Scientific Research Publishing

Journal of Intelligent Learning Systems and Applications, 2025, 17(1), 8-24
https://www.scirp.org/journal/jilsa
ISSN Online: 2150-8410
ISSN Print: 2150-8402

Machine Learning and Simulation Techniques for Detecting Buoy Types from LiDAR Data

Christopher Adolphi, Masha Sosonkina

Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, Virginia, USA
Email: cadol001@odu.edu, msosonki@odu.edu

How to cite this paper: Adolphi, C. and Sosonkina, M. (2025) Machine Learning and Simulation Techniques for Detecting Buoy Types from LiDAR Data. *Journal of Intelligent Learning Systems and Applications*, **17**, 8-24.
https://doi.org/10.4236/jilsa.2025.171002

Received: December 24, 2024
Accepted: February 7, 2025
Published: February 10, 2025

Abstract

Critical to the safe, efficient, and reliable operation of an autonomous maritime vessel is its ability to perceive the external environment through onboard sensors. For this research, data was collected from a LiDAR sensor installed on a 16-foot catamaran unmanned vessel. This sensor generated point clouds of the surrounding maritime environment, which were then labeled by hand for training a machine learning (ML) model to perform a semantic segmentation task on LiDAR scans. In particular, the developed semantic segmentation classifies each point-cloud point as belonging to a certain buoy type. This paper describes the developed Unity Game Engine (Unity) simulation to emulate the maritime environment perceived by LiDAR with the goal of generating large (automatically labeled) simulation datasets and improving the ML model performance since hand-labeled real-life LiDAR scan data may be scarce. The Unity simulation data combined with labeled real-life point cloud data was used for a PointNet-based neural network model, the architecture of which is presented in this paper. Fitting the PointNet-based model on the simulation data followed by fine-tuning the combined dataset allowed for accurate semantic segmentation of point clouds on the real-world data. The ML model performance on several combinations of simulation and real-life data is explored. The resulting Intersection over Union (IoU) metric scores are quite high, ranging between 0.78 and 0.89, when validated on simulation and real-life data. The confusion matrix-entry values indicate an accurate semantic segmentation of the buoy types.

Keywords

Maritime Autonomy, LiDAR, Unity Simulation, Machine Learning, PointNet

1. Introduction

Critical to the safe, efficient, and reliable operation of an autonomous maritime vessel is its ability to perceive the external environment through onboard sensors. To enable such a perception, methods are needed to accurately convert raw sensor readings into actionable higher-level features, such as the location of buoys and their types, through object detection and classification. In [1], the objective was to identify buoys around an unmanned vessel with efficient image processing techniques. Object-type classification is notoriously difficult to do with image processing techniques. While the authors were able to successfully detect buoys with few false positives, they did not distinguish between the two types of buoys. To determine the type and location of buoys, the current paper presents an experimental setup for the classification of objects represented as point clouds using machine learning. The point clouds used in this research are sets of points in space collected from LiDAR sensor data. The investigation includes real-life data collection and labeling, simulation data generation from Unity [2], and training a model to label each of the points as belonging to one of three classes: *spherical* buoy, *cylindrical* buoy, or *other*. This notion of labeling each part of an image or LiDAR scan is called semantic segmentation. The simulation was created to complement a limited amount of training data obtained from hand-labeling the LiDAR scans, with the assertion that it would assist in training a model. The main contributions of this paper are a marine LiDAR simulation and how to employ the generated simulation data in semantic segmentation of LiDAR scans in the situation when the real-life data available for training is insufficient.

1.1. WAM-V

A WAM-V 16-foot catamaran (see **Figure 1**) unmanned vessel equipped with an onboard Velodyne VLP-16 sensor [3] with synchronized camera feeds was used to collect LiDAR data at the pier of the Old Dominion University (ODU) campus sailing center (see **Figure 2**). Once the boat is on the water, it is relatively cheap in terms of time to collect vast amounts of LiDAR scans. It is very time-consuming, however, to hand-label the scans for training a machine learning model.

Figure 1. WAM-V unmanned vessel equipped with Velodyne VLP-16 sensor.

Figure 2. Buoys captured by the camera on WAM-V.

1.2. Unity Simulation

A simulation of a maritime environment and the output of a LiDAR sensor has been developed in Unity Game Engine (Unity). Unlike the real-world environment where data was collected from a small area, the simulation is capable of randomly modifying the environment by adding other objects, such as buildings, and waves. Such complexity was added to the Unity simulation to better mirror a maritime environment and diversify the data, which is helpful for machine learning (ML) models. Similar to [4]-[6], utilizing the simulation data for training the ML model was more advantageous given the ease of data collection, automatic labeling, and the ability for customization.

1.3. Machine Learning Method

The machine learning technique, described and used in this paper, consists of semantic segmentation of point clouds. In particular, PointNet++ [7] with a SphereFormer [8] enhanced architecture is utilized. PointNet++ is a machine learning model that can predict a classification label for each point in a point cloud, or produce a global vector that predicts something about the point cloud as a whole. In this case, the model categorically classifies each point of a point cloud as a spherical buoy, cylindrical buoy, or other. The individual labeling of each point provides fine details regarding the space the buoys occupy. The SphereFormer module helps LiDAR scan-based models perform better with points farther away from the origin of the sensor [8]. The model was first trained on a large set of simulation data and then fine-tuned on a mixture of real and simulation data. The data was mixed for two key reasons. Firstly, there are limitations to transferring learned behavior from simulation to reality, where the simulation does not depict reality faithfully. Secondly, there is not enough real data on which to solely train the model. Mixing the two datasets solves both of these problems.

1.4. Organization

Section 2 presents the related work. Section 3 presents the ML tools used here and

the developed ML model architecture. In Section 4, the methodology used for generating and processing simulation data from Unity is described. Following these are the experiments presented in Section 5 and their results discussed in Section 6. Section 7 concludes and mentions future work.

2. Related Work

This section overviews several object detection techniques that use point clouds and how they differ from the model presented in this paper.

2.1. Enriching 3D Object Detection in Autonomous Driving

Autonomous driving datasets are the most common type of labeled LiDAR data, whereas the authors are not aware of the examples of marine datasets that are widely available. In the CARLA Simulator [4], a vehicle is placed in a randomized environment to collect data. Unreal Engine [9] is used for the CARLA simulation environment. Various sensors are available, such as LiDAR, cameras, lane sensors, collision detectors, and inertial measurement units. Different environments and randomized environmental conditions are also available in CARLA, including traffic and other non-playable characters.

In [5], CARLA was modified to produce a point cloud with automatically generated semantic segmentation labels. The researchers took their generated data and mixed it with real-life data from the Semantic-Kitti [10] dataset. They found that 1) training solely on simulation data did not perform as well as training on real data, 2) mixing the two types of data improved the model performance, and may save time on the rigorous labeling of real-world data. Most notably, infrequent objects such as emergency service vehicles were supplemented by increasing their appearances in the simulation dataset. This increased the model's predictive power of specific objects in the real-life dataset. In the current paper, similar results are achieved when simulating maritime autonomy and LiDAR in Unity while additionally exploring how various combinations of real-life and simulation data may affect the results.

2.2. Object Detection for Autonomous Vehicle Operation

In Complex-Yolo [11], object detection is performed with point clouds. This method prioritized speed which is necessary for autonomous vehicle operation. In [11], the authors first flattened the point cloud from a birds-eye view, and then they encoded the height, intensity, and density of the points into the RGB channels of an image. Then 2D bounding boxes were generated for each of the objects, which included the angle, the class such as car, pedestrian, or cyclist, and the position, length, and width of the object. The flattened point cloud can lose some finer details of the input point cloud, as a trade-off for speed, and networks that directly operate on points are better suited for point-cloud predictions. In the current paper, fine-grained details of the dataset are retained with the proposed model architecture as described in Section 3 and model is tested in the maritime

environment.

3. Machine Learning Model Architecture Used

3.1. PointNet++

In this research, the main network employed is PointNet++ [7]. It uses several PointNet [12] models to represent local regions of space better. Since PointNet can produce a good global feature representation of the point cloud, it is used within [7] to produce several local feature vectors of increasing length and decreasing point density.

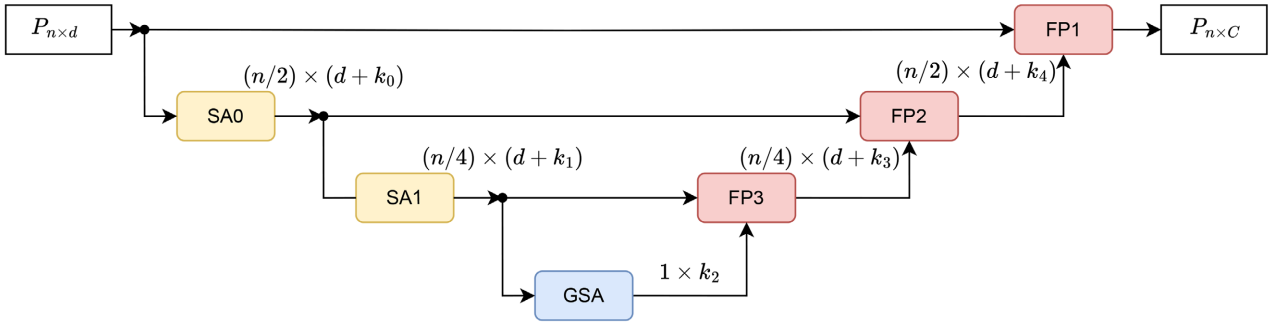


Figure 3. PointNet++ feature sampling and propagation for point cloud of n points ($d=3$) segmentation into C classes. SA0 and SA1 down-sample the point cloud while increasing the number of features per point. The global set abstraction (GSA) module generates a single feature vector for the point cloud. FP3 through FP1 uses *k*-nearest neighbors interpolation to concatenate the larger feature sets with the smaller feature sets [7].

Figure 3 shows how PointNet++ down-samples a point cloud while increasing the feature size $k_0 < k_1 < k_2$, then up-samples it while decreasing the feature size $k_2 > k_3 > k_4$. This type of architecture resembles a U-Net [13], where *skip connections* and a U-shaped down-sampling and up-sampling architecture are employed. Here, each skip connection represents the local features for a subset of points. The feature-length increases with each iteration of PointNet since there is a smaller number of points representing larger and larger regions. They are all combined in the end to give several feature sets of the point cloud with different levels of detail.

3.2. SphereFormer

SphereFormer [8] is a transformer [14] module that utilizes radial window self-attention combined with a cubic self-attention window to process sparse point cloud data. Transformers comprise attention layers, which process data within attention windows. In the case of SphereFormer [8], the attention windows capture groups of points and within each group, the features of points are used together to generate new features within that group. The radial window is a key factor in why this module improves upon models that process LiDAR point clouds. The radial window acknowledges the feature of LiDAR scans where the sensor rays get further apart radially as the distance from the origin of the sensor increases. Be-

cause of this feature, the surfaces of objects that are farther away appear more sparse within a cubic window. In a radial window, the volume of the window increases as the distance from the origin increases.

The SphereFormer module is not designed to be a standalone machine learning model for point clouds. Instead, it is meant to be inserted into existing models such as PointNet-based models or graph neural networks. In this work, it is inserted between layers of a PointNet++ machine learning model to boost its ability to learn to interpret point clouds from LiDAR scans. The SphereFormer module resulted in a high Mean Intersection over Union performance score on the Semantic-Kitti [10] dataset, and it specifically excelled for objects that were farther away, which is why it is utilized in this paper.

3.3. Model Architecture

The following machine learning model architecture was developed (see **Figure 4**), in which the PointNet++ architecture has been fitted with SphereFormer modules.

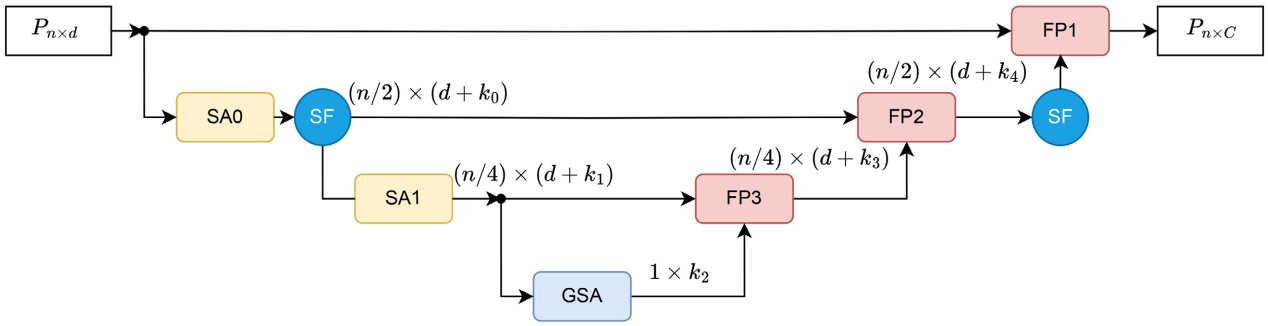


Figure 4. PointNet++ feature sampling and propagation for point cloud of n points ($d=3$) segmentation into C classes. SphereFormer modules were inserted at the second and the next to last stages of the U-Net style architecture (cf. **Figure 3**).

The original PointNet architecture is able to perform semantic segmentation on point clouds while being inherently order-invariant. This gave it an advantage over most other networks at the time by turning a point cloud into a voxel representation first [12] [15] instead of working with point cloud directly. The PointNet++ architecture was chosen for this research as it is a substantial improvement over the original PointNet network [7] since it allows the model to learn finer details of a point cloud. The SphereFormer modules are used to further enhance this model's learning ability, but for a LiDAR-specific domain. Other networks that use order-invariant techniques similar to PointNet++ [7] are usually derived from a Graph Neural Network (GNN), which operates on a graph. In a GNN, message-passing steps update the values of the graph nodes at every step [16]. A node can have any number of neighbors, so the messages passed from the neighbors need to be aggregated with an associative function, similar to the one used in [12]. Many ML techniques may be considered to be GNNs, such as the attention mechanism [14], KPConv [17], as well PointNet++ itself. In this paper, the PointNet++

technique has been used due to its superior ability to capture fine-grained details.

4. LiDAR Sensor and Simulation Data

4.1. Sensor

The VLP-16 LiDAR sensor fires its sixteen infra-red lasers (and corresponding detectors) at a rate of approximately 18,000 times per second in a regular spinning pattern [3]. These lasers reflect off objects nearby and the detected time-of-flight allows for the calculation of the distance to the point (in spherical coordinates) and the intensity is used to get the object reflectivity [3]. Translating the spherical coordinates produces a set of 3D Euclidean coordinates that represent the point cloud.

Robot Operating System (ROS) [18] is a robotics platform for control systems. In particular, messages can be sent with ROS between different sensors to different processes. These messages and their metadata, such as the timestamp and topic, can be recorded and saved to a ROSBAG-type file format. In this work, the LiDAR sensor messages along with other data about the WAM-V are saved to a ROSBAG file, and this file is used to load the point cloud data for labeling.

A remote-controlled catamaran unmanned vessel (see **Figure 1**) maneuvered around various buoys. The vessel moved such that the vessel-mounted Velodyne VLP-16 LiDAR sensor (see **Figure 5**) detected buoys at various distances, heights, and angles. Varying paths taken around the buoys by the WAM-V created a diverse set of LiDAR point-cloud data.

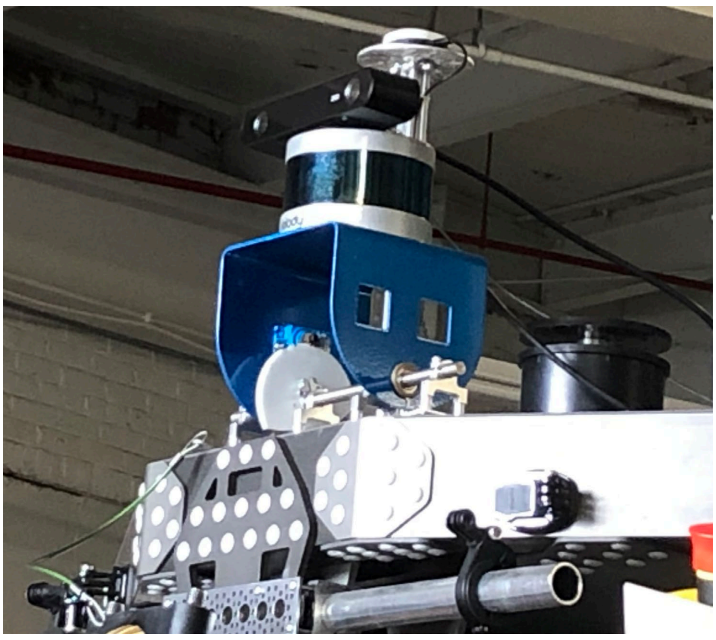


Figure 5. Vessel-mounted Velodyne VLP-16 sensor.

The ROSBAG file contains records from the Velodyne VLP-16 sensor having Velodyne coordinates for all the points that collided with objects in the maritime

environment. Utilizing LiDAR Toolbox in MATLAB [19] enabled the identification followed by the labeling of points in the point cloud.

4.2. Simulation

Unity Game Engine [2] is a development platform designed mainly to assist in creating video games, but can also be used for simulations. Unity was chosen due to a large pool of assets, such as 3D models and scripts, available within Unity's large community. To simulate the local maritime environment, the Unity simulation needed to introduce environmental factors, such as waves, which was done via the Crest [20] software, shore trees, and docks. The main objective of the simulation was to produce high-fidelity, synthetic LiDAR data, so lighting conditions and materials were a low priority, whereas rendering high-resolution depth images for simulating the LiDAR sensor was the highest priority. Random perturbations to the environment were also mainly aimed at augmenting the LiDAR data. The success of the simulation in producing quality synthetic data was assessed by how well the ML model developed in this research performs when assisted by the inclusion of the simulation data.

4.2.1. Building the Maritime Environment

A 3D model of the WAM-V vessel was placed in a Unity scene with randomly generated objects. The WAM-V vessel model follows tracks around simulated environments (see, e.g., **Figure 6**). The Crest wave simulation allows for variable wave parameters. At the start of every loop of the simulation, random wave parameters are used to calculate different Gerstner waves [21] to simulate realistic ocean wave behavior.

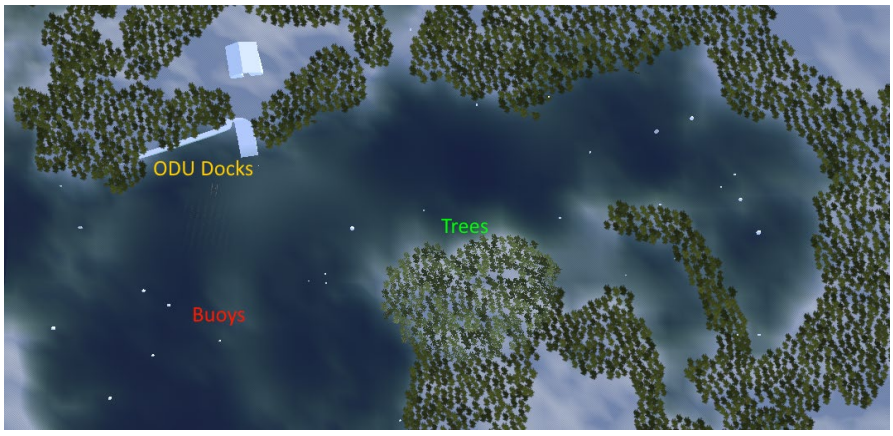


Figure 6. One of the simulation environments, meant to model the real-life data. This environment includes the docks from the ODU campus (labeled "ODU Docks"), trees (labeled "Trees"), and randomly placed buoys (labeled "Buoys").

When generating points with the simulated LiDAR sensor, two depth maps are utilized. The first depth map contains all of the objects in the scene, such as the buoys, buildings, and ground. The second depth map solely pertains to the dis-

tance from the sensor to the surface of the water. The minimum depth of the two maps is taken to get the true depth value and generate a point. When using Crest [20] to simulate waves, the water depth map causes points to appear on the surface of individual waves. However, during data collection with the WAM-V vessel, points collected from the LiDAR scans that are under the Z-plane where the vessel touches the water are removed. During low swell activity, this is a good estimation of where the water is at a given point. However, passing boats or other wave sources could falsely appear to be buoys if the current model is used. To match the real-life data collection, this Z-plane filter was also used in the Unity simulation, thereby curtailing somewhat the Unity simulation capabilities of generating points on the surface of waves.

4.2.2. LiDAR

The LiDAR system in Unity was simulated using a compute shader. Compute shaders are a high-level graphical processing unit (GPU) programming language for writing parallel code. In this case, each point is handled individually, sampling from four different textures and four cameras pointing in the cardinal directions. From sampling the textures, we get the layer and object IDs as well as the object and water depth of each point. Taking many samples from spherical coordinates results in a point cloud similar to that of a real LiDAR sensor. In Figure 7, a sample snapshot of the simulation shows a labeled point cloud.

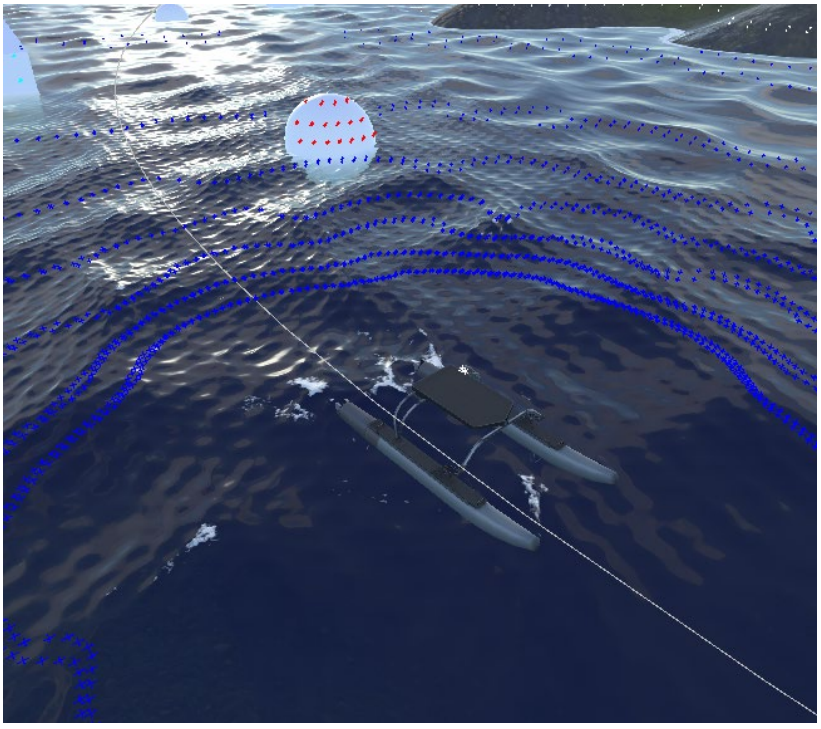
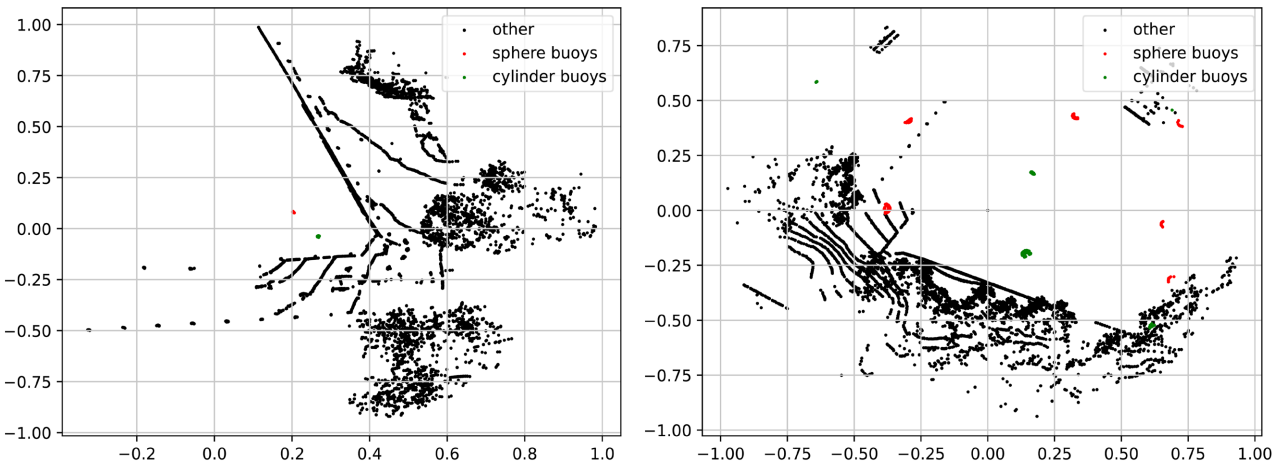


Figure 7. Unity simulation LiDAR points with depth and segmentation information sampled from simulated Gerstner waves, buoys, and other objects. Blue points are water and red are spherical buoys. Other features include the ground (white points) and a random floating object (cyan points).

4.2.3. Post-Processing the LiDAR Simulation Data

To better mirror the real-world LiDAR sensor, the simulation data output required some post-processing. Real LiDAR data collected from the Velodyne VLP-16, shows that a large slice of the point cloud is missing from the back side of the WAM-V vessel, which is due to obstruction of the view caused by the WAM-V vessel itself. Conversely, the simulation data does not exhibit such an omission, and thus, needs to be adjusted. Fortunately, simulation data may be easily post-processed since the point cloud data includes horizontal and vertical IDs for each point. These IDs may be used to quickly determine which sensor produced which point and perform adjustments, such as reducing the field of view of either axis, creating dropout, and changing the resolution. In particular, the vertical field of view is randomly set for each point cloud between 20 and 35 degrees to enhance the generalizing ability of the model. The horizontal field of view extends entirely around the LiDAR sensor, and it is randomly sampled to create artificial obstructions that more closely mimic the obstructed view of the real-life craft.

Figure 8 compares the LiDAR sensor data from the real docks, (a), and data generated by the Unity simulation and post-processing, (b). The images exhibit a similar object sparsity, in general, which may point to the simulation data representing obstructions to a similar degree. Yet, the buoys (red and green dots) are more uniformly distributed in the simulation-generated image and there are more of them (Figure 8(b)). The latter underscores the goal of the simulation of generating more training data reliably containing buoy objects and is leveraged in the experimental procedure to obtain quality semantic segmentation as described in Section 5.



(a) ODU campus docks

(b) Unity-simulated docks

Figure 8. Training examples from the real-life (a) and simulation (b) datasets. The LiDAR scan is flattened onto the XY plane, and the LiDAR range is normalized between −1 and 1.

5. Experiment Design and Results

The two experiments performed are similar, in that they both utilize the same model architecture and similar datasets. The first experiment (see Section 5.1) will

show how training on simulation data alone affects the model's performance on real data. In the second experiment (see Section 5.2), the trained model will be fine-tuned to both simulation and real data combined at different ratios. Note that, since the hand-labeled real dataset obtained from the Old Dominion University (ODU) campus docks is very small, especially concerning the number of cylindrical buoys, it is unable to facilitate sufficient training. Hence, the generated (and expandable) simulation dataset is used for training.

5.1. Results: Training on Simulation Dataset

After training on the simulation data, validation on the real data showed that the model lacked some predictive capabilities. **Figure 9** depicts the results in the form of the *confusion matrix*, which is a technique to report in a square table layout the number of true positives, true negatives, false positives, and false negatives. The diagonal of the matrix represents all instances that are correctly predicted. In this work, three classes are to be predicted: *sphere*, *cylinder*, and *other* (see **Figure 8**). The first two classes are to distinguish among the two types of buoys and the latter is for all the 'non-buoy' objects. The small number of classes justifies employing the confusion matrix to assess the results despite the noticeable prevalence of the *other* objects. It may be observed in **Figure 9** that a point on *sphere* or *cylinder* is more often labeled as *cylinder* since there are more spherical buoys in the dataset. The matrix entry corresponding to (*other*, *other*) has the highest value correctly predicting the largest number of non-buoy objects in the real dataset. Hence, the confusion matrix indicates that the model is good at distinguishing a buoy from other objects, while it is very poor at finding out which type of buoy it is. Following the ideas in [5], this work proposes to use the real data to supplement the training set in the LiDAR Unity maritime environment when the amount of real data is limited. The next Section 5.2 describes the obtained improvements.

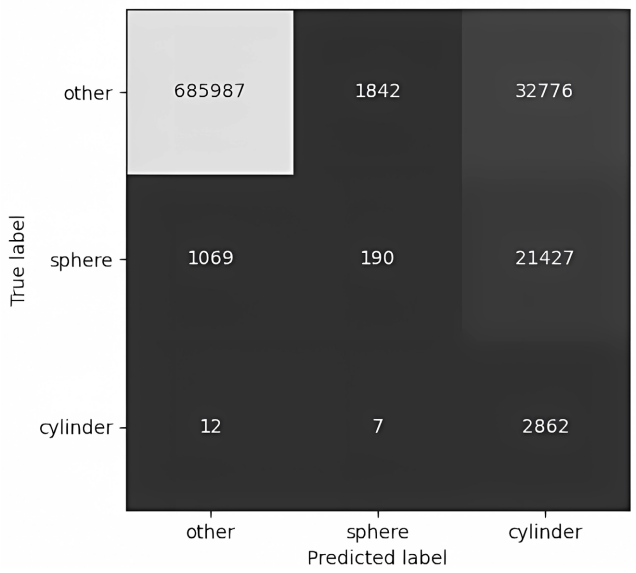


Figure 9. Confusion matrix resulting from training entirely on the simulation data.

5.2. Results: Fine-Tuning with Real-Life Dataset

A simple union of the real and simulation datasets exhibits a large imbalance between the simulation and real data samples, at around 74 simulation samples per 1 real sample. Compare also **Figure 8(a)** and **Figure 8(b)** for the differences in the number of objects. This imbalance may skew the training on the combined dataset. Therefore, the model is first trained on the entire simulation dataset (as in the first experiment in Section 5.1. After the training, the model is fine-tuned on a particular combination of the two datasets. Each such combination is defined by the proportions of simulation and real data in the mixed dataset used in the fine-tuning.

To fine-tune the model, its weights are initialized from the weights saved from training entirely on the simulation dataset, and the training proceeds on the combined dataset. Several cases of the combined dataset were considered, each containing a particular ratio of simulation to real data amounts. Specifically, the amount of the real data was fixed at $\mathcal{D}_R$ while the amount of simulation data $\mathcal{D}_S$ was taken proportionally in each case as $r \times \mathcal{D}_R$, where the factor $r = 0,1,2,3,4,5$.

In all the different r-factor cases, **Figure 10** displays the IoU metric calculated for the validation on the real and simulation datasets (blue and orange dots, respectively) as well as for the fine-tuning the model (green dots). Note that the IoU results with abscissa equal to -1 correspond to the model without fine-tuning, as described in Section 5.1. Without any exposure to the real data, the training model overfit the simulation data, and therefore, scored very low on the real data. At $r = 0$, the model is trained only on the real data and predicted it—IoU = 0.8—while performing poorly on the simulation data, which may also be due to overfitting.

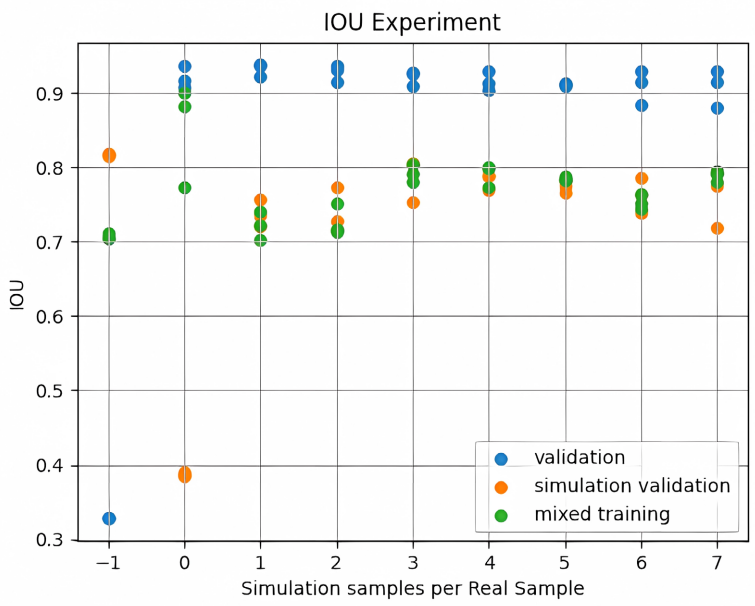


Figure 10. IoU scores after fine-tuning on a combination dataset. The x-axis shows the non-negative r factors for the size increase of the simulation dataset over the fixed real-life dataset to form the combined dataset. The IoU values with abscissa $r = -1$ are for training without real-life data as in the experiment shown in Section 5.1.

The model with the factor $r = 4$ is showcased here since it has a high real data IoU while having a similar IoU on the simulation data, indicating that it may be less likely to overfit either of the datasets, which is also supported by the curve of loss values shown in Figure 11. The IoU has improved to 0.84 for the real-life data validation from both $r = -1$ and $r = 0$ cases. For the simulation data validation, the IoU of 0.80 is an improvement over the $r = 0$ case and a very slight decrease of 0.02 from the simulation-data-only training ($r = -1$). These IoU scores are similar to those of 0.74 and 0.81 obtained by using the SphereFormer on the Semantic-Kitti dataset benchmarks [10].

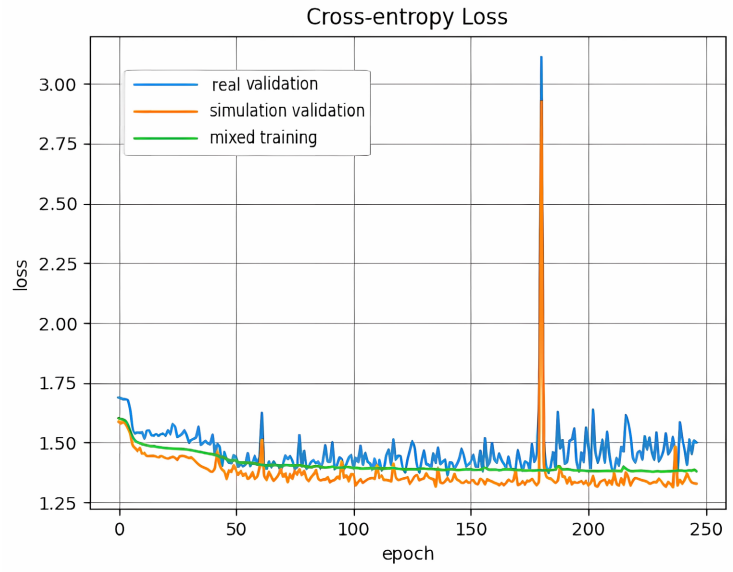


Figure 11. Loss curves produced from the model obtained in the $r = 4$ case.

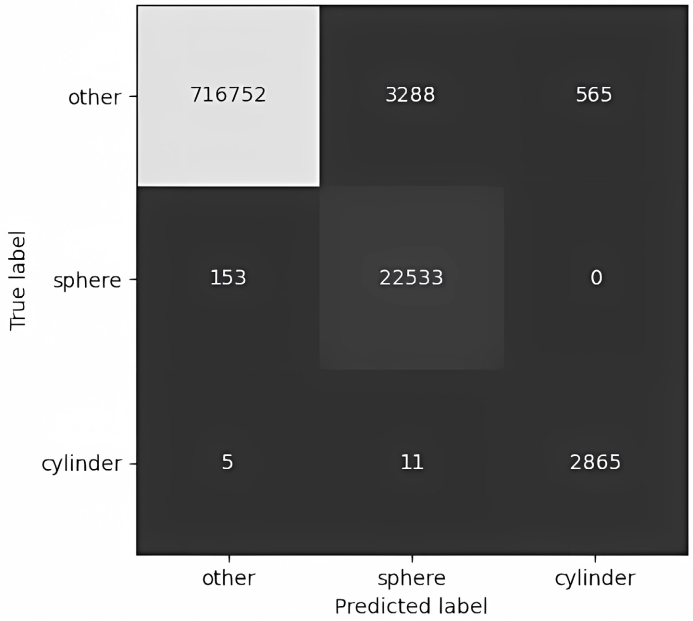


Figure 12. Confusion matrix from the model obtained in the $r = 4$ case.

The showcased factor $r = 4$ dataset resulted in the confusion matrix shown in Figure 12 presenting an overall improvement from the first experiment, where the model failed to differentiate between the two buoy types, and more often mislabeled other objects as buoys. The other positive factor r cases achieved similar confusion matrices.

It may be observed in Figure 10 that, for the other positive r factors, the IoUs are higher on the real-life data, ranging between 0.86 and 0.89, than that of the showcased $r = 4$ model (IoU of 0.84). For those r's, however, the IoU scores were all either 0.78 or 0.79 on the simulation data. Hence, for those r's, substantially higher differences may be observed between the IoU scores of the simulation and real-life data as compared with the difference in the $r = 4$ case. The high differences may indicate that the model is overfitting the real data in those cases. Hence, the case of $r = 4$ was showcased in Figure 11 and Figure 12 since it had the closest high IoU scores between real and simulation validation sets.

6. Experimental Results Discussion

The two experiments (see Section 5) contrast the model's performance on the real data when trained only on simulation data versus being trained on real and simulation data. In the first experiment, poor results were achieved when semantic segmentation was performed on real data by the model trained only on the simulation data. It could be that there is something substantially different about the two datasets, such that when the model is presented with a point cloud that was taken from a real-life LiDAR system, it is so far outside of the training domain that it cannot make the predictions accurately.

This is not to say that the model was unable to interpret the real data at all. Looking at the confusion matrix in Figure 9 gives a better clue as to what is happening and indicates that the model actually makes some useful predictions about the data, although many *other* object points are misclassified as being cylindrical buoys. It can be inferred that the model is roughly able to tell the difference between buoys and other objects. Since many spherical buoys are misclassified as cylindrical buoys, the model struggles to tell the two different buoy types apart. This could be the biggest difference between the simulation and real datasets. Perhaps, the cylindrical buoys generated by the simulation are too distinct from the spherical ones. In other words, the simulated cylindrical buoys are too perfect looking, whereas, in real life, even humans labeling the data can struggle to tell the difference between the cylindrical and spherical buoys floating in the water from the sparse LiDAR scans.

In the second experiment, the shortcomings of the results of the first one were addressed. After realizing that the simulation was not quite realistic enough to allow the model to learn the rather small differences between the two buoy types in real life, the simulation dataset was combined with a fixed amount of real-life data in certain increasing proportions, which was done due to the small amount of hand-labeled real data. Interestingly, the experiments showed that the mixed

dataset improved the model's performance from the case when only real data was input ($r = 0$ in Figure 10). Specifically, in all the positive r factors, this performance improvement was as much as 0.1 greater IoU than that for the $r = 0$ case. It is possible that adding the simulation data at a comparable amount to the real data forced the model to better generalize to predict correctly both types of buoys, thereby fixing the issues with the first experiment.

Model generalization ability was also the major reason why the $r = 4$ model was showcased: If including the simulation data helped the model generalize, then it would be better to pick the most balanced model, or in other words, the model performing well on both real and simulation data. Another significant reason is that the real-life dataset is too small for reliable training, and the simulation data added more samples of buoys for the model to learn from. This consideration even more underscores the need for the model's quality performance on the simulation data and supports the choice of the $r = 4$ model.

7. Conclusions and Future Work

The main focus of this paper is semantic segmentation of LiDAR scans using PointNet++ combined with SphereFormer modules trained on the data generated from a Unity simulation and fine-tuned with a mix of simulation and real-life data hand-labeled from LiDAR scans. When creating the simulation data, a variety of elements were improved to make the simulated LiDAR scans more realistic. Such elements included adding trees, having the simulated watercraft move randomly throughout the scene, and adjusting the density of the randomly generated buoys. The PointNet++ and SphereFormer combined model was then trained on the simulation dataset. After training the model performed very well on simulation data, but poorly on real-life data. To improve the results, a fixed amount of real-life data was used together with varying amounts of simulation in model fine-tuning. The resulting model performed significantly better on all the real and simulation dataset combinations considered in this work. The balanced overall performance showcased has an IoU of 0.84 and 0.80 on real-life and simulation-generated datasets, respectively. It has been shown in this work that both IoU scores are important to produce a balanced generalizable model and to use the simulation data as much as possible for training because the real-life data processing is very human-labor intensive, and thus may not be sufficient for training.

Future Work

Another approach to treating simulation and real-life datasets, which is left as future work, may be to create a model based on CycleGAN [22], which is a generative model able to transfer among domains. Here, it would transfer between simulation and real-life maritime LiDAR scan domains to generate an unlimited amount of realistic data for training the model.

In future work, it may be better to remove the Z-plane filter from the Unity simulation and have the ML model learn to differentiate the water from other ob-

jects. By coupling with less restrained Unity simulation, the ML model may be also more robust to different weather conditions.

Acknowledgements

This work was supported in part by the Old Dominion University 2022 Data Science Seed Grant and by the U.S. Department of Education under grant number 230725. The authors are grateful to Prof. Yiannis Papelis and Dr. Ahmet Saglam who provided the WAM-V LiDAR scan data, to Prof. Yaohang Li for helpful model architecture discussions, and to Dorie Parry who labeled the data. The authors thank the anonymous reviewers for their comments that helped to improve the paper.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Saglam, A. and Papelis, Y. (2023) Efficient Maritime Object Detection and Validation for Enhancing Safety of Uncrewed Marine Systems. *Proceedings of the* 25*th International Conference on Harbor, Maritime and Multimodal Logistic Modeling & Simulation* (*HMS* 2023), Greece, 18-20 September 2023, 7.

[2] Unity (2022) Unity Real-Time Development Platform. https://www.unity.com/

[3] Velodyne Lidar, I. (2019) VLP-16 User Manual. https://www.manualslib.com/manual/1407706/Velodyne-Vlp-16.html

[4] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A. and Koltun, V. (2017) CARLA: An Open Urban Driving Simulator. *Proceedings of the* 1*st Annual Conference on Robot Learning*, California, 13-15 November 2017, 1-16.

[5] Jaiswal, C., Penumatcha, H., Varma, S., AlHmoud, I.W., Islam, A.K. and Gokaraju, B. (2024) Enriching 3D Object Detection in Autonomous Driving for Emergency Scenarios: Leveraging Point Cloud Data with CARLA Simulator for Automated Annotation of Rare 3D Objects. *SoutheastCon* 2024, Atlanta, 15-24 March 2024, 1137-1143. https://doi.org/10.1109/southeastcon52093.2024.10500173

[6] Haarnoja, T., Moran, B., Lever, G., Huang, S.H., Tirumala, D., Humplik, J., *et al.* (2024) Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning. *Science Robotics*, **9**, 1-17. https://doi.org/10.1126/scirobotics.adi8022

[7] Qi, C.R., Yi, L., Su, H. and Guibas, L.J. (2017) Pointnet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space.

[8] Lai, X., Chen, Y., Lu, F., Liu, J. and Jia, J. (2023) Spherical Transformer for Lidar-Based 3D Recognition. 2023 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), Vancouver, 17-24 June 2023, 17545-17555. https://doi.org/10.1109/cvpr52729.2023.01683

[9] Epic Games (2024) Unreal Engine. https://www.unrealengine.com

[10] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., *et al.* (2019) Semantickitti: A Dataset for Semantic Scene Understanding of Lidar Sequences. 2019 *IEEE/CVF International Conference on Computer Vision* (*ICCV*), Seoul, 27 October-2 November 2019, 9296-9306. https://doi.org/10.1109/iccv.2019.00939

[11] Simon, M., Amende, K., Kraus, A., Honer, J., Samann, T., Kaulbersch, H., *et al.* (2019)

Complexer-Yolo: Real-Time 3D Object Detection and Tracking on Semantic Point Clouds. 2019 *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (*CVPRW*), Long Beach, 16-17 June 2019, 1190-1199. https://doi.org/10.1109/cvprw.2019.00158

[12] Charles, R.Q., Su, H., Kaichun, M. and Guibas, L.J. (2017) Pointnet: Deep Learning on Point Sets for 3D Classification and Segmentation. 2017 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Honolulu, 21-26 July 2017, 77-85. https://doi.org/10.1109/cvpr.2017.16

[13] Ronneberger, O., Fischer, P. and Brox, T. (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. In: *Lecture Notes in Computer Science*, Springer, 234-241. https://doi.org/10.1007/978-3-319-24574-4_28

[14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., *et al.* (2023) Attention Is All You Need. https://arxiv.org/abs/1706.03762

[15] Redmon, J., Divvala, S.K., Girshick, R.B. and Farhadi, A. (2015) You Only Look Once: Unified, Real-Time Object Detection. http://arxiv.org/abs/1506.02640

[16] Scarselli, F., Gori, M., Ah Chung, T., Hagenbuchner, M. and Monfardini, G. (2009) The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, **20**, 61-80. https://doi.org/10.1109/tnn.2008.2005605

[17] Thomas, H., Qi, C.R., Deschaud, J., Marcotegui, B., Goulette, F. and Guibas, L. (2019) KPConv: Flexible and Deformable Convolution for Point Clouds. 2019 *IEEE/CVF International Conference on Computer Vision* (*ICCV*), Seoul, 27 October-2 November 2019, 6410-6419. https://doi.org/10.1109/iccv.2019.00651

[18] Stanford Artificial Intelligence Laboratory (2024) Robotic Operating System. https://www.ros.org

[19] MathWorks (2023) Lidar Toolbox: Design, Analyze, and Test Lidar Processing Systems. https://www.mathworks.com/products/lidar.html

[20] Wave Harmonic (2022) Crest. https://github.com/wave-harmonic/crest

[21] NVIDIA (2004) GPU Gems: Chapter 1: Effective Water Simulation from Physical Models. https://developer.nvidia.com/gpugems/gpugems/part-i-natural-effects/chapter-1-effective-water-simulation-physical-models

[22] Zhu, J., Park, T., Isola, P. and Efros, A.A. (2017) Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. 2017 *IEEE International Conference on Computer Vision* (*ICCV*), Venice, 22-29 October 2017, 2242-2251. https://doi.org/10.1109/iccv.2017.244