005) *Probabilistic Robotics*. Cambridge, MA: MIT Press.

Williams C and Seeger M (2000) Using the Nyström method to speed up kernel machines. In: *Advances in Neural Information Processing Systems (NIPS)*.

Wipf D and Nagarajan S (2008) A new view of automatic relevance determination. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 20, pp. 1625–1632.