# Deep occupancy maps: a continuous mapping technique for dynamic environments

**Ransalu Senanayake**[1]
The University of Sydney

**Thushan Ganegedara**[2]
The University of Sydney

**Fabio Ramos**[3]
The University of Sydney

## Abstract

Learning a model of an environment that is correctly able to distinguish occupied and unoccupied areas is important for maneuvering robots in unstructured environments. A common technique to tackle such problems is to train a classifier with hand-crafted features that encode occupancy information. However, finding good features quickly becomes computationally prohibitive and impractical for complex and large environments. In this paper, we propose a "fully" convolutional neural network which can build global continuous occupancy maps by learning from raw local unorganized point cloud data, conveniently captured using a range sensor such as LIDAR. We propose a novel way of, 1) transforming data into a grid representation, and 2) perform convolution on robot's position rather than on occupancy levels. With this formulation, the map can produce both static and long-term maps in large environments without altering the model. Since the model is a function of locations, it is possible to query the occupancy probability at any position in the environment. Experiments indicate both computationally-efficient and accurate results over other continuous occupancy mapping techniques that require manual feature extraction.

## 1 INTRODUCTION

Intelligent transportation systems are rapidly improving human lifestyle. However, it is a challenging task to produce efficacious intelligent transportation systems. This can be primarily attributed to complex and contingent behavior of objects, especially in dynamic environments. Machine learning has recently been widely used in developing such intelligent transportation systems for modeling both perceptions and actions [1], ensuring robots work reliably and efficiently. In particular, deep learning has been successfully applied in many visual learning tasks, due to its rich representation learning capability enabled by large number of layers. For example, deep learning is widely used for autonomous driving from pedestrian detection to scene affordance estimation [2] proving to be a versatile *black-box* learning tool.

A promising avenue for using deep learning is in understanding occupancy of the 2D or 3D environments. A reliable occupancy maps is an imperative aspect in path planing and safer navigating in previously unmapped unstructured environments in both indoor and outdoor settings [3, 4], distinguishing dynamic areas from the stationary [5], identifying paths that pedestrians frequently take [6], etc. LIDAR laser range finders, which are ubiquitous and found in almost all autonomous vehicles as well as in indoor robotic platforms, are commonly used for building occupancy maps.

### 1.1 Continuous occupancy mapping

The seminal paper by Elfes [7] proposed a method to model the occupancy level of a static environment by dividing the world into a fixed sized grid and then updating the occupancy probability

Submitted to Workshop on Machine Learning for Intelligent Transportation Systems at the 30th Conference on Neural Information; (NIPS 2017), Long Beach, California. Email: [1]rsen4557@uni.sydney.edu.au, [3]tgan4199@uni.sydney.edu.au, [3]fabio.ramos@sydney.edu.au

of each cell individually. Due to this pre-discretization and independent updates, it not only limits building maps with different resolutions, but also the resolution directly affects the quality of the map as the cells can neither be too large nor small. In order to overcome these issues, Gaussian process occupancy maps (GPOMs) [8] have been proposed. Unlike grid maps, GPOMs produces spatially continuous occupancy maps—maps with arbitrary resolutions—without having to relearn the model. Nevertheless, being a Bayesian nonparametric method, the computational complexity of GPOMs in both the learning phase and querying phase grows cubically with the number of data points $N$ [9]. Even the fastest implementation of GPOMs, variational sparse dynamic GPOM (VSDGPOM) [5, 10] has a $\mathcal{O}(M^2N)$ runtime complexity which curtails using large datasets and mapping large areas, because in such cases the user selected parameter $M$ needs to be large enough to represent the entire dataset. In order to build continuous occupancy maps that do not retard the algorithm with the number of data points, Hilbert maps (HMs) [11, 12, 13] have been proposed. They operate in a high dimensional feature space and utilizes a logistic regression classifier. However, the number of features required to build the map increases with the area of the map.

Central to both VSGPOMs and HMs are kernels that operate in RKHS [14]. These kernels mainly help to capture the longitude-latitude spatial dependencies that are not possible to explicitly learn with grid maps [7]. In order to capture rich features using kernels, it is required to pre-specify which areas of the environment are more likely to have complex patterns. Such feature locations are called inducing points and hinged features in VSDGPOMs and HMs, respectively, and they are typically guessed by a clustering technique such as density based spatial clustering of applications with noise (DBSCAN) [5]. In this paper, we propose a technique to use convolutional neural networks (CNNs) where convolutional filters are used to model longitude-latitude spatial relationships. Importantly, unlike other continuous occupancy mapping techniques, the proposed method learns features fully automatically and hence it is not required to pre-specify them.

CNNs are also capable of modeling non-linear patterns that are otherwise captured by RKHS kernels in continuous occupancy maps. For instance, in VSDGPOMs, a kernel function is evaluated between data points to build a prior distribution for the Bayesian model, where as in HMs, a kernel function is used to project longitude-latitude pairs into a very high (i.e. finitely many) dimensional feature map. In contrast, since a neural network is the weighted sum of some functions that passes through activation functions, they inherently are non-linear classifiers. Considering the recent success in different application domains, we limit the discussion on nonlinear classifiers to deep neural networks and kernel methods [15], although other non-linear classification techniques such as decision trees or ensemble methods exist [16].

### 1.2 Fully Convolutional Neural Networks (FCNNs)

CNNs are behind the recent leap of image recognition tasks in autonomous vehicles [1, 2, 17]. However, most of these tasks have straightforward and well-understood objectives, e.g. classifying a given image into a category (e.g. vehicle, person) using a set of convolution layers followed by a set of densely connected layers [18, 19, 20]. However, our objective is to generate a global occupancy map from sample of local patches of occupancy. Since fully connected (densely connected) neural network layers consider interactions between all pixels, they would vitiate the pixel-wise correspondence between adjacent layers [21] and lose spatial information. For the same reason, we do not use pooling as this would cause to ignore certain pixels (i.e. max-pooling) or average them (i.e. average-pooling) during both inference and prediction phases. Following this reasoning, we utilize a fully convolutional neural network [21] where the network constitutes of only convolutional layers.

For the best of our knowledge, [22] is the only instance where deep learning has previously been used for a continuous occupancy mapping related task. However, it uses a variational autoencoder for image completion with the intention of reinforcing features in Hilbert maps, and the map is built completely using the RKHS kernel based Hilbert mapping technique discussed in Section 1.1. In contrast, our method generate the map as the output of the neural network and the features are learned solely from the neural network, thus removing any bias induced by hand-crafted features.

The proposed method method,

1. Produces spatially smooth continuous occupancy maps
2. Produces long-term occupancy maps (the average occupancy in an area over time) in dynamic and complex environments efficiently

3. Learns features fully automatically
4. Can update the occupancy map as new laser scans are captured in an online manner (if needed)
5. Can adapt to large environments without any modifications to the existing architecture
6. Does not rely on auxiliary motion models or object trackers

## 2 Deep occupancy maps (DOMs)

In this section, we present how to represent the occupancy as a function. Firstly, in Section , how to create a dataset to train the fully convolutional neural network is described. Unlike CNNs used for typical computer vision applications where the input is an image and the output is another image or a label, the inputs to the proposed CNN are logitude-latitude virtual grids. This virtual meshgrid generation from raw laser scans is described in Section 2.1. Details of where these data are used in the CNN is provided in section 2.2, and finally, how to train and query the CNN is described in Section 2.3.

### 2.1 Data generation and virtual grids

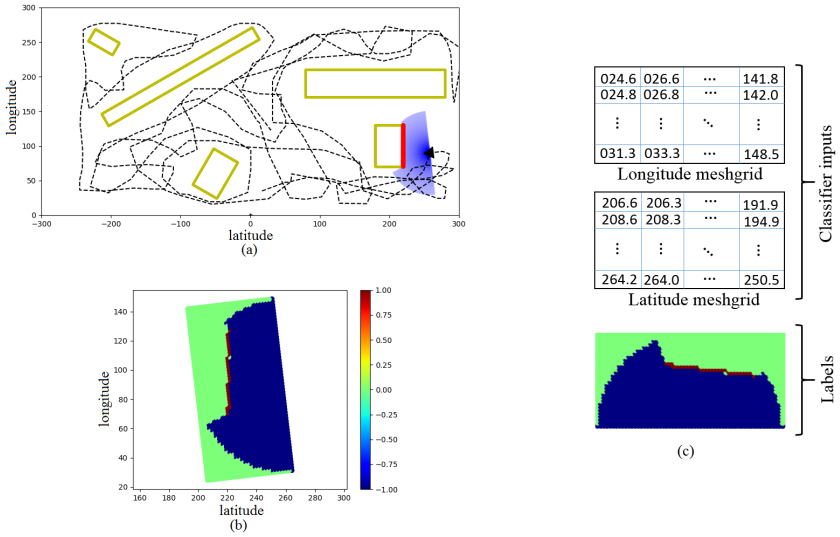



Figure 1: Data generation (a) An example of an environment. Obstacles are indicated in yellow and the paths the robot traverses is indicated in black (b) A virtual grid around the area the robot can see is created and occupancy levels are indicated by three colors $\{$blue,green,red$\} = \{-1, 0, 1\} = \{$free,unknown,occupied$\}$ (c) Longitudes and latitudes are separated into two meshgrids and labels into another grid. Note that all three matrices have the same size.

As in [8, 11], we assume that a 2D lidar range sensor mounted on a moving robot is used to collect data and the robot's pose (longitude, latitude, and direction) is known from an external sensor. Figure 1 (a) shows a sample path the robot maneuvers to collect laser hit points with respect to the robot's pose. There, at an arbitrary pose, the laser beams are shown in blue and laser hit points are shown in red.

The first step is to create input-output pairs for training the CNN, and this data can be generated at each pose on the fly while the robot maneuvers. To this end, when the robot is at a given pose, an area that encloses the entire laser scan of the size $2R \times R$ with maximum laser beam distance equals of $R$ is delimited. Then, this realm is divided into a fixed size grid which is then used to generate both classifier inputs and labels for training the CNN. However, although is method also uses a fixed size grid, the proposed method does not suffer from the limitations of occupancy grid mapping discussed in Section 1.1, because i) grids are to merely provide an input to the CNN which will capture spatial relationships, ii) the world is not discretized before the robot starts mapping and the virtual grid changes in the global coordinate system with different viewpoints depending on the location of the robot.

As illustrated in Figure 1 (c), the center points of the grid cells that lie in the 2D Cartesian plane are separated into two "meshgrids," one for latitudes and the other for longitudes. That is, for grid cell indices $i$ and $j$, $\{c^{(i)(j)}, (x_{lon}^{(i)(j)}, x_{lat}^{(i)(j)})\}_{i=0,j=0}^{2N-1,N-1}$ are separated into three different 2D matrices—longitude meshgrid $\{x_{lon}^{(i)(j)}\}_{i=0,j=0}^{2N-1,N-1}$, latitude meshgrid $\{x_{lat}^{(i)(j)})\}_{i=0,j=0}^{2N-1,N-1}$, and $\{c^{(i)(j)}\}_{i=0,j=0}^{2N-1,N-1}$. Here, $x_{lon}^{(i)(j)}$ and $x_{lat}^{(i)(j)}$ are the longitude and latitude locations in the global coordinate systems given in $i^{\text{th}}$ and $j^{\text{th}}$ cells of the virtual grid. All cells $\{c^{(i)(j)}\}_{i=0,j=0}^{2N-1,N-1}$ are filled as $\text{occupied} = 1$, $\text{free} = -1$, or $\text{unknown} = 0$ based on how laser beams pass through four sides of each cell. If a beam passes only one side of the cell, then the cell is occupied; if it passes more than one side, then the cell is free; and if it does not pass through any of the sides of the cell, then the accupancy state is unknown. If contradictory decisions are taken for a particular cell because of two or more beams passing through different sides of the same cell, such a cell is also labeled as unknown.

In summary, the content of the two meshgrid matrices are longitude and latitude locations whereas the the content of the label matrix is the actual occupancy level. In other words, a location in the environment indicated by the $(i, j)^{\text{th}}$ cell of the two meshgrids has a corresponding occupancy level indicated by the $(i, j)^{\text{th}}$ cell of the label matrix. In the next section, let us discuss how to use these three matrices in the CNN.

### 2.2 DOM-FCNN architecture

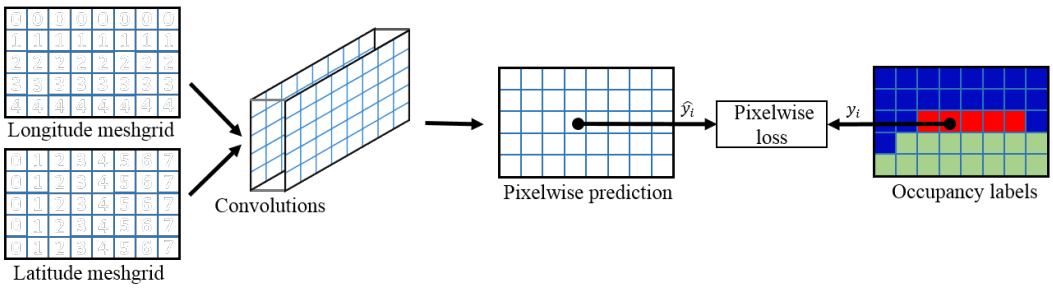



Figure 2: Architecture. Inputs to the neural network are longitude-latitude meshgrids shown in Figure 1 (c). Inputs passes through a series of convolutions and deconvolution, yet maintaining the pixelwise correspondence. The loss is calculated based on the predicted output and labels from Figure 1 (c).

Unlike in typical CNNs where the input is an image, the proposed architecture takes longitude-latitude meshgrids generated in Section 2.1 as inputs. The key insight is that we want to capture local dependencies using filters of the CNN that are otherwise captured by RKHS kernels in the other occupancy mapping techniques. On the other hand, since we work with a virtual meshgrid system, as discussed in Section 2.1, the size of the inputs ($W = 2R$ and $H = R$) to the CNN only depends on the maximum lidar distance $R$. This gives the proposed method the capability of mapping arbitrarily large environments without widening the CNN.

The $i^{th}$ input to the network, $x_i \in \mathbb{R}^{H,W,C}$ where $H$, $W$, and $C$ are the height, width, and the number of channels of the input, respectively. Here, $C = 2$ and the two channels are longitude and latitude meshgrids. The $l^{th}$ convolution layer, consists of $D_l$ kernels defined as, $\{K_l^0, K_l^0, \ldots, K_l^{D_l}\}$. The outputs of the $l^{th}$ layer is denoted by $g_l = \{g_l^0, \ldots, g_l^{D_l}\}$, where $g_l^i = \text{LeakyRelu}(g_{l-1} * K_l^i + b_l^i)$ with $g_0 = x$, $b_l^i \in \mathbb{R}$ is the bias and LeakyRelu is the leaky ReLU activation. The network uses a stride of 1 and padding the boundary with zeros in order to ensure the output $g_l^i \in \mathbb{R}^{H,W,C}, \forall l = \{1, \ldots, L\}, i = \{1, \ldots, D_l\}$ where $L$ is the number of convolutional layers. The network, however, shuld not be unnecessarily deep as it might loose pixelwise correspondence.

Next, followed by the $L$ convolution layer, $L$ deconvolution layers, with the $l^{th}$ deconvolution layer having $\hat{D}_l$ kernels, $\hat{K}_l^0, \ldots, \hat{K}_l^{D_l}$ is defined. Let the output of the $l^{th}$ deconvolution layer be $\hat{g}_l = \{\hat{g}_l^0, \ldots, \hat{g}_l^{\hat{D}_l}\}$, where $\hat{g}_l^i = \text{LeakyRelu}((\hat{g}_{l-1}^i * \hat{K}_l^i)^\top + \hat{b}_l^i), \forall l = \{1, \ldots, L-1\}$, where $\hat{g}_0 = g_L$ $(\cdot * \cdot)^\top$ defines the deconvolution or transposed convolution operation. Note that deconvolutions should be setup in such a way that the size of the output layer should be as same as the input meshgrid.

Finally the last layer, i.e. $l = L$ is defined as $\hat{g}_L^i = \text{Tanh}((\hat{g}_{L-1}^i * \hat{K}_L^i)^\top + \hat{b}_L^i)$. This output indicates the occupancy level $\in [-1, 1]$ of a pixel.

### 2.3 Training and querying

It is important to take into consideration the *imbalance* in the dataset while calculating the loss. For example, in the environments shown in Figures 4 and 5, the amount of occupied map points is relatively smaller compared to the unoccupied map points. Therefore, we use *importance reweighing* in the loss function. By representing the neural network as function $f^{\mathbf{w}}(\mathbf{x})$ which outputs the occupancy level of a longitude-latitude pair $\mathbf{x}$ anywhere in the space and corresponding labels $y$, for a batch of data of size $B$ with $B_+$ occupied data points, $B_-$ unoccupied data points, and $B_{unk}$ occupancy unknown data points, our objective is to minimize $\sum_{i=1}^{B} \alpha_i (f^W(\mathbf{x}_i) - y_i)^2$, where,

$$\alpha_i = \begin{cases} 1 - (B_+/B), \text{ if } y_i = 1 \\ 1 - (B_-/B), \text{ if } y_i = -1 \\ 1 - (B_{unk}/B), \text{ if } y_i = 0. \end{cases} \tag{1}$$

We can now use stochastic gradient descent to learn $\mathbf{w}$ of the CNN using the input-label pairs that are generated in Section 2.1. In other words, the pixel-wise loss is computed for given meshgrid frames.

Unlike grid maps where probabilities of the discretized space are stored as the map, here we store weights of the CNN as the map which can be query at anytime. The required query locations ($W \times H$ area) should be in the same form as the two meshgrids in Section 2.1. On the other hand, because the map is merely represented as a function, irrespective of the intermediate discretization process, DOM-FCNN can predict occupancy of any location, i.e. continuous maps. To visualize large maps, $f^{\mathbf{w}}$ can be queried with a moving meshgrid window. Note that the outermost query points of the $W \times H$ window can be distorted because of padding and they can be easily re-evaluated using a different query window.

## 3 Experiments

Two different datasets have been used in the following experiments to demonstrate the key features of DOM-FCNN;

1. Large Environments (Figure 4 (a)): Dataset 1 is collected by a robot with a 60 m lidar that traverses in the environment in a random trajectory. To map the $600 \times 300$ m$^2$ area, we only used 1300 robot poses. We query the map with a 1 m resolution.
2. Dynamic Environments (Figure 5 (a)): Dataset 2 is obtained using a stationary robot which has a 100 m lidar. Dynamic objects constantly move in a road across the robot's view. That is, at a given time, the occupancy of a point in this trajectory (road) is a value between $-1$ and 1. Since the dataset is sequentially collected over 330 time steps, the deep occupancy map provides the average map—long-term map. This is the same environment used in [5].

Table 1: Configuration of DOM-FCNN. Each convolution and deconvolution layer is denoted by Conv $(i \times j) \times k$ where $i \times j$ is the convolution filter size and $k$ is the channel depth. All layers used a stride of 1.

| Layer | Specifications |
|---|---|
| | $W \times H \times 2$ Input |
| Conv1 | $(3 \times 3) \times 16$ (LeakyRelu) |
| Conv2 | $(1 \times 1) \times 16$ (LeakyRelu) |
| Conv3 | $(3 \times 3) \times 32$ (LeakyRelu) |
| Conv4 | $(1 \times 1) \times 32$ (LeakyRelu) |
| Conv5 | $(3 \times 3) \times 32$ (LeakyRelu) |
| Conv6 | $(1 \times 1) \times 32$ (LeakyRelu) |
| Deconv3 | $(1 \times 1) \times 32$ (LeakyRelu) |
| Deconv2 | $(1 \times 1) \times 16$ (LeakyRelu) |
| Deconv1 | $(1 \times 1) \times 1$ (Tanh) |

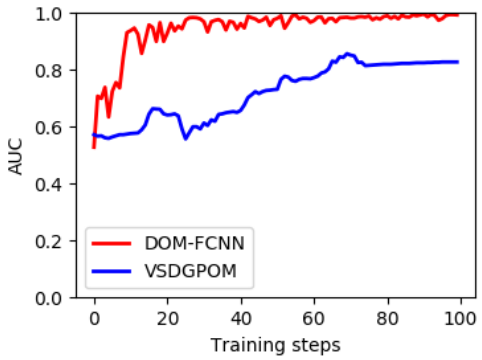
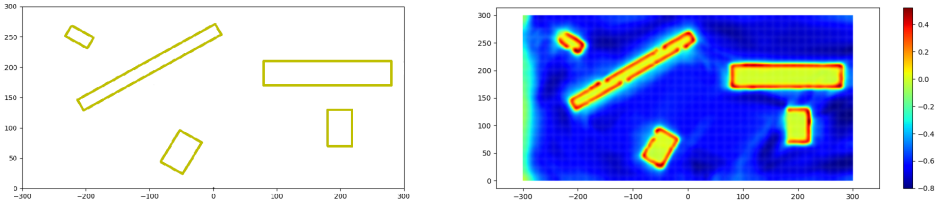



Figure 3: The learning curve for dataset 1.

The parameters of DOM-FCNN are summarized in Table 1. $W \times H = 60 \times 30$ and $W \times H = 100 \times 50$ for datasets 1 and 2, respectively. The algorithm was implemented in TensorFlow and the $[\text{batch\_size}, \text{height}, \text{width}, \text{n\_channels}]$. Here, $\text{n\_channels} = 2$ are the longitude and latitude channels shown in Figure 2. Although experiments were run on a 10 GB Nvidia-K40c, using a simple CPU would not thwart learning in a feasible time.

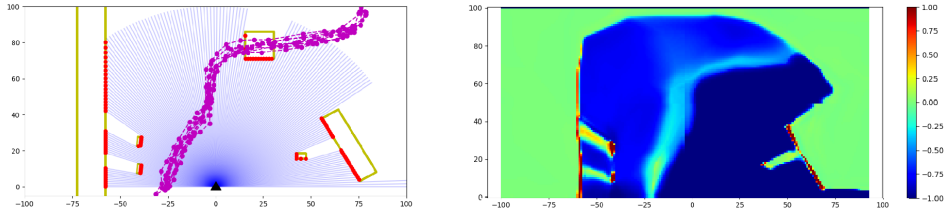
Table 2: Average performance metrics

| | Accuracy (%) | | AUC | | Training Time (s) | | Query Time (ms) | |
|---|---|---|---|---|---|---|---|---|
| Method | Dataset 1 | Dataset 2 | Dataset 1 | Dataset 2 | Dataset 1 | Dataset 2 | Dataset 1 | Dataset 2 |
| DOM-FCNN | 94.9 | 98.2 | 0.997 | 1.000 | 268 | 25.28 | 1.519 | 0.441 |
| VSDGPOM | 72.4 | 88.5 | 0.827 | 0.949 | 1126 | 3440 | 4.355 | 9.48 |



(a) (b)

Figure 4: Dataset 1. (a) A static environment with obstacles. (b) DOM-FCNN map



(a) (b)

Figure 5: Dataset 2. (a) The robot resides at $(0, 0)$. Laser beams and laser hit points at a given time are shown in blue and red, respectively. The path of moving objects is shown in magenta. Note that there are three parked vehicles the background occluding the static walls. (b) DOM-FCNN map

The $600 \times 300$ static map and the long-term map are shown in Figures 4 (b) and 5 (b), respectively. Since $\text{sigmoid}(x) \equiv \big(\tanh(x/2) + 1\big)/2$, log odds of occupancy can be denoted as $\exp(f^{\mathbf{w}}(\mathbf{x}))$. Hence, the average occupancy probability (i.e. the long-term probability) along the trajectory of moving vehicles is between 0 and 1. We also can conclude that the probability beyond the trajectory is clearly slightly lower than the fully observable area as sometimes the former area is partially occluded. The occupancy behind the three parked vehicles is also low. We used manually labeled points as the test dataset. The comparison of DOM-FCNN with VSDGPOM which requires feature extraction minimally is given in Table 2. Clearly, the proposed method is not only more accurate, but also it is faster.

## 4 Conclusions and future work

In this paper, we proposed an end-to-end learning technique, DOM-FCNN, that is capable of learning the topographical occupancy of large and dynamic environments without hand-crafted features. From a macroscopic view, one may think the whole framework as a collection of virtual occupancy grid maps being reprocessed into a parametric function which can be evaluated to produce continuous occupancy maps. We demonstrated the key properties of the method using simulated datasets. As the immediate step, we will use real-world datasets. As this is a sequential learning task, we also plan to adapt the neural network architecture according to the data distribution on the fly [23] to further cope with complex patterns with a minimal budget.

## References


[1] A. Seff and J. Xiao, "Learning from Maps: Visual Common Sense for Autonomous Driving," in *arXiv preprint arXiv:1611.08583*, 2016.

[2] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2722–2730, 2015.

[3] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[4] D. Hahnel, R. Triebel, W. Burgard, and S. Thrun, "Map building with mobile robots in dynamic environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[5] R. Senanayake, S. O'Callaghan, and F. Ramos, "Learning highly dynamic environments with stochastic variational inference," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[6] S. T. O'Callaghan, S. P. Singh, A. Alempijevic, and F. T. Ramos, "Learning navigational maps by observing human motion patterns," in *Robotics and automation (ICRA), 2011 IEEE international conference on*, pp. 4333–4340, IEEE, 2011.

[7] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. RA-3(3), pp. 249–265, 1987.

[8] S. T. O'Callaghan and F. T. Ramos, "Gaussian process occupancy maps," *The International Journal of Robotics Research (IJRR)*, vol. 31, no. 1, pp. 42–62, 2012.

[9] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[10] J. Hensman, A. Matthews, and Z. Ghahramani, "Scalable variational gaussian process classification," in *the Eighteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[11] F. Ramos and L. Ott, "Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent," in *Proceedings of Robotics: Science and Systems (RSS)*, 2015.

[12] K. Doherty, J. Wang, and B. Englot, "Probabilistic map fusion for fast, incremental occupancy mapping with 3d hilbert maps," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 0–0, 2016.

[13] R. Senanayake and F. Ramos, "Bayesian hilbert maps for dynamic continuous occupancy mapping," in *1st Annual Conference on Robot Learning (CoRL)*, 2017.

[14] A. Smola, A. Gretton, L. Song, and B. Schölkopf, "A hilbert space embedding for distributions," in *International Conference Algorithmic Learning Theory (COLT)*, pp. 13–31, Springer-Verlag, 2007.

[15] Z. lu, A. May, K. Liu, A. B. Garakani, D. Guo, A. Bellet, L. Fan, M. Collins, B. Kingsbury, M. Picheny, and F. Sha, "How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets," in *arXiv preprint arXiv:1411.4000v2*, 2014.

[16] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[17] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[20] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *arXiv preprint arXiv:1602.07261v2*, 2016.

[21] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

[22] V. Guizilini and F. Ramos, "Learning to reconstruct 3d structures for occupancy mapping," in *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[23] T. Ganegedara, L. Ott, and F. Ramos, "Online Adaptation of Deep Architectures with Reinforcement Learning," in *arXiv preprint arXiv:1608.02292*, 2016.