Detecting and Mapping Trees in Unstructured Environments with a Stereo Camera and Pseudo-Lidar

Brian H. Wang, Carlos Diaz-Ruiz, Jacopo Banfi, and Mark Campbell[1]

Abstract— We present a method for detecting and mapping trees in noisy stereo camera point clouds, using a learned 3-D object detector. Inspired by recent advancements in 3-D object detection using a pseudo-lidar representation for stereo data, we train a PointRCNN detector to recognize trees in forest-like environments. We generate detector training data with a novel automatic labeling process that clusters a fused global point cloud. This process annotates large stereo point cloud training data sets with minimal user supervision, and unlike previous pseudo-lidar detection pipelines, requires no 3-D ground truth from other sensors such as lidar. Our mapping system additionally uses a Kalman filter to associate detections and consistently estimate the positions and sizes of trees. We collect a data set for tree detection consisting of 8680 stereo point clouds, and validate our method on an outdoors test sequence. Our results demonstrate robust tree recognition in noisy stereo data at ranges of up to 7 meters, on 720p resolution images from a Stereolabs ZED 2 camera. Code and data are available at **https://github.com/brian-h-wang/pseudolidar-tree-detection**.

I. INTRODUCTION

Agile autonomous navigation in complex and unstructured environments requires a robot to plan ahead and process noisy sensor measurements, in order to determine the existence and locations of any obstacles in its planned path. Consider a quadrotor UAV flying through a forest—tree trunks and branches may appear in the quadrotor's flight path with little warning, and agile navigation therefore depends on the quadrotor's ability to use noisy sensor data to reason about the uncertain presence or absence of obstacles ahead.

Current state of the art approaches to agile autonomous navigation typically use either lidar [1], [2] or stereo cameras [3]. Lidar sensors, while accurate, are heavy, power-hungry, and expensive. Stereo cameras are more lightweight and less costly, but the quality of stereo depth information degrades significantly with range [4]. For this reason, mapping approaches designed for obstacle avoidance based on stereo cameras generally discard sensor measurements past a limited range, and/or limit the maximum robot speed in complex or crowded environments where sensor noise is more prevalent. Additionally, the computational cost of updating 3-D occupancy grid maps grows significantly as the maximum mapping range and/or grid resolution is increased, creating further difficulty for planning around obstacles in complex environments.
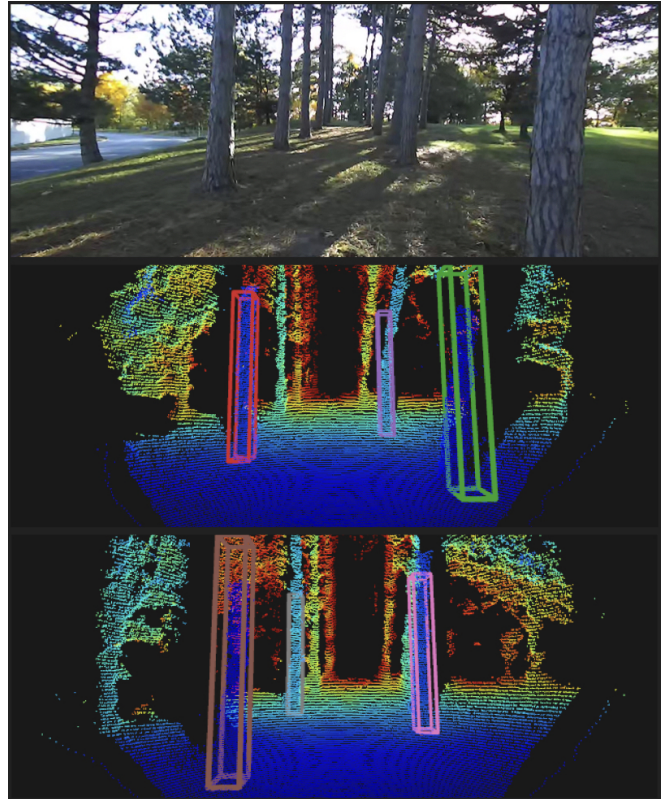
Fig. 1. Results of our tree detection and mapping system, demonstrated on sample frames in our test sequence. The colored bounding boxes indicated different tracked objects within our tracker.

Motivated by the problem of autonomous navigation in unstructured outdoors environments such as forests, and inspired by recent advancements in 3-D object detection using a *pseudo-lidar* representation for stereo point clouds [4], [5], we present, to the best of our knowledge, the first method for detecting and mapping distant trees using a stereo camera alongside a state of the art 3-D object detector. Figure 1 demonstrates example results of our method, which includes the following components:

1) A labeling pipeline which generates a training data set for the tree detector from recorded stereo camera sensor data. This pipeline allows rapid training data creation with minimal user effort by clustering a fused global point cloud map.
2) A PointRCNN 3-D object detector [6], which detects trees in sparsified stereo point clouds, using the novel pseudo-lidar approach proposed in Wang *et al.* [4].
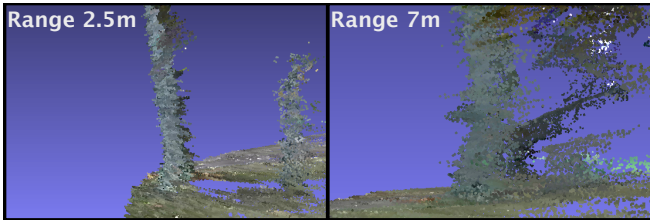3) A Kalman filter, which performs data association on

Fig. 2. Example of the distortion that occurs in stereo point clouds at longer sensing ranges, using data from a ZED 2 stereo camera. *Left:* Side view of a tree at a range of approximately 2.5 meters from the sensor, as seen in the 3-D point cloud generated from a stereo image pair. *Right:* The same tree, at a range of 7 meters, appears significantly more distorted.

detections in order to robustly map trees. The filter outputs accurate estimates of the positions and sizes of trees in front of the camera, and is scalable due to estimating a limited set of state variables per tree.

By learning patterns from large numbers of labeled noisy stereo point clouds, our detection and mapping system is able to recognize trees that appear distorted in stereo data, an example of which is shown in Figure 2. We note that distorted objects such as this are infeasible to manually annotate with bounding box labels for training a 3-D detector, as there is no unbiased way for someone to judge the true position and size of the tree. For this reason, previous pseudo-lidar detection methods use training labels that come from annotated lidar data [4], [5]. Our automatic labeling method avoids this problem, enabling the first stereo image training and detection pipeline with no dependence on lidar sensors. Our method is also modular, and can complement existing 3-D occupancy grid mapping approaches.

We validate our perception system on a test sequence of stereo point clouds captured in a forest-like environment, demonstrating accurate tree detection and mapping at ranges up to 7 meters away from the camera, where stereo depth noise becomes significant. To further promote creation of data sets and obstacle detectors for forests and other challenging environments for robotic navigation, our code for point cloud labeling, pseudo-lidar object detection, and tree mapping, as well as our collected data set, are publicly available at `https://github.com/brian-h-wang/pseudolidar-tree-detection`.

II. RELATED WORK

A. Perception for Navigation in Unstructured Environments

Recent works have demonstrated robust and effective perception pipelines for enabling autonomous outdoors navigation. Ryll *et al.* [3] use an Intel RealSense D435i stereo camera to construct a 3-D occupancy grid that a UAV can use for navigation. However, the occupancy grid requires consistent sensor measurements to form a reliable map for planning; as a result, while the quadrotor could achieve peak speeds of 8 m/s in open areas, flight speed was greatly reduced around obstacles and narrow passageways. Mohta *et al.* [1] use a similar perception component, the main difference being the usage of a nodding 2D lidar scanner

as the main sensor for mapping. Due to computational considerations, lidar measurements are fused into a local occupancy grid map covering a distance only up to 7.5m away from the quadrotor. These navigation pipelines have demonstrated impressive results in autonomous quadrotor flight, however the difficulty of using longer-range sensor measurements remains a roadblock towards faster flight in more complex environments, with [1] specifically identifying the size of the local map as a limiting factor on flight speed.

Other works have explored the problems of detecting trees and navigating in forests specifically. Giusti *et al.* [7] use a convolutional neural network for detecting forest trails, allowing a quadrotor to navigate through the forest by visually following the trail. Mohta *et al.* [2] demonstrate quadrotor navigation in a sparse forest environment, and Tian *et al.* [8] present a multi-UAV exploration framework using trees as landmarks for SLAM, clustering trees in order to reduce the computational burden of mapping a forest using an occupancy grid. Both of these pipelines use lidar sensing for mapping and tree detection. Finally, Durand-Petiteville *et al.* [9] present a method for detecting tree trunks in orchards using stereo cameras, by reasoning about the shadows that tree trunks cast within the stereo point cloud. This method is computationally efficient and demonstrates that trees can be effectively recognized in stereo point clouds, however since it is designed specifically for usage in orchards, it depends on a number of heuristic assumptions about the shapes and sizes of the trees.

B. 3-D Pseudo-lidar Object Detection

Motivated in large part by autonomous driving, researchers have made significant progress in perception and scene understanding with 3-D sensors [10], [11], [6], [12], [13]. Recent works have greatly narrowed the gap between 3-D object detectors that use lidar, and those that use stereo images as inputs. Wang *et al.* [4] showed that this gap was caused mainly by the representation used for stereo data, rather than the quality of depth estimation, as was previously assumed. By representing stereo depth images as a 3-D point cloud, then using the resulting point cloud as an input to a 3-D lidar object detector, they achieved massively improved detection accuracy over previous stereo image object detectors. Wang *et al.* refer to this representation as *pseudo-lidar*, since the stereo point cloud mimics the measurements produced by a lidar sensor. You *et al.* [5] introduced improvements that further increased the accuracy of stereo-based 3-D detection. Due to the reduced cost and weight of stereo cameras as compared to lidar, these advancements are especially impactful for UAVs and other resource-constrained robotic platforms.

III. APPROACH

Our approach to tree detection and tracking consists of three main components, described in detail in the following sections: a pipeline for generating training data (Section III-A), a 3-D tree detector (Section III-B), and a tree state estimator (Section III-C).
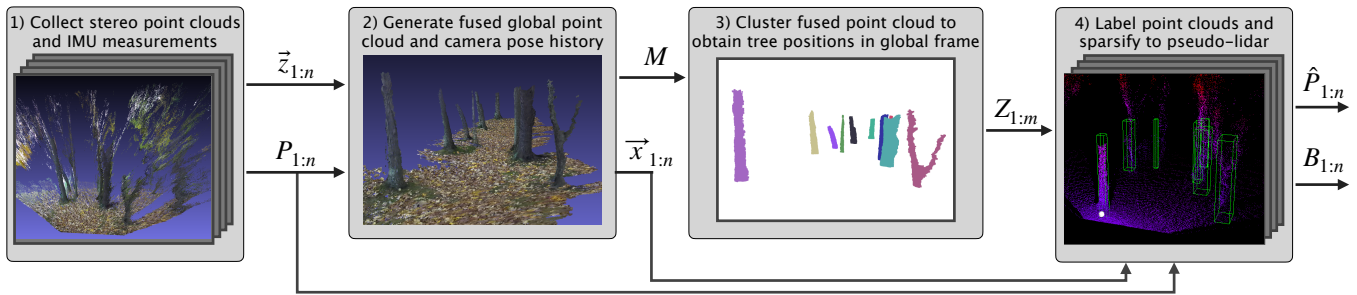
Fig. 3.   Our pipeline for generating labeled stereo point cloud training data for the 3-D tree detector. See section III-A for definitions of terms.

A. Training Data Generation Pipeline

In the following section, we describe our pipeline for generating stereo image training data with minimal user supervision. Figure 3 illustrates the steps of this process. We implement our pipeline in Python, using the Open3D [14] library for 3-D data processing.

1) Data collection: Our approach uses these inputs:

- P_k, point clouds from a 3-D camera sensor.
- f_x, f_y, c_x, c_y, the camera intrinsic parameters.
- $\vec{z}_k$, measurements from an IMU and any accompanying sensors used for camera pose estimation. These are used to ensure accurate mapping and localization, which are required for generating reliable training labels.

For the camera point clouds and other sensor data, the index k is over the time step indices in the data collection, $k \in \{1, \ldots, n\}$. We use a Stereolabs ZED 2 stereo camera for our data collection. As described in [4], given a pair of stereo images, the point cloud P_k can be computed by matching pixels in the left image to corresponding pixels in the right image, then computing a depth value to each pixel based on the camera parameters.

We note that active RGBD sensors such as the Intel RealSense or Microsoft Kinect can also be used to obtain the required 3-D point cloud measurements and are therefore compatible with our pipeline. However, these sensors tend to perform less accurately outdoors due to their usage of infrared sensors. Thus, the ZED 2 stereo camera better addresses the use case we explore in this paper.

2) Global point cloud fusion and pose estimation: Given the sensor measurements and camera parameters as inputs, we compute the following:

- $\vec{x}_k$, the pose of the camera at each time step.
- $M \in \mathbb{R}^{p_G \times 3}$, a *global point cloud map* created by fusing all P_k into a global reference frame. p_G is the total number of points in the global point cloud.

Multiple approaches for visual-inertial 3-D SLAM have been proposed, which generate the pose and point cloud map given a series of 3-D point clouds and inertial measurements [15], [16], [17], [18]. Any of these approaches can, in principle, be used to generate the required inputs for our labeling pipeline. Our current implementation uses the spatial mapping API of the Stereolabs ZED SDK[1], which uses

[1] https://www.stereolabs.com/docs/spatial-mapping/

stereo vision alongside the built-in IMU, magnetometer, and barometer on the ZED 2 sensor to perform point cloud fusion and pose optimization.

3) Global point cloud clustering: After constructing the global point cloud, we apply a clustering algorithm in order to determine the number, shape, and position of individual tree trunks in the point cloud, proceeding as follows:

1) Detect the ground plane in the global point cloud using RANSAC, and remove any points within a distance threshold to the ground.
2) Cluster the remaining points using the DBSCAN clustering algorithm [19]. The output of this step is a list of clusters $Z_i \in \mathbb{R}^{p_i \times 3}$, where p_i is the number of points in cluster i.
3) Remove any cluster i where $p_i < p_{\min}$, a threshold on cluster size. This filters out small erroneous clusters from sources such as low-hanging foliage.
4) Output the remaining tree clusters $Z_i, i \in \{1, \ldots, m\}$.

This clustering approach is sensitive to noise in the 3-D points; thus, it cannot be used to effectively detect trees in the individual noisy stereo point clouds. However, noise in the fused global point cloud is significantly reduced in comparison, and the global cloud can therefore be accurately clustered to determine the number, shape, and position of individual tree trunks, as demonstrated in Figure 3.

In addition, we have developed a labeling tool which can be optionally used to review the generated clusters, remove any incorrect clusters, and manually annotate any trees missed by the DBSCAN clustering by drawing bounding boxes around groups of unclustered points in the global point cloud (with the ground plane removed). This step ensures the generated clusters are free of errors, which would otherwise propagate into the detector training data.

4) Automated label generation and point cloud sparsification: Given a cluster of 3-D points Z_i for each tree in the global frame, we can step through the individual stereo point clouds P_k and, using the computed pose of the camera $\vec{x}_k$ in the global frame, transform the clusters Z_i to the local camera frame to determine which trees are visible at time k.

Our label generation procedure is as follows. For each time step $k \in \{1, \ldots, n\}$:

1) Initialize the set of bounding box annotations for frame k to $B_k = \emptyset$.

2) Using the computed pose (position and orientation) of the camera $\vec{x}_k$, define the homogeneous transformation matrix $T_{W \to C,k}$ which transforms points from the global world frame to the local camera reference frame.

3) For each cluster of 3-D points $Z_i, i \in \{1, \ldots, m\}$, use $T_{W \to C,k}$ to transform the cluster to its representation in camera frame coordinates, which we denote as Z_i^C.

4) Fit a 3-D rectangular bounding box $b_{i,k}$, aligned to the axes of frame C, around the points Z_i^C. We align boxes to C, because otherwise the correct orientation of the box is undefined, since trees have no clear front side.

5) If no points in the local point cloud P_k fall within $b_{i,k}$, discard $b_{i,k}$. This removes trees which are outside the sensing range of the camera, or are completely occluded and therefore impossible to detect. Otherwise, add $b_{i,k}$ to the set B_k of labels for point cloud P_k.

In addition, following You *et al.* [5], we sparsify each stereo point cloud P_k into a sparse pseudo-lidar representation $\hat{P}_k$ by downsampling P_k to 128 scan lines, evenly distributed according to vertical angle to the sensor, from $\theta = -35°$ to $\theta = +35°$ (covering the vertical field of view of the ZED 2 camera). We choose 128 scan lines because You *et al.* used 64 scan lines to imitate a lidar with a vertical field of view of approximately 30°; since the ZED 2 vertical field of view is 70°, doubling the number of pseudo-lidar scan lines maintains a roughly similar density of scan lines. The sparsification step is required because state of the art methods for 3-D object detection assume lidar point cloud inputs, and exploit the sparsity property of lidar data [6]. Sparsification has the added advantage of greatly reducing the size of the stereo point clouds—sparsifying ZED 2 stereo point clouds generated using 720p resolution stereo images results in an over tenfold decrease in the number of points—significantly reducing the storage requirements for a large data set.

The end result of this pipeline is a set of sparse stereo point clouds $\hat{P}_k$, each labeled with a set of ground truth bounding boxes B_k indicating tree positions and sizes, to be used for 3-D object detector training. By automatically clustering a global point cloud, then optionally having a user spend a few minutes reviewing and fixing the clustering results, we can generate annotation bounding boxes for the hundreds or thousands of individual stereo point clouds used to create the fused global point cloud. We note that even when the user needs to add additional manual annotations, annotating a group of points in the ground plane-removed global point cloud takes only a few seconds of user time, and this single annotation is then used to label many tree instances in the individual sparse point clouds $\hat{P}_k$.

Our approach assumes a forest environment, with all obstacles in the point cloud map being trees. In order to consider more general environments, with multiple object classes present, a potential extension to our pipeline is an extra labeling step where a user manually reviews each computed cluster and determines an appropriate class from a list of predefined options. This process would require little extra user time on top of our current pipeline, and would enable training multi-class 3-D object detectors.

B. 3-D Object Detector

We use the PointRCNN detector, developed by Shi *et al.* [6] and publicly released online[2], to detect trees in 3-D stereo point clouds. Previously, You *et al.* [5] applied PointRCNN to stereo point clouds, using the sparse pseudo-lidar representation. Our training data for the detector consists of the sparsified local stereo point clouds $\hat{P}_k$, as well as the 3-D tree bounding boxes $B_{i,k}$ generated through our labeling pipeline.

C. Kalman Filter Estimator

We treat 3-D bounding box detections from PointRCNN as uncertain measurements in a Kalman filter mapping framework, in order to consistently estimate tree positions and sizes and account for missed detections. Our goal is to estimate the joint probability $p(\mathbf{a}_k, \mathbf{t}_k | \mathbf{d}_k)$ at time k, where $\mathbf{t}_k$ is the set of estimated tree states (position and size), $\mathbf{d}_k$ is the set of observed sensor measurements (frame-wise bounding box detections), and $\mathbf{a}_k$ the assignment of measurements to mapped trees. We decouple the joint distribution into the continuous estimation problem $p(\mathbf{t}_k | \mathbf{a}_k, \mathbf{d}_k)$, solved recursively via a Kalman Filter (KF), and the discrete data assignment problem, $p(\mathbf{a}_k | \mathbf{t}_k, \mathbf{d}_k)$, solved via global nearest-neighbors (GNN).

1) Kalman Filter: We define a state vector for each tree based on its global frame position and size,

$$\mathbf{t}_k^i = \begin{bmatrix} x & y & w \end{bmatrix}^T, \tag{1}$$

where x, y is the position of the center of the tree in top-down birds-eye view coordinates, w is the tree diameter, and $\mathbf{t}_k^i$ corresponds to the i-th tree at the k-th time. For the rest of the paper, the superscript i is omitted for clarity; but it is given that multiple trees are being tracked. The x-axis points along the direction the camera is facing, and the y axis points to the left. Our state vector effectively represents each tree as a cylinder, ignoring its height. We deem this an appropriate choice for representing tree trunks in our use case, as we would like to enable a robot to navigate between tree trunks in a forest, rather than flying over the tree canopies.

We assume the states of the model for each tree to be constant relative to the global frame, and driven by a small amount of process noise

$$\dot{\mathbf{t}}_k = \begin{bmatrix} e_x & e_y & e_w \end{bmatrix}^T \tag{2}$$

where e_x, e_y, e_w are process noise values with small covariances which can be tuned to balance the time response of the filter and the pose error. This noise allows the state estimates to adjust over time as additional data arrives.

The Kalman Filter prediction step is

$$\hat{\mathbf{t}}_{k+1} = \mathbf{A}\hat{\mathbf{t}}_k \tag{3}$$

$$\hat{\Sigma}_{k+1} = \mathbf{A}\hat{\Sigma}_k\mathbf{A}^T + \mathbf{Q} \tag{4}$$

where $\hat{\mathbf{t}}$ is the state estimate, $\mathbf{A}$ is the state transition matrix (in this case identity), $\hat{\Sigma}$ is the state error covariance, and $\mathbf{Q}$ is the process noise covariance matrix.

[2]https://github.com/sshaoshuai/PointRCNN

Each measurement consists of direct measurements of all three states, derived from the bounding box detector. More specifically, we extract the following measurement vector $\mathbf{d}_{k+1}^m$ for the m-th measurement at the $k+1$ time:

$$\mathbf{d}_{k+1}^m = \begin{bmatrix} d_x & d_y & d_w \end{bmatrix}^T, \tag{5}$$

where d_x, d_y denote the center-bottom position of the bounding box relative to the camera coordinate system, and d_w is its width. We treat the width of a bounding box as a noisy measurement of the corresponding tree's diameter. The measurement function from the detections $h_d(\cdot)$ is

$$\mathbf{d}_{k+1} = \mathbf{H}\mathbf{t}_{k+1} + \mathbf{r}_{k+1}. \tag{6}$$

The measurement function is modeled with additive zero-mean Gaussian white noise $\mathbf{r}_{k+1}$. This yields the predicted measurement and corresponding innovation covariance to be:

$$\hat{\mathbf{d}}_{k+1} = \mathbf{H}\hat{\mathbf{t}}_{k+1} \tag{7}$$

$$\mathbf{S}_{k+1} = \mathbf{H}\Sigma_{k+1}\mathbf{H}^T + R. \tag{8}$$

The tree location and diameter estimates are then updated according to the standard Kalman Filter update equations.

2) Data Association: We use the GNN algorithm to optimally assign measurements to tracks, using the Hungarian algorithm [20] to minimize the Mahalanobis distance

$$D_{k+1}^{j,i} = (\mathbf{d}_{k+1}^j - \hat{\mathbf{d}}_{k+1}^i)^T S_{k+1}^{-1}(\mathbf{d}_{k+1}^j - \hat{\mathbf{d}}_{k+1}^i), \tag{9}$$

where the j-th measurement is compared to the i-th estimated measurement, derived from the i-th track, and S is the corresponding innovation covariance matrix.

Tracks are initialized when a fixed number a_{minHits} of measurements are associated to the same track; our current implementation has $a_{\text{minHits}} = 3$. Track deletion, on the other hand, requires that a current track does not receive any measurement updates over a set amount of time steps a_{maxAge}, or that the object exits the field of view (FOV); we use $a_{\text{maxAge}} = 100$. For our 60FPS camera, this equates to 1.67 seconds. This method of track deletion addresses short temporary occlusions. We use χ^2 hypothesis testing for validation gating, where very unlikely ($<5\%$) measurement associations are discarded as clutter or false positives.

IV. EXPERIMENTS AND RESULTS

A. Data collection and processing

In order to train our tree detection system, we collected data in forest-like environments near the Cornell University campus, using a handheld Stereolabs ZED 2 camera to gather over 40,000 stereo image pairs at 720p resolution, split up into seven individual sequences. Due to the camera's 60 FPS framerate, in order to avoid including multiple highly similar frames in our detector training data, we downsampled the data to only include every fifth frame in our training set, for a set of 8680 stereo point clouds in total. In terms of number of frames, this is similar to the popular 3-D object detection KITTI data set, which includes 7481 training frames [21].

Each of the seven data capture sequences was fused into a separate global point cloud, which were each then
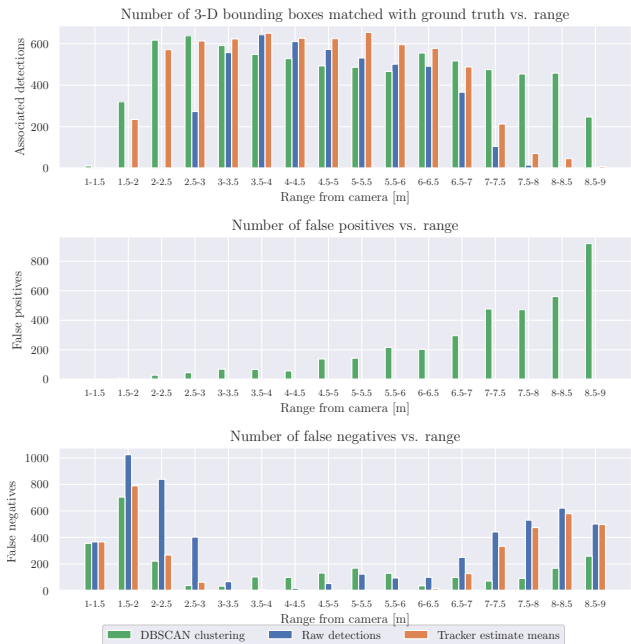

Fig. 4. Counts of successfully matched 3-D bounding boxes, false positives, and false negatives, shown binned over different ranges from the sensor. We compare raw detector bounding box outputs and tracker estimates with baseline detections generated by running DBSCAN clustering on noisy stereo point clouds.

clustered using our automatic annotation pipeline. For the ground plane removal, we ran RANSAC for 1000 iterations, and removed any points within 0.5 m of the fit plane. For the global point cloud clustering, we set the DBSCAN parameters to $\varepsilon = 0.1$ and *minSamples* $= 10$. Finally, after DBSCAN clustering, we remove any clusters containing less than 2000 points (To give a sense of scale, the full global point clouds from our seven training data captures each contained between 1.1 and 2.2 million points, though the majority of these would be removed with the ground plane). When creating the sparsified pseudo-lidar point clouds, we also remove stereo points past 15 meters from the camera, to avoid creating training bounding boxes that include only very few 3-D points.

Our training data sequences contain 110 distinct trees in total. 98 of these were correctly automatically clustered by our global point cloud clustering process. The automatic clustering generated 5 incorrect clusters which a manual annotator removed. Finally, the manual annotator added 12 clusters which were missed by the DBSCAN clustering—most of these were very thin tree trunks which fell under the minimum threshold for DBSCAN cluster sizes. In total, we trained PointRCNN on 8680 pseudo-lidar point clouds annotated with 34,386 bounding box training labels.

B. Evaluating tree detection and estimation accuracy

We evaluate our trained PointRCNN tree detector and tracker on a test sequence of 2000 stereo point clouds, captured in a separate forest-like environment. This site contains two rows of trees, and is physically separated from
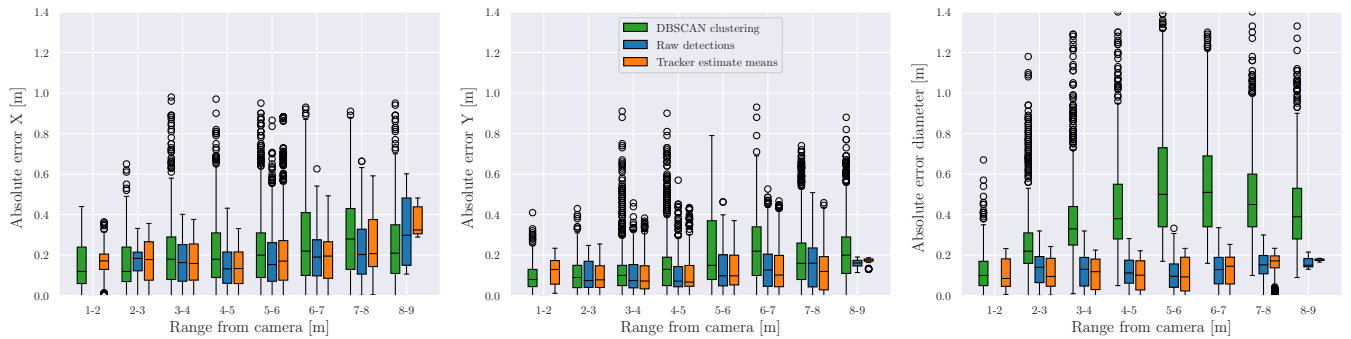
Fig. 5. Absolute errors on *x* (forward axis) position, *y* (lateral axis) position, and tree diameter over different ranges. An absent bar indicates no bounding boxes present over a given range bin.

the area where we collected our training data, in order to evaluate our detector's performance in an environment outside the training domain. Figure 1 visually shows our test site; for full results of our tree mapping system, please see the video accompanying this paper. We create ground truth tree annotations of this outdoors test sequence using our clustering and bounding box generation process, additionally using our manual annotation tool to ensure correct labels.

As a baseline for comparison, we run our ground plane removal and DBSCAN clustering method directly on the individual stereo point clouds P_k, fit a bounding box around each DBSCAN cluster, and treat these as detections. This baseline exactly mimics our clustering procedure used for automatic training data generation, except applied on noisy stereo point clouds rather than on the fused global point cloud. We leave all parameters used for this baseline clustering identical to the parameters used for generating training labels for the detector, except for running RANSAC ground plane removal for only 50 iterations (this saves computation time and did not significantly affect clustering, since the ground area in P_k is much smaller than in the global point cloud and therefore does not require as precise a plane fit).

Figure 4 shows the total number of bounding boxes successfully associated with ground truth boxes (top), false positive bounding boxes (middle), and false negatives (bottom) for the baseline DBSCAN detections, tree detector and the tracker. For evaluation, we match bounding boxes to nearby ground truth instances using linear assignment, with a maximum distance cutoff of 1 meter. This evaluation analyzes how often our detection and mapping system successfully recognizes the presence of trees ahead. We bin statistics over various ranges, in order to analyze how performance changes with distance from the sensor. As seen in the results, the DBSCAN baseline correctly recognizes the majority of ground truth trees at shorter ranges, when depth errors are minimal. Past around 5 meters, however, the number of false positive DBSCAN bounding boxes grows rapidly, as DBSCAN begins to mistakenly find clusters in noisy areas of the stereo point clouds. In contrast, our detector, trained on data generated largely by the same DBSCAN clustering method applied to a fused global point cloud, does not raise these false postives, and demonstrates a low false negative

rate up to around 7 meters, significantly further than the accurate range of DBSCAN point cloud clustering. Past 5 meters, the Kalman filter effectively compensates for missed detections, reducing the false negative rate of the system. Our detector noticeably misses trees at very short ranges; at close distances, the stereo point cloud is very dense, which may make shorter-range detections difficult for PointRCNN, which was designed with lidar data in mind, to learn. This is however a minor drawback, since at such a close range depth measurements are quite reliable and could be easily integrated into a local grid-based map.

In order to examine the quality of tree bounding boxes, Figure 5 reports the errors in *x*- and *y*-position and tree size for the DBSCAN baseline bounding boxes, PointRCNN detections, and Kalman filter estimates. Errors for both the PointRCNN detections and filtered tree states remain consistently low over the ranges shown, indicating that the detections closely match the ground truth positions and sizes of trees in the test sequence. In contrast, the DBSCAN baseline demonstrates low error up to around 3 meters, but begins to drastically grow in error after this point, with errors in tree sizes becoming particularly large.

V. CONCLUSION

We present a method for detecting and mapping trees in stereo camera point clouds. In our experiments, this system accurately recognizes trees at ranges up to 7 meters, even when trained on labels generated from clustering which is accurate in noisy stereo point clouds only up to approximately 3-4 meters. Our automatic labeling pipeline enables a detector to learn to recognize trees in noisy longer-range stereo points, which are often discarded in current mapping approaches, increasing the extent of usable sensor information available to the robot. Additionally, our pipeline bypasses the need for time-consuming manual labeling, and does not depend on supervision from a lidar sensor. Future work will focus on further extending the range of learned 3-D object detectors, enabling robots to navigate while accounting for distant uncertain obstacles.

ACKNOWLEDGMENT

REFERENCES

[1] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico *et al.*, "Fast, autonomous flight in gps-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[2] K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Experiments in fast, autonomous, gps-denied quadrotor flight," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7832–7839.

[3] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 732–738.

[4] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8445–8453.

[5] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving," *arXiv preprint arXiv:1906.06310*, 2019.

[6] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.

[7] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2015.

[8] Y. Tian, K. Liu, K. Ok, L. Tran, D. Allen, N. Roy, and J. P. How, "Search and rescue under the forest canopy using multiple uas," in *International Symposium on Experimental Robotics*. Springer, 2018, pp. 140–152.

[9] A. Durand-Petiteville, E. Le Flecher, V. Cadenat, T. Sentenac, and S. Vougioukas, "Tree detection with low-cost three-dimensional sensors for autonomous navigation in orchards," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3876–3883, 2018.

[10] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 7652–7660.

[11] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "Rangenet++: Fast and accurate lidar semantic segmentation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4213–4220.

[12] P. Kaul, D. De Martini, M. Gadd, and P. Newman, "Rss-net: Weakly-supervised multi-class semantic segmentation with fmcw radar," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 431–436.

[13] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 529–10 538.

[14] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[15] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[16] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 Ieee/rsj International Conference on Intelligent Robots and Systems (iros)*, 2017, pp. 1366–1373.

[17] M. Grinvald, F. Furrer, T. Novkovic, J. J. Chung, C. Cadena, R. Siegwart, and J. Nieto, "Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3037–3044, July 2019.

[18] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1689–1696.

[19] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[20] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.

[21] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.