3D Convolutional Neural Networks for Landing Zone Detection from LiDAR

Daniel Maturana[1] and Sebastian Scherer[1]

***Abstract*— We present a system for the detection of small and potentially obscured obstacles in vegetated terrain. The key novelty of this system is the coupling of a volumetric occupancy map with a 3D Convolutional Neural Network (CNN), which to the best of our knowledge has not been previously done. This architecture allows us to train an extremely efficient and highly accurate system for detection tasks from raw occupancy data. We apply this method to the problem of detecting safe landing zones for autonomous helicopters from LiDAR point clouds. Current methods for this problem rely on heuristic rules and use simple geometric features. These heuristics break down in the presence of low vegetation, as they do not distinguish between vegetation that may be landed on and solid objects that should be avoided. We evaluate the system with a combination of real and synthetic range data. We show our system outperforms various benchmarks, including a system integrating various hand-crafted point cloud features from the literature.**

I. INTRODUCTION

The capability to autonomously find and select good landing sites is essential for the flexible and safe operation of unmanned rotorcraft. Unmanned helicopters, as vertical takeoff and landing (VTOL) vehicles, have the potential advantage of being capable of landing in a wider variety of sites than conventional takeoff and landing vehicles. However, for autonomous helicopters to take full advantage of this capability they must be able to accurately assess the safety of sites in which they can land.

Current state-of-the-art systems in LiDAR landing zone detection ([1], [2]) evaluate the safety of landing zones based on simple geometric features of point clouds, such as residuals from a robust plane fit. Nevertheless, this approach may miss potential landing sites covered in low vegetation such as grass, despite the fact that grass height of up to $30\,\mathrm{cm}$-$45\,\mathrm{cm}$ are considered safe [3]. This occurs because cluttered vegetation is seen as highly rough terrain, despite being compliant enough to be safely landed on. While a clear landing site is preferable over one covered in vegetation (other things being equal), the ability to find safe landing zones in terrain covered by low vegetation extends the type of environments in which the unmanned rotorcraft can operate.

Engineering a system capable of reliably detecting landing safe landing sites under these conditions is a difficult task. The appearance of vegetation, terrain and potential obstacles is highly heterogeneous and difficult to capture with hand-built rules. These difficulties are compounded by the noisiness and sparsity that often occurs in real-world point cloud data.

[1]Robotics Institute, Carnegie Mellon University, Forbes Ave 5000, Pittsburgh PA 15201 USA { dimatura, basti } at cmu.edu

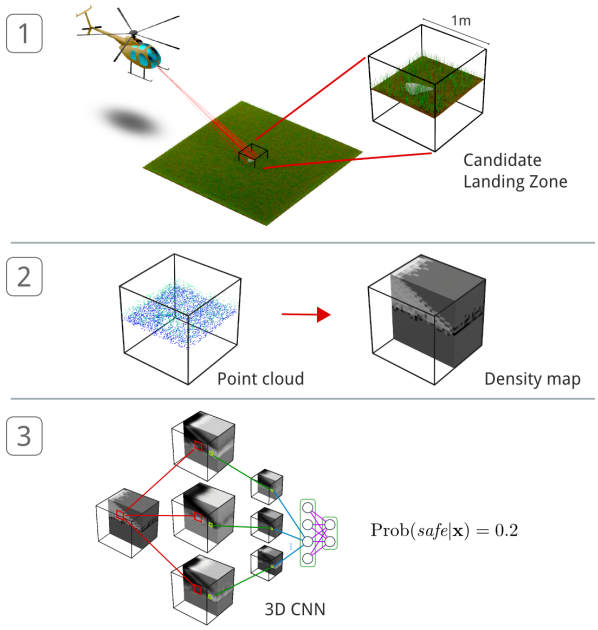

Fig. 1. Outline of our system: *1)* The helicopter acquires scans of vegetated terrain. The terrain is divided in $1\,\mathrm{m}^3$ subvolumes. We focus on the subvolume indicated by the box outline. *2)* The scans are converted to registered point clouds and are used to update a volumetric density map encoding spatial occupancy, which is the input $\mathbf{x}$ to the network. *3)* The 3D convolutional network predicts the safety of the subvolume from the density data $\mathbf{x}$.

Therefore, it is natural to turn to machine learning and train a classifier to solve this task. However, this poses its own challenges. We are interested in efficiently assessing the safety of areas in the order of $100\,\mathrm{m}^2$ to $200\,\mathrm{m}^2$ with a resolution of $1\,\mathrm{m}^2$. Our sensor can collect thousands of points per m^2 in a matter of seconds, but only a few may actually be discriminative to assess landing zone safety; how do we effectively maintain the relevant information from this large stream of data? We also need to be able to assess the safety of candidate landing sites "on-the-fly", along with a measure of confidence, for the system to be useful. Finally, how do we obtain labelled data? Acquiring and labeling the point cloud data for different vegetation characteristics, LiDAR viewpoints, obstacle poses, etc. is an expensive and laborious process. In this paper we address these challenges with the following contributions:

- A system to efficiently and reliably assess the safety of landing zones covered in low vegetation (Figure 1). The system combines a volumetric occupancy map with a 3D Convolutional Neural Network (CNN), which to our knowledge has not been previously done. The

occupancy representation encodes information about free and occupied space and can be incrementally updated from point clouds. Safety evaluation of a volume with the 3D CNN takes less than $5\,\mathrm{ms}/\mathrm{m}^3$.
- The use of semi-synthetic point clouds to generate realistic automatically labelled point clouds. We use these to train the CNN to perform probabilistic landing zone safety predictions using only raw occupancy data as input, without any hand-crafted features.

II. RELATED WORK

A. Landing zone detection

An early approach using simulated LiDAR for landing zone detection is Johnson et al. [4]. They propose a system for landing zone selection based on a relatively simple geometric analysis of terrain roughness and slope. Whalley et al. [1] use similar geometric criteria and demonstrate its success in real data. Scherer et al. [2] also perform a similar geometric analysis as part of a larger system that also incorporates factors such as terrain/vehicle interaction, wind direction and mission constraints. Warren et al. [5] propose a similar approach based on geometric criteria, using visual structure from motion instead of LiDAR. As we will see in the experiments, these simple geometric approaches break down when the landing site is covered by low vegetation, as this makes the landing site appear as excessively rough.

B. Ground filtering and terrain modelling

In the area of remote sensing there is a rich body of literature devoted to extracting bare earth surfaces from aerial LiDAR point clouds, also known as ground filtering; see [6] for an overview. These algorithms are designed to discriminate ground points from non-ground points, regardless of what kind of objects the non-ground points correspond to. In our case we need to distinguish between safe and unsafe non-ground objects.

In robotics there are several terrain modelling approaches designed to model a ground surface for navigation purposes (e.g. [7]). These suffer from the converse problem of not making any distinctions between ground and non-ground points, which means vegetated terrain will always appear as highly rough and unsafe.

C. LiDAR Semantic classification

Several works perform of classification of various semantic classes in LiDAR point clouds, including vegetation and ground ([8], [9], [10], [11]). However, in our case we are not interested in the semantic labels of each point or voxel in a scene, but to assess its overall safety. Regardless, we have taken various hand-engineered point cloud features from work in this area to build one of our baselines (section IV).

Another relevant line of work proposes to discriminate traversable grass from hard obstacles using local statistics of laser scan lines ([12], [13]). These systems assume measurements are taken from forward-facing scanners on ground vehicles, and do not model the ground, which is important for our task.

D. Convolutional Neural Networks

CNNs [14] are a class of deep learning algorithms that use the spatial structure of the input (usually image data) to reduce the amount of parameters to be learned. Modern variations of CNNs have shown state-of-the-art performance in various computer vision tasks [15], resulting in widespread interest from the community.

There has been work in applying CNNs to semantic labelling [16] and grasp detection [17] from RGBD data. However, these works simply treat depth images analogously to RGB images, which ignores their 3D structure and impedes integration over time. Moreover, in our case, where the point clouds come from a LiDAR sensor in movement, there is no direct analog to a depth image.

3D CNNs have been applied to other domains. Viewing video data as a volume with time as the third dimension, Ji et al. [18] apply 3D CNNs to human action classification. Flitton et al. [19] apply a CNN-like method to the classification of computed tomography imagery. From an architectural viewpoint these CNNs are similar to ours, as they perform 3D convolution, but otherwise the data and tasks are quite different.

Recently, Lai et al. [20] applied a form of unsupervised feature learning related to deep learning to semantic labelling of indoor scenes from RGBD data. Like this work, they use a volumetric representation, however, it is not convolutional but dictionary-based, which is generally less efficient.

III. APPROACH

A. Overview

The input of our system is a stream of globally registered LiDAR point clouds and a predefined region of interest with candidate landing sites to be evaluated. The output is probabilistic safety prediction for each landing site.

To this end our system has two main components. The first is a volumetric density map for the area containing landing zones of interest. In our system this map usually covers a horizontal area of $100\,\mathrm{m}^2$ to $200\,\mathrm{m}^2$ and a height of $2\,\mathrm{m}$ to $4\,\mathrm{m}$. The second is a CNN used to independently predict the safety of $1\,\mathrm{m}^3$ subvolumes within this map.

We note that the output of the system described in this paper is not meant to be the sole or final arbiter on landing zone decisions; rather, it is designed meant to output a prediction to be integrated into a more complex evaluation system that also takes in consideration mission constraints and objectives, atmospheric conditions, and other factors ([2], [21]).

B. Volumetric density mapping

Given the region of interest and a point cloud stream, we incrementally build a volumetric density map, similar in spirit to occupancy grids [22]. Intuitively, we expect space containing vegetation to be seen as relatively "porous" by the LiDAR sensor, compared to space containing solid objects. Researchers have used this intuition to model vegetation by counting the ratio of LiDAR hits to pass-throughs [23] for each grid cell in a map. We use a probabilistic formulation of this idea, in which we interpret the posterior mean of a

Bernoulli with a uniform Beta prior as the density of each cell. This formulation was used by [24] to create range keypoints.

More formally, let $\{z^t\}_{t=1}^T$ be a sequence of registered range measurements that either hit ($z^t = 1$) or pass through ($z^t = 0$) a given grid cell with coordinates (i, j, k). We assume an "ideal beam" inverse sensor model for LiDAR, where we simply use 3D ray tracing [25] to obtain the number of hits and pass-throughs for each grid cell.

To track the state of each cell we use the Beta parameters α_{ijk}^t and β_{ijk}^t, with a uniform prior $\alpha_{ijk}^0 = \beta_{ijk}^0 = 1$ for all (i, j, k). The update for each grid cell affected by the measurement z_t is

$$\alpha_{ijk}^t = \alpha_{ijk}^{t-1} + z^t$$
$$\beta_{ijk}^t = \beta_{ijk}^{t-1} + (1 - z^t)$$

and the posterior mean for the cell at (i, j, k) is

$$\mu_{ijk}^t = \frac{\alpha_{ijk}^t}{\alpha_{ijk}^t + \beta_{ijk}^t} \tag{1}$$

While the mean may be calculated incrementally, in practice we simply keep track of the number of hits and pass-throughs for each cell and compute the density directly from Equation 1 as needed. In our current implementation we store these values in a scrolling grid, which discards measurements outside a moving bounding box to keep memory usage constant. In all the experiments in this paper we use voxels of size $0.05\,\mathrm{m}^3$ with maps of up to $20\,\mathrm{m} \times 20\,\mathrm{m} \times 4\,\mathrm{m}$, which only takes around $25\,\mathrm{MB}$ of space when using two bytes per cell.

Currently we simply use the posterior mean of each cell as input to the network. This throws away some information regarding the uncertainty in our beliefs. We performed experiments using the posterior variance as an additional input channel, but did not find significant improvements.

We have also explored the use of occupancy maps [22] as an alternative representation. However, our preliminary experiments found slightly inferior performance.

C. Safety prediction

1) Volume subdivision: To predict the safety of a region we first uniformly subdivide the XY plane of our map into non-overlapping $1\,\mathrm{m}^2$ tiles. For each tile we estimate the ground surface height from a robust minima calculation of the height of all the occupied cells within. We then place a $1\,\mathrm{m}^3$ subvolume on each tile, centered on z around the ground surface height estimate. Finally each subvolume is evaluated independently with the CNN. In this work we use a resolution of $0.05\,\mathrm{m}^3$, so each subvolume has voxel dimensions $20 \times 20 \times 20$, with padding of up 6 voxels per dimension.

A possible shortcoming of this strategy is that the it is nontrivial to calculate the ground surface height precisely. In practice, we have not found this to be a problem, as the CNN is designed to be robust to small translational shifts.

2) Volumetric Convolutional Neural Nets: CNNs have three distinguishing features that make them useful for our scenario. First, they can explicitly make use of the spatial structure of our problem. In particular, they can learn local spatial filters useful to the classification task. In our case, we expect the filters at the input level to encode structures such as planes and corners. Second, by stacking hidden neural network layers the network can construct a hierarchy of more complex features over larger regions of space, eventually leading to the final classification. For example, if a region of unobserved space is seen behind a region of occupied space, that is a strong cue that there is a potential obstacle occluding the LiDAR rays. Third, they can be trained end-to-end from the raw volumetric density data to provide a probabilistic safety prediction for each subvolume. The probabilistic output allows management of risk, and use of uncertainty to guide sensing actions.

Let $\{\mathbf{x}^n, y^n\}_1^N$ be a dataset where each $\mathbf{x}^n \in \mathbb{R}^{I \times J \times K}$ is a subvolume containing density data and y^n is the corresponding ground truth label (*safe* or *unsafe*). We aim to learn a predictor $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ assigning a posterior probability of *safety* to any given subvolume.

To model this posterior probability we designed a volumetric convolutional neural network patterned after the networks in [15] extended from 2D (image) data to 3D (volume) data.

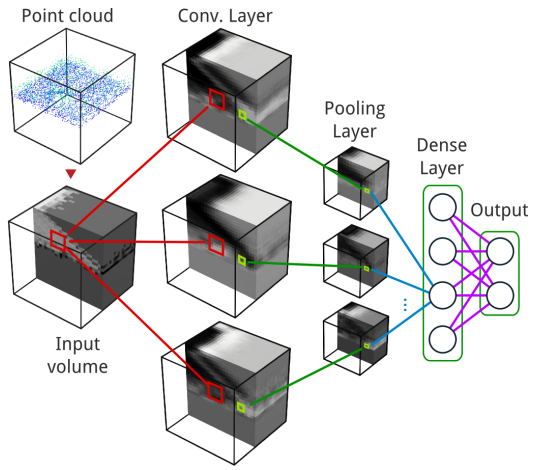

Fig. 2. Volumetric convolutional neural net architecture with a single convolutional layer. The input in the left shows the cross-section of a subvolume computed from the cloud (darker is denser). In the convolution layer we show cross-sections of three feature maps extracted from this patch. These feature maps are pooled and propagated into the dense and classification layers.

The CNN model consists of an input layer, followed by one or two convolutional layers, a single dense layer, and a classification output layer (Figure 2):

1) The input to the network is the subvolume $\mathbf{x}$, a 3D tensor with dimensions $I \times J \times K$, where $\mathbf{x}_{ijk} = 2\mu_{ijk} - 1$ is the current density estimate (Equation 1), scaled and centered to $(-1, 1)$.
2) The convolutional layer computes F_1 *feature maps* from the input data by convolving it with F_1 learned filters with spatial dimensions (f_h^1, f_h^1, f_v^1) (we consider

only square shapes for the horizontal axes). Its learned parameters are the filter weights W_1 and the bias b_1. The kth feature map is calculated as

$$\mathbf{v}_1^k = \text{relu}\left(W_1^k * \mathbf{x} + b_1^k\right)$$

where $*$ is the 3D convolution operation, the bias is added componentwise and relu$(\cdot)$ is simply $\max(0, x)$ applied componentwise (rectification). We then apply *max-pooling* with scale hyperparameter P_1. This step downsamples the feature map across the three spatial dimensions by using the maximum over non-overlapping blocks of P_1^3 voxels.

In this work we experiment with networks having one or two convolutional layers. In the latter case the two convolutional layers are consecutive (considering the rectification and pooling as part of the layer) and the second convolutional layer has the same form, but with parameters (W_2, b_2) and hyperparameters (f_h^2, f_v^2, F_2, P_2).

3) The dense layer consists of N_h units, each one connected to all the feature maps of the last convolutional layer. The output is a vector $\mathbf{v}_h \in \mathbb{R}^{N_h}$ with the kth value computed as

$$\mathbf{v}_h^k = \text{relu}\left(W_h^k \mathbf{v}_i + b_h^k\right)$$

where W^k and b_k are parameters and i is the index of the last convolutional layer.

4) The output layer gives a posterior probability computed as a logistic regression layer

$$\hat{y} = \sigma\left(W_o \mathbf{v}_h + b_o\right)$$

where $\sigma(x) = 1/(1 + \exp(-x))$ and W_o and b_o are parameters.

3) Model selection, learning and inference: The hyperparameters of the model are the number of convolutional layers and (F_i, f_h^i, f_v^i, P_i), where i ranges over the number of convolutional layers. These are explored through randomized search, as described in section IV.

The parameters of the model are $\Theta = \{W_i, b_i, W_h, b_h, W_o, b_o\}$ where i ranges over the number of convolutional layers. To learn these we use stochastic gradient descent on the regularized negative log-likelihood:

$$\Theta^* = \arg\min_\theta -\sum_n \log P(\hat{y}^n | \mathbf{x}^n; \Theta) + \lambda f(\Theta)$$

where $P(\cdot)$ is the probabilistic output of the logistic regression layer and $f(\cdot)$ is the L_2 norm of the weight parameters. We also apply dropout regularization [15] (with $p = .5$) on each layer's output.

We use the Theano library [26] to compute the gradients and accelerate computation with the GPU. At runtime, inference is done by a simple forward propagation from the subvolume input to the network output.

IV. EXPERIMENTS

A. Overview

To explore the parameters and hyperparameters of our volumetric CNNs we began with various simple experiments on a small synthetic dataset. We then did two comparison tests: one on a larger, more complex synthetic dataset, and another in a semi-synthetic dataset, consisting of scenes combining real and synthetic data.

B. Datasets

1) Synthetic datasets: In our synthetic datasets we recreate the scanning pattern of our LiDAR sensor by simulating its pulse repetition rate, angular resolution, and the nodding behavior of its sensor mount. Our sensor is a RIEGL configured to scans lines perpendicular to the direction of flight with up to 100° horizontal field of view (FOV). It is mounted on a custom motorized platform for nodding, optionally allowing up to 100° vertical FOV scanning. The sensor assembly is mounted on an helicopter which uses an INS for global registration of the point cloud.

We also add Gaussian noise to the range, based on the manufacturer specifications ($\sigma = 25\,\text{mm}$). Small motions for the origin of the sensor pose were added by a Gaussian random walk with zero mean and $\sigma = 2\,\text{cm/s}$. Scenes were scanned until a point density of $3000\,\text{points/m}^2$ was reached, consistent with the density obtained near the landing zones in the autonomous landing missions from [21].

The generated LiDAR rays are then intersected against a synthetic scene consisting of a ground surface, grass blades and obstacles, with two variations.

1) In the first synthetic dataset the ground surface is a mesh in which the height of the vertices are perturbed using Perlin noise with a height range of $(-0.05\,\text{m}, 0.05\,\text{m})$. The grass blades are simulated by 3-triangle strip of maximum width $3\,\text{mm}$, and normally distributed height and inclination. The grass placement is generated according to a homogeneous spatial Poisson process with a configurable intensity, as in [12]. The box is generated with $(0.15\,\text{m})^3$ dimensions at a uniformly random location and yaw angle on the plane.
2) In the second dataset the box obstacles were replaced with a selection of 3D models obtained from the Trimble 3D Warehouse. In particular, we use 11 models of rocks (modelled after actual rocks by Intresto[1]), a tire, and cinder blocks (Figure 3). The height of the objects ranges between $15\,\text{cm}$ to $40\,\text{cm}$, and are all less than $50\,\text{cm}$ across.

In each dataset various parameters were systematically sweeped as shown Table I. For the first dataset 20 instances were generated per parameter setting, resulting in 26860 scenes. In the second dataset 40 instances were generated, resulting in 28760 scenes. In each case only half the instances have an obstacle; these are considered *unsafe*, while the rest are considered *safe*.

[1] www.intresto.com.au

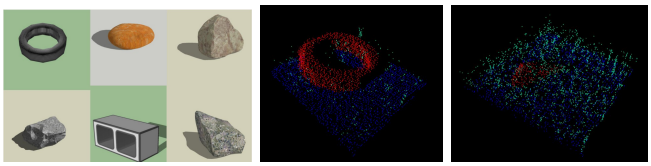

Fig. 3. Left: renderings of 6 of our CAD models. Middle and right: Two synthetic point clouds colored by point cloud type. Best seen in color.

TABLE I

SIMULATION PARAMETER SWEEP FOR SYNTHETIC DATASETS.

Dataset	Parameter	Values
1	Blade per m^2	$100, 200, \ldots 1200$
	Scanline angular resolution $^\circ$	$0.05, 0.1, 0.2, 0.41$
2	Blade per m^2	$100, 200, \ldots 1600, 1800$
	Blade width mm	$3,\ 5,\ 8$
Both	Blade mean height m	$0.1, 0.2, 0.3$
	Sensor distance in x-axis m	$-5.0, -10.0$
	Sensor height m	$4.0, 8.0$

Finally, since we wish the learned models to be invariant to the global orientation of the grid frame and to small errors in our ground height estimation procedure, the point cloud datasets are augmented dynamically during training by creating 20 perturbed copies of each point cloud. Each copy is rotated with a randomly chosen yaw angle and translated with a uniform random shift of $(-0.3\,\mathrm{m}, 0.3\,\mathrm{m})$ along the z direction.

2) Simulator validation: In order to validate the simulator for the first synthetic dataset we constructed a calibrated target and scanned in the laboratory. The target and a render of its synthetic version, along with a range histogram of 1500 points and an occupancy grid are depicted in Figure 4. Note that the blades are randomly generated and not meant to match the real grass on an individual blade level.

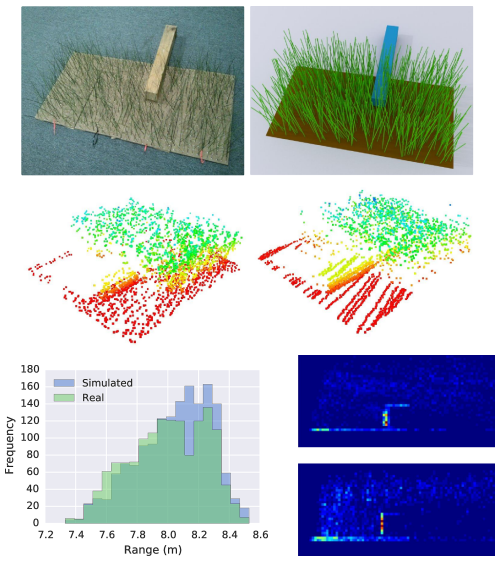

Fig. 4. Top row: the real point cloud and its simulated counterpart. Bottom row: the calibrated target and a render of its synthetic version.

3) Semi-synthetic data: In reality, the structure of the vegetation and terrain are much richer than our simple Poisson/Perlin processes. Moreover, we have observed that our simple model of range sensing is not very accurate for vegetation, despite being quite reasonable for solid objects. This has also been observed by Deschaud et al [27].

Therefore we propose the use of semi-synthetic point clouds, consisting of real point cloud data for vegetation and ground combined with simulated scans for solid obstacles. Since our sensor setup reports the estimated pose of the sensor at each measurement time, we can simply use our simulator with sensor rays obtained from actual point clouds, and build semi-synthetic scenes by inserting virtual obstacles in the world frame and altering rays if they intersect with the obstacle (see Figure 5). As before, we add noise to the simulated rays. To generate the semi-synthetic scenes we used

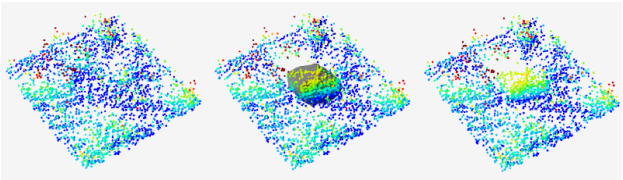

Fig. 5. Left to right: Real point cloud data, with an inserted CAD rock model used to simulate an obstacle, and the resulting semi-synthetic cloud.

data from 8 flights. The first two are from data collection flights performed in the Rock airport in Pittsburgh, PA on October 2013. The remaining six are autonomous landing missions performed on a site in Quantico, VA on February 2014 as part of the AACUS project [21]. Figure 6 shows images of the vegetation in these sites. We manually selected

Fig. 6. Left: Vegetation in the Quantico landing site. Right: vegetation in the Pittsburgh Rock airport.

areas known to be safe for landing and from these sampled on average 1000 patches for each flight resulting in 21000 patches. We then insert random synthetic CAD obstacles with random positions and orientations in half of the patches. The point cloud dataset was also subject to the same augmentation procedure as the synthetic datasets.

C. Evaluation metrics

We use Receiver Operating Characteristic (ROC) curves to compare the different algorithms. For any given threshold on a detector with continuous output, we may commit two types of errors (Figure 7): erroneously labelling an unsafe subvolume as safe (a false positive), or failure to label a

safe subvolume as safe (a false negative). A ROC curve is generated by varying this threshold and evaluating the false positive rate (FPR) and true positive rate (TPR). An ideal algorithm would always have TPR equal to 1, and random chance would have TPR=FPR.

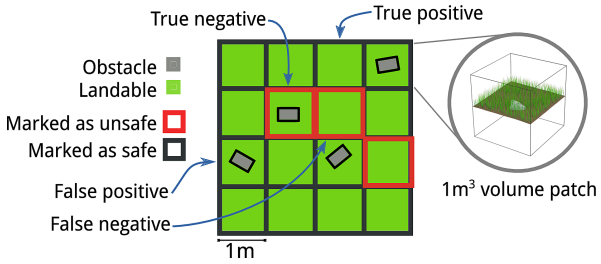

Fig. 7. Possible outcomes for landing zone safety prediction. The gray squares represent obstacles.

Note that we are implicitly assuming each volume has a .5 prior probability of being safe; this can be easily changed to incorporate prior knowledge by scaling the output appropriately.

D. Baselines

Our first baseline is based on the residuals of a robust plane fit, as in [2] and similar geometric methods. The sum of the residuals (with each clipped to $0.1\,\mathrm{m}$ for robustness) was used as the continuous output in the ROC curve.

The second baseline is a Random Forest classifier [28], using the implementation of [29]. It is trained with the same raw volumetric data as our networks. We used 20 trees with no maximum depth. Random forests are known to be robust and usually effective classifiers that work well with high-dimensional data. This baseline has no built-in invariance to spatial shifts.

The third baseline is a system built with various well-known features from the literature on point cloud classification. These include three spectral shape features [30], three directional features and three height-related features [8], shape features from [9] and range variance features from [12]. We calculate each feature on a per-voxel basis, with $3 \times 3 \times 3$ voxel spatial smoothing for the spectral features. We carefully tuned these features to work well in this data, as they were part of our first approach to solve this problem.

We construct a K-means codebook with 512 words from $1/4$ of the training data and represent each volume with a softly-quantized Bag of Words (BoW) [31]. Finally, we classify the BoW with a random forest classifier trained using the same parameters as before. This approach is similar to approaches that until recently were considered state-of-the-art for various tasks in computer vision. While the BoW ignores any spatial structure in the data, the features encode some local spatial context (e.g. height relative to the ground).

E. Results

1) CNN hyperparameters: The first synthetic dataset was reserved to choose the hyperparameters of our CNN. To this effect we performed random hyperparameters search using Hyperopt [32]. We vary the number of convolutional layers $(1, 2)$, filter shapes $(f_h^1, f_v^1, f_h^2, f_v^2) \in \{3, 5, 7\}^4$, number of hidden nodes $N_h \in (2^7, \cdots, 2^{12})$, number of filters $(F_1, F_2) \in \{32, 64, 128\}^2$ and pooling factors $(P_1, P_2) \in \{1, 2, 4\}^2$. Training is limited to 6 epochs with a minibatch size of 20. We use de facto standard learning rate (0.01), momentum (0.9) and L_2 regularization (0.0005) parameters for learning, with half of the data for training and half for validation. After three days and 400 different hyperparameter evaluations, we found many of the best networks had virtually identical performance. We chose a representative single-convolution layer net and a two convolution-layer net for further evaluation. The first net uses filters $f_h^1 = f_v^1 = 7$, $F_1 = 64$, $P_1 = 4$ and $N_h = 512$. The second uses filters $f_h^1 = f_v^1 = 7$, $f_h^2 = f_v^2 = 5$, $F_1 = 32$, $F_2 = 64$ and $N_h = 512$. In shorthand, we use `C7-F64-P4-D512` for the first net and `C7-F32-P2-C5-F64-P2-D512` for the second.

TABLE II
AREA UNDER CURVE (AUC) FOR OUR EXPERIMENTS.

Method	AUC Synth2	AUC Hybrid
Feature BoW 512 RF	0.931	0.66
CNN C7-F64-P4-D512	0.97	0.93
CNN C7-F32-P2-C5-F64-P2-D512	0.97	0.95
Plane Residuals	0.51	0.50
Raw volumetric RF	0.80	0.73

2) Results on second synthetic dataset: We trained each model on half of the (shuffled) data and evaluated performance on the second half. To avoid overfitting there are no obstacle models in common between the training and testing dataset. ROC results are shown in Figure 9 and Table II.

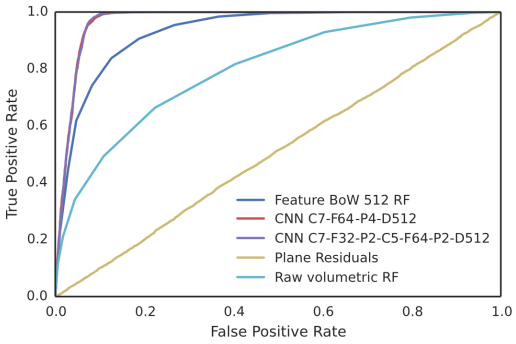

Fig. 8. ROC curve on the second synthetic dataset.

The two CNN approaches take the lead, and perform almost indistinguishably. This suggests they are learning similar hypotheses despite their different architecture, or that they are reaching some limit related to the dataset. The plane residuals perform barely above chance. This is due to the fact that by construction, our clouds are relatively dense and always have at least some clutter.

3) Results on semi-synthetic dataset: We use the same protocol on the semi-synthetic dataset. Results are shown in Figure 9 and Table II.

In this dataset the performance drops considerably. This

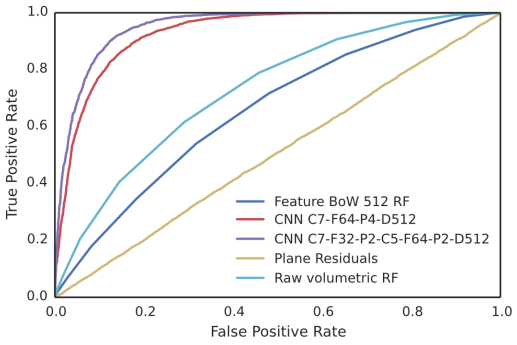

Fig. 9. ROC curve on the semi-synthetic dataset.

is not surprising considering that the semi-synthetic dataset is considerably noisier, and both the ground and vegetation have a more heterogeneous appearance. In this case `C7-F32-P2-C5-F64-P2-D512` shows a superior performance to `C7-F32-P4-D512`, suggesting the extra layer is beneficial in this scenario. It is interesting to see that in this case the raw volumetric random forest overtakes the BoW approach. This indicates the spatial information being discarded by the BoW is discriminative. The CNN, with an intermediate degree of spatial invariance, achieves the best results in both cases.

4) Timing: Depending on the parameters of the network, training for 6 epochs takes between two to six hours on our Core 2 DUO equipped with a 3GB GTX580 GPU. On the other hand, labeling a $1\,\mathrm{m}^3$ patch takes less than $5\,\mathrm{ms}$, and ray tracing (which is done on the CPU) to compute hit/passes and density on 3000 points takes less than $1\,\mathrm{ms}$ per 1000 points for a $400 \times 400 \times 40$ voxel grid. While not a fair comparison, as it runs on the CPU, the BoW algorithm by itself takes around $200\,\mathrm{ms}$ per volume. Aroud half of the time is spent performing feature extraction, and the other half is spent performing quantization.

5) Qualitative results: For a video of our method running in real time, see the accompanying video submission (Figure 10).

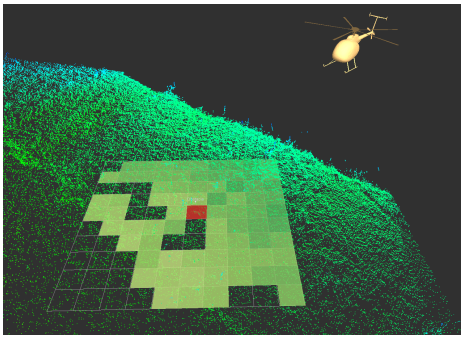

Fig. 10. Our method evaluating a $10\,\mathrm{m} \times 10\,\mathrm{m}$ region, from the accompanying video submission.

Some representative successes and failures from the semi-synthetic dataset are shown in Figure 11. Our method sometimes results in false negatives when the vegetation is dense enough to resemble rocks, or results in false positives when the obstacle is very small or mostly occluded. In order to overcome these problems we are considering the integration of other sensor modalities and active sensing.

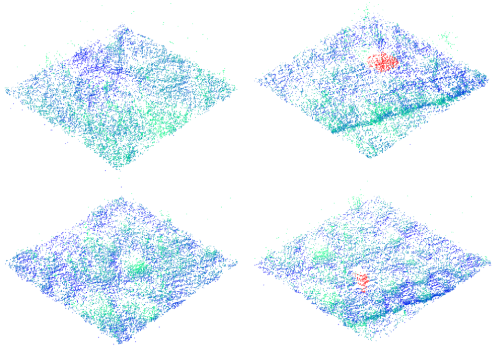

Fig. 11. Top: two correct assessments (true positive and true negative). Bottom: Two failures (false negative and false positive). Obstacles are shown in red. In the first failure case, there are several dense bushes which are similar to rocks. In the second, only a very small portion of the obstacle is visible.

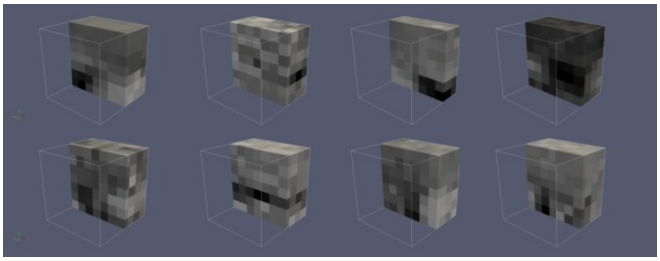

Fig. 12. Selected filters from `C7-F32-P4-D512` on the second synthetic (top row) and the hybrid (bottom row) datasets, where darker means denser.

6) What is the network learning?: A common way to visualize what CNNs learn is to examine the weights of the first convolutional layer, which have the same spatial structure as the input data. Figure 12 shows four $7 \times 7 \times 7$ filters of the single-convolutional layer in each of the datasets for `C7-F32-P4-D512`. The filters seem to have features corresponding to corners, blobs and plane-like structures.

Understanding the higher levels of the network is less straightforward. Simonyan et al [33] proposed a techique to hallucinate the "ideal" input for each category predicted by the network. We apply this method on the network trained on the first synthetic dataset (Figure 13), which only has simple box-like obstacles. We observe the ideal unsafe volume has multiple box-like sets of planes, whereas the ideal safe volume has visible ground and free space.

V. CONCLUSIONS

In this paper we have proposed a system for efficient and reliable detection of safe landing zones covered in vegetation. The key component of our system is a novel volumetric convolutional neural network system operating on density grid maps extracted from LiDAR range data.

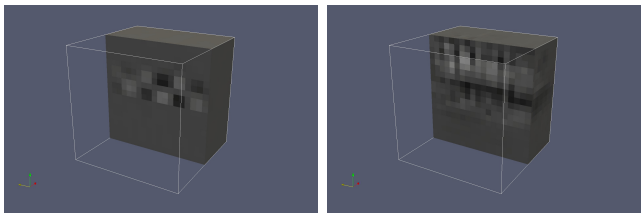

Fig. 13. Cross-sections of hallucinated unsafe and safe volumes from the first synthetic dataset.

Our experiments with synthetic and real data show that our system outperforms various baselines, including a system incorporating many well-known hand-engineered features, despite being trained directly on raw data. This mirrors similar results in computer vision [15]. The performance of our system suggest this is a viable option to increase the ability of unmanned rotorcraft to operate in a wider range of environments.

In the immediate future, we plan to perform more extensive evaluations, incorporating more 3D models and more real data and data with real obstacles. We are also considering the incorporation of additional information beyond range, such as LiDAR intensity, full waveform data or data from other sensing modalities such as camera or radar.

In the longer term we are interested in using the volumetric CNN framework for other tasks, such as semantic point cloud labelling. We would also like to close the sensing-planning loop by using the uncertainty reported by our system to actively guide sensing.

ACKNOWLEDGMENT

This research was sponsored under a fellowship by United Technologies Research Center. We would like to acknowledge the use of datasets collected under the ONR AACUS program (contract N00014-12-C-0671). We would also like to thank Near Earth Autonomy for their assistance with data collection.

REFERENCES

[1] M. Whalley, M. Takahashi, P. Tsenkov, G. Schulein, and C. Goerzen, "Field-testing of a helicopter UAV obstacle field navigation and landing system," in *AHS 65th Annual Forum*, 2009.

[2] S. Scherer, L. Chamberlain, and S. Singh, "Autonomous landing at unprepared sites by a full-scale helicopter," *RAS*, vol. 60, no. 12, pp. 1545–1562, Dec. 2012.

[3] *Helipad Marking And Lighting (STANAG 3619)*, 4th ed., 2007.

[4] A. Johnson, J. Collier, and A. Wolf, "Lidar-based hazard avoidance for safe landing on mars," in *AAS/AIAA Space Flight Mechanics Meeting*, 2001, pp. 1091–1099.

[5] M. Warren, L. Mejias, X. Yang, B. Arain, and F. Gonzalez, "Enabling Aircraft Emergency Landings using Active Visual Site Detection," in *FSR*, 2013.

[6] X. Meng, N. Currit, and K. Zhao, "Ground Filtering Algorithms for Airborne LiDAR Data: A Review of Critical Issues," *Remote Sensing*, vol. 2, no. 3, pp. 833–860, 2010.

[7] R. Hadsell, J. A. Bagnell, D. Huber, and M. Hebert, "Non-Stationary Space-Carving Kernels for Accurate Rough Terrain Estimation," in *RSS*, 2008.

[8] D. Munoz, N. Vandapel, and M. Hebert, "Onboard contextual classification of 3-D point clouds with learned high-order markov random fields," in *ICRA*, 2009.

[9] R. Shapovalov, "Non-associative Markov networks for 3D point cloud classification," *PCV*, 2010.

[10] B. Douillard, J. Underwood, V. Vlaskine, A. Quadros, and S. Singh, "A pipeline for the segmentation and classification of 3D point clouds," in *ISER*, 2010.

[11] C. Wellington, A. Courville, and A. Stentz, "A Generative Model of Terrain for Autonomous Navigation in Vegetation," *IJRR*, vol. 25, no. 12, pp. 1287–1304, 2006.

[12] J. Macedo, R. Manduchi, and L. Matthies, "Ladar-based discrimination of grass from obstacles for autonomous navigation," in *ISER*, 2001.

[13] A. Castano and L. Matthies, "Foliage discrimination using a rotating ladar," *Robotics and Automation*, vol. 1, pp. 1–6, 2003.

[14] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, "Handwritten digit recognition with a back-propagation network," in *NIPS*, 1990, pp. 396–404.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, pp. 1929–1958, 2014.

[16] R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng, "Convolutional-Recursive Deep Learning for 3D Object Classification," in *NIPS*, 2012.

[17] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," in *RSS*, 2013.

[18] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *TPAMI*, vol. 35, no. 1, pp. 221–231, 2013.

[19] G. Flitton, T. P. Breckon, and N. Megherbi, "A 3D extension to cortex like mechanisms for 3D object class recognition," in *CVPR*, 2012, pp. 3634–3641.

[20] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3D scene labeling," in *ICRA*, 2014.

[21] S. Choudhury, S. Arora, and S. Scherer, "The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety," in *AHS 70th Annual Forum*, 2014.

[22] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *ICRA*, 1985.

[23] A. Kelly, A. Stentz, and O. Amidi, "Toward reliable off road autonomous vehicles operating in challenging environments," *IJRR*, vol. 25, no. 5-6, pp. 449–483, 2006.

[24] G. D. Tipaldi and K. O. Arras, "FLIRT - interest regions for 2D range data," in *ICRA*, 2010.

[25] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Eurographics '87*, Aug. 1987, pp. 3–10.

[26] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *SciPy*, 2010.

[27] J.-E. Deschaud, D. Prasser, M. F. Dias, B. Browning, and P. Rander, "Automatic data driven vegetation modeling for lidar simulation," in *ICRA*, 2012, pp. 5030–5036.

[28] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[30] J.-F. Lalonde, N. Vandapel, D. F. Huber, and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," *JFR*, vol. 23, no. 10, pp. 839–861, Oct. 2006.

[31] A. Coates and A. Y. Ng, "Learning feature representations with K-means," in *Neural Networks: Tricks of the Trade - Second Edition*, 2012, pp. 561–580.

[32] "Making a science of model search," in *ICML*, vol. 28, 2013.

[33] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *CoRR*, 2013.