Procedural Generation of Synthetic Forest Environments to Train Machine Learning Algorithms

Rui Nunes[1], João Ferreira[2] and Paulo Peixoto[3]

***Abstract*— The demand for the development of forestry robotics has been increasing. As with most robotics applications, Machine Learning is the engine driving innovation in this field. However, Machine Learning development for robotic perception tasks is highly dependent on the availability of annotated datasets. Contrasting with urban environments, public datasets for forest applications are rare, hard to collect and currently not enough to train models capable of operating autonomously. This paper proposes a solution to mitigate the data shortage problem: a system that uses procedural generation to create virtual forests and collects synthetic data from these environments using virtual sensors. More specifically, the system generates RGB images and point clouds with pixel-wise and point-wise annotations, respectively, as well as depth maps, substantially reducing the time and effort invested in dataset construction. The system proved capable of generating 1000 frames with all the above-mentioned data types in 3 hours of autonomous operation. The generated data is ready to be used in Machine Learning model training. Finally, qualitative preliminary results obtained by a semantic segmentation model trained on the generated dataset, which has been made publicly available in a community-wide repository, are presented.**

I. INTRODUCTION

Forestry has a substantial importance in the economy of many industrial countries [1] [2]. However, we are facing an increasing lack of manpower due to low salaries (especially taking into account the harshness of its operations), and also the progressive abandonment of rural areas and of practices such as pastoralism. One way of solving these problems would be by introducing and developing (semi-)autonomous vehicles and robots, which would potentially reduce running costs and remove the many health hazards involved in forestry, while still keeping the human "in the loop". The need to develop robots and systems capable of autonomously performing tasks in field robotics applications that rely on robotic perception has increased over the past few years. Machine Learning (ML) is the technological backbone that is powering this development, however, many challenges arise when trying to combine an already proven technology (ML) in contexts where annotated training data is hard to get, due to the lack of experiments or the complexity of task itself. Deep Learning (DL) techniques (which are a subset of ML) are the go-to solution to perform the referred tasks.

[1] Institute of Systems and Robotics (ISR-UC) rui.jose.nunes@gmail.com

[2] João Filipe Ferreira is with the Department of Computer Science, School of Science and Technology, Nottingham Trent University, UK joao.ferreira@ntu.ac.uk

[3] Paulo Peixoto, University of Coimbra (UC), Institute of Systems and Robotics (ISR-UC), Electrical and Computer Engineering Department (DEEC), Coimbra, Portugal peixoto@isr.uc.pt

The lack of data is one of the most challenging problems to overcome. When creating a ML model one needs to gather data that represents that problem. Afterward, this data needs to be annotated to train the model on what it represents. The amount of data needed to achieve the desired level of performance is directly coupled with the problem at hand.

According to F. Lateef and Y. Ruichek [3], researchers are resorting to semi and weakly supervised methods making DL models less reliant, as opposed to investing time collecting datasets. For most applications, dataset construction is an enormous task that can render many research teams powerless due to time constraints.

For some applications, such as autonomous driving, there are many labeled datasets and tools to boost the development of ML algorithms. On the other hand, forest navigation and perception is an underdeveloped field with few or no datasets available to enable research and development. This is the reason why many ML applications are not further developed or even put into practice.

This paper presents a novel solution to attenuate this problem for the specific application of forestry robotics: a system capable of generating synthetic data from procedurally generated worlds to train ML algorithms for forest applications. Thanks to the power of procedural generation and virtual sensors, the generated data is automatically labeled and requires virtually no effort to acquire. The preliminary results presented in this paper show that this is an effective technique with the potential to boost the development of ML solutions for forest applications. The system generates Red Green Blue (RGB) images with pixel-perfect segmentation maps, depth maps and point clouds with point-wise labeling, as presented in Fig. 1. The proposed system was developed using the game engine Unity. Built for game development, Unity provides many of the needed tools such as graphics pipelines and support for implementing logic via scripting.

The main contribution of this work is to provide the ability to generate data to train ML models for forest applications without the need to spend time doing so. Additionally, researchers can modify the generated environments by simply tweaking the procedural generation algorithm parameters. Therefore, researchers can devote their time to improving ML models architecture while generating diverse datasets in the background.

II. PROBLEM FORMULATION

The lack of data, raw or labeled, to develop and train ML algorithms is a problem that keeps robotic perception solutions from reaching better, or even reliable, results. This

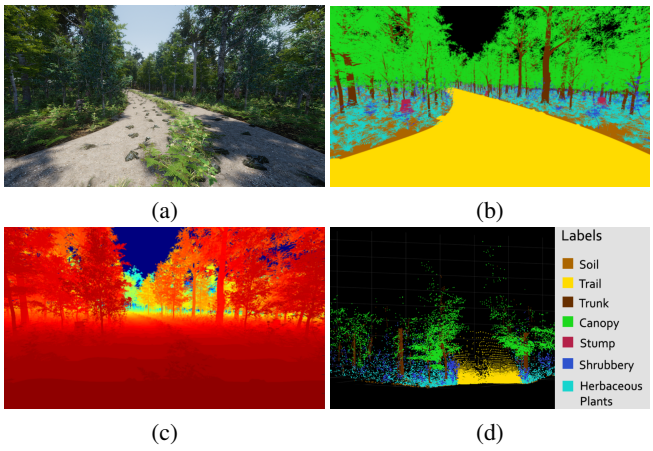

Fig. 1. Different types of data generated by the system for a single position of the virtual sensors. Labels apply to both point cloud and semantic segmentation map.

problem is the sum of a series of factors that will be discussed below.

A. Deep Learning is Data-Hungry

As previously mentioned DL is the most used field of ML to perform robotic perception tasks. DL models are data-hungry [4]. Their neural network structure allows them to be degrees of magnitude more independent than more traditional Artificial Intelligence (AI) algorithms, but this comes at the expense of needing additional training which is only possible with good and relatively big datasets.

The amount of annotated data needed for each problem is tightly coupled with the problem itself, meaning that models programmed to detect and deal with many features and objects will require much more data to cover all the cases needed to build the internal model. In a forest, the unstructured nature of the environment makes this a complex problem that requires substantial amounts of data to be tackled.

B. Acquiring Data

Collecting data implies operating sensible and power dependent equipment. In a forest, this is a laborious and time-consuming task, that, in turn, leads to a raw data shortage problem. In [5] and [6] the authors collect data in the forest and present the data collection setup, helping to understand the effort involved.

When collecting data, one needs to make sure that all the necessary cases are covered to generalize the model, which might not happen. In later stages, this might force the teams back to square one to solve overfitting or other poor performance problems. It is hard to be aware of the coverage of the already collected data, making this problem hard to counter. Authors of the Neural Network (NN) for semantic point segmentation in large point clouds named RandLaNet [7], state that classes that were less represented in the training dataset show poor performance. Making sure that the data collected covers all classes equally can be challenging, depending on the problem. In a forest environment, this becomes close to impossible, simply because there are a lot more objects of some kinds than others. For instance, the canopy is going to be everywhere, whereas rocks will only appear in some situations, leading to dataset imbalance problems.

C. Data Annotation

ML algorithms require datasets with labels to be trained. Annotation is the process of labeling data with a meaningful description. This process needs to be conducted or at least supervised by humans to ensure that the labels are correct. Incorrect labels are a huge problem in DL and, unfortunately, annotation is a very error-prone process due to the monotonicity of the task. C. Northcutt, A. Athalye and J. Mueller [8] analysed commonly-used computer vision, natural language, and audio datasets an estimate an average of 3.3% errors across them. NN trained on wrong labels, will learn those errors, deeply impacting performance.

There are many ways to annotate data. For RGB images, the most common is to draw a polygon mask or a more simple bounding box around each object, assigning it the correct label. This is a very heavy and tiring task to do for thousands or millions of images. Semi-automated tools can be used to speed up the process. Annotating images from forest environments becomes more complex than annotating images from urban scenarios, given that forests are unstructured environments with fine details.

Labeling point clouds is an even harder task. When looking at a point cloud on a computer screen, one feels the need to rotate and pan around the virtual environment to perceive shapes and identify objects, making it the most time-consuming task in the data processing pipeline. Point clouds taken from a forest setting are even harder to label, again, due to the unstructured nature of these environments. Mahony et al. [9] classifies dataset annotation as a huge bottleneck in the development of 3D DL.

D. In Practice - Semantic KITTI

Semantic KITTY [10] is a large dataset of Light Detection And Ranging (LiDAR) point clouds, based on the KITTI[11] Vision Odometry Benchmark, with added point wise annotations. The annotation process is detailed and allows us to extract some conclusions on the effort and techniques used.

First, multiple scans of the same area where superimposed in other to label all the points consistently. Then, all sequences of point clouds were divided into 100m by 100m tiles. For scans that were part of more than 1 tile, a small boundary is shown, relative to the neighbour tiles, when labeling. No bounding boxes or other annotations from the original KITTI dataset were used, in order to ensure that the point-wise labels are accurate.

The dataset consists of 23201 scans for training and 20351 for testing, totaling 4549 million points. The dataset took a total of 1700 hours of labeling effort or roughly 71 whole days.

III. RELATED WORK

The idea of creating a system capable of rendering and collecting datasets from a virtual scene has been explored for autonomous driving applications. Mostly, pre-built virtual worlds were used, without resorting to procedural generation. Nevertheless, the processing of capturing the data is similar.

Richter et al. [12] used the game GTA V as a source of data. A system capable of propagating labels from frame to frame was developed and allowed to generate a dataset with 25 thousand images and pixel-perfect semantic segmentation in 49 hours. A dataset composed of $2/3$ of game data and $1/3$ of CamVid [13] data outperformed models trained on the complete CamVid training set by 2.8 percentage points. This is a substantial improvement considering that the synthetic data was generated in a fraction of the time.

Yue et al. [14] also utilized the environment in the game GTA V, but to create annotated point clouds with point-wise annotation, again for autonomous driving application purposes. However, the used method generated point clouds with a low level of detail, since the developed system was limited by the game object's physics colliders. This problem was tackled by Hurl et al. in the generation of the Precise Synthetic Image and LiDAR (PreSIL) dataset [15], utilizing the depth buffers from the GPU to generate point clouds.

SynthCity [16] is a dataset created from 3D models of cities and other urban environments found online. The data is composed of point clouds with point-wise annotation. The entire virtual world was built by hand.

IV. PROCEDURAL GENERATION OF FORESTS

Procedural generation of the environment frees the user from the task of creating the environment with hand-placed objects, as done in [16]. The algorithms used to generate the environment rely on random number generators to assemble a different environment each time the system is used. These are configurable algorithms that can be tuned to produce different types of environments. This section will present the different modules that compose the procedural generation system, focusing on its interfaces: the parameters available to the user and the produced output, a virtual forest. This forest is seen through the sensors present a virtual entity that will hereby be designated as robot.

A. Chunk Manager

The first task of generating and managing a procedural world is to control what part of the forest is visible. In practice, this means creating, hiding and destroying the virtual world around the robot. To achieve this behavior the world is divided into finite pieces called chunks. The chunk manager is the sub-system that provides these features and is controlled by the following variables:

- **Robot position:** The robot position in the world;
- **Visible Radius:** Distance, in meters, around the robot for which chunks are visible;
- **Delete Radius:** Distance, in meters, around the robot from which chunks are not visible and can be deleted.

As the robot moves through the environment new chunks are created and the chunks left behind are disabled and eventually deleted. There is the need to delete chunks because memory is a finite resource and there is no gain in saving these chunks on permanent memory. As seen in Fig. 2, chunks in between the visible radius distance and the delete radius distance are disabled but not deleted since these are locations that the robot will likely visit again.

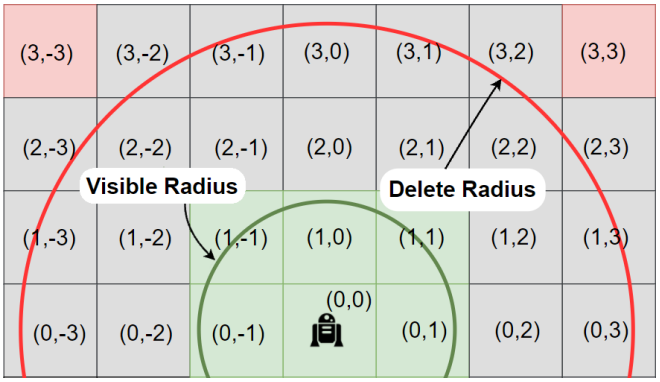

Fig. 2. Each square represents a chunk, with the respective chunk coordinates. Green chunks inside the visible radius are active. Chunks in between the visible and delete radius are disabled, becoming invisible, but kept in memory. Chunks outside the delete radius are deleted from memory.

As aforementioned, each chunk is a part of the virtual forest. This forest features terrain, trails and objects such as trees, grass, rocks, etc. The next subsections will address the individual components that make up a chunk.

B. Terrain Generation

From a more low-level computer graphics perspective, the terrain object is no more than a set of vertices and triangles evenly distributed along their axis, with an assigned height value, constituting a mesh. For the current application, it makes sense that chunks are square (along X and Z).

Assigning a random value to the height of each vertex in the mesh creates unnatural-looking terrain. However, using Perlin noise, terrains with smooth transitions between heights are created. Thus, terrain generation is controlled by the following parameters:

- **Number of Octaves:** The number of Perlin Noise layers that will be combined. It is a positive integer;
- **Scale:** Controls the distance to the noise map. Can be thought of as zooming in and out. Useful to create rougher or smoother terrains. It is a positive real number;
- **Lacunarity:** Adjusts the frequency of each Perlin Noise octave, controlling it's level of detail. Can be any real number greater than 1;
- **Persistence:** Controls the weight (amplitude) each Perlin Noise octave has on the final shape. If the value is less than 1, each octave will contribute less, which is desirable. Therefore, this parameter ranges between 0 and 1;

- **Max height:** The maximum height, in meters, that the terrain can reach;

The tuning of these parameters allows the creation of many different terrain morphologies, ranging from meshes that resemble plains to mountains with smoother or rougher terrain. Figure 3 presents some examples.

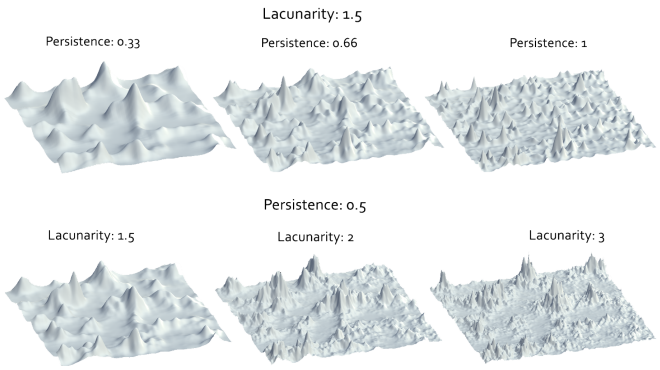

Fig. 3. Effect of persistence and lacunarity on the generated terrain mesh. These meshes have a 240 meter side size and the values for other parameters remain constant: noise scale of 40, a maximum height of 50 meters and 5 octaves.

C. Trail Generation

Robots equipped with perception algorithms will most likely be working in areas with trails, making it important to have a trail generation module for the virtual environment. Real-world forest trails can have complex structures since the trails tend to affect and even change the terrain itself. For instance, on a slope, the trail will be carved leaving a clear footprint on the landscape, which is quite characteristic in forests. That said, the trails generated by this system are planar with the terrain mesh, being an area where this system can improve. This makes the problem much easier to solve, at the expense of losing some realism. In the current version of the system, trail generation can also be switched off if there is the need to generate data from forests with steep slopes.

The trail generation algorithm makes it possible to instantiate grass and small rocks throughout the trail to enhance realism. Grass spawns from the middle of the track and can spread over a configurable distance. This implementation makes it easy to create one of the most common effects seen on forest trails: grass predominant in the middle of the trail due to vehicles passing. The trail generation is controlled by the following adjustable parameters:

- **Width:** Trail width in meters;
- **Trail texture:** Texture to apply to the trail mesh;
- **New trail spawn probability:** Probability that when a chunk is being generated, a new trail will be started;
- **Probability of the trail going forward:** How likely it is for a trail to end at the opposing chunk border, instead of deviating to its left or right; It is useful to either create straight and long trails, or trails with more turns.
- **Section shape:** A curve used to shape the trail, extruding it from the terrain mesh;
- **Grass density:** The maximum number of grass objects that can be instantiated along the middle of the track; 0 corresponds to no grass instantiated;
- **Grass spread:** How far from the middle of the trail can grass objects spawn;
- **Rock density:** The maximum number of rocks that can be instantiated per track;

D. Object Placement

Having already generated the terrain mesh and the trails, the next step is to populate the terrain with the remaining objects. Object instantiation is done in a fixed order to ensure the correct composition of the environment: First, rocks are instantiated, then trees, stumps and shrubbery and, finally, the lower vegetation such as grass and other herbaceous plants. Following this order, it is possible to ensure that the generated environment makes sense eliminating problems such as different objects occupying the same space.

The objects that populate the environment were acquired from the Unity Asset Store in three distinct asset packs. All of them have models of trees, stumps, rocks, shrubbery, herbaceous plants and other details. They also have textures for soil and trails. These provide enough variability to create diverse and distinct environments.

Some forests have rocks scattered about randomly. The object placement subsystem supports rock instantiation with a random pattern. For each chunk, there is a minimum and a maximum number of individual rocks that can be spawned anywhere within the chunk's border. The exact number of rocks instantiated in each chunk is randomly determined within the defined range.

Next, trees and shrubbery objects are placed. In order to generate a realistic looking forest, trees need to be able to overlap their canopy whilst keeping theirs trunks isolated. An algorithm suited algorithm for this task is the Fast Poisson Disk Sampling (FPDS) [17]. The original implementation is designed to handle the placement of objects (originally called samples) with a fixed radius in n-dimensions. For this application, the original algorithm was modified to be able to place objects with different radii in Two-Dimensional (2D), since the goal is to place them in the XZ plane. The density of the vegetation is manipulated multiplying the radii of occupation of each object by a factor. The resulting value is then used by the placement algorithm.

One of the challenges object placement algorithms impose is to find an efficient strategy to keep track of occupancy. This can be done by keeping an occupancy grid or by having a data structure that is capable of searching for neighbors efficiently (in space). Since the objects to be placed in this application have different radii, an occupancy grid would have to be built with the cell size accounting for the smallest object, which revealed inefficient. Thus, quad-trees were used to keep track of the objects in the 2D space since they have proven to be suited for neighbor search operations. A list

of positions where objects can still be placed around is also used, as suggested in [17].

Finally, herbaceous plants are placed. Considering that these plants are located everywhere, the whole chunk is swept and the objects are instantiated based on a grid. They can be placed in the middle of shrubbery, right next to tree trunks or even with parts occluded by rocks, but not in trails. For each grid location, a neighbor position is generated (with a random direction and distances) to avoid the placing of unnaturally looking undergrowth.

E. Lighting

Using Unity to develop this system comes with the advantage of having lighting and post-processing tools that are readily available to deploy in the generated scenes.

Sunlight is simulated with an infinite light object. This is an infinite plane of light that emits in a defined direction with a configurable intensity and color. With this feature, it is possible to simulate different times of the day. Using different skyboxes, light with different colors is reflected on the scene (reacting to the skybox), as seen in Fig. 4. This feature is valuable to generate diverse RGB datasets since ML models are known to be sensitive to light conditions.

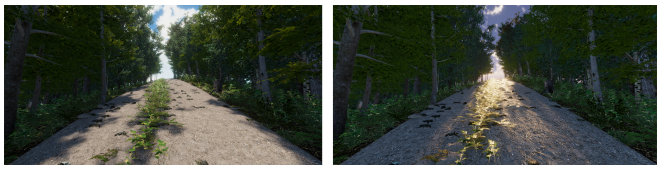

Fig. 4. Same scene rendered with two different light (sun) positions and skyboxes.

V. VIRTUAL SENSORS

Having generated the environment the next step is to collect data from it. To achieve this, physical sensors were modelled and deployed in the generated environment. This section will give an overview of the implementation and capabilities of each virtual sensor.

A. Classes

Most of the value in the proposed system comes from autonomous data labelling. With this in mind, the following classes were defined and are used in the generated semantic segmentation for RGB images, as well as in the point wise segmentation performed on point clouds: Background, Terrain, Traversable, Trunks, Canopy, Shrubs, Herbaceous plants and Rocks.

Some of these classes can be merged in a post-processing step if it makes sense for the case at hand. In tasks such as landscaping for wildfire prevention, it might make sense to classify shrubs and herbaceous plants as the same class of live flammable material ("fuel" for short).

B. LiDAR

LiDAR is a technology that uses light to create a 3D representation of the environment, a point cloud. Light pulses are emitted and their time of flight (from the sensor to the objects and back) is used to calculate the distance between the objects and the device. To create a virtual model of a LiDAR scanner, first a laser pulse was simulated and then replicated in many directions to make up common LiDAR scanning patterns.

Simulating the described light pulses seems as simple as using Unity's physics API to issue a raycast operation. However, the fact the environment is a forest imposes some challenges. In a forest there is an abundance of leaves and fine details that are not common in other scenarios, for instance urban scenarios. While this problem is easily solved for visual rendering purposes, projecting a detailed texture on a rough mesh, raycasting uses the far less detailed mesh colliders, as seen in Fig. 6.

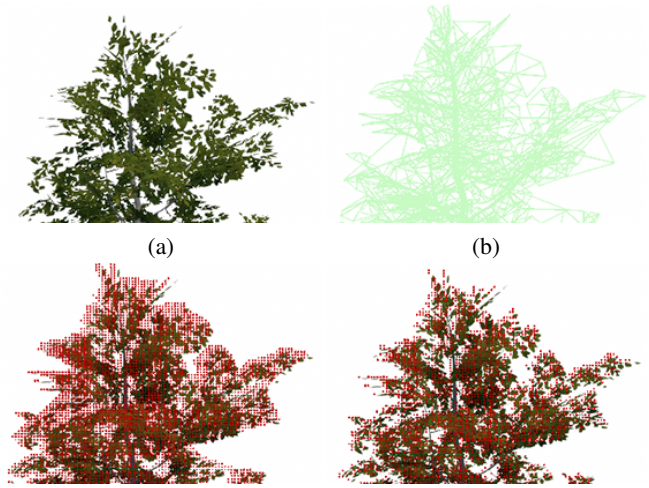

Fig. 5. Using the top of a virtual tree model to compare (a) the visual render, (b) the mesh collider with which the physics engine interacts, (c) the result of a virtual LiDAR implementation that only accounts for the mesh collider and (d) the result of a virtual LiDAR implementation that accounts for the mesh collider as well as the texture transparency

To overcome this problem a solution was developed merging the information from raycast operations with the transparency information present on the textures. Points are only registered when there is a collision with a mesh collider and the texture at that point is not transparent, yielding the results presented on Fig. 5.

Moreover, a model to simulate LiDAR noise was developed using data from studies on physical LiDAR devices [18], [19] and [20]. The presented data was used to create mathematical functions that model the noise of a specific device to generate the error value. This error is then used as the mean value of a Gaussian distribution with a configurable sigma and the value returned by the Gaussian noise generator is used as the error value. The normal distribution is used to add some variability and avoid a patterned noise generation.

Finally, three scanning patterns were developed for the

virtual LiDAR. A traditional line scanning pattern (repeating), and two non-repeating patterns: circular Field of View (FOV) and rectangular FOV, inspired on the commercially available Livox Mid-40 and Livox Horizon.

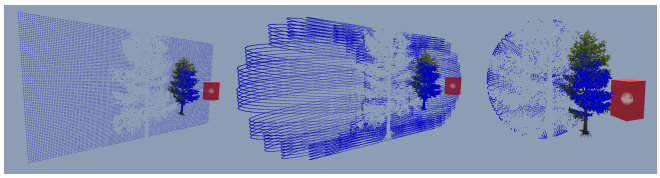

Fig. 6. Scanning patterns developed for the virtual LiDAR projected onto a plane with a tree to test for occlusion. Repetitive line pattern, non-repetitive rectangular pattern and non-repetitive circular pattern, respectively in the image. The red box represents the robot.

C. RGB Camera

Unity provides a readily available camera component that renders the 3D scene onto a display or a texture. Rendering the image to a texture allows it to be saved to a file, effectively creating an RGB image. This camera can be configured in terms of sensor size and FOV or focal length.

Semantic segmentation images are created using an idea proposed by Unity Technologies [21]. This idea uses camera replacement shaders (that make the camera ignore all the material shaders and render the entire scene with that shader), in combination with material property that allow to programmatically set custom properties for each material such as textures. Thus, a second camera was added to the scene (with the same transform and physical properties of the RGB camera) to generate semantic segmentation images. This camera runs a replacement shader that renders each object with a solid color, producing pixel perfect semantic segmentation maps, as illustrated in Fig. 7.

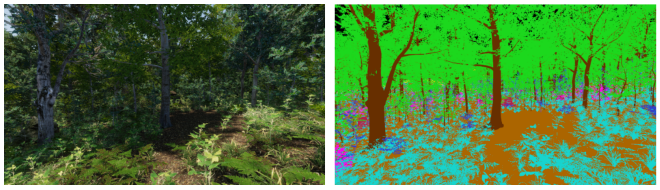

Fig. 7. RGB image and it's corresponding semantic segmentation.

D. Depth Camera

The depth maps are created intercepting the depth buffer generated by the GPU in the rendering pipeline. As the name suggests this buffer stores the distance to the nearest object, for each pixel that is to be rendered.

Depth maps generated from Red Green Blue Depth (RGBD) cameras or stereo camera rigs encode distance in meters. Unity's depth buffers values range from 0 to 1 and are non-linear since more precision is needed for objects that are closer to the camera.

Information regarding the curve that maps depth buffer values to meters was not found in the documentation, so a virtual experimental setup was created in Unity to solve this problem. By placing a plane fronto-parallel to the camera and moving it away in small increments whilst measuring the value on the depth buffer, it was possible to generate data to create a curve with the MATLAB curve fitting tool. This curve maps values on the depth buffer to meters. This is a post-processing step, meaning that an application was developed in MATLAB to convert the images that are captured in the rendering pipeline. The reason to do this outside of the main system is that this would be costly in terms of resources. That can be done with greater performance in MATLAB, avoiding an additional overhead on the data collection process. An example of the generated depth maps can be seen in Fig. 8.

Fig. 8. Generated RGB image with the corresponding depth map.

The following expression, where d means distance and v represents the value on the depth buffer, maps each value on the depth buffer to a distance in meters:

$$d(v) = 183.7e^{-28.04v} + 33.87e^{-6.99v} \quad (1)$$

The Root-mean-square Error (RMSE) for the curve is 0.18m.

VI. PERFORMANCE ASSESSMENT

An important criterion for evaluating the performance of the proposed system is the time it takes to generate data for different types of environments and sensor configurations. As noted earlier, the system does not run in real-time. The chunks take time to be generated and, most notably, the virtual LiDAR also takes some time to scan the environment. The time that different system modules take to perform their tasks scales with the density of the vegetation and with the resolution of the data collected by the sensors. Bigger chunks and greater visible radius also impact performance.

Different environmental and virtual LiDAR settings have an impact on the number of images the system can produce in a given time interval. To evaluate these effects, two experiments were conducted:

1) In the first experiment, the sensor settings were kept equal during the three tests with the LiDAR scanning 50.000 points per frame. The density of the vegetation in the environment was increased from low to medium and finally high. Regarding the number of trees per chunk, for each density, on average, the following numbers were used: 31 trees for low density, 96 trees for medium density, and 210 trees for high density.
2) In the second experiment, the environment settings were kept equal during the three tests with a medium-density configuration of the environment. The number of points scanned by the LiDAR was increased from 25.000 to 50.000 and finally to 100.000. This number

of points is consistent with the different integration times for the Livox Horizon: 100ms (10Hz), 200m (5Hz), and 500ms (2Hz), respectively.

TABLE I

TIME IT TAKES FOR THE SIMULATOR TO GENERATE 1000 FRAMES, FOR DIFFERENT ENVIRONMENT CONFIGURATIONS.

Environment Density	Low	Medium	High
Time to Generate 1000 Frames (min)	65	180	185
Average Time per Frame (sec)	3.9	10.8	11.1

TABLE II

TIME IT TAKES FOR THE SIMULATOR TO GENERATE 1000 FRAMES, FOR DIFFERENT LIDAR CONFIGURATIONS.

Number of Points per Scan	25.000	50.000	100.000
Time to Generate 1000 Frames (min)	95	180	225
Average Time per Frame (sec)	5.7	10.8	13.5

The reason for the experiments to only cover tests where the resolution of the LiDAR varies, is that the virtual LiDAR is by far the bottleneck in terms of processing time. On average, the virtual LiDAR takes 79% of each frame's total processing time. The chunk size was fixed at 50m and the visible radius at 100m for all tests. Regarding the camera, the RGB images, segmentation maps and depth maps had a fixed resolution of 1280 x 720 pixels. Testing was performed on a machine equipped with an Intel(R) Core(TM) i7-7700K 4.20GHz, 32GB of RAM and an NVIDIA GeForce GTX 1080 GPU.

The presented system is able to generate 1000 synthetic frames of a forest with a medium density, each one composed of an RGB image with semantic segmentation, a depth map and a point cloud with point-wise annotations for 50.000 points, in 3 hours. Tables I and II summarize the performance of the proposed simulator using the considered hardware. It is capable of generating in 3 hours 1000 synthetic images of a medium density forest, each consisting of an RGB image with semantic segmentation, a depth map, and a point cloud with point-wise annotations for 50,000 points. Using this system to generate a dataset with an equivalent amount of points as Semantic KITTI would take roughly 170 hours. This time was calculated using the results from Table II, knowing that Semantic KITTI has 43.552 scans, each one with 104.449 points on average. Compared to the 1700 hours it took for them to annotate the dataset manually this is 10x faster. It should also be noted that:

- The system would not only generate the point clouds, but also the other mentioned types of data;
- The generated data has perfect labels. When humans annotate data they inevitably introduce errors and inaccuracies that harm the performance of the models;
- The referred 170 hours already account for data collection and annotation;
- Apart from the initial parameter configuration the system runs completely unattended.

VII. PRELIMINARY RESULTS

A preliminary evaluation of the usefulness of synthetic data to train ML algorithms in the context of forest perception was performed. The chosen application domain was semantic segmentation based on RGB images. This section presents some preliminary qualitative results that will be discussed below.

A readily available ML framework for semantic segmentation was used to perform the evaluation [22]. This framework provides different DL models, based on Convolutional Neural Networks (CNNs). After some initial tests, the model PSPNet [23] was chosen. The main reason for this choice was the fact that the framework provided a pre-trained version of this model on the Cityscapes dataset [24]. This dataset was collected in an urban scenario, not in a forest. However, using transfer learning from a similar domain offers a number of advantages, such as shorter training time, better neural network performance, and the absence of a large amount of data to be trained from scratch.

The dataset generated for this evaluation was composed of 3154 frames. The herbaceous plants and shrubbery classes were merged into the "fuel" class since these are similar classes in terms of appearance. The dataset was made publicly available on a community-wide repository [25].

The model was trained solely on synthetic data and then inference was performed on real-world RGB images. Figure 9 presents some of the obtained results. From a qualitative perspective and considering the used data, some conclusions can be drawn:

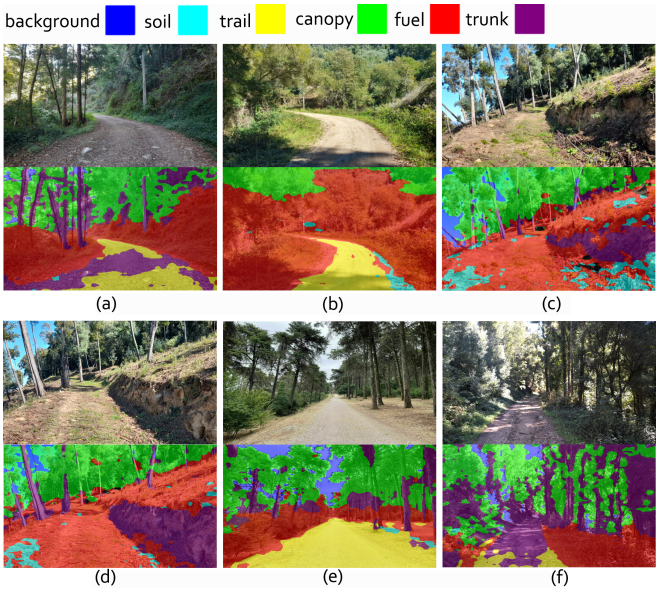

Fig. 9. Qualitative results produced by the trained model.

- The model learned to differentiate fuel from the canopy in some situations but requires more training to be considered reliable. This limitation was expected since

visually, the canopy and the objects from the fuel class are similar. Since the model, most likely, learned to associate canopy to trunks, identifying trees whose trunks are hidden is a difficult task.

- The training data had all the trails generated with the same texture, similar to the areas segmented as trails in (b) and (e). This leads to problems when segmenting trails with a different appearance, such as in (a) and (f). This problem can be attenuated by generating data with different looking trails;
- Similarly, areas on the RGB images that resemble the textures used to generate the training data can be miss classified. This happened on the right side on (c), where the terrain was identified as belonging to the trunk class. It also happened in (d), where the trail was segmented as fuel because it has small plants all over. The model was built based on images where these plants only appeared in the middle of the trails;
- On (e) some trunks are classified as fuel. This represents a problem for a forest clearing robot. On (f) the same happens on the bottom right of the image. This happens because tree trunks are wrapped with vegetation. The model did predict those labels accurately. Nevertheless, ML algorithms should be trained to identify these situations and predict where the trunks are, even among the vegetation.

VIII. CONCLUSIONS

As previously shown, the developed system can potentially contribute to improve the performance of ML based robotic perception algorithms on the forest domain. By tuning the parameters of procedurally generated virtual forests, one can generate specific scenarios that target situations where models are underperforming, thus helping to improve their accuracy.

The presented preliminary evaluation suggests that the generated synthetic data has the potential to improve the quality of trained ML models. However, this work does not aim to replace real data with synthetic data to address the problem of annotated dataset shortage for the forest domain, but to provide an attenuating solution to the problem. Further testing is needed to draw more definitive conclusions, namely assessing the performance impact of combining synthetic and real data during training.

REFERENCES

[1] M. Couceiro, D. Portugal, J. F. Ferreira, and R. P. Rocha, "SEMFIRE: Towards a New Generation of Forestry Maintenance Multi-Robot Systems," *IEEE/SICE International Symposium on System Integration*, 2019.

[2] A. Agrawal, B. Cashore, R. Hardin, G. Shepherd, C. Benson, and D. Miller, "Economic Contributions of Forests," *United Nations Forum on Forests*, 2013.

[3] F. Lateef and Y. Ruichek, "Survey on semantic segmentation using deep learning techniques," *Neurocomputing*, vol. 338, pp. 321–348, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523121930181X

[4] A. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne, "Data Management Challenges for Deep Learning," *Proceedings - 45th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2019*, no. August, pp. 140–147, 2019.

[5] A. Giusti, J. Guzzi, D. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, D. Scaramuzza, and L. Gambardella, "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots," *IEEE Robotics and Automation Letters*, 2016.

[6] L. Lind, "Deep learning navigation for UGVs on forests paths," Master's thesis, KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, Stockholm, Sweden, 2018.

[7] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "Randla-Net: Efficient semantic segmentation of large-scale point clouds," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 11 105–11 114, 2020.

[8] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," *ArXiv*, vol. abs/2103.14749, 2021.

[9] N. O. Mahony, S. Campbell, A. Carvalho, L. Krpalkova, D. Riordan, and J. Walsh, "Point Cloud Annotation Methods for 3D Deep Learning," pp. 3–8.

[10] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-October, no. iii, pp. 9296–9306, 2019.

[11] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[12] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for Data: Ground Truth from Computer Games," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9906 LNCS, pp. 102–118, 2016.

[13] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[14] X. Yue, B. Wu, S. A. Seshia, K. Keutzer, and A. L. Sangiovanni-Vincentelli, "A LiDAR Point Cloud Generator: from a Virtual World to Autonomous Driving," *ICMR 2018 - Proceedings of the 2018 ACM International Conference on Multimedia Retrieval*, pp. 458–464, 2018.

[15] B. Hurl, K. Czarnecki, and S. Waslander, "Precise Synthetic Image and LiDAR (PreSIL) Dataset for Autonomous Vehicle Perception," pp. 2522–2529, 2019.

[16] D. Griffiths and J. Boehm, "SynthCity: A Large Scale Synthetic Point Cloud," in *ArXiv preprint*, 2019.

[17] R. Bridson, "Fast poisson disk sampling in arbitrary dimensions," *ACM SIGGRAPH 2007 Sketches, SIGGRAPH'07*, p. 2006, 2007.

[18] S. Soudarissanane, R. Lindenbergh, and B. Gorte, "Reducing the error in terrestrial laser scanning by optimizing the measurement set-up," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, vol. 37, pp. 615–620, 2008.

[19] S. Soudarissanane, R. Lindenbergh, M. Menenti, and P. Teunissen, "Incidence Angle Influence on the Quality of Terrestrial Laser Scanning Points," *Iaprs*, vol. XXXVIII, no. May 2014, pp. 183–188, 2009.

[20] J. Laconte, S. P. Deschênes, M. Labussière, and F. Pomerleau, "Lidar measurement bias estimation via return waveform modelling in a context of 3D mapping," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 8100–8106, 2019.

[21] U. Technologies, "Image synthesis for machine learning," https://bitbucket.org/Unity-Technologies/ml-imagesynthesis/src/master/, 2017.

[22] divamgupta, "Image segmentation keras: Implementation of segnet, fcn, unet, pspnet and other models in keras," github.com/divamgupta/image-segmentation-keras, 2021.

[23] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *CoRR*, vol. abs/1612.01105, 2016. [Online]. Available: http://arxiv.org/abs/1612.01105

[24] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] R. Nunes, J. Ferreira, and P. Peixoto, "SynPhoRest - Synthetic Photo-realistic Forest Dataset with Depth Information for Machine Learning Model Training," Mar 2022, doi: 10.5281/zenodo.6369445.