Semi-automated Cleaning of Laser Scanning Campaigns with Machine Learning

PATRICK MARAIS, University of Cape Town, South Africa
MATTEO DELLEPIANE, PAOLO CIGNONI, and ROBERTO SCOPIGNO, CNR-ISTI, Italy

Terrestrial laser scanning campaigns provide an important means to document the 3D structure of historical sites. Unfortunately, the process of converting the 3D point clouds acquired by the laser scanner into a coherent and accurate 3D model has many stages and is not generally automated. In particular, the initial cleaning stage of the pipeline—in which undesired scene points are deleted—remains largely manual and is usually labour intensive. In this article, we introduce a semi-automated cleaning approach that incrementally trains a random forest (RF) classifier on an initial keep/discard point labelling generated by the user when cleaning the first scan(s). The classifier is then used to predict the labelling of the next scan in the sequence. Before this classification is presented to the user, a denoising post-process, based on the 2D range map representation of the laser scan, is applied. This significantly reduces small isolated point clusters that the user would otherwise have to fix. The user then selects the remaining incorrectly labelled points and these are weighted, based on a confidence estimate, and fed back into the classifier to retrain it for the next scan. Our experiments, across 8 scanning campaigns, show that when the scan campaign is coherent, i.e., it does not contain widely disparate or contradictory data, the classifier yields a keep/discard labelling that typically ranges between 95% and 99%. This is somewhat surprising, given that the data in each class can represent many object types, such as a tree, person, wall, and so on, and that no further effort beyond the point labeling of keep/discard is required of the user. We conducted an informal timing experiment over a 15-scan campaign, which compared the processing time required by our software, without user interaction (point label correction) time, against the time taken by an expert user to completely clean all scans. The expert user required 95mins to complete all cleaning. The average time required by the expert to clean a single scan was 6.3mins. Even with current unoptimized code, our system was able to generate keep/discard labels for all scans, with 98% (average) accuracy, in 75mins. This leaves as much as 20mins for the user input required to relabel the 2% of mispredicted points across the set of scans before the full system time would match the expert's cleaning time.

CCS Concepts: • **Computing methodologies → Machine learning**; • **Applied computing → Archaeology**;

Additional Key Words and Phrases: Heritage data, laser scan, point clouds, full dome scan, cleaning, machine learning

ACM Reference format:
Patrick Marais, Matteo Dellepiane, Paolo Cignoni, and Roberto Scopigno. 2019. Semi-automated Cleaning of Laser Scanning Campaigns with Machine Learning. *ACM J. Comput. Cult. Herit.* 12, 3, Article 16 (June 2019), 29 pages.
https://doi.org/10.1145/3292027

This work was partially carried out in the framework of the SCIADRO project, co-funded by the Tuscany Region (Italy) under the Regional Implementation Programme for Underutilized Areas Fund (PAR FAS 2007-2013) and the Research Facilitation Fund (FAR) of the Ministry of Education, University and Research (MIUR). Patrick Marais also received funding from the National Research Foundation (NRF) in South Africa to support this research collaboration.
Authors' addresses: P. Marais, Dept. of Computer Science, University of Cape Town, Rondebosch 7701, South Africa; email: patrick@cs.uct.ac.za; M. Dellepiane, P. Cignoni, and R. Scopigno, Visual Computing Group, CNR-ISTI, Via G. Moruzzi 1, 56124 Pisa, Italy; emails: {matteo.dellepiane, paolo.cignoni, roberto.scopigno}@isti.cnr.it.

1556-4673/2019/06-ART16 $15.00
https://doi.org/10.1145/3292027

1 INTRODUCTION

The use of laser scanning equipment to capture 3D surface data from cultural heritage sites has become fairly routine. This data is typically acquired using a laser scanner that produces a full-dome scan of the structures surrounding the device. Due to the regular grid scanning pattern that the laser scanner employs, the collected range data can be represented as a panoramic depth map in which each pixel encodes the distance along a ray from the scanner origin through that pixel to the intersected environmental surface point. This data can easily be turned into a collection of 3D point coordinates—often with associated colour—usually known as a *point cloud*.

Unfortunately the data that the scanner returns cannot immediately be used to construct a 3D model of the scanned buildings or environment. Unwanted points arise due to noise in the scanning process (for example, the laser may scatter off certain materials) as well as objects that are inadvertently captured. The latter category is particularly common, with people, animals, and objects such as telephone poles often superimposed on the background heritage structure. The process of manually removing these unwanted points is called *cleaning*, and for large complex single scans it may take up to two hours [24] by a skilled user. The cleaning process is exacerbated by the large number of scans that may be required to cover a site. A computerized system that could help to reduce overall cleaning time would be beneficial in such a work-flow. However, this cleaning step is difficult to generalize: the exact set of points to be removed often depends on the specifications of the campaign, making it difficult to fully automate the process.

This article introduces a semi-automated system and work-flow to help reduce the user workload of the cleaning process for individual laser scans. The system uses machine learning and exploits the grid structure of the range scans by applying a number of image processing algorithms to counter some of the noisiness in the classifier predictions. Motivated by the observations in Reference [20], we do not try to learn a general classifier that will work on any scene type, but rather incrementally train a classifier tailored to the characteristics of the current scanning campaign. We consider geometric information only—colour per point may be available, but this is not guaranteed for all cultural heritage scans, and poor quality image-to-model registration and differences arising from scene lighting can cause the colour information to be unreliable.

The approach outlined in this article is intended to be exploratory, probing the extent to which machine learning can be used in the ill-defined task of point-cloud cleaning. To support this approach, the evaluation considers a diverse set of cultural heritage scanning campaigns that clearly illustrate the challenges inherent in this application.

The framework is based on a random forest classifier that seeks to classify each point in a sequence of scans as either "keep" or "discard," starting from one or more manually cleaned scans. The classifier is trained by generating a set of geometric features for each point and then recording which points the user elects to discard as they clean the scan. The framework has been designed so as not to impose any additional workload to the standard cleaning process: the user starts to clean the scans as in a standard system, and the classifier incrementally learns what has to be discarded in the subsequent scans. In such an approach, there is no explicit, usually onerous, training step where labels and classes are assigned to scene parts. To the best of our knowledge, this is the first attempt to use machine learning in the context of *point-cloud cleaning* using only the two intuitive class labels noted above.

The classifier is retrained as cleaning proceeds from one scan to the next, allowing it to become more accurate as more data are generated throughout the cleaning phase. To accommodate additional data, we have developed a re-weighting scheme that adds misclassified points to the initial training data as each scan is cleaned. The re-weighting employs the natural sample weighting mechanism of random forests and allows us to nudge the classifier towards better predicting such points. The candidate points are easily identified, since during cleaning the user will have to select/deselect points that were mislabelled, and these points (and their features) are stored until the classifier needs them.

As an additional aid to help cleaning, we have also developed an image-based post-process that uses the classification confidence to guide a point-cloud denoising process. The aim of this phase is to avoid a "messy" classification, which results in many isolated error components scattered through the point cloud, whilst matching or exceeding the original predictor classification accuracy. Fewer error components/points should allow the user to spend less time moving about the point cloud locating and correcting classification errors. To accomplish this, we return to the gridded point-cloud representation and apply a sequence of denoising and depth-aware region growing on the 2D range image that promotes coherence of the classification. The region growing is seeded with regions for which the classification confidence is very high; this helps to reduce propagating low-confidence point classifications through the cloud.

These techniques support a work-flow that fits naturally within the process of point-cloud cleaning and which is particularly well suited to the ill-defined requirements of cleaning a large number of related scans.

The remainder of this article is laid out as follows: Section 2 introduces background concepts and discusses related work. Section 3 motivates the cleaning framework. Results are presented and discussed in Section 4. Section 5 concludes the article and presents some areas for future work and improvements.

2 BACKGROUND AND RELATED WORK

Acquiring 3D structure—in the form of a "point cloud"—through laser scanning is well-established technology and comes in a multitude of forms. Acquisitions can be aerial (for example, a UAV or aircraft), mobile (such as Google Maps' urban scanning vehicles), or terrestrial [15].

Our work is intended to reduce the workload in the point processing pipeline relevant to a set of terrestrial laser scans acquired from multiple, full-dome (360 degrees about the scanner origin) view points [24]. The back-end of the pipeline addresses issues such as meshing and modelling; our concern is with the preparation for this stage: cleaning. The purpose of the cleaning stage is to remove unwanted points from a collection of point clouds. These are typically points arising from scanner acquisition noise/errors, including scattering effects, as well as valid points that are not wanted in the final model input. Typically, the latter category is very general, often depends on campaign specification, and includes people and animals, trees and foliage, vehicles, telephone poles and so on. Previous approaches for automatic cleaning mainly focus on only one category, such as vegetation [9, 17]. In general, cleaning is context-dependent; in some applications, we wish to include a structure such as column, in another we may consider it inappropriate and flag it for removal. This lack of consistency is a major factor that complicates the wish to learn some more general cleaning model through machine learning.

Point-cloud cleaning in cultural heritage (CH) preservation, the goal of this work, is typically more challenging than cleaning point clouds acquired in urban/industrial settings. CH scenes vary dramatically: encompassing relatively well-preserved and regular structures in urban/rural settings to overgrown crumbling ruins in a rain forest. There is little consistency across different scanning campaigns, but within a campaign, we expect that there is some measure of consistency in the organization of the points in 3D. This is the central premise of our approach. In other application fields, such as, for example, industrial premises or landslide monitoring, there is generally more consistency in the types of data to be kept and removed. In these cases, ad hoc solutions, such as pipe extraction or vegetation removal, can lead to good results. Nevertheless, the applicability of such methods to the CH context is limited; hence, the aim of this work to deal with data sets where previous information on which points to discard is low.

Previous work using machine learning in this context, for example Reference [20], has focused on learning a label for each point that splits the point cloud into classes such as trees, ground cover, walls, and so on. This approach is influenced heavily by the traditional machine-learning approaches used to segment shapes/structures from 2D and 3D data sets [21, 22, 26]. For most image/scene segmentation tasks, the goal is to recover pixels/points that belong to a small subset of object classes. This requires labelled training data that contains examples of each possible object class and all its likely variations.

Unfortunately, given the enormous variety of background and foreground object types encountered during general cleaning, acquiring an extensive set of truly representative object labels is implausible. Instead, we retain the existing cleaning work-flow and augment it with a more focused machine-learning system. More specifically, we note that the problem is not a general object labelling problem, but rather a binary labelling problem in which we seek to label each point as either *keep* or *discard*, with no care about which objects those points belong to. This is a very different view of the data from the more traditional object-centric approaches. Furthermore, because the scheme is intended to augment manual cleaning, compute time is important and the machine learning approach used cannot take hours to learn/classify a scan in the campaign.

Machine learning (ML) is often portrayed as a black box that can solve any problem, given enough training data. Unfortunately, this is a simplification, since the type and quantity of data, the kinds of algorithms utilized, and the context all play a significant role in determining whether a machine-learning approach can be used to automate (or semi-automate) some process. Machine learning generally requires labelled data. For some problems this is readily available, or it can be harvested from the Internet. In many cases, however, data is not readily available, so methods that can learn from smaller data sets are thus preferable. Deep-learning methods, such as Convolutional Neural Networks (CNNs), attempt to discover abstract representations of the data that can be used to better solve classification or regression problems [19]. While they have been successfully used in many image/voxel-based classification problems [14], they require a great deal of data. While it is possible to use unlabelled data in some cases, and have the ML method discover correlations/structure within the data, the volume of data required does not diminish and training times remain high. This is problematic for our application domain, where there is no sensible repository to learn from and where the possible types of cleaning targets are effectively unlimited. Recent work on deep learning for point clouds [23] attempts to directly address semantic point classification of variable density structures *without* enforcing a regular re-sampling of the input, as required by a CNN. Unfortunately, this approach still requires a large amount of training data and is specifically designed to learn and classify a pre-determined set of man-made objects.

In addition to neural network architectures, *graphical models* [3] have also become very popular as a means of learning local relationships between pixels/points. The simplest graphical models are based on the Markov Random Field (MRF), which describes relationships between a point and its immediate neighbours. Graph-based image segmentation has been around for many years, mostly based on MRFs with carefully modelled potentials that exploit prior information about image edges, neighbourhood intensity consistency, and so on [4, 31]. A more sophisticated Conditional Random Field (CRF) can be used to describe long-distance relationships, which is useful for many image segmentation tasks [3]. The DeepLab [6] system is an example of an approach that combines a CRF with multiple CNN layers to generate semantic labels for each image pixel. Again, however, the inclusion of deep learning makes this an expensive approach.

Recently there has been much interest in semantic labelling of RGB-D (colour + depth) images, which are often acquired from short-range scanning systems such as Microsoft Kinect [12]. Several graphical model approaches [13, 16, 18, 30] use CRFs to label RGB-D images, which are essentially point clouds. A more powerful variation of the MRF, the associative MRF, has also been used for a similar purpose [21, 27]. These techniques seek to solve a similar problem to our own and would thus appear to be viable classification solutions. However, they require a pre-defined set of class/category labels, and they generally try to decompose the scene points into "consistent" groups of pixels/points. While we have only two class labels, the underlying classes are so broad that they cannot be encapsulated in a learned model that applies to *all* possible heritage environments. Furthermore, simple geometric or depth image consistency is not enough to reliably map points to the keep and discard classes.

The above observations suggest a more conservative approach: rather than attempting to learn a general model, learn a contextually relevant model from the current campaign you are processing. We have access to an expanding set of labelled training data, generated by the cleaner as they work, that can be used to refine the classifier. The core requirements of our approach are (a) the user should not have to do substantially more work, and (b) the time required to run the machine learning should not be longer than the time to manually clean the scans.

We have noted that deep-learning schemes are problematic due to the volume of data required as well as the compute power needed for fast training. In our approach, we require that the user fully clean the first scan only, as they would normally do. This simple binary labelling is then used to bootstrap the classifier for the following scans in the campaign. Later user input should be limited to correcting the (hopefully small) mis-classifications produced by the ML component.

A widely used ML scheme that performs well in many contexts and does not require a very large amount of training data is the random forest (RF) [5]. This creates a set of decision trees, built from permuted versions of the input training data, and aggregates their predictions to arrive at a single outcome. Random forest can be parallelised, has been widely used in many domains, and has been a popular method for general point-cloud classifications for some years [29]. A great deal of work has been done on modifying the core RF technique to improve performance under different assumptions. One variant we considered is the *on-line random forest* (ORF) [25]. This assumes the underlying distribution the classifier is modelling changes over time, as new data arrives, and tests only a single random feature value per node during tree construction. The RF model only updates itself when enough new data has been seen, and the update does not require rebuilding the entire tree. While incremental retraining is computationally desirable, our tests showed that the method requires many more deep trees to approximate the probability distribution to the same accuracy level as the base ("offline") RF method and was generally less accurate. A variant of this approach, *streaming random forests*, was introduced in the Semantic Paint system [28]. While the performance of this method is significantly better than ORF, at least for the problem examined in this article, it is not clear how performance compares to an offline random forest.

A scheme that attempts to combine deep learning with the lower resource requirements of RF has recently been developed [33]. While this scheme does offer better training times than many other deep schemes, its principal benefit is that it requires much less data to obtain good results. Unfortunately, since it uses a collection of random forest classifiers, the training times are still significantly slower than our system, which uses only a single random forest.

Finally, while voxel-based CNNs are now providing good results [10] for point classification, the caveats we noted earlier still hold, and we do not consider these a good match for our specific problem. For these reasons, we decided to base our scheme on a random forest, more specifically the approach proposed by Reference [11], which utilises a fast multi-resolution feature vector.

3 POINT-CLOUD CLEANING FRAMEWORK

Cultural heritage preservation deals with structures that are often very old and in a state of decay: parts are missing or crumbling and eroded and there is often a great deal of vegetation cover that occludes structures. The variety of structure types is also high: ranging from well-maintained modern-era buildings to crumbling ruins that are barely recognizable. Using machine learning to classify such point data thus needs careful consideration. Attempting to learn a general model for such data would not be practical; instead, we have opted for a model that is tailored to a given scanning campaign. One might be tempted to choose a small set of scans and learn a model to apply to the rest of the scan set. Unfortunately, this is problematic, since the set we choose may not cater for the variability in the campaign, which may still be substantial, and the choice of the training subset will have a dramatic effect on later performance. Instead, we propose that the learning process is ongoing and incorporates new information from misclassified points as new scans are processed and cleaned.

The point training/prediction procedure of the framework is presented in Algorithm 1. The main components of the system are incremental learning (which is strongly affected by data balancing and feature calculation choices) and post-processing the classification obtained by the point classifier. These components are presented in detail in Sections 3.1 and 3.2.

3.1 Incremental Learning

As noted above, rather than choosing a small set of campaign scans and training a fixed classifier, we allow for an initial training phase followed by periodic incremental training. In this context, *incremental* refers to the addition of training data to the starting training data set, as each scan is processed. While the choice of the starting training example(s) will initially affect the accuracy of the classifier, even with a poor starting choice the addition of more training samples as the cleaning proceeds will allow the classifier to generalize.

After running the predict step to assign a label to each point, the user cleans the data by labeling points as either “discard” or “keep.” We can compare the predicted class for a point with the label assigned by the user (using, for example, a paintbrush or lasso tool to make a selection). We thus know which points have not been correctly predicted by the classifier and can use this information to refine the classifier. To do this, we use the confidence estimate attached to each *incorrect* prediction to weight the sample and resubmit it to the classifier on a retraining pass. More specifically, a misclassified point P_i is assigned a weight:

$$w(P_i) = \max(1, \mathrm{int}(\kappa\ C(P_i)), \tag{1}$$

where $\kappa \geq 1$ is a scalar factor and $C(P_i)$ is the confidence assigned to the (incorrect) classification by the classifier. The correct label is attached to this point, and along with the newly calculated sample weight and the point feature vector, the sample is added to the set of samples to be included on the next retraining cycle. The weighting function ensures that points that were misclassified with high confidence will receive the correct label and a *high* sample weight during retraining: this should nudge the classifier away from repeating such a mistake. Other weighting functions could be adopted, but it is worth noting that a simple linear function is sufficient for our purposes.

The retraining criteria we have experimented with are (1) a specified number of scans have been processed or (2) the prediction accuracy has fallen below some specified level.

3.1.1 Data Balancing. Classifiers can be very badly affected by the number of training examples available for each class [1]. Schemes to balance data often use either downsampling (choosing a subset of input samples for each class) or upsampling (through repetition or interpolation). Furthermore, many classifiers allow one to attach sample or class weights that bias the classifier towards a given sample/class. While class and sample weights are supported by random forests, their uses as a balancing mechanism is ill-advised. Each tree in a random forest is built using a “bootstrap sample”: a new training sample is constructed by randomly re-sampling the original input set (feature vectors and labels) with replacement. If the minority class has few samples, even if these are associated with very large class or sample weights, then the random resampling may not include *any* instances of that class and the tree will not be able to reliably predict this class. Adding more trees can help, but for very small classes, a great many trees would be required, the results are likely to remain poor, and the training time increases linearly with the number of trees.

To balance the data, we choose to generate a new initial training set that is a resampled version of the original training data ensuring that equal numbers of each class are available. For the initial training phase, the majority class is found and the resampling target is set to 20% of this to reduce initial training time. The larger class is randomly down-sampled while the minority class consists of the original minority data plus an additional number of randomly repeated samples, such that the size of the two class training sets coincides. This was done primarily to reduce training time on the initial pass: one could simply up-sample the minority class to the majority class size, but that will increase classifier training times significantly. Under no circumstances is interpolation used: only original feature vectors and labels are used. Note that sample weighting is still used—but only to emphasize the role of misclassified samples; it is not used to balance data but serves a different role.

To balance the classifier as we proceed with incremental training, we do up-sample the minority label class to the majority. In this case, we expect that the classifier will not generate a great many misclassifications, particularly later in the scan processing, and thus the computational burden will be less. Because we always add the same number of samples per class at each phase, the entire data set remains balanced. We do not use samples from the previously accumulated training data when upsampling the minority class: if the minority class in a

ALGORITHM 1: Point label prediction

```
// Input scan set, S
$S \leftarrow \{s_1, \dots, s_n\}$
// Choose a set of scans for manual cleaning
$M \leftarrow$ UserSelectScans($S$)
// Compile training data: feature vectors (F), labels ($\ell$), weights (=1)
$TD \leftarrow \emptyset$
for each point p in $M$ do
    $TD \leftarrow TD \cup (\mathbf{F}(\mathbf{p}), \ell(\mathbf{p}), 1)$
end for
$TD \leftarrow$ BalanceTrainingData($TD$)
$C \leftarrow$ TrainClassifier($TD$)
SaveClassifier($C$)
SaveTrainingData($TD$)
$TP \leftarrow \emptyset$
for each scan $s$ in $S \setminus M$ do
    // run the classifier to get per point prediction labels
    $L \leftarrow$ PredictPointLabels($s, C$)
    // user input: select incorrectly predicted points
    $Q \leftarrow$ UserInput($s, L$)
    // compute sample weights $w(\mathbf{q})$ and add training data to pool, $TP$
    for each point q in $Q$ do
        $TP \leftarrow TP \cup (\mathbf{F}(\mathbf{q}), \ell(\mathbf{q}), w(\mathbf{q}))$
    end for
    if retrain criterion reached then
        $TP \leftarrow$ BalanceTrainingData($SP$)
        // retrain classifier with accumulated training data and weights
        $TD \leftarrow$LoadTrainingData()
        $TD \leftarrow TD \cup TP$
        $C \leftarrow$ TrainClassifier($TD$)
        // save new classifier and training data (feature vectors, labels, weights)
        SaveClassifier($C$)
        SaveTrainingData($TD$)
        $TP \leftarrow \emptyset$
        $TD \leftarrow \emptyset$
    end if
end for
```

new scan exhibits a previously unseen point distribution and there are few samples, then using samples from earlier scans to up-sample the minority class will likely overwhelm this new information.

3.1.2 Density Mitigation and Feature Calculation. For a stationary full-dome laser scanner, the point density across a surface can vary dramatically, depending on the distance of the surface from the scanner origin as well as the orientation of the surface with respect to the scanner. To regularize the density, we follow the approach of Reference [20] and overlay the entire scan volume with a grid of cubic cells of fixed size. The points in each occupied cell are then averaged to "homogenize" the point cloud: the large point cloud is reduced to a much smaller, more regularly sampled point cloud.[1] This cloud is then used for all subsequent processing; most importantly,

[1]One could also compute the medioid of the points in a cell, but this is computationally more expensive.

Table 1. Point Features Used by Classifier

Name	S / M	Definition	Notes
Height	S	$\mathbf{p}_z$	The z value returned by the scanner in the scanner coordinate frame
Distance	S	$\lvert\mathbf{p}\rvert$	Distance from scanner (depth)
Curvature 1	S	κ_1	Principal curvatures at $\mathbf{p}$
Curvature 2	S	κ_2	
Cylinder 1	M	$Z_{\text{max}} - Z_{\text{min}}$	Cylindrical neighbourhood features
Cylinder 2	M	$\mathbf{p}_z - Z_{\text{min}}$	
Cylinder 3	M	$Z_{\text{max}} - \mathbf{p}_z$	
Anisotropy	M	$\frac{\lambda_1-\lambda_3}{\lambda_1}$	Features computed from eigen-values/vectors of the structure tensor
Planarity	M	$\frac{\lambda_2-\lambda_3}{\lambda_1}$	
Sphericity	M	$\frac{\lambda_3}{\lambda_1}$	
Omnivariance	M	$\sqrt[3]{\lambda_1\lambda_2\lambda_3}$	
Linearity	M	$\frac{\lambda_1-\lambda_2}{\lambda_1}$	
Surface variation	M	$\frac{\lambda_3}{\lambda_1+\lambda_2+\lambda_3}$	
Eigen entropy	M	$-\sum_{i=1}^{3}\lambda_i \log \lambda_i$	
Verticality	M	$1-(0,0,1) \bullet \mathbf{e}_3$	
Density	M	$(k+1)/\frac{4}{3}\pi r_k^3$	for k nearest neighbours

the classifier learns and classifies this down-sampled cloud. This speeds up the whole pipeline significantly, at the cost of some loss of prediction accuracy. To propagate the labels to the full resolution variable-density cloud, a map is stored per cell that records which input points were originally down-sampled to that cell. To implement the downsampling, we insert all the points into an octree and augment the tree with the maps. A simple traversal of all leaf nodes allows us to aggregate all the points in a cell and set up the appropriate maps. It is worth noting that octree cells close to the laser scanner origin will contain many more points, so misclassifications for such points at the "cell" level will be much more noticeable.

The features we use consider only geometric (point) data, since these are always available and, for much CH data, more reliable than colour features. We also conducted a *variable importance analysis* for the RF feature variable set that showed that the laser return intensity was a very weak feature; we thus decided to discard it. Neighbourhood features are computed using either a fixed radius, R, or the k nearest neighbours. Both of these parameters are fixed for all experiments—see the testing section for more information. A feature vector is computed for each point in the down-sampled cloud, as shown in Table 1.

Some features are computed at the highest resolution level (denoted as "S" in the table) and others over multiple resolution levels ("M"). In particular, the principal curvatures, κ_1 and κ_2, are computed over a single spherical neighbourhood per point, at the highest resolution level, for a fixed R; computing this feature over multiple scales would push up feature computation time quite significantly. Furthermore, some of this information (e.g., planarity) is encapsulated in the structure tensor features, which are multi-scale. The features based on a cylindrical neighbourhood are computed for a given 3D point $\mathbf{p}$ over a vertical cylinder with radius R. For a given cylindrical neighbourhood, Z_{max} and Z_{min} represent, respectively, the largest and smallest z coordinates for all points $\mathbf{p}$ in the cylinder. To compute the structure tensor features, the k nearest neighbours to $\mathbf{p}$ are found and the eigen values, $\{\lambda_i\}$, and corresponding eigen vectors, $\{\mathbf{e}_i\}$, of the structure tensor are computed [29]. The eigen values/vectors are arranged in decreasing order, so $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$. For the density feature, r_k is the smallest radius that encloses all k neighbours.

For multi-resolution (MR) features, we utilise 6 levels [29]. More specifically, for each MR feature level, we sub-sample the point cloud by doubling the aggregation cell size and compute the features using the points from this sparser sampling. The original aggregation level is considered to be the first level of the combined MR feature vector. To compute an individual MR feature, we take a point **p** in the dense level sampling and extract its neighbours, using the appropriate neighbourhood, in the dense cloud as well as each of the the coarser MR clouds. The feature vector for each neighbourhood is then calculated, and all these feature vectors are concatenated to form the final MR feature vector for the point. The single resolution features are appended to this to generate the final composite point feature. Note that for features computed using a k nearest neighbourhood, for fixed k, the coarsening of the point clouds introduces a natural smoothing effect in the composite MR features[11].

The total number of features in our point feature vector, assuming 6 MR levels, is 76. These values are all represented as single precision floating point values.

We could add other geometric features, as well as image-based features, but our intention was not to exhaustively explore the best feature set for cleaning, but rather to develop a framework in which additional features can be added as desired.

We do not normalize the feature vectors—this is not necessary for random forest [5].

3.2 Post-processing the Classification

The core stages of classification pipeline we have adopted are shown in Figure 1. Each 3D point is initially assigned a predicted label, based on the current random forest classifier. The representation then switches to a 2D depth map panorama with each pixel representing a 3D point, and low-confidence points are subject to further pixel-based processing that seeks to smooth the initial predictions by carefully interpolating high confidence labels.

The point classifier often produces a noisy labelling with incorrectly classified clusters of points surrounded by larger regions of correctly predicted points. This makes the cleaning process harder, since the user has to spend more time correcting these errors. In many cases, the point clusters correspond to regions where the classifier has lower confidence in its labelling, so we would like to force these regions to match the surrounding high confidence predictions. However, one can't simply assign low-confidence points to the dominant label in a neighbourhood, since this ignores depth discontinuities and there may be no local neighbours. Our overall goal is to reduce the number of small isolated components while not impacting too greatly the initial classification result.

We use the range image (2D) domain for this processing, since the neighbourhood relationship between points is often more clearly defined in this representation. For example, if one looks at a 3D point cloud (even when down-sampled) there are many regions where we have thin, irregular strips of surface samples that seem uncorrelated, but can be clearly seen as neighbours in the depth image.

The data we work with has additional characteristics that complicate the label propagation task:

- We do not have RGB-D data, but geometry (or D) only. This removes a large source of neighbourhood and region correlation that is exploited by popular MRF/CRF methods;
- Isolated points or point clusters are often not noise points, but valid data such as parts of wall separated by scanner acquisition problems around edges. It would thus be wrong to make the simplifying assumption that small structures such as these are noise.
- The depth maps we process generally have large depth changes across the image and steep depth gradients. This issue is not typically encountered when processing image data from short-range depth cameras like those used in Microsoft's Kinect, which are widely used in the classification literature.
- Pixels that are part of a single structure in 3D may be separated by gaps consisting of no-response samples produced by scanner artifacts. In this case requiring that adjacent pixels, or even pixels that are further away, be consistent in depth may not be enough to reconnect these structures. This problem is exacerbated

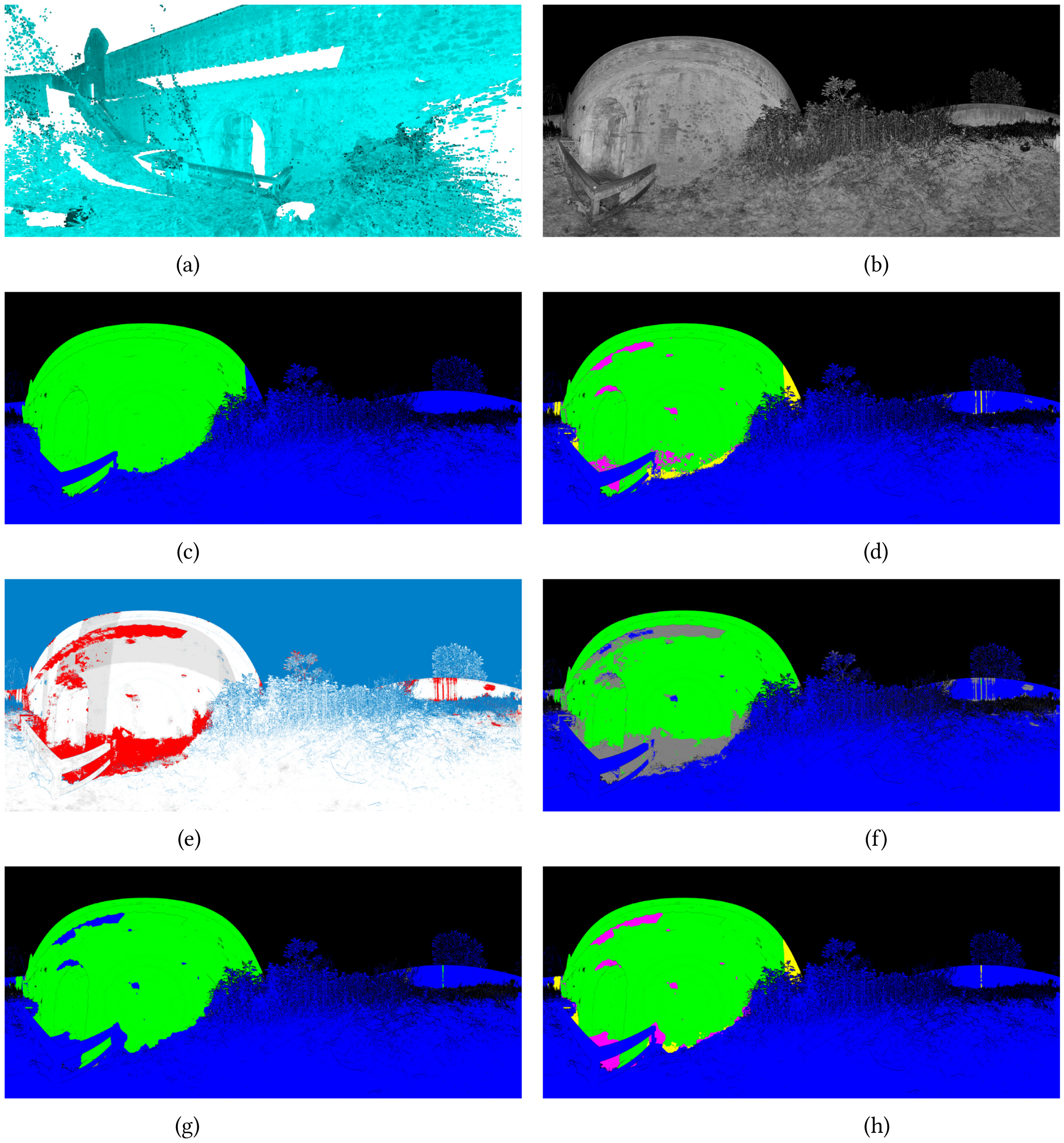

Fig. 1. **Point classification through range map.** (a) view of point cloud, close to scanner origin; (b) laser return intensity; (c) ground-truth labels—green is keep, blue is discard, other colours are classification errors; (d) initial classification with no post-processing; (e) confidence map—values below confidence threshold are red, no-return points are light blue, and the grey/white values represent points above the threshold; (f) grey values represent points for which values need to be filled (after morphological filtering); (g) the final depth aware filled image—any remaining grey points are filled by the graph fill stage; (h) the final post-process classification image, which is mapped 1-1 back to the points.

when there are strong depth gradients across the region, which mean that the depth change can be very large between adjacent samples.

The use of MRF/CRF that generally enforce neighbourhood consistency of pixels in this more complex scenario would require careful design of the potential terms, and it is not clear how this should be done, or whether this approach is ultimately a good one for our (depth only) data. Instead, we have developed a more heuristic label propagation approach that uses a series of processing phases that attempt to directly address these issues. We deal with the easier cases first and defer harder label assignment choices until later. The phases are: (1) Thresholding and morphological filtering; (2) Depth-aware region filling; and (3) Label assignment for isolated components.

The first phase selects all low-confidence depth samples, which are set as "unlabelled" and then removes small component clusters from these for later processing. This creates unclassified "holes"; however, these holes are not missing geometry or depth, and thus conventional techniques for depth/geometric hole-filling, such as [8, 32], are not considered.

In the second phase, good label information is incrementally propagated into unlabelled regions, in a way that respects depth continuity and is somewhat robust to high depth gradients.

The final phase deals with the more complex cases. First, we attempt to connect structures broken by no-response samples and those removed in phase 1, even in the presence of strong depth gradients. To do this, we start from components with reliable labels, either from the initial labelling or from label propagation, and then determine whether we can connect unlabelled components to these in a way that minimizes the depth discontinuity at any point along their boundaries. This assigns labels to all reasonably well-behaved unlabelled components. The remaining pathological cases are then assigned the closest label in the 2D image plane. These steps are explained in more details below.

We refer to the per-point (pixel) classification labels attached to the range image representation as the *classification image*.

Phase 1—Thresholding and morphological filtering: To filter out the low-confidence regions, we use a simple confidence threshold, and all pixels that fall below this threshold are set to *unclassified*. We then construct a binary image in which valid pixels are assigned 1 (foreground) and all other pixels are set to 0 (background). A morphological opening with a 3×3 disc is applied to this image, which removes small and thin foreground structures. All the foreground pixels that are removed (small clusters of labelled points mostly) are also set to unclassified in the classification image. All unlabelled pixels will be assigned a valid class label by the next two phases.

Phase 2—Depth-aware region filling: Next, we apply a depth-aware region growing operation to fill the holes created by phase 1. We perform a number of passes across the image buffer, processing *all* current pixels deemed to be boundary pixels, and attempt to label them. We visit each pixel in scan-line order, skipping pixels until we find an unclassified *boundary pixel*—an unclassified pixel with at least some neighbours that have a label type other than "unclassified" (labels discard, keep, and no-return). A $b \times b$ block of pixels, centred on the unclassified boundary point, is extracted and for all pixels except no-return pixels, we compute the depth dissimilarity:

$$S(i) = |D(i) - D(0)|$$

between the pixel i and the depth at the unclassified pixel, $D(0)$. We then sort these values in ascending order and compute the derivative (forward difference) across the 1D array. Note that the first dissimilarity array entry will always be 0 and will correspond to the boundary pixel we are processing. A large derivative will indicate that at least some neighbouring pixels are located across a strong depth discontinuity. We then inspect the derivative array and the labels attached to the corresponding samples. We use a small set of rules to determine how to assign a label based on the size and location of the derivative maximum. If the derivative maximum is weak or badly behaved, then we delay making the label assignment and leave this for a later pass, since more information may be added as we complete each image pass.

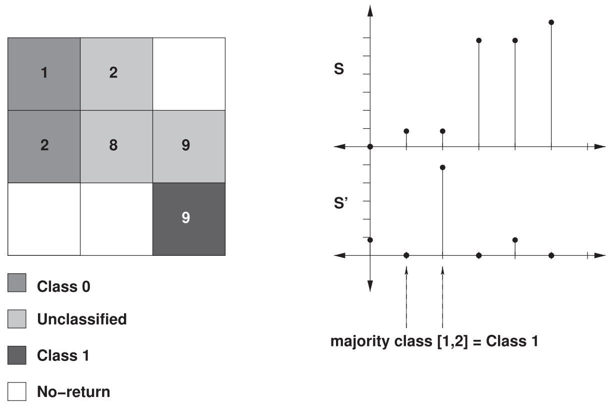

Fig. 2. **Example label assignment.** The center unlabelled pixel requires a label. The numbers in the pixel array are the depth values, which generate the dissimilarity array and (forward difference) derivative array on the right. In this example, the closest depth value to the centre, with a valid class label, is the pixel with depth 9. Formally, we assign the majority class label in the range $[1, i]$, where i is the array index of the derivative maximum.

In the simplest case, we scan the derivative array from index 0, looking for the position i of the largest maximum value. We then assign the unclassified pixel the majority label (ignoring no-returns) based on the labels attached to array samples in $[1, i]$—see Figure 2. For well-behaved neighbourhoods, this works well. However, we also have failure cases that need to be dealt with immediately or deferred until the next pass. We apply the following sets of rules in the order indicated:

Rule 1: isolated points. If a point has no neighbours (except for no-return values), then it is labelled "discard." If the only neighbours are unlabelled, then it is skipped and the decision is deferred to the next pass.

Rule 2: weak discontinuity. As we scan for the largest derivative jump, we apply a threshold, $\frac{1}{2}S_{\max}$. Values below the threshold are ignored. $S_{\max}$ is the largest value in the dissimilarity array. If all the values fail the test, then the entire set of samples is assumed to lie on a noisy surface patch with no noteworthy depth discontinuities, and we simply take the majority class label (discard/keep) as the assigned label. However, if the threshold test is passed, then we proceed to Rules 3 and 4.

Rule 3: max at origin. If the largest jump in the derivative array occurs at sample 0 (large jump from sample 0 to sample 1), then we have limited information to make a decision. If the next sample, at index 1, is classified, then we assume the jump is due to noise and assign this class label, which is closest in depth. Otherwise, we cannot say where or if a real discontinuity exists (for example, there can be steep non-linear depth gradients on sloping surfaces, in addition to noise, which make it hard to identify a large jump), so we inspect all samples in the array and assign the majority class to the pixel.

Rule 4: regular maximum. If the maximum index, i, is not 0, then we inspect all ordered samples in $[1, i]$ and assign the majority class to the unlabelled pixel. If the majority class label is unlabelled, then we defer a decision until the next iteration.

Each boundary pixel is processed independently during an image pass: a new classification image is updated with newly assigned labels. Existing valid class labels (discard/keep) and no-returns are immediately written through into the new image; at no time are updated values written back into the classification image we are reading from. If an unclassified pixel cannot be assigned a class label at this point, then it is written into the new image as an unclassified pixel and will be resolved in the next pass. At the end of the current pass, the new classification image becomes the source classification image and the next pass begins. The iteration terminates when there is no further decrease in the number of unclassified pixels. This indicates that no further filling is possible without violating depth continuity or that only isolated unlabelled points remain. The final classification image is then passed into Phase 3. We used 3×3 blocks for all our tests; larger blocks require more sorting passes, although they do provide more context.

Phase 3—Label assignment for isolated components: We need to assign labels to the remaining unlabelled components that are isolated by depth discontinuities or invalid (no response) pixels. We do not simply label an entire component with the closest valid pixel in the classification image, since this ignores depth differences and pixel proximity in the image.

Instead, we set up a graph problem in which the (labelled and unlabelled) pixel cluster components are vertices and the edges that connect them are carefully chosen to avoid associating distant components. As explained later, the undirected graph edges are weighted with a measure of depth disparity between linked components. Once the graph is built, each unlabelled component performs two independent shortest-path searches: each search aims to find the shortest path from the unlabelled component to the "closest" labelled component of the appropriate class. This may require a number or searches for each class, but they are extremely fast (of the order of milliseconds). By minimizing depth changes as we move through the graph, we can find a compromise labelled component and copy its label to all pixels in the unclassified component we are considering. Note that each component labelling search happens independently, so the entire labelling operation could be parallelised if desired. This independence allows us to avoid order dependencies and limits the propagation of bad labelling decisions.

To build the graph, we first run a connected components analysis on the classification image generated by the first two phases. This allows us to label all the pixels belonging to connected blobs with a unique identifier. To simplify processing, we utilize three connected component images: one for each class label (keep and discard) and one for unlabelled pixels. In these images, the pixel class (one of keep, discard, or unlabelled, as appropriate) is considered foreground and everything else is background (no-returns and all other label pixels). To accelerate graph construction and queries, we filter out small unlabelled components; these are removed based on a fixed threshold and are essentially invisible to the graph algorithms. They will have a class assigned at the end of this phase.

Next, we extract the boundary pixels for each blob and write these into another image: identifying a boundary pixel simply requires checking the 3×3 neighbourhood for a candidate to see if there are any background samples. Extracting the boundary will reduce the calculation required in later steps. Having computed these separate images, we then merge all the component images to form a single multi-label component image and do the same to merge all boundary images into one boundary image. We are able to identify the class of any boundary or component in these images by storing some simple map that associates component indices to class type.

For each component (graph node), we wish to connect it to the components that are "nearby" in the component image. A common approach is to build a k-nearest neighbour graph [2], but this can lead to very dense graphs (even for small k) and allows very distant components to be linked. Instead, we define a small region around each component boundary point within which components may link to others and use ray-casting to decide which neighbours to connect to. More specifically, for a given component boundary point, we cast a ray to all other boundary samples in the region (hence the need for our boundary extraction—to avoid testing interior points). We compute the absolute depth difference between these points and keep a map that associates to each pair of components the smallest depth distance encountered so far. This map is constantly updated as the ray-casting proceeds. Once components have been processed in this way, we inspect the map and extract the linkage information for components as well as the weight—which is simply the minimum depth delta between components, measured along their connectable boundary. The intuition behind this is that one can "path" through a component and emerge anywhere on its boundary to find the shortest (depth) hop to another neighbouring component. The ray-casting is very fast (it uses 2D Bresenham and terminates early if there is a non-boundary intersection). Effectively, the approach explained above ensures that only components that have boundaries that are "visible" to one another and in close image proximity can be linked. These ideas are illustrated in Figure 3.

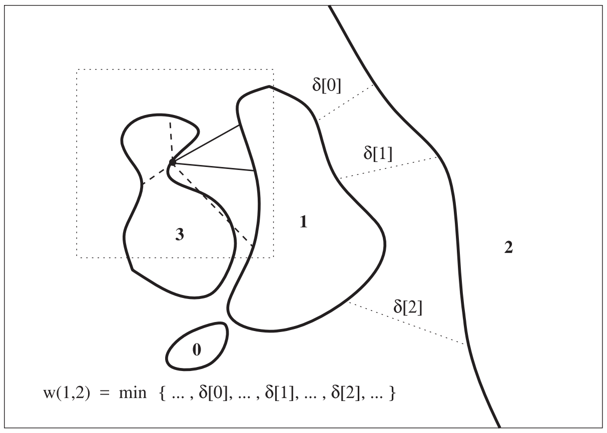

Fig. 3. **Connecting components into the graph.** The figure shows several components that need to be connected into the graph. On the left, a boundary pixel has cast a number of rays within the indicated square neighbourhood. Dashed lines are discarded due to self-intersection. Only two rays are shown to intersect the boundary of component 1, although in practice all boundary pixels on 1 that are visible to the point on 3 will be considered. Weights are then computed for valid rays. On the right, several representative ray casts connecting components 1 and 2 are shown: the weight assigned to the edge (1,2) is the minimum of all the ray cast weights from 1 to 2 (and 2 to 1).

Finally, we assign labels to the small set of unclassified points that remain. These are points that were excluded from the graph processing. At this stage, we have no further information to work with, so we simply assign the closest labelled neighbour in the classification image.

It is worth pointing out that if the classifier gives consistently poor predictions, then the post-processing will not smooth out the data in a useful way. The underlying assumption is that most high-confidence predictions are correct.

3.3 Applying the Labelling to the Model

Once all the unclassified pixels in the classification image have been labelled, the labels are copied to the corresponding "down-sampled" 3D points. Then all the points that mapped to a given octree cell are assigned the same label. As noted earlier, if a cell contains a large number of points, and the proxy point was misclassified, then this will result in a seemingly large classification error. Nonetheless, the down-sampling process is essential for performance, and as the classifier improves, this effect becomes less noticeable.

4 RESULTS AND DISCUSSION

Here, we present the hardware and software we used for our tests, the parameters we chose, and provide details on our test data. We then discuss our results.

4.1 Test Data and System

Tests were run on a PC with 16GB of memory and a quad-core Intel Core i7 CPU clocked at 3.4GHz, running Ubuntu 16.04 (64-bit).

The framework is implemented in C++ 11 and uses OpenCV v3.3, Point Cloud Library (PCL v1.8) and the Boost library. More specifically, PCL is used to manage point clouds and to compute 3D point features, OpenCV is used for machine learning, and Boost is used for tasks such as command line parsing and graph operations.

Table 2. Data Sets Used in Testing

Model	Resolution	# points	# scans	Description	Discarded data
Songo Mnara	5,056 × 2,019	10.2M	10	Ruined buildings with many trees and foliage	Structures around the area of interest, vegetation also inside the area of interest
Ball Court	4,096 × 1,736	7.1M	10	Mayan ruin with trees and foliage	Structures around the area of interest, vegetation
Montelupo	4,790 × 2,154	10.3M	6	Old church site with lots of clutter and foliage	Vegetation around the area of interest
	9,580 × 4,307	41.2M	2		
Bagni di Nerone	3,828 × 1,723	6.5M	15	Ancient Roman bath site	Structures around the area of interest, vegetation, railings
Church	1,910 × 862	1.7M	15	Underground church	Mainly railings and grates, outside environment
Monument	1,911 × 1,723	3.3M	14	Statue in courtyard	Everything but the monument
Signoria	3,822 × 1,723	6.6M	15	Busy Piazza in Florence	Mainly people and vehicles, scattered data due to window glass
Stairs	2,390 × 1,076	2.6M	11	Interior steps with adjacent chambers	Wooden doors, non-interesting rooms

The data set consists of eight small scanning campaigns, spanning a range of external and internal cultural heritage environments. The scans are stored as PTX files,[2] since this format provides a simple and direct representation of raster format produced by the scanner. This is not a space-efficient format, since the data is stored as text and contains redundant vector entries for no-return points—if space is an issue, then these files could be parsed into a more efficient raster-preserving format.

Information on each of the data sets we use is shown in Table 2.

4.2 Parameter Selection

The classifier is built on the random forest implementation provided by OpenCV 3. Unfortunately, support for RF parallelism was unexpectedly removed in this version; classifier performance was thus significantly lower than we expected, compared to OpenCV 2. RF is embarrassingly parallel and should scale linearly with the number of CPU cores.

RF has a number of parameters that control the growth of the forest and classification accuracy. Since we are not trying to learn a single general classifier, it would not make sense to fine-tune parameters through, for example, a grid-search with n-fold cross-validation. Instead, we performed a small number of test runs to determine a set of plausible parameters and then used these for all our tests:

[2]See http://paulbourke.net/dataformats/ptx/ for description.

Table 3. Parameters Used for Testing

Param.	Detail	Value
G_r	resolution of sub-sampling grid	0.02m
R_c	cylinder feature radius	0.1m
R_c	curvature calc. radius	0.2m
κ	RF re-weighting multiplier	10
L	Multi-resolution levels	6
s_r	feature sub-sample percentage for the first scan	20%
k	nearest neighbour number for feature calc.	10
PP_s	post-process: min. component size for graph node	10
PP_c	post-process: confidence level threshold	0.8
PP_r	post-process: pixel radius for graph linkage	15

- For the number of node split trials, we used $\sqrt{N_F}$, where N_F is the number of elements in the feature vector. This is the recommended value for classification with random forest [5]. Increasing the value will lead to better classification, but at higher computational cost. Informal testing showed that a higher value was better, but since our RF implementation is serial, we kept the default.
- We limited the tree depth to 32 and the maximum number of trees to 32. We forced all trees to be grown by setting the target out-of-bag error to 0; more trees generally equate to better performance but greater training and classification times.
- The minimum number of points in a node for a split to occur was set at 20. This means trees cannot grow to any depth and are thus less likely to over-fit. This, in turn, means we should be able to use fewer trees.

In addition to the RF parameters, Table 3 lists the defaults used across all experiments.

4.3 Prediction Performance Metrics

Since we have ground-truth segmentations from cleaned scanning campaigns, we are able to directly measure predictor performance. We have chosen to measure the usual classifier metrics, true/false positives (TP/FP), and true/false negatives (TN/FN) for each class, as well as the pre- and post-process component distribution. However, we do not report precision and recall—which are based on TP/FP/TN/FN rates—since these measures are asymmetric and depend on which class is taken to be the positive class. Instead, we report the *accuracy*, A, which is simply

$$A = \frac{TP + FP}{TP + FP + TN + FN}. \tag{2}$$

This is a symmetric measure and reports what users are most interested in: whether the predictor is predicting points correctly.

The post-process treats the initial prediction as a range image with classification labels instead of depth as each pixel value. To get a sense of how well the post-process does at reducing small isolated error components, we generate size distribution histograms and also record the total number of error components produced. These measure the distribution of components sizes of incorrectly labelled pixel clusters left behind after the post-process has been completed. Note that we only record incorrectly labelled components; it is quite possible to have small correctly labelled components, as noted earlier. To compute the histograms, we perform a connected component analysis on the error image. This is a binary image that sets all incorrectly labelled class pixels, relative to the ground-truth classification image, to 1. This image will include components that the classifier labelled incorrectly with high confidence.

Table 4. How κ Affects Training Set Samples Added, x1000 (Montelupo Data Set)

	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	Total
$\kappa = 1$	422	188	73	26	161	43	21	10	944
$\kappa = 5$	422	189	74	32	138	44	23	10	932
$\kappa = 10$	422	190	68	33	131	41	23	11	**919**
$\kappa = 10^*$	422	190	61	35	150	43	25	11	937
$\kappa = 20$	422	191	57	35	139	46	24	9	923

4.4 Training Set Size and the κ Parameter

Since we do not constrain the work-flow of the cleaner, they are free to clean the scans in any order they wish. If the current classifier is a good approximation of the underlying distribution, then the number of mispredictions falls sharply with each new scan and the classifier needn't be updated after each scan. However, when a scan with previously unseen geometric characteristics is processed, the number of bad predictions increases. This, in turn, means that the number of mispredicted samples that are added to the current set of training samples for the next RF retraining phase increases.

The κ parameter is used to weight new training samples, and the choice of this parameter impacts on the accuracy of the classifier at each step. Generally, a high value will mean that misclassified samples are heavily emphasized so if similar data is encountered in the next point cloud, then the classifier will do well and fewer new training samples will be added to the growing pool of training data. For low values of κ, the classifier does well if the following point clouds are similar, but if a new point distribution is encountered, then the classifier performs badly, and many more training samples are added as a consequence. Table 4 shows the approximate number (in 1000s) of new training samples added as each new scan is encountered, as well as the total number of training samples in the final classifier in the sequence.

The initial scans are similar to scan #1; however, scan #5 is very different, with the scanner right next to a wall. This causes a high prediction error (across the adjacent wall), and many new training samples are added to the pool to capture this new point distribution. The value $\kappa = 10$ responded best in this example, providing a good compromise between general prediction accuracy and quick response to new kinds of point distributions. An obvious question is whether the probability is needed in the weighting scheme. The table entry for $\kappa = 10^*$ shows what happens when probability is removed for the $\kappa = 10$ case—the classifier performs more poorly, and more training samples need to be added in general, most notably after a large correction. We expect such corrections to occur frequently for scanning campaigns, so the ability to respond quickly and to reduce the number of subsequent training samples required is important.

4.5 Discussion

Both quantitative and qualitative results are presented here. Quantitative results present measures of system performance, while qualitative results consist of a discussion around the classification images produced for each scan campaign. Classification images are used rather than the structured point cloud, since it is easier to see where classification errors are clustered. The role of the κ parameter was explored in Section 4.4, so we will not discuss this further.

4.5.1 Quantitative Results. The summary results for classification performance are presented in Tables 5 (accuracy) and 6 (timing). Detailed per-scan results for accuracy (Tables A.1, A.2, and A.3) and timing (Tables A.4, A.5, and A.6) can be found in the Supplementary material.

Examining Table 5, we observe that the average classification performance, when measured using accuracy (Equation (2)), is generally good. However, the accuracy varies depending on which view one takes of the data.

Table 5. Overall per Campaign Classification Accuracy, for $\kappa = 10$

Campaign	Avg. Geom %	Avg. No-Post %	Avg. Post %	Min. Post %	Max. Post %	STDDEV. Post %	Comp. Reduce Factor
Montelupo	94.6	93.3	93.7	64.8	99.1	11.8	x9.3
Songo Mnara	96.2	96.4	97.3	94.7	98.9	1.2	x5.4
Ball Court	80.3	69.8	71.9	56.6	97.4	13.6	x4.1
Bagni	95.3	97.2	98.1	92.7	99.8	2.2	x10.3
Monument	98.9	97.1	97.6	84.8	99.8	4.0	x7.3
Church	98.3	99.1	99.1	94.1	99.9	1.5	x2.4
Signoria	95.5	97.0	97.9	96.5	99.3	0.9	x4.1
Stairs	84.7	83.9	84.3	37.5	99.0	18.6	x10.7

It is important to emphasize that the classifier is trained on (and classifies) point data, not image data. Furthermore, the data that is trained/classified is *down-sampled* point data—the point set produced by the initial homogenization/re-sampling step. The table entry for "Acc. (geom.)" measures this error. The down-sampled data has to be up-sampled back to the original grid resolution, and this is where differences arise. The up-sampling step simply labels all the points that mapped into a sampling grid voxel with the same predicted label. Naturally, this means that any error in the down-sampled prediction will be magnified significantly in the high-resolution point cloud. The effect is particularly damaging for misclassified points close to the scanner origin: in this region the point density is highest and many points are averaged in each sampling voxel. For this reason, some of the data sets show results where a scan has high accuracy on the down-sampled cloud but very much lower accuracy on the up-sampled cloud. The "Acc. (no post)" entry refers to the directly up-sampled classification, with no post-processing stage. This is a more accurate measure of actual classification performance for our approach. The entry "Acc. (post)" reports the final high-resolution mesh classification accuracy when post-processing is used. We would like this number to be better, or at least no worse, than the unprocessed classification while significantly reducing the number of error components. Note that for some scans the predictor produces bad classification values and marks most of the scan as "unknown." For such scans, the post-process will produce poor results—worse than simply up-sampling the direct point prediction, regardless of prediction confidence—since it has to produce values for most of the points and is seeded with incorrect data. This is clear from the "Min. Post" column in the table. The Stairs campaign produces one such case, which is expanded on in more detail later.

In an attempt to quantify the quality of the classification, we report the error component sizes based on the classification image, as noted in Section 4.3. Table 5 provides condensed per-campaign information, while Tables A.1 to A.3 (in the Supplementary material) report the number of error components for each class (0 = keep, 1 = discard) with and without post-processing, for each scan. In these tables, the column labelled with <5 lists total component numbers when only components of size 1–4 pixels are considered. We also present a summary component size histogram for the Church campaign—see Figure 4. The full set of histograms—Figures A.2 to A.5—are listed in the Appendix. For all histograms, the last bin contains the count for all components of size ≥50 pixels/points.

It is clear that for all data sets, post-processing reduces the number of components, in some cases by a factor of 10. Table 5 (last column) lists the error component reduction factor for each campaign as a result of post-processing. While post-processing tends to grow larger components, on average, there are less of them and classification accuracy is the same or better than using the unprocessed classification. Fewer, more visible clusters should translate into less work to clean the scan after the initial classification. Also note that many of the very

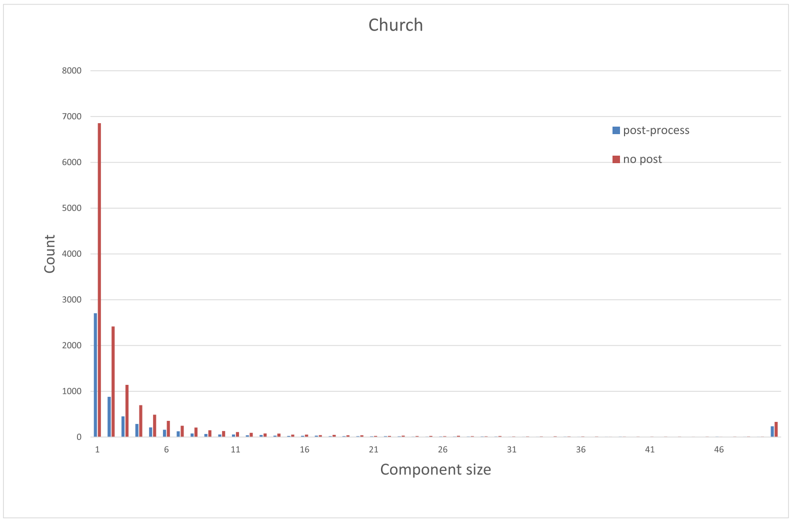

Fig. 4. **Connected component histogram.** The graph shows the result of building a histogram of connected component sizes—with and without post-processing—across the Church campaign. As intended, the post-process reduces the number of isolated components significantly.

Table 6. Overall Processing Times (in seconds) for Each Campaign, $\kappa = 10$

Campaign	# scans	load + MR gen.	train: feature compute	train: RF	classify: feature compute	classify: run	post:	Avg. post	Avg. total
Montelupo	8	269	340	1,810	999	75	386	48	485
Songo Mnara	10	184	50	1,481	798	143	209	21	286
Ball Court	10	111	65	7,264	522	120	145	15	823
Bagni	15	158	93	2,076	1,431	168	178	12	273
Monument	14	86	61	852	878	69	75	5	144
Church	15	47	29	436	299	50	23	2	59
Signoria	15	198	179	14,047	2,802	446	236	16	1,193
Stairs	11	56	73	3,497	512	56	84	8	389

Each column shows the processing time for that component of the pipeline.

small clusters arise from hard-to-see errors in the ground-truth labels, as noted above, so the error component numbers are an over-estimation. Finally, the small error components tend to be scattered along or close to depth boundaries. It is likely that the registration and meshing procedures used to build a final model are robust to such small errors and the cleaner would not have deal with them.

Table 6 shows that the post-processing phase generally requires a small part of the total scan processing time. The time increases significantly when the classifier produces low-confidence predictions and a large fraction of the classification image pixels need to be filled. If the point set is very large, then the computational burden will be very high. Fortunately, the average post-process time is still low when compared to overall scan processing times. The depth-aware fill is the slowest part of the post-process due to the large number of sort operations required. We did not invest time in optimizing this stage, since it seemed fast enough, in relation to training and feature computation, but this could be an area for future work.

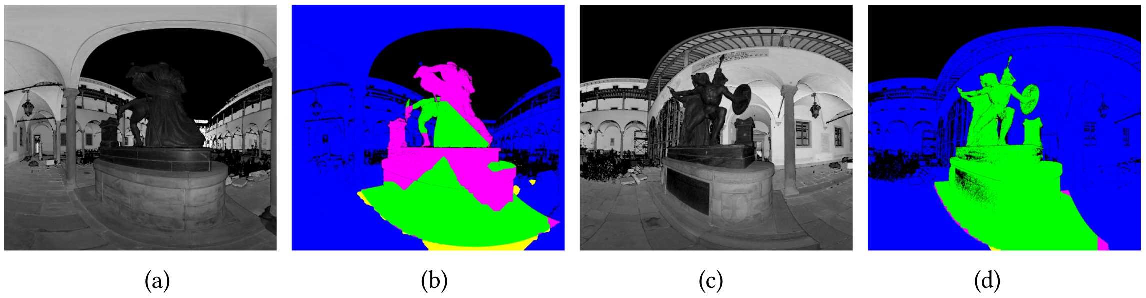

Fig. 5. **Monument examples.** Images (a) and (c) show the laser return intensity, while (b) and (d) show the post-processed predictions.

Feature computation for most of our examples was below 2mins per scan. The feature compute time depends on both the number of points in the scan and the point distribution in space. Thus, it can be hard to predict what the compute cost will be. We observed that the cylindrical neighbourhood queries are very costly, accounting for perhaps 50% of the total feature computation time. We used a simple approach of projecting all points to the ground plane and then performed a (2D kd-tree) R-radius neighbour query for every point; at fine resolution levels this is time-consuming and returns many points, even on the down-sampled input cloud. Since this feature has high importance (it is used often during RF tree construction), we would not want to discard it. However, the fine resolution levels of the feature are less important, so one option would be to simply avoid calculating the cylinder Z features on these fine levels (probably the first two levels).

Classification per scan is fast: less than 30s, on average, for all our examples. In general, the largest fraction of the run-time is taken up by training. We expect this to increase slowly over time, as more data is added to refine the classifier for the current campaign. If the scans are more consistent, then less data will be added as each new scan arrives and training time will taper off. The Signoria campaign had a particularly high training time: 3.9h across 15 scans. Due to the inconsistent labelling and the complexity of the discard/keep classes, the classifier was unable to reach any kind of equilibrium, and a large number of re-weighted training samples were added at each new scan, greatly inflating training times (which scale with the size of the training set). Although this example is extremely challenging and would take a long time to clean manually, it does highlight a weakness in the scalability of the approach that would need to be addressed for large campaigns. However, as noted earlier, the RF implementation is not parallel, so one could reduce the training and classification times dramatically by implementing a correctly parallelised RF. Each tree can be built independently, and each node trial (determining a kd-tree split plane from the the feature variable set) can be done in parallel.

4.5.2 Qualitative Results. Each scan campaign is discussed below. To reduce the number of figures, only two scans from each campaign are presented: one with low prediction accuracy and one with high prediction accuracy. This is sufficient to highlight any important issues encountered in the relevant campaign. The full data sets are presented in the Supplementary material; these supplementary figures/tables are referred to in the following discussions and have an "A" attached to the reference. The Ball court campaign is discussed in detail, since it exhibits a number of important characteristics that intrinsically limit classification accuracy. In all the classification results, green represents "keep," blue represents "discard," yellow corresponds to points incorrectly labelled as "keep," and magenta to points incorrectly labelled as "discard."

Monument: This campaign captures a single statue in an enclosed courtyard—see Figure 5. (The full set of scans can be found in Figures A.15 and A.16.) In this campaign, the majority of points are discarded and contain a complex array of objects, such as beams, plants, and so on. The low prediction accuracy shown in the figure is a result of a sudden viewpoint change—the scanner moves close to the statue, and this causes the point distribution

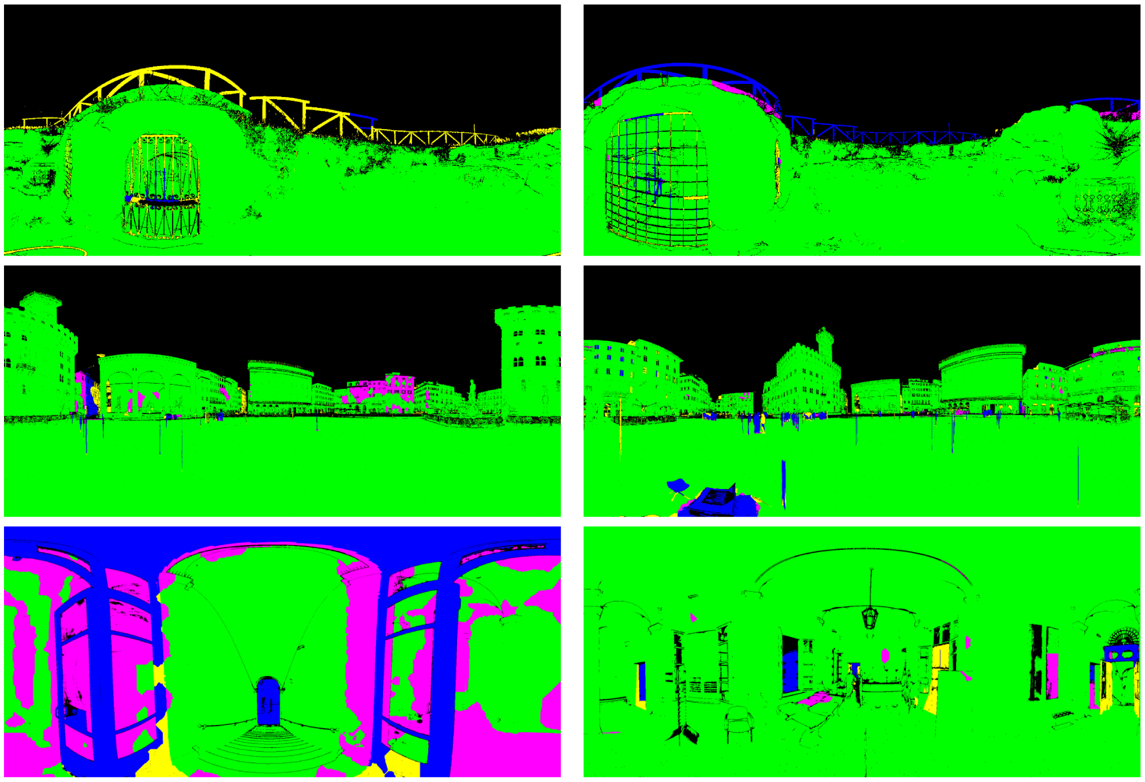

Fig. 6. **Example classifications by campaign.** Each row shows two examples from a campaign. **Row 1:** Church; **Row 2:** Signoria; **Row 3:** Stairs

and density to change suddenly. However, the predictor recovers on the next scan. The high-prediction image shows a scan after the predictor has corrected itself.

Church: In this campaign, the interior and exterior of a partially buried church are scanned—Figure 6, Row 1. This data set is challenging, since the cleaning target in the interior consists of pipework and thin bars located at varying distances from the scanner. The exterior is also very different, so when the first exterior scan is processed, the predictor does poorly. However, it recovers on the next scan. These scans are shown in the sample. The full set of scans, Figures A.17 and A.18, show the interior cases. While the overall accuracy is high, the points to be removed are located on thin structures and not well recovered in general. This is caused by several factors: (1) the initial scan down-sampling further erodes thin structures, (2) the features used are not well suited to detecting thin structures, and (3) the training data is overwhelmingly drawn from the "keep" class. This latter point is important and affects all scans where there is a very large imbalance between the input training set sizes. Although we balance the data for each class, there is simply a great deal more geometric variability to sample in the very large class, while the minority class has very little.

Signoria: This campaign covers the *Piazza della Signoria* in Florence—Figure 6, Row 2. This is a busy square with a large number of tourists in motion and many buildings of all shapes and sizes. The scanner also captures points that are very distant and thus come from areas of the point cloud with very low density. To further complicate matters, the choice of which distant structures to keep is somewhat arbitrary, as is the decision to keep or discard vegetation, vehicles, and shop façades along the boundary of the square. As the figure shows, the

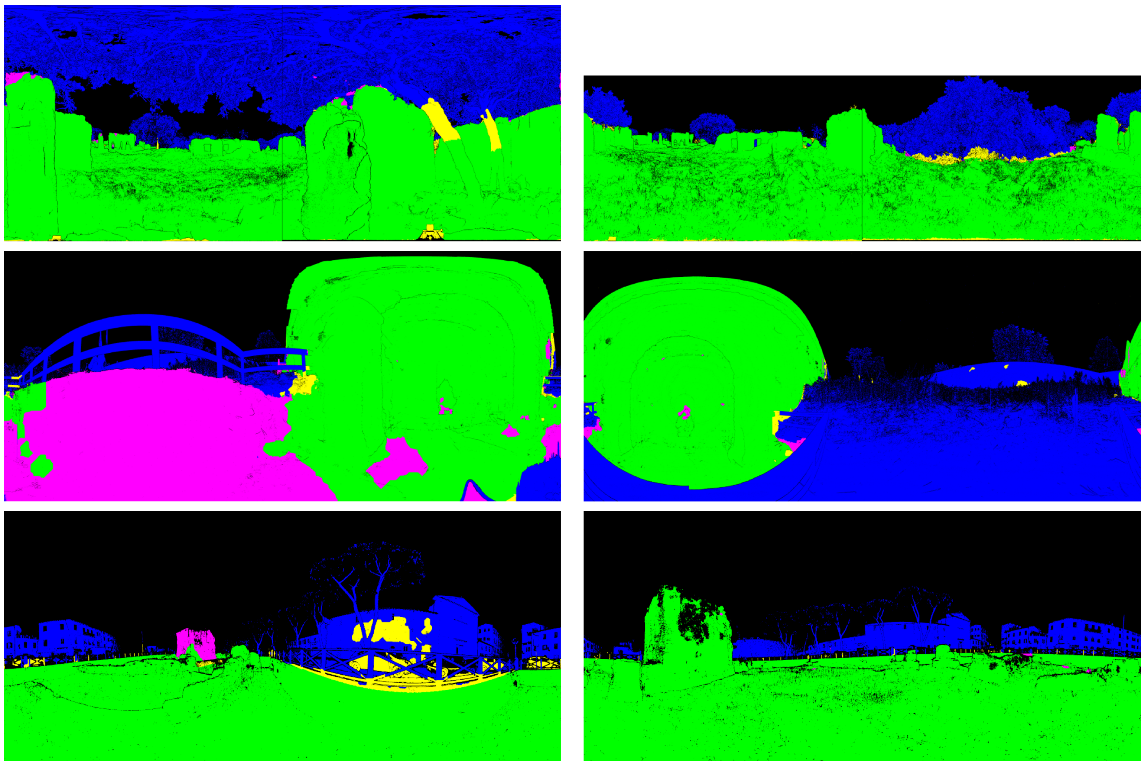

Fig. 7. **Example classifications by campaign.** Each row shows two examples from a campaign. **Row 1:** Songo Mnara; **Row 2:** Montelupo; **Row 3:** Bagni di Nerone.

classifier struggles with the inconsistency, although it is able to produce some reasonable classification outputs. As noted in the performance results, the large classifier training time is due to the very high variability of keep/discard in the data set, which means that a lot of new training data is added for each new scan. The full set of scans is presented in Figures A.19 and A.20.

Stairs: the scans come from a staircase inside a building and also include adjacent chambers—Figure 6, Row 3. The keep and discard points sets have a great deal of overlap, and the scanner also acquires points below the horizontal plane (descending steps). The full scan set—Figures A.21 and A.22—shows the effect of introducing a new scan type (stairs to chambers). The classification of the first predicted scan in all campaigns is generally poor, since the predictor will only have seen one scan at that point. However, in the Stairs campaign, the first predicted scan (Figure A.21, second image on top row) produces a result that is essentially random. In this case, the second scan is completely different from the first. The classifier thus has low confidence for almost all scene points, and the seed points chosen for the post-process are not reliable. The post-process thus propagates bad labels across large depth-consistent surfaces, and this results in the poorest result across all scans, in all campaigns. This highlights the need to have at least some consistency within a campaign. In this campaign, having the user clean the first two (very different) scans would introduce enough training-set variability to avoid having such a poor classification result later in the scan sequence.

Songo Mnara: This campaign covers a ruined site with crumbling walls, trees, and plants—Figure 7, Row 1. This campaign is generally well predicted, although there is some inconsistent ground-truth labelling of tree

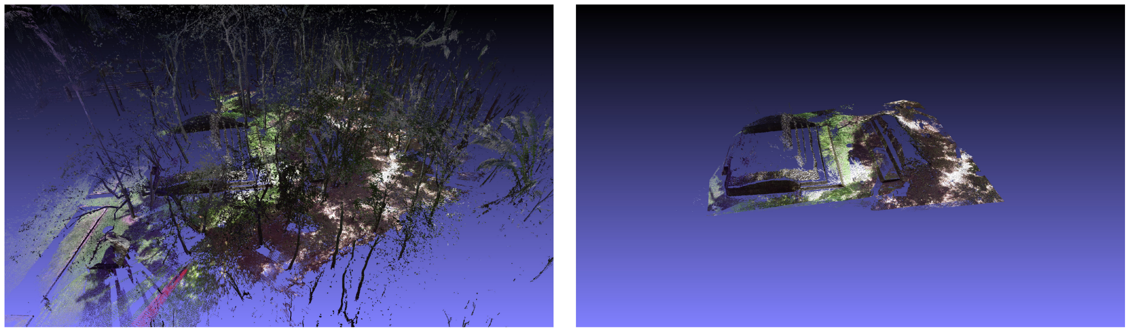

Fig. 8. **Ballcourt dataset.** Left: the original uncleaned point cloud (all scans aligned). Right: the cleaned point cloud. In this dataset, the vast majority of the acquired data is removed, and the points that remain have some similarity to the removed data.

trunks near the scanner that causes later prediction errors. The scanner also acquires a large depth range and is sometimes placed right next to a wall (generating "holes" in some scans). The predictor works well, because the keep/discard regions are qualitatively distinct and there is less of the arbitrary labelling that occurs in many other campaigns. The full data set is shown in Figures A.8 to A.10.

Montelupo: This site contains the exterior of an old building in a walled area with plant growth as well as a separate chamber interior—Figure 7, Row 2. The sample scans show only the exterior wall and building. The low accuracy prediction occurs because all previous scans were taken from a distance. For this scan, the scanner is placed against a wall and, since this is a new geometric configuration, the prediction does poorly. Since the adjacent wall occupies a large part of the full dome scan, the poorly predicted information is propagated across a large segment of the scan. Nonetheless, as the full campaign shows—Figures A.6 and A.7—the classifier recovers after this scan and produces good results overall.

Bagni: This site covers part of the ruined site in Pisa—Figure 7, Row 3—and includes surrounding city buildings and clutter that have to be removed. There are also large distances involved for some of the scans, with distant buildings and thin structures that need to be removed. The campaign also includes the interior of a chamber. As the low-prediction example shows, some scans are not well recovered—likely due to an overdependence on the "distance to point" feature, which is expecting the city building to be more distant based on previously seen scans. While the overall classification accuracy is high, fine detail—such as the bars of the railings/barrier around the site—remains a challenge. As noted earlier, thin structures are badly affected by the point-cloud resampling and distance from the scanner. The full data set is presented in Figures A.11 and A.12.

Ballcourt: This campaign surveys a ruined Mayan structure surrounded by jungle—see Figure 8. The classifier exhibits very poor accuracy on the Ballcourt campaign. Here the down-sampled accuracy drops and the high-resolution point-cloud accuracy is dramatically worse—by 20% for one scan. Applying the post-process improves matters slightly, but the results remain poor. The reasons for this poor performance are:

- large scale and consistently poor classification close to the scanner,
- extensive contradictory labelling, and
- small and scattered labelling errors in the ground-truth input.

The first issue is a direct consequence of the other two. The labelling problem is best appreciated by looking at the ground-truth labels and the predicted (post-processed) results—see Figure 9. The initial training set (not shown) is similar to the first ground-truth image in this figure. It contains trees as discard and ground cover/grass and building as point types to be kept. The classifier makes some errors, but the first scan is well

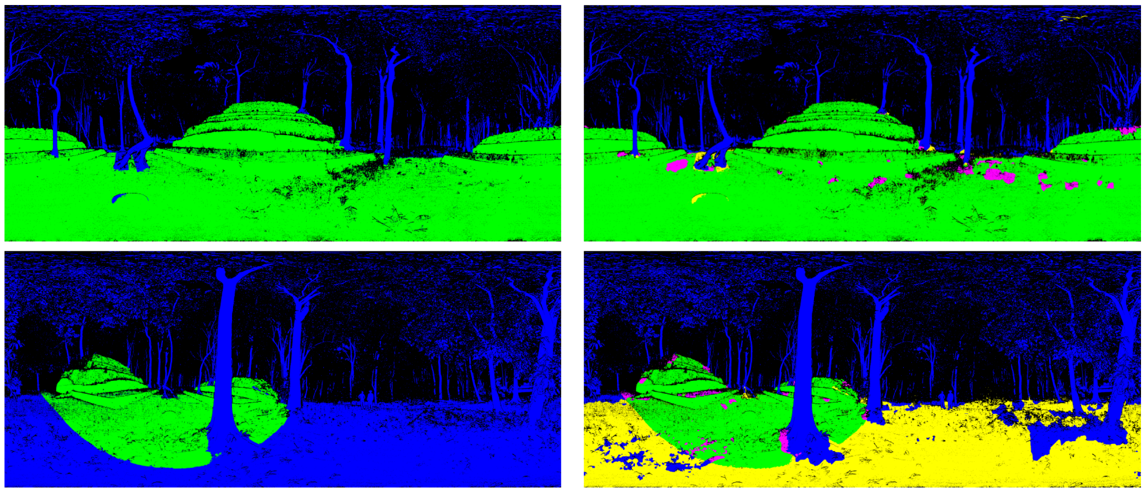
Fig. 9. **Ballcourt data, showing contradictory labelling.** The left image shows the ground-truth labels, and the right images show the post-processed prediction.

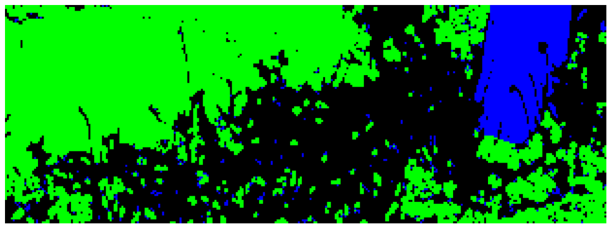
Fig. 10. **Errors in input ground truth.** The closeup image shows a section of the Ballcourt campaign, which includes labels for discard (blue) and keep (green). However, the keep labels have a large number of isolated points that are marked as discard, even though they appear to be part of the keep class. The post-process explicitly seeks to reduce small isolated point clusters, and thus these points end up being flagged as classification errors.

classified, given that only 20% of the input labels were used. The next image in the sequence is very badly classified: based on prior training data, the classifier expects ground to be part of the "keep" class. It still manages to pick up most trees and the higher parts of structures. Nonetheless, the dominance of ground in this scan means that the error is very large and is magnified further, since the scanner is surrounded by this point type. The classifier notes that ground is now "bad" and adds many point features back with a heavy weighting to compensate. The full set of ground-truth labels and post-processed results is listed in the Appendix (Figures A.13 and A.14). These show that the next scan does a lot better, although the classifier has been skewed by the inconsistent labels. Unfortunately, the following scan in the sequence (the fourth scan) introduces a large patch of samples that contradict what the classifier had seen in the previous scan. As expected, the classifier does badly. This alternating association of ground points between the "keep" and "discard" classes essentially asks the classifier to reconcile contradictory data. Interestingly enough, except for the ground points, the other elements such as trees and buildings are mostly recovered. We will return to this point later.

Although it is not apparent from the ground-truth labels, there are also widespread but isolated errors in the input. This is illustrated in Figure 10. This shows part of a scan near the base of a structure, with a tree

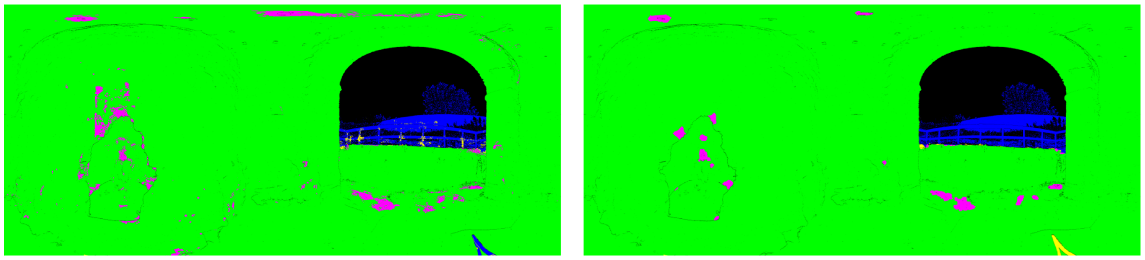
Fig. 11. **Effect of post-process.** The classification image on the left is directly up-sampled from the 3D point prediction. The image on the right has been post-processed.

to the right. Note the very small isolated (blue) point clusters interspersed with the green region. Based on a reasonable assumption of regional smoothness, these would seem to be incorrectly labelled. Since the post-process is designed to discourage very small isolated components, these points are effectively set to green and thus marked as classification errors when compared to the ground-truth labelling. This is the main cause of the very large keep/discard error component numbers noted below. The impact of these errors is to further bias the retraining stage, since the classifier is given yet more incorrect or contradictory information.

General comments: Errors in the keep/discard classification are most evident close to depth discontinuities: these are not only from incorrect high-confidence predictions, but also from incorrect filling of unclassified pixels during the post-process. Although the post-process is intended to propagate good predictions and to avoid doing so over depth discontinuities, it is limited by the accuracy of the predicted labels it is extrapolating. There are also isolated unclassified points for which a neighbouring label is the only viable match, even if it is across a steep depth discontinuity. Nevertheless, on average, the post-process ensures a cleaner classification with significantly less misclassified point clusters—see Figure 11. As we noted above, some classification errors are a result of problems with the ground-truth labelling.

The impact of the features we used on classification accuracy was not explored in any detail: while we used a common set of features for 3D point classification, these were not necessarily optimal for our data sets. In particular, the point re-sampling we employ only partially mitigates point density differences through the cloud and is not ideal when there are many thin structures that need to be cleaned from a campaign. Informal feature importance tests showed that Z-based features (height and cylindrical features) and depth from scanner are favoured by the classifier. While these ensure more robustness against the diversity of keep/discard labels, they may overwhelm weaker features that are better at representing intrinsic neighbourhood properties.

Finally, it is important to emphasize how the cleaning process differs from the object classification or point segmentation task. For most learning applications, the goal is to associate each point with a class label, and membership of that class is typically hard, e.g., a point is part of a tree or part of a pole. In the cleaning task, the general goal is to achieve a binary labelling, keep and discard, but the problem is complicated by the common practice of also discarding part of the point cloud for what we have termed "modelling" reasons. In this case, a label that is nominally correctly assigned by the classifier based on previous learning can be arbitrarily invalidated. To learn the true classification rule in such circumstances requires learning to establish rules about modelling decisions. Unfortunately, these rules appear to be somewhat arbitrary, since they are (a) based on aesthetics and personal preference in many cases and (b) depend on the kind of data being classified. This suggests that a modification to the point cleaning work-flow might be a better option. For example, one could restrict cleaning to the pure problem (keep/discard labelling) noted above and then run a post-edit step to delete parts of the labelled point set that are not desirable for some reason. This might be a simple matter of using a 2D/3D painting/lasso interface to delete unwanted points.

Table 7. Expert vs System Timings (in seconds): Bagni di Nerone Campaign

#	Acc. G%	Acc. P%	Feat.	Train	Classify	Post.	Edit	Total	Expert
1	100.0	100.0	93	60	-	-	540	696	540
2	97.0	97.8	76	80	9	10	-	175	480
3	97.6	98.6	99	105	10	11	-	225	360
4	92.7	92.9	153	199	10	31	-	393	240
5	98.9	99.3	151	-	9	30	-	190	180
6	94.4	98.8	130	250	8	19	-	407	420
7	98.3	99.0	95	-	11	10	-	116	360
8	96.9	97.9	81	326	11	9	-	427	420
9	99.5	99.8	100	-	14	7	-	121	360
10	99.1	99.7	87	-	14	7	-	108	420
11	98.2	98.5	90	-	13	7	-	110	360
12	93.0	92.7	101	483	13	13	-	610	420
13	98.8	99.5	110	-	15	5	-	131	360
14	97.7	99.1	87	573	16	9	-	685	420
15	98.6	99.2	71	-	14	10	-	95	360
Tot.		(98.0)	1,524	2,076	168	178	540	4,489	5,700

Acc. G records the classifier prediction accuracy (with no post-process), *Acc. P*, with postprocessing. The average prediction accuracy (post-processed), excluding the first manually labelled scan, is 98%.

4.6 Classifier Generalization Test

To determine if there is much "transferability" between scanning campaigns, we took the classifier generated from the full Songo Mnara data set and used this to classify all the Montelupo scans. As Supplementary Figure A.1 shows, the initial scan is badly misclassified, but the classifier recovers as it adds more data from the Montlupo campaign. However, the recovery can be attributed to the new data from Montelupo *overwhelming* the original Songo Mnara training data, as shown by the very large number of training samples added in the first scan. There is one notable change: the Songo Mnara training data contains many scanning positions close to walls, so the Montelupo scan that caused a large drop in performance—see Figure A.7(b)—is handled much better using the mixed classifier. Nonetheless, the slightly lower classification accuracy coupled with the the greatly increased training times suggest that it is better to develop a classifier per scanning campaign, as we originally proposed.

4.7 System vs Expert User

In an attempt to gauge real-world performance, we present timings produced by our system (using ground-truth labels) and timings produced by an expert user with 15 years of experience in cleaning cultural heritage scans. In total, 15 scans from a new Roman site, Bagni di Nerone—see Table 2—were cleaned using the MeshLab software tool [7].

The expert user times were estimated by the cleaner based on a job start/end timer. Timings do not include loading/saving times. To avoid unnecessary processing, we imposed a constraint whereby classifier retraining, using all the accumulated features since the last retrain, only occurs when the scan classification accuracy (with no post-processing) falls below 98%. This reduced the number of training iterations from 15 to 8 while retaining reasonable results. The classification images are shown in Figures A.11 and A.12.

Table 7 shows the timings obtained when the scans are classified by our system and the expert user. The "Acc. G" column reports the prediction accuracy, with no post-processing, which is used to determine if a new retraining cycle is required. The final post-process prediction accuracy is listed under "Acc. P." The average (post-processed) prediction accuracy, excluding the initial labelled scan, is 98%.

We included the original estimated cleaning time for scan 1, since this is required to bootstrap the classifier. The "Edit" column in the table should indicate the time required to clean the misclassified points in the remaining scans; since we do not have these, we have not reported any numbers here. Instead, we compare the total system time for all processing, excluding editing, and the times reported by the expert user. As the table indicates, the non-editing processing time is about 20mins less than the expert user. This means that 20mins are available to clean the remaining 14 scans, after prediction, before the semi-automated system and expert user times match. Since the number of misclassified points is small and captures many small discard regions such as railing posts and trees correctly, this task seems achievable. Furthermore, when the misclassifications are large, they are fairly smooth blobby regions that could be easily selected with a lasso or similar selection tool. While the average scan cleaning time was more than 6mins for the expert, this included full scan cleaning with no assistance. It is also important to note that if the random forest is parallelised, then the processing time for classification and training could shrink by up to 28mins in this example, assuming a four-core CPU (which is fairly standard on modern PCs). This would allow 48mins for the user to to correct 14 scans, which would be much easier to achieve. Additional processing time optimization, such as reducing feature compute times, could also be implemented.

5 CONCLUSION

Point-cloud cleaning is a tedious and time-consuming part of the processing pipeline used to transform data from a terrestrial laser scanner into a useful 3D model. The problem is exacerbated when many scans have to be cleaned, since a large and complex scan may require more than 30mins to clean. Clearly, any system that can reduce this time will help to accelerate the scan-processing work-flow. In this article, we introduce a framework that seeks to use machine learning to semi-automate the process of point-cloud cleaning with a focus on applications in the cultural heritage domain. The heritage structures encountered in the CH domain vary dramatically, as do the environments around them. This makes the application of machine learning challenging, since there is little general consistency across data sets. Instead of trying to learn a general cleaning classifier, we develop an approach that incrementally learns a Random Forest classifier for a given campaign, requiring only a single cleaned scan to start. The classifier refines over time, by noting the kinds of points it has misclassified before, based on feedback gathered as the user corrects mislabelled points. A weighting scheme based on the confidence assigned to incorrect points is used to nudge the classifier towards better prediction of problematic point structures. To further reduce user effort, a post-process is applied to the classified output by treating the point cloud as a 2D "classification" image and attempting to reduce small isolated or inconsistent point predictions. This exploits the natural raster scan order in which 3D points are acquired and uses information on local depth discontinuity to reassign point labels.

The results show that this approach consistently produces classified point clouds, across a variety of scanning campaigns, in which, on average, most points are correctly assigned as keep/discard. Furthermore, as desired, the number of small mislabeled components is greatly reduced compared to using the classifier alone. While the processing times can be significantly improved, as noted in this article, optimization of the method was not the focus of this investigation. Nonetheless, an informal cleaning experiment is presented that shows that even in its unoptimized state, the proposed framework results in a potential time-saving over a fully manual cleaning process when compared to an expert user.

5.1 Future Work

The system can be improved in several ways. The simplest improvement is the parallelisation of the RF classifier. This is a matter of finding a better ML library or layering OpenMP on top of the implementation provided by OpenCV. The more important areas to be addressed are:

Scalability: The main bottleneck at the moment is the potentially unbounded growth of the training set for the RF classifier. While this growth should be slow for consistent data, sometimes it grows more quickly than we anticipated, leading to slow training times. A possible solution is to resample the training sets when it exceeds a given size by randomly choosing a smaller number of samples for each set. Each sample would retain the weight it was allocated earlier in the training. Extensive testing would be needed to assess the impact of this change on prediction accuracy.

Consistency: A problem we highlighted earlier was the tension between *labelling* and *modelling* decisions when choosing points to discard. A possible solution, easy to accommodate in the work-flow, is to have the user—once a prediction has been presented—select points that should NOT be considered when retraining the classifier. These points will be ignored when determining point classification errors, but the user can reassign them if desired, ensuring that the classifier solves only the labelling problem.

Aligned datasets: While we did not require an alignment among the scans, this information, if present, could be successfully exploited to improve the convergence of the classifier; for example, we could transfer the classification of samples from a scan into other ones according to their distance. This could allow the user to work on only a few scans and have the whole dataset automatically cleaned.

Improved smoothing: The heuristic approach we used to smooth the classification image appears to work fairly well, given the limited and complex data we have available per pixel. Nonetheless, it would be interesting to see whether a carefully crafted CRF/MRF could be used to further improve the quality of the final classification image.

It would also be interesting to try other classifiers, with the caveat being that they need to train and predict quickly enough to ensure the system remains useful. Additional features could be explored, including image-based features based on texture, since these may be less subject to colour or intensity return variations. Although adding features will generally increase training time, a detailed analysis of feature importance could determine which features in the extended set are truly useful and a smaller, but information dense, set of features could be determined. Finally, it would be interesting to see how well the framework functions when applied to non-CH scanning campaigns.

ACKNOWLEDGMENTS

Finally, we would also like to thank the following groups for access to their point-cloud data:

- Em.-Prof. Heinz Rüther and the Zamani Project (*http://www.zamaniproject.org/*) for Songo Mnara;
- Visual Computing Group at CNR-ISTI (Pisa) for the data sets: Montelupo, Bagni di Nerone, Church, Monument, Stairs and Signoria;
- Stefan Lindgren from HumLab, University of Lund, for the Ballcourt data set.

REFERENCES

[1] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. 2004. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explor. Newslett.* 6, 1 (2004), 20–29.

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag TELOS, Santa Clara, CA.

[3] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ.

[4] Yuri Boykov, Olga Veksler, and Ramin Zabih. 2001. Fast approximate energy minimization via graph cuts. *IEEE Trans. Patt. Anal. Machine Intell.* 23, 11 (2001), 1222–1239.

[5] Leo Breiman. 2001. Random forests. *Machine Learn.* 45, 1 (2001), 5–32.

[6] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. 2018. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Trans. Patt. Anal. Machine Intell.* 40, 4 (2018), 834–838.

[7] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: An open-source mesh processing tool. In *Proceedings of the 6th Eurographics Italian Chapter Conference*. 129–136. http://vcg.isti.cnr.it/Publications/2008/CCCDGR08.

[8] David Doria and Richard J. Radke. 2012. Filling large holes in lidar data by inpainting depth gradients. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'12)*. IEEE, 65–72.

[9] J. Elseberg, D. Borrmann, and A. Nüchter. 2011. Full wave analysis in 3D laser scans for vegetation detection in urban environments. In *Proceedings of the 23rd International Symposium on Information, Communication and Automation Technologies.* 1–7. DOI: https://doi.org/10.1109/ICAT.2011.6102101

[10] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D. Wegner, Konrad Schindler, and Marc Pollefeys. 2017. Semantic3D. net: A new large-scale point cloud classification benchmark. Retrieved from *arXiv preprint arXiv:1704.03847.*

[11] Timo Hackel, Jan D. Wegner, and Konrad Schindler. 2016. Fast semantic segmentation of 3D point clouds with strongly varying density. *ISPRS Ann. Photogram., Remote Sens. Spatial Inform. Sci.* 3, 3 (2016).

[12] Jungong Han, Ling Shao, Dong Xu, and Jamie Shotton. 2013. Enhanced computer vision with Microsoft Kinect sensor: A review. *IEEE Trans. Cyber.* 43, 5 (2013), 1318–1334.

[13] Alexander Hermans, Georgios Floros, and Bastian Leibe. 2014. Dense 3D semantic mapping of indoor scenes from RGB-D images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'14).* IEEE, 2631–2638.

[14] Jing Huang and Suya You. 2016. Point cloud labeling using 3D convolutional neural network. In *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR'16).* IEEE, 2670–2675.

[15] Juha Hyyppä, Anttoni Jaakkola, Yuwei Chen, and Antero Kukko. 2013. Unconventional lidar mapping from air, terrestrial and mobile. In *Proceedings of the Photogrammetric Week Conference.* 205–214.

[16] Byung-soo Kim, Pushmeet Kohli, and Silvio Savarese. 2013. 3D scene understanding by Voxel-CRF. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'13).* IEEE, 1425–1432.

[17] J. Kisztner, J. Jelínek, T. Daněk, and J. Ružička. 2016. 3D documentation of outcrop by laser scanner—Filtration of vegetation. *Perspect. Sci.* 7 (2016), 161–165. DOI: https://doi.org/10.1016/j.pisc.2015.11.026

[18] Hema S. Koppula, Abhishek Anand, Thorsten Joachims, and Ashutosh Saxena. 2011. Semantic labeling of 3D point clouds for indoor scenes. In *Proceedings of the Conference on Advances in Neural Information Processing Systems.* 244–252.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems.* 1097–1105.

[20] Rickert Mulder and Patrick Marais. 2016. Accelerating point cloud cleaning. In *Proceedings of the 14th Eurographics Workshop on Graphics and Cultural Heritage (GCH'16).* Eurographics Association, 211–214. DOI: https://doi.org/10.2312/gch.20161410

[21] Daniel Munoz, Nicolas Vandapel, and Martial Hebert. 2008. Directional associative Markov network for 3-D point cloud classification. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission.*

[22] Guan Pang and Ulrich Neumann. 2013. Training-based object recognition in cluttered 3D point clouds. In *Proceedings of the International Conference on 3D Vision (3DV'13).* IEEE, 87–94.

[23] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. Pointnet: Deep learning on point sets for 3D classification and segmentation. *Proc. Comput. Vis. Patt. Recog., IEEE* 1, 2 (2017), 4.

[24] Heinz Rüther, Christoph Held, Roshan Bhurtha, Ralph Schroeder, and Stephen Wessels. 2012. From point cloud to textured model, the Zamani laser scanning pipeline in heritage documentation. *South African J. Geomat.* 1, 1 (2012), 44–59.

[25] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. 2009. On-line random forests. In *Proceedings of the 12th IEEE International Conference on Computer Vision Workshops (ICCV'09).* IEEE, 1393–1400.

[26] Richard Socher, Brody Huval, Bharath Bath, Christopher D. Manning, and Andrew Y. Ng. 2012. Convolutional-recursive deep learning for 3D object classification. In *Adv. Neural Inform. Proc. Syst.* 656–664.

[27] Rudolph Triebel, Kristian Kersting, and Wolfram Burgard. 2006. Robust 3D scan point classification using associative Markov networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'06).* IEEE, 2603–2608.

[28] Julien Valentin, Vibhav Vineet, Ming-Ming Cheng, David Kim, Jamie Shotton, Pushmeet Kohli, Matthias Niessner, Antonio Criminisi, Shahram Izadi, and Philip Torr. 2015. SemanticPaint: Interactive 3D labeling and learning at your fingertips. *ACM Trans. Graph.* 34, 5, Article 154 (Nov. 2015), 17 pages. DOI: https://doi.org/10.1145/2751556

[29] Martin Weinmann, Boris Jutzi, Stefan Hinz, and Clément Mallet. 2015. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS J. Photogram. Remote Sens.* 105 (2015), 286–304.

[30] Daniel Wolf, Johann Prankl, and Markus Vincze. 2015. Fast semantic segmentation of 3D point clouds using a dense CRF with learned parameters. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'15).* IEEE, 4867–4873.

[31] Jingyu Yang, Ziqiao Gan, Kun Li, and Chunping Hou. 2015. Graph-based segmentation for RGB-D data using 3-D geometry enhanced superpixels. *IEEE Trans. Cyber.* 45, 5 (2015), 927–940.

[32] Na-Eun Yang, Yong-Gon Kim, and Rae-Hong Park. 2012. Depth hole filling using the depth distribution of neighboring regions of depth holes in the Kinect sensor. In *Proceedings of the IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC'12).* IEEE, 658–661.

[33] Zhi-Hua Zhou and Ji Feng. 2017. Deep forest: Towards an alternative to deep neural networks. Retrieved from *arXiv preprint arXiv:1702.08835.*

Received March 2018; revised October 2018; accepted November 2018