GEOG 596B

Capstone Project

Leveraging Autonomous Vehicle or Mobile Platform Sensor Data for Urban Infrastructure Management

B. Alonzo

12/6/2018

Advisor: Jan Oliver Wallgrün

Contents

1 Introduction 3
2 Background 4
2.1 Autonomous Vehicles 4
2.2 Mobile Sensor Platforms 5
2.3 Change Detection Algorithms 6
3 Objectives 7
4 Workflow 7
4.1 Simulation 8
4.2 Sensor Data Analysis 11
4.2.1 Change Detection Analysis 12
4.2.2 PointNet 15
4.3 Infrastructure Asset Inventory 16
5 Results 16
5.1 F1 Scores 16
5.2 Analysis Results 17
5.3 Tools and Scripts 18
6 Conclusion 20
7 References 23

ABSTRACT

In this work we describe a geographic information systems (GIS) problem where organizations attempt to manage assets like signs or trees in geospatial inventories, but struggle in maintaining these up-to-date as they usually just rely on field work to identify issues. At the same time, mobile sensor platforms with LiDAR and camera sensors are becoming more widely available and potentially more so in the future with the development of autonomous vehicles with these sensors. We look at ways to leverage point cloud data from these sensors and perform change detection analysis to identify assets that may need inspection. Open source tools are employed to simulate an urban environment and generate synthetic point cloud data. We process the data to prepare it for two change detection methods, one with a simple voxel grid intersection (VB), and the other a machine learning (ML) approach using the PointNet project. The results go through an F1 score analysis to help determine test accuracies for both methods. The VB approach outperforms the ML method used, but both can still be further optimized.

1 INTRODUCTION

Recent and ongoing advances in autonomous vehicle technology by major companies is spurring innovation with the potential to improve the capabilities and lower the cost of vital LiDAR, camera, and other sensors needed by these vehicles for independent navigation. Whether as a byproduct of autonomous car usage or increased accessibility to dedicated mobile platforms with similar sensors, we can expect more point cloud data in the future for environments in proximity to transportation networks. This work examines the viability of leveraging such data to help solve geospatial problems for those environments. The work focuses on minimizing the challenge of maintaining a geospatial database with urban infrastructure assets up-to-date by analyzing mobile platform point cloud data to detect changes over time. The work follows a synthetic approach to generate analysis data by creating a simulation environment with an Unreal Engine 3D map or model. There is also an additional emphasis on utilizing open source software tools to minimize costs and facilitate sharing scripts or other data from any part of the workflow. We use the CARLA Simulator[1] (Dosovitskiy et al., 2017), meant for autonomous vehicle research, to generate a synthetic point cloud of areas visible to a virtual autonomous vehicle. The exported sensor data is processed with CloudCompare and made ready for further analysis using point cloud data change detection methods. We also test two separate methods for this analysis including a machine learning (ML) approach with a PointNet[2] (Qi et al., 2016) TensorFlow[3] implementation in Python[4], and the other a simpler voxel grid (VB) intersection technique. The results of these techniques applied to change detection make predictions on whether it is likely that a detectable change has taken place for any asset between time specific conditions. This information serves to update the geospatial database and help prioritize further confirmation more directly.

[1] https://github.com/carla-simulator/carla
[2] https://github.com/charlesq34/pointnet
[3] https://www.tensorflow.org/
[4] https://www.python.org/

2 BACKGROUND

2.1 AUTONOMOUS VEHICLES

Although long promised, technical hurdles have so far held back the realization of our science fiction vision of a world of self-driving cars. Recently, the confluence of the required technologies seems to put a world with autonomous vehicles within reach, with major companies like Google, Tesla, and General Motors pouring significant resources in the development of this technology that is showing tangible progress that will lead to the deployment of this technology in real world applications. In order to perform any autonomous tasks a vehicle with this capability is outfitted with the necessary hardware. As shown in Figure 1, this may include a combination of sensors like GPS, depth cameras, radar, and LiDAR in addition to a dedicated computer unit and critical software. These sensors allow the car to perceive its environment as it navigates. They also allow these vehicles to essentially become remote sensing platforms capable of generating big data with an important geospatial component. There are also dedicated vehicles for gathering this type of data, such as those employed by Google to create Street View. These mobile sensor platforms, especially LiDAR, may see significant cost reductions as a result of increased research, competition, and economies of scale as autonomous vehicles are increasingly deployed. These sensors will routinely capture images and generate LiDAR point clouds of the road and any adjacent infrastructure including sidewalks, traffic signs, utility poles, building façade, and other assets.

Cities and utility companies already keep detailed inventory systems to help keep track of maintenance and other costs for these assets. It is also increasingly common that these inventory systems integrate GIS databases since location is crucial to effectively manage them. However, keeping the information in these databases up-to-date can be challenging and costly since it usually requires field crews to be deployed. As a result, some of these assets can go years or months before being updated. As autonomous vehicles are increasingly deployed, and the data generated by their sensors become more available, it opens the opportunity for geospatial professionals to leverage such data to improve or automate their workflows. Figuring out effective ways to do that is the central objective of the project described in this proposal.

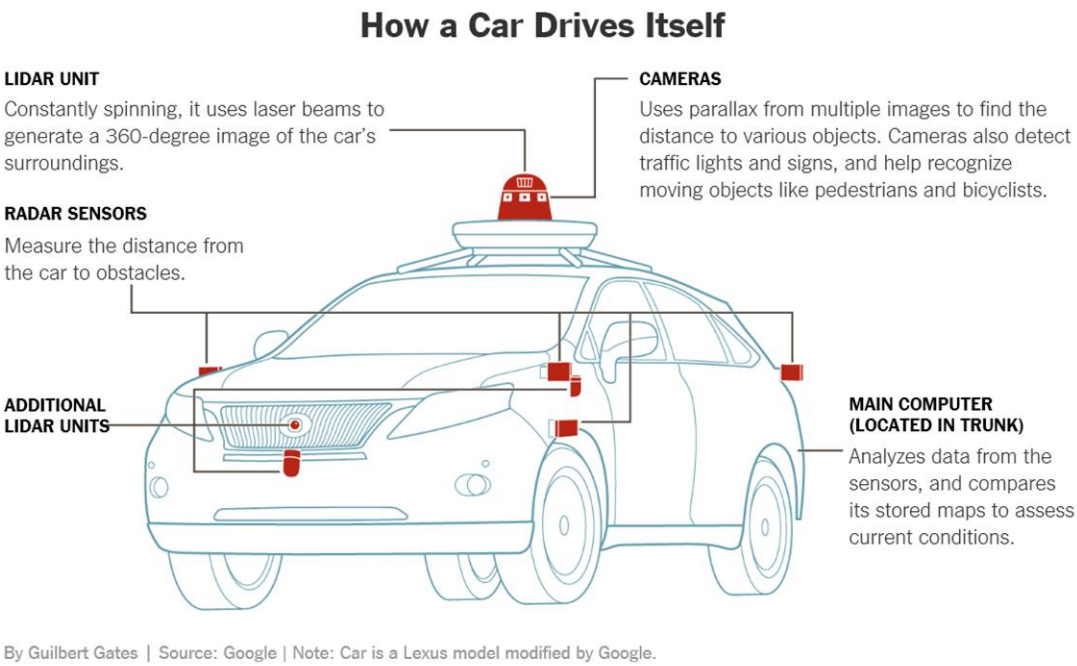

Figure 1: Autonomous vehicles come equipped with a range of sensors to perceive their environment as they attempt to perform independent navigations tasks. *(Source: Google)*

2.2 MOBILE SENSOR PLATFORMS

Currently there are mobile LiDAR hardware options with impressive capabilities, but these solutions are still expensive. Leica Geosystems has the Pegasus 2[5] laser scanner or LiDAR sensor that is intended for a mobile platform such as a vehicle as shown in Figure 2 below. There are some alternatives from other companies, such as the ROBIN[6] by 3D Laser Mapping which is a smaller version that can be mounted on a large drone or carry in a backpack. The current prohibitive cost of these platforms limits their use, but they may become more common in the future. These dedicated platforms can collect denser point clouds due to a higher resolution.

[5] https://leica-geosystems.com/en-US/products/mobile-sensor-platforms/capture-platforms/leica-pegasus_two
[6] https://www.3dlasermapping.com/robin-mobile-mapping-system/

Figure 2: Pegasus 2 platform by Leica Geosystems mounted on a vehicle. *(Source: Leica Geosystems)*

Whichever platform is used to produce the point cloud data and related images captured with the camera sensors, the data can be a resource to track streetscape changes overtime by implementing some change detection algorithms.

2.3 CHANGE DETECTION ALGORITHMS

Some work related to change detection is available in the literature. Historically a common approach for change detection was to calculate the point distance to a reference model (N. Aspert et al., 2002) as is implemented in MESH[7]. However, the need to convert between point cloud and mesh object was a major downside. There is some irony that this early approach could work rather well with simulated environments we describe later since a 3D mesh for a model can be exported more easily than generating a point cloud. A later technique common in GIS (Vögtle et al., 2004) for urban areas compared digital surface models (DSMs) from a reference and test condition to classify buildings, vegetation, and terrain. The technique relied on segmentation of the data into separate 3D objects, the building object class are compared for overlapping areas and assigned the values *non-modified, heightened, or decreased*. This technique targeted change detection in buildings after an earthquake and had limited applicability at the streetscape asset level.

One additional change detection algorithm implementation is described by Liu et al. in 2016 which uses the Apache Spark[8] application. The approach leverages cloud computing parallel processing to build and process voxel grids and then calculating geometric differences by comparing each voxel in the different point clouds. Another method is described in 2005 by Girardeau-Montaut et al. consisting of a direct point cloud comparison using an octree, which is a recursive and consistent subdivision of 3D space. This method relies on determining the Hausdorff distance between sets of points, but that means the user needs to determine a distance threshold. More recently in 2013 Aijazi et al. described a technique for change detection in which point clouds are classified to remove temporary points, and then comparing the similarity between 3D cells or voxel grids and creating a similarity map. This approach is rather complex, but shares some basic similarities with the grid occupancy voxel based (VB) approach.

[7] https://github.com/naspert/MESH
[8] https://spark.apache.org/

3 OBJECTIVES

This work intended to examine possible ways to leverage resulting sensor data from autonomous vehicles or related mobile sensor platforms to help solve a common GIS data maintenance problem. In this scenario an urban jurisdiction maintains a detailed GIS database of road-adjacent infrastructure assets, but often finds feature locations and attributes to be outdated. Part of our work focus was on ways to keep these assets updated by integrating autonomous vehicle or related platform sensor data. Specifically, it aimed to determine how to implement change detection and object classification methods in point cloud data from autonomous vehicles to help automate updates to geospatial features in a database for infrastructure assets.

One of the most important requirements for the project involved acquiring or generating sensor data needed for analysis. This meant repeatedly taking point cloud and image data for a study area before and after infrastructure assets are changed in any way. This introduced the major challenge of consistently and reliably acquiring such data, which was vital for the analysis. Lacking consistent access to the necessary sensor hardware meant alternative options had to be considered. This was solved by using a simulated environment for the study area with virtual autonomous vehicles and sensors. This allowed synthetic sensor data to be generated on demand, thus enabling an iterative and methodical approach to evaluate analysis tools and algorithms. The work relied on open source software tools and algorithms and supporting these options was a significant goal for this project. Major milestones to help achieve the main goals included these major workflow components, which are also detailed in Figure 3:

- Use a virtual 3D model of study area to simulate real world conditions
- Generate sensor point cloud data for analysis
- Perform change detection and feature extraction analysis on point cloud data by comparing a voxel intersect approach (VB) and a machine learning (ML) method
- Identify managed assets and assign change status
- Automate GIS database updates based on analysis results
- Visualize results in a web application

Fundamentally this work sought to answer questions regarding the feasibility of leveraging the sensor data at all, describing the methods and tools that proved most efficient, and the impact of the synthetic nature of the source data. In a more tangible sense It also includes a connected system of open source analysis tools, asset database, and visualization of results.

4 WORKFLOW

Unreal Engine 4 (UE4)[9] from Epic Games is a 3D video game engine that allows the creation and modification of a model that may represent an actual study area or a fictional site. The CARLA Simulator[10] is a major plugin created for UE4 that adds an autonomous vehicle simulation component while integrating its own Python API to program simulation parameters such as route and sensor information. CARLA is actively developed by the Computer Vision Center at the Autonomous University of Barcelona with support from Intel and Toyota. These two components make up the simulated portion of the environment. The analysis portion includes the use of the CloudCompare[11] software, which was

[9] https://www.unrealengine.com/en-US/what-is-unreal-engine-4
[10] https://github.com/carla-simulator/carla
[11] https://www.danielgm.net/cc/

developed by researchers with the Electricity of France state power company. CloudCompare is used to manipulate and visualize point cloud data. The change detection analysis relies on Python to compare a machine learning approach (ML) and a voxel grid intersection (VB) method. The infrastructure asset GIS database starts as GeoJSON features for simplicity but may be loaded into a PostgreSQL system with PostGIS[12]. Visualization and management of these features can be made available to possible stakeholders through a web map application developed with JavaScript libraries. Most of these software packages are open source and any contributions made during this work are also open and available on GitHub[13].

The major tasks performed are divided into three main sections or components, starting with the simulation, moving to sensor data analysis, and finally the asset inventory GIS database (Figure 3).

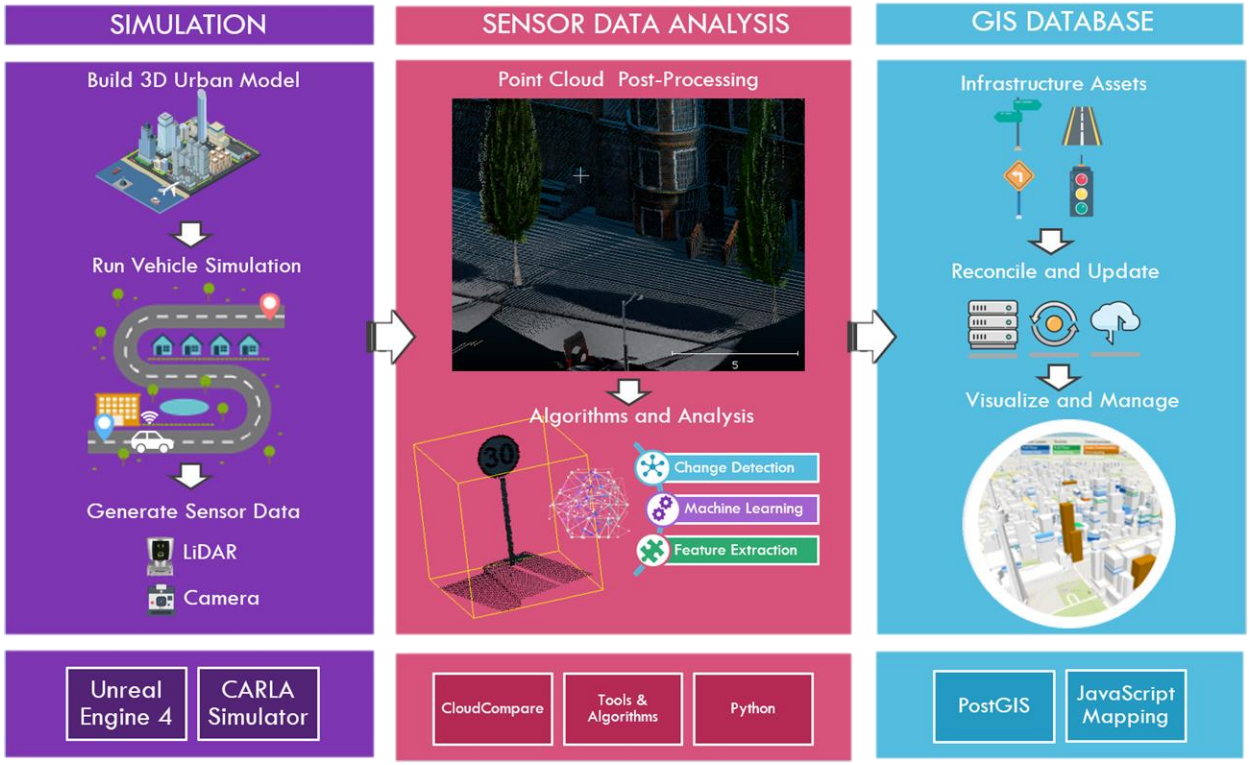

Figure 3: The three main sections of the workflow are divided into simulation, sensor data analysis, and GIS database results and visualization.

4.1 SIMULATION

Video game engines like UE4 refer their 3D model environments as 'maps' and we follow this convention. Creating your own UE4 map can be very time-consuming and, in this work, we are limited to a pre-existing fictional urban area. However, new releases of the CARLA Simulator are making it easier to build your own UE4 map by allowing you to use other software that can import GIS data as a starting point. This should help minimize the time needed to create a custom UE4 map. Some effort towards virtualizing a small study area in San Diego has been made and will be made available in follow-up work.

[12] https://postgis.net/
[13] https://github.com/bienalon/ChangeDetection_MobileLiDAR

The first portion of the work entailed generating the point cloud data using the CARLA Simulator. We compiled the source code for version 0.8.3 of the CARLA Simulator under the Unreal Engine 4.18 (UE4) library on Windows 10 following the project documentation. The default map represents a somewhat realistic virtual town layout with a mix of land use types. The study is focused on a large square block of a residential area as shown on Figure 4.

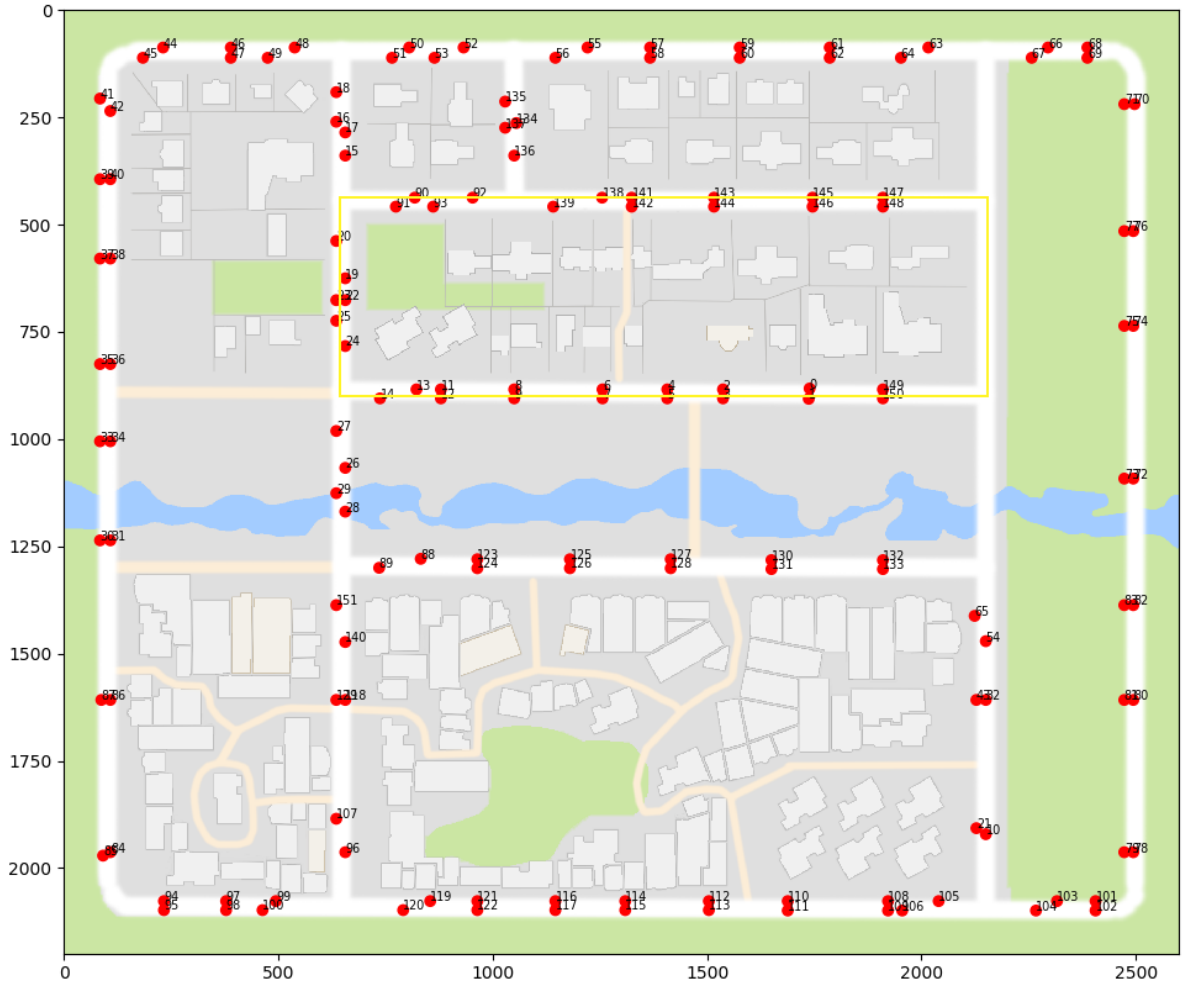

Figure 4: 2D layout of the CARLA Simulator map for Town 1 with a rectangular area highlighted in yellow showing the study area.

The game map was opened in the UE4 Editor environment to allow edits to the assets adjacent to the streets that were intended to be captured. Figure 5 shows the main interface for the UE4 Editor and CARLA plugin with the default map loaded. This is where edits to the simulated urban environment were made before running the process to generate the sensor data.

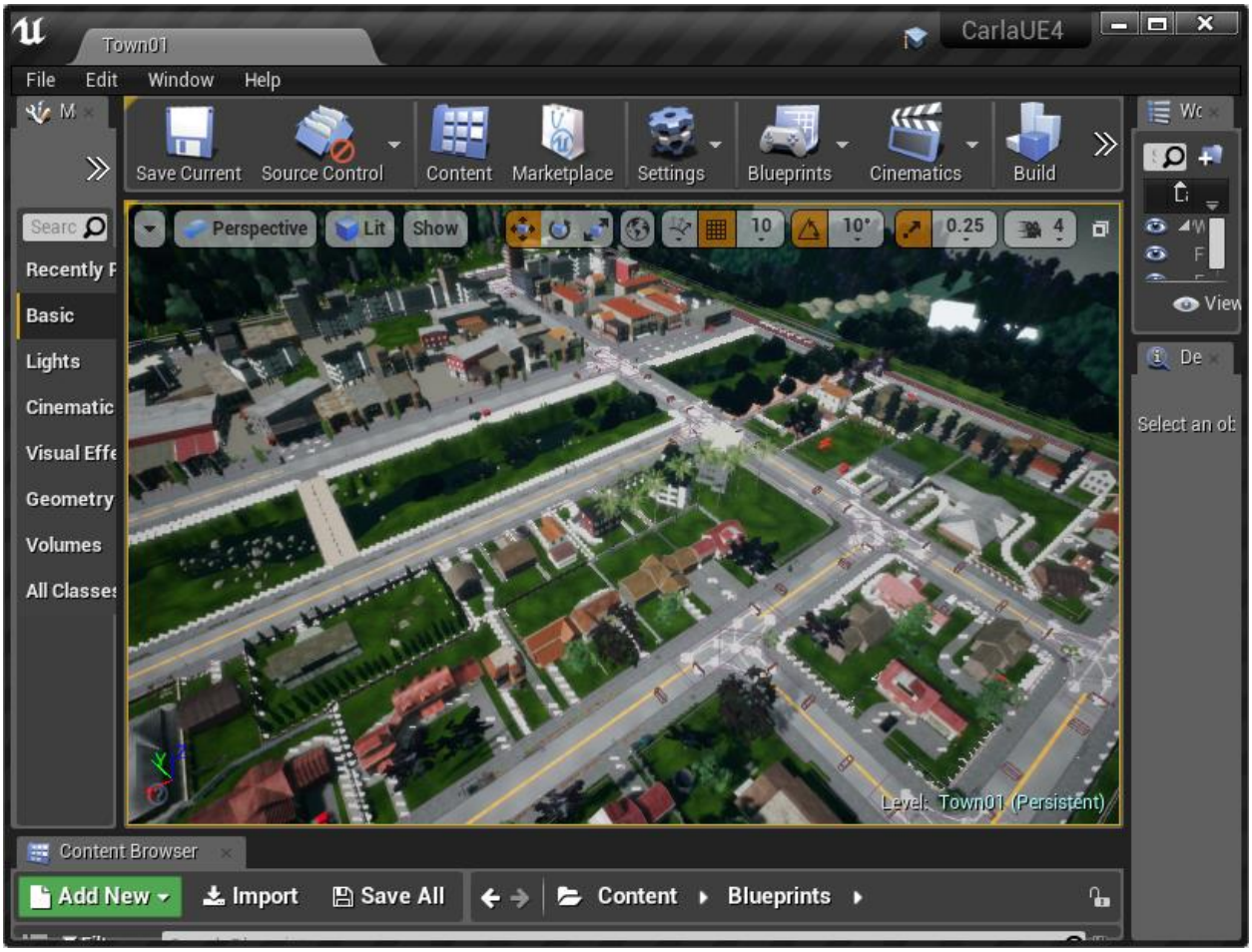

Figure 5: The Unreal Engine 4 Editor with integrated CARLA Simulator plugin and showing the default town model or map.

Additional assets were also added to have a wider selection of asset samples for the analysis. In total 200 assets meant to be tracked were placed in the map to represent the reference or initial conditions set of data. These assets include lamps, trees, traffic signs and others as shown in Table 1.

Asset Type	**# of Samples**		
Acerifolia Tree	5	Maple Tree	19
Arbusto Pine Tree	2	Parking Meter	20
Bare Acerifolia Tree	1	Pedestrian Lamp	10
Bare Sassafras Tree	2	Quercus Tree	3
Bench	9	Saccharum Tree	1
Bus Stop	4	Sassafras Tree	10
Cypress Tree	11	Speed Limit	9
Date Palm	1	Stop sign	3
Electric Pole	4	Totai Palm	4
Fan Palm	10	Traffic Light	12
Fire Hydrant	6	Trash Can	4
Lamp	37	White Fir Tree	6
Mail Box	7		

Table 1: List of assets added to the simulated environment that are street adjacent and meant to tracked between reference and test conditions.

Once the map and updated assets were ready the CARLA Simulator Python API was used to specify a vehicle actor with a LiDAR and camera sensor array. The vehicle with the sensor platform was then instructed to drive around the designated streets in the scope area of the map. At the same time, the custom script would take a sensor snapshot every 20 frames, post-process the data, and save an RGB point cloud file of the data in the Stanford Polygon File (PLY) format with a sample shown on Figure 6. The automated process generated 344 files with an average size of about 7.5 MB. This is the 'reference' environment and all derived data will be referred to as such.

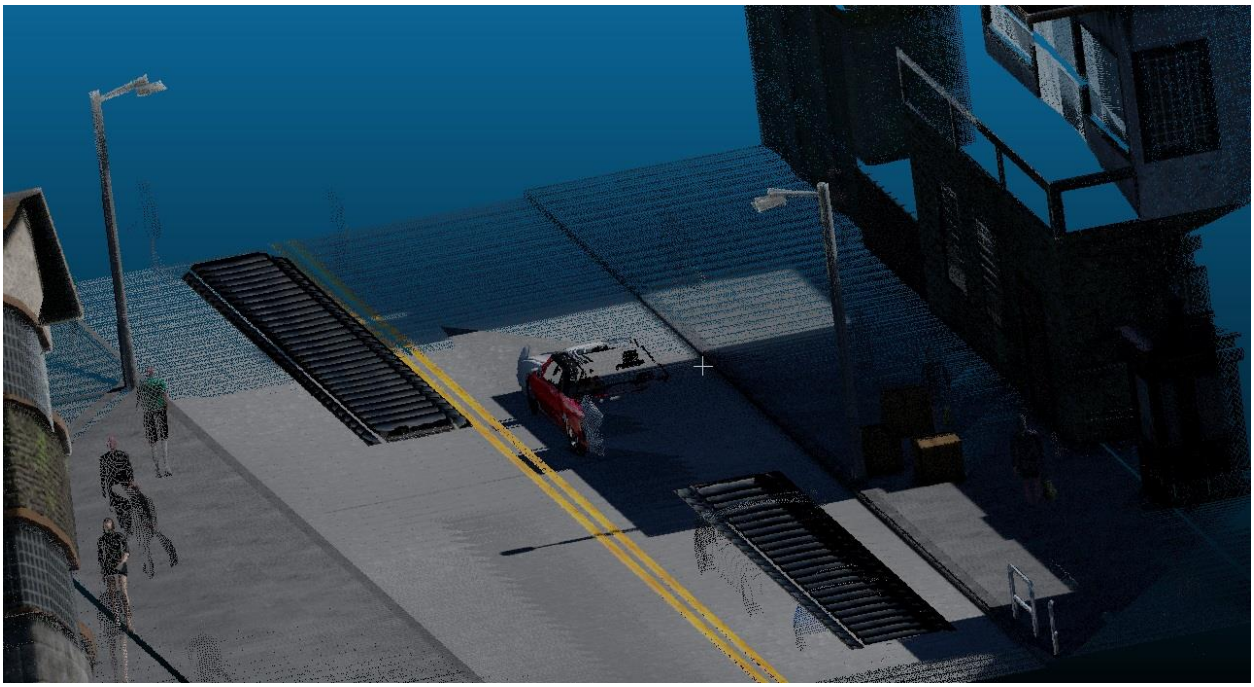

Figure 6. PLY format export from CARLA Simulator with point cloud data generated with RGB color values from virtual depth camera sensor.

The entire CARLA Simulator project folder was then duplicated to serve as the 'test' environment to generate the data needed to compare to the initial batch after some deliberate changes. The UE4 Editor was again used to make changes to the initial assets in the map, by moving, rotating, covering, or otherwise changing the some of the initial assets. Once the map reflected the new changes, the simulation ran again for the same path with the same parameters, but now capturing the changes to the infrastructure assets. This process may have been repeated multiple time as needed for analysis and machine learning training. The simulated or synthetic camera and LiDAR data included 376 PLY point cloud files to be used for the next portion of the analysis.

4.2 SENSOR DATA ANALYSIS

The sensor data export process generates a set of PLY format files per frame with point cloud information. Figure 7 shows the CloudCompare interface loaded with point cloud data created with the CARLA Simulator. Using Python and including the modules PyntCloud[14], and Pandas the small PLY files from each batch were merged into two temporally distinct point clouds covering the entire study area. Since the PLY files included significant overlap, the merged point clouds were optimized using

[14] https://github.com/daavoo/pyntcloud

CloudCompare by removing all duplicate points. The optimized reference point cloud came in at 836 MB, while the test file ended up being about 871 MB. If some of the point clouds became misaligned during processing for any reason, Open3D[15] (Zhou et al., 2018) was used to register and align them based on overlapping similarities. At this point we needed to determine the locations of all the tracked assets as they would be in an asset inventory in a GIS system. All 200 assets were digitized into a vector layer in QGIS 2.18 with attributes that included feature type, a unique feature name, and fields describing the coordinates of a square bounding box centered around each asset points. The purpose of the bounding box was to designate the area of interest around each asset to limit the size of the point cloud for analysis and to allow processing of point cloud data per individual asset. Using Python with the PyntCloud and GeoJSON modules we iterate though each asset point in a GeoJSON file and crop the reference and test point clouds with the bounding box limits resulting in two new batches of 200 ply files each.

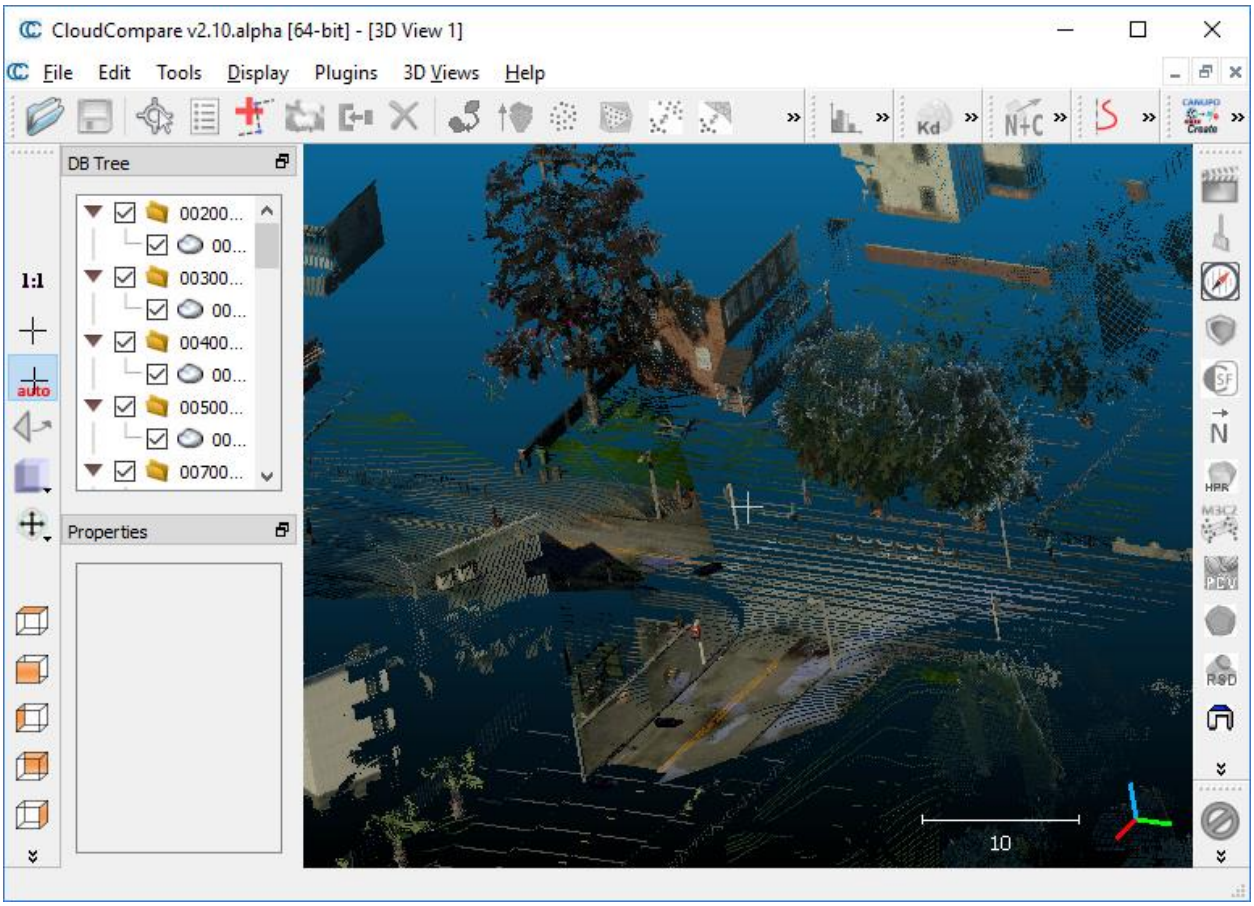

Figure 7: CloudCompare application showing point cloud generated with the CARLA Simulator ready for merging and other analysis.

4.2.1 Change Detection Analysis

This basic principle is implemented with a custom Python script again using the PyntCloud module. Our script iterates through the asset GeoJSON file to open each corresponding and identically named ply file in the reference and test folders. For each point cloud pair, the script temporarily merges them both and adds a voxel grid of shape 64x64x64 covering the extent of the combined point cloud as shown on

[15] http://www.open3d.org/

Figure 8. This process ensures that subsequent voxel grids in the analysis share the same shape and can be intersected using standard Numpy array methods. Once the overall voxel grid for the extent of the merged point clouds is created, we query the populated voxels that have points for each separate reference and test point cloud. This creates two new voxel grids unique to each condition, but that still share an overall shape. These are then intersected using the *Numpy.intersect1d* function which gives us results with the number of populated voxels shared by both point clouds, as well how many are only either occupied in the reference or test point clouds.

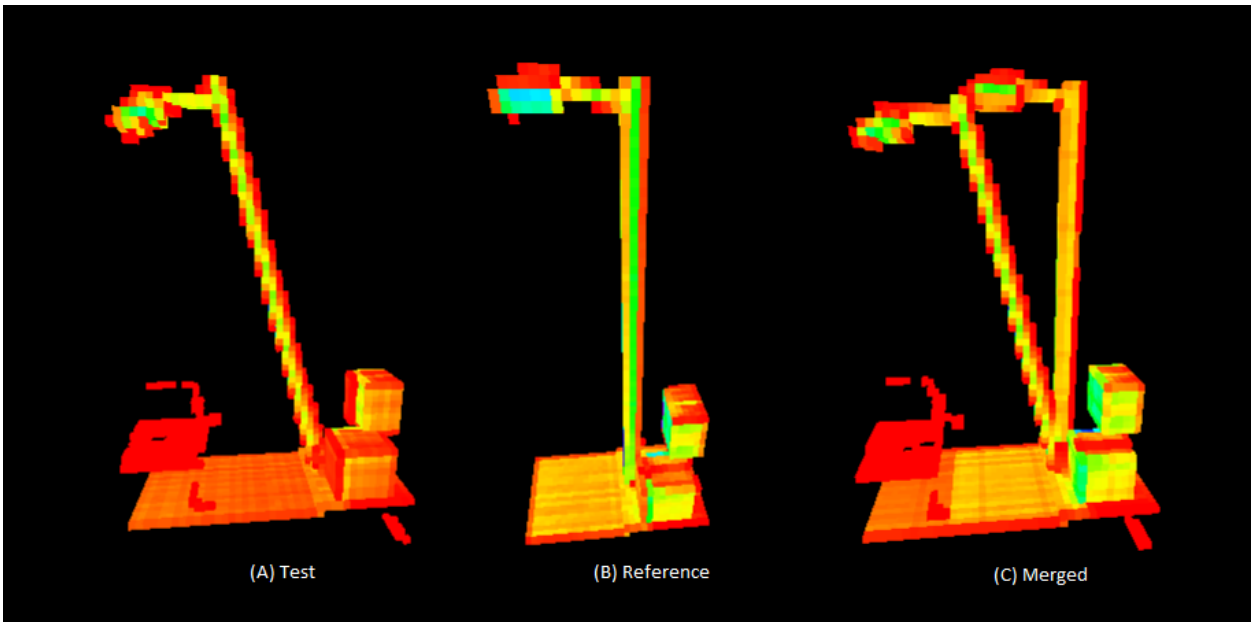

Figure 8: Voxel grid visualization with the test condition (A) showing a tilted lamp, the reference condition (B) showing a normal lamp, and the merged point cloud voxel grid with both conditions.

The script then adds fields to the GeoJSON containing these three values, which are then used to determine the 'match' percent value by simply using the formula:

$$Ri = \frac{Vi}{(Vi + Vr + Vt)}$$

The formula shows the intersection rate (Ri) is determined simply by dividing the voxel intersect (Vi) by the sum of itself, the voxel reference (Vr), and the voxel test (Vt).

Some example values resulting from this process include asset 'SpeedLimit4' which had voxel intersect value of 0.92, meaning that it was likely unchanged and further verification confirms that to be the case. It is shown in Figure 9 with little variation between the reference and test point clouds. In contrast 'SpeedLimit9' had a voxel intersect value of 0.09 indicating significant change, which can be confirmed by the apparent orientation difference due to the rotation event shown in Figure 10.

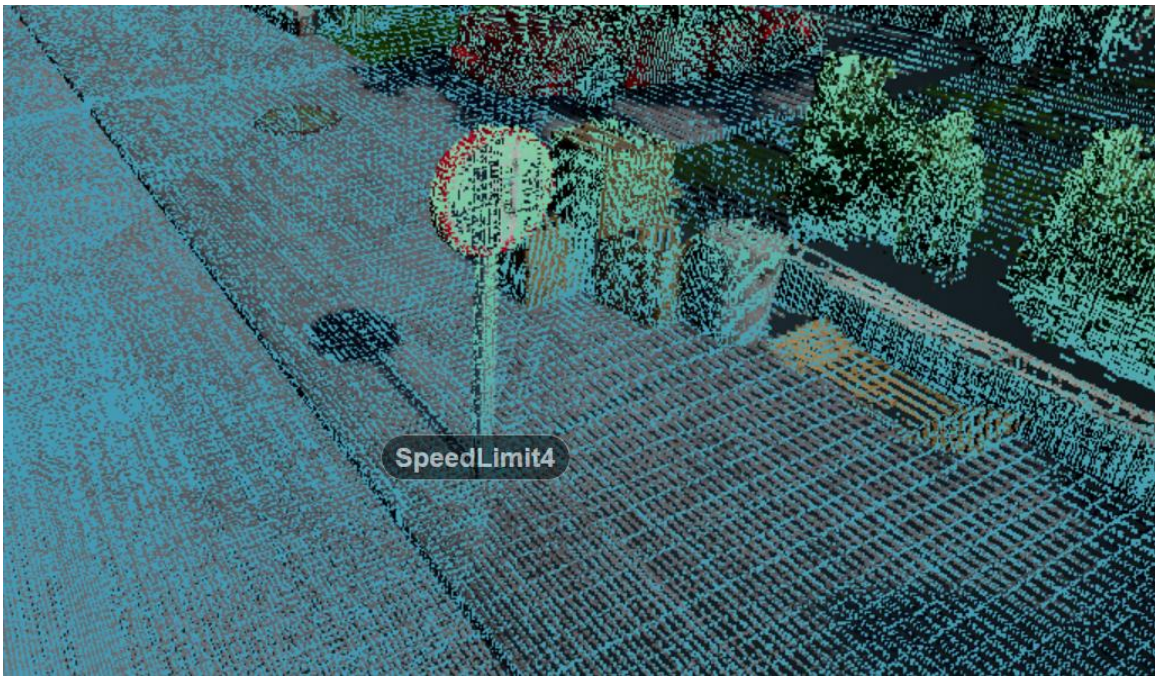

Figure 9: Asset Speed Limit 4 is shown with green test condition against RGB reference condition points overlapping and not indicating any change.

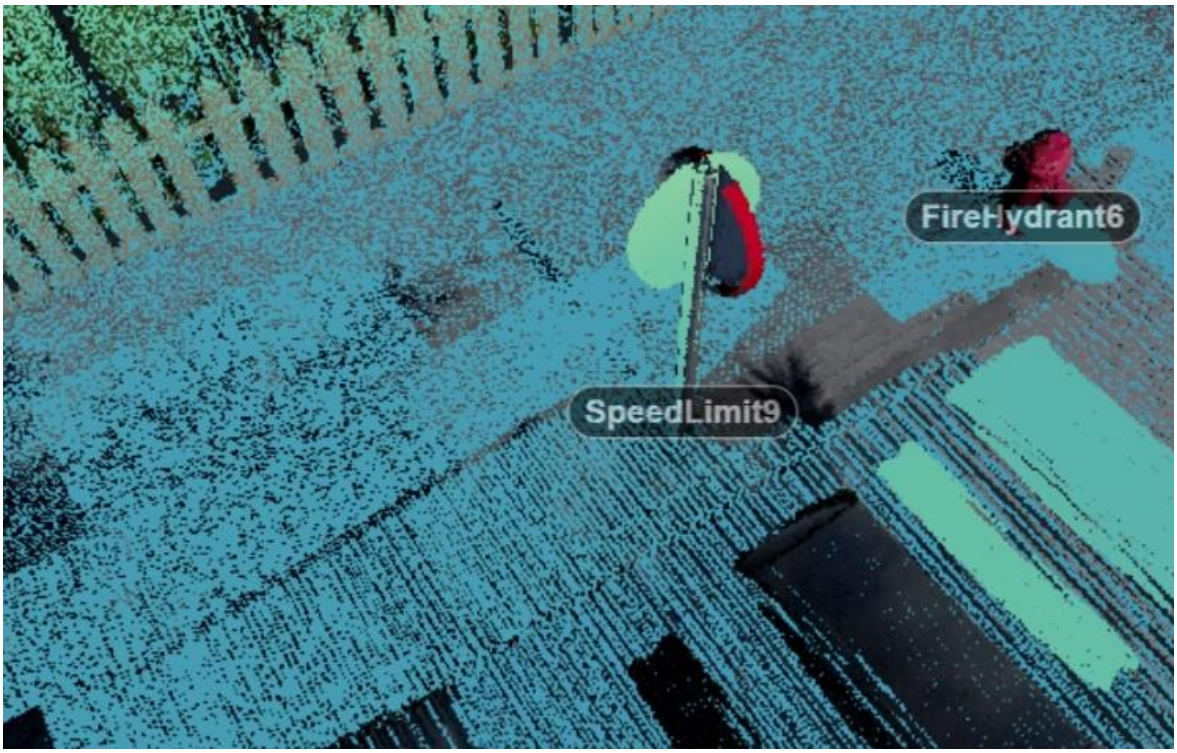

Figure 10: Asset Speed Limit 9 is shown as having been rotated along its vertical axis which denotes a significant change flagged correctly by a voxel intersect.

4.2.2 PointNet

We wanted to determine how a machine learning method would do in supporting this main workflow by using object classification. PointNet is an implementation by the original authors based on their published paper (Qi et al., 2016). The paper describes a deep learning neural network that takes direct point cloud data as input and able to perform object classification, and both part and semantic segmentation. Figure 11 shows that he PointNet architecture uses max pooling to aggregate optimal values resulting from the learning process as point functions select points deemed critical or representative.

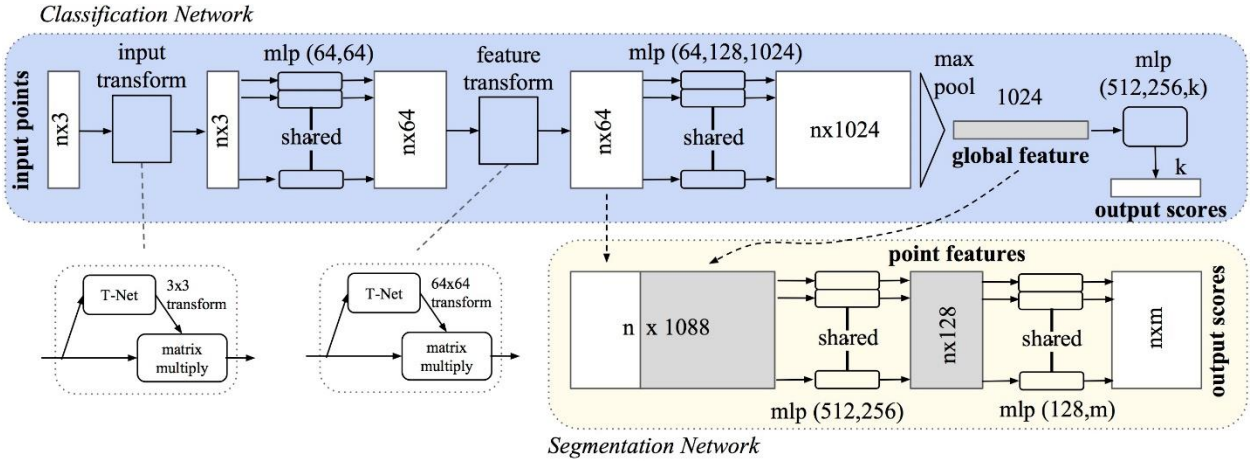

Figure 11: This workflow schematic shows the core components of the PointNet architecture and includes the path the point cloud data takes to give final prediction results. *(Source image: PointNet paper (Qi et al., 2016))*

The PointNet authors provide Python code on GitHub for implementing their neural network architecture using TensorFlow, CUDA, and cuDNN, and therefore requiring a compatible NVIDIA GPU. The available PointNet project is hardcoded to work with the ModelNet40 (modelnet.csprincton.edu) dataset. The dataset is in the h5 format and contains 2048 point clouds with 2048 points sampled from ModelNet40 3D models. As the name implies, the dataset models represent 40 different object classes. Deviating from this data structure shape of 2048x2048x3 required changes to the provided code base.

These constraints on the expected input meant that to use our own data with PointNet changes to the code would be necessary. It also meant our input data would have to be preprocessed significantly to be able to feed it to the neural network. Our point cloud data varied in density and complex shapes like trees would require more than 2048 points to retain a discernable shape. The PointNet code was changed to take 25 object classes for our corresponding number of asset types. The points in each point cloud was increased from 2048 to 8192 points, and the total number of point clouds inside the h5 file went down to just 200 including our total number of assets. Additional Python code was developed to process or asset ply filed by up-sampling if lower than 8192 points and subsampling when higher. These point clouds were then assigned class label codes and bundled into an h5 file in the expected format. This process was performed separately for the reference point cloud batch as well as the test batch. These files were fed to the PointNet neural network as the required training and evaluation datasets, and the training portion was run through the default 250 training epochs. Due to the high GPU hardware requirements the training took place on Microsoft Azure Cloud on an NC6 Standard virtual machine.

After training the model with the new input data we ran the evaluation script which output a simple text file with a sequential list of all 200 assets identified by class ID and a corresponding value with the predicted class. The script was also modified to also include another column in the output file with a confidence score for each asset prediction.

In this work we attempt to use the object recognition or point cloud classification capabilities of the PointNet project in a change detection context. Since PointNet is designed to look for similarities or differences between a training (our reference condition data) and an evaluation (our test condition data) point cloud, we could argue that ideally a class prediction from PointNet is a measure of similarity. When PointNet evaluates an input point cloud based on the TensorFlow model resulting from the ML training it assigns a confidence or probability value for all the possible classes (asset types) in the data. Its final class prediction is simply the highest scoring value, and these are derived from the *softmax* function implemented in the final layer of the neural network classifier in PointNet. As the returned value is normalized between 0 and 1, we use it as a confidence score, but for the one that is the actual class of the asset instead of the one for the predicted class, which is just the highest value returned by the TensorFlow function used. This confidence score is used in the results analysis as a threshold value for the F1 score analysis.

4.3 INFRASTRUCTURE ASSET INVENTORY

In a GIS context an asset inventory can be kept in a range of geospatial formats, from a simple local shapefile or GeoJSON to a more robust relational database. Ideally, such data would be stored in something like an ArcGIS Enterprise geodatabase or the open source Postgres/PostGIS database. Our inventory of 200 street-adjacent assets is kept initially in a GeoJSON for easier data manipulation with local Python scripts that integrate it closely with several steps in data processing including storing results. However, in a production environment this would be reconciled with a version stored in a PostGIS database with those assets.

5 RESULTS

5.1 F1 SCORES

As part of the analysis results, we attempt to compare the relative accuracy of each change detection method to help us interpret the data to have better informed conclusions. The ML method code was modified to generate a TensorFlow prediction confidence value for each asset class on a range between 0 and 1. The VB method created a result providing the decimal percent of intersecting voxels in the merged grid. Both these values were used to independently define threshold values to run an F1 score analysis.

Table 2 shows how assets were randomly divided into training and evaluation datasets, and each asset based on the threshold were categorized as true positive (TP), true negative (TN), false positive (FP), and false negative (FN). The training dataset totals were used to calculate precision, recall, and the F1 scores for threshold increments of 0.1 between 0 and 1. These threshold values from low to high were meant to check a broad range to see how the accuracy results clustered and to identify the highest scoring F1 score. The best performing threshold for each method (0.8 for ML and 0.3 for VB) were used to calculate

the F1 scores on the evaluation dataset for both the ML and VB methods. For the evaluation dataset the final F1 scores for our samples were 0.56 for the ML method and 0.84 for the VB approach with a 0.28 difference between both scores. These preliminary results show a significant difference between the two tested methods with the VB method outperforming our ML implementation in change detection.

TRAINING DATASET

Method	TP	FN	FP	TN	Threshold	Precision	Recall	F1 Score
VB	13	11	1	41	0.10	0.93	0.54	0.68
ML	8	16	1	41	0.10	0.89	0.33	0.48
VB	19	5	1	41	0.20	0.95	0.79	0.86
ML	9	15	1	41	0.20	0.90	0.38	0.53
VB	20	4	1	41	0.30	0.95	0.83	0.89
ML	10	14	1	41	0.30	0.91	0.42	0.57
VB	20	4	2	40	0.40	0.91	0.83	0.87
ML	10	14	1	41	0.40	0.91	0.42	0.57
VB	21	3	3	39	0.50	0.88	0.88	0.88
ML	11	13	1	41	0.50	0.92	0.46	0.61
VB	22	2	6	36	0.60	0.79	0.92	0.85
ML	13	11	1	41	0.60	0.93	0.54	0.68
VB	22	2	10	32	0.70	0.69	0.92	0.79
ML	13	11	1	41	0.70	0.93	0.54	0.68
VB	22	2	15	27	0.80	0.59	0.92	0.72
ML	15	9	4	38	0.80	0.79	0.63	0.70
VB	24	0	29	13	0.90	0.45	1.00	0.62
ML	16	8	7	35	0.90	0.70	0.67	0.68

EVALUATION DATASET

Method	TP	FN	FP	TN	Threshold	Precision	Recall	F1 Score
VB	26	5	5	98	0.30	0.84	0.84	0.84
ML	19	12	18	85	0.80	0.51	0.61	0.56

Table 2: F1 score results table shows for both methods (ML and VG) the number of true positives, true negatives, false positives, and false negatives for given thresholds. The table shows the precision, recall, and F1 score derived for each threshold.

5.2 ANALYSIS RESULTS

Based on the evaluation dataset thresholds, we can draw a couple of examples of false positive and false negative results for each change detection method. For the ML approach out of the 18 false positives in the evaluation dataset 66% were trees. For example, asset 'QuercusTree1' was misclassified by predicting it was a traffic light when there was really no change and the confidence score for the actual asset type was 0.009, which indicates PointNet is wrongly confident that it was not a Quercus tree. One potential factor for these results is that tree point clouds tend to have more variation between different areas due to their organic branching. They also tend to spread out and it could be more likely that a cropped point cloud area around a tree and branches contains other structures or objects in close proximity. Perhaps PointNet is focusing on a batch of points that seem interesting for some reason or have a pattern like another simpler class. In contrast, the 12 false negatives for the same dataset and

method contain 83% lamps or speed signs and hardly any trees. These had their classes correctly predicted with high confidence scores by the ML method and were therefore flagged as likely unchanged. The actual changes that were missed for these in this case could include significant tilts like those shown in Figure 11.

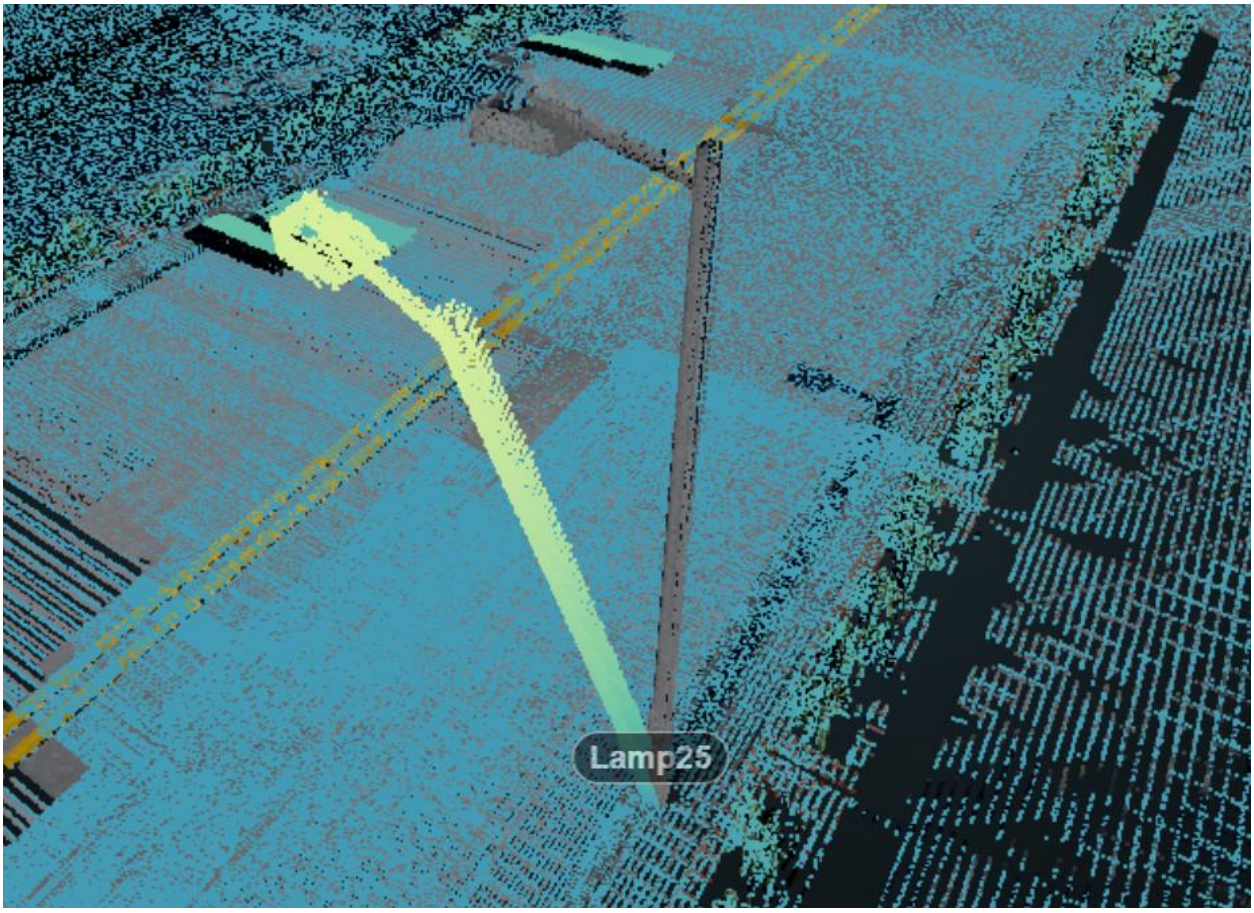

Figure 11: Lamp 25 showing reference condition in gray and test condition in yellow/green.

For the VB method the 5 false positives consisted of trees and fire hydrants. Perhaps the fire hydrants are too small and too many voxels are being created with the default 64x64x64 shape, thus making it more sensitive to variation with voxels containing few points. This issue will have to be addressed in a future update. The false negatives are almost all lamps with changes that included less spatial displacement, like rotation or tilts, but still had voxel intersects with most between 0.60 to 0.76.

5.3 TOOLS AND SCRIPTS

The workflow required the development of a set of scripts to implement its various portions. Python was chosen as a common thread to facilitate potential automation and the integration of the different components, most of which already exposed their capabilities through this popular programming language. Most significant programming contributions are intended to remain open source and be shared on a GitHub repository for the project. This way other developers or researchers may find some of the code useful for their own projects.

The CARLA Simulator requires use of its Python API to control the path of simulated autonomous vehicles, sensor placement and parameters, and frequency or format of the generated data. As a result, we developed a custom Python script to execute this portion of the simulation process. The script was also modified as needed when required by the export process. Developers should compare this script with some of the standard samples provided by the CARLA Simulator project to see if it more closely meets the needs of their intended application. It is still recommended to use the API documentation for any type of customization. In addition, there is a Python script for preliminary merging and subsequent cropping around each asset location. Instead of very long scripts performing large portions of the workflow, we prefer to break those up into re-usable smaller scripts and functions that perform certain parts of the workflow. This should allow developers to more easily leverage these scripts within their own workflows, which will likely differ from this example.

In support of the change detection analysis, developed code include several Python scripts implementing various recommended algorithms or for training machine learning modules. For the ML approach the PointNet implementation used for this work relied almost entirely on Python. The original code was customized to accommodate needs specific to our workflow and these changes will be shared by creating a fork from the original repository on GitHub. It may be better to use this fork than the original repository if developers intend to use their own input data. Closely related to this, several additional scripts were developed to prepare exported simulation data to be used with PointNet. These perform a range of tasks that include normalizing point counts, reshaping arrays, shifting coordinates and scaling, assigning classes, and bundling asset point clouds into H5 format files. For the VB method, several scripts or functions were developed to create voxel grid arrays, merge conditions for comparison, perform an intersect, and calculate resulting values.

These scripts also record results of both the ML and VB processes in a GeoJSON file, by reading, adding, and population fields or attributes. Python scripts were also developed to help analyze the results by performing an F1 Score analysis. Some of the tasks include separating datasets into training and evaluation, assigning true positive, false negatives, etc. to each asset based on a given threshold, calculating precision and recall, and recording the results as CSV files.

Some scripts help automate the integration of the simulation, analysis, inventory, and visualization components of the project. The focus of this integration would mostly be in identifying analysis results for managed asset features and appropriately editing the geospatial database to reflect the changes detected during the analysis. The final portion of the project consists of a web application to visualize the contents of the infrastructure asset database and point cloud results. The application is based on JavaScript mapping libraries like OpenLayers and Cesium JS as part of the Potree viewer. Some of the Python scripts used and that are already on GitHub are listed in Table 3. These are likely to be updated in the future and could include file name changes. Other files are anticipated to be added to the repository on a regular basis in the near-term.

Script File	Description
crop.py	Crops point cloud around each asset point
f1_score_analysis.py	Takes geojson with attributes and performs F1 score analysis
join_ml_eval.py	Incorporates PointNet CSV results into geojson
merge_ply.py	Merges PLY files exported from CARLA Simulator
normalize_points.py	Translates coordinates and scales points to normalize shapes
obj_to_ply.py	Convert 3D model object to mesh or point cloud
process_h5.py	Labels and loads point clouds into HDF5 files for PointNet
process_pcd.py	Performs voxel grid intersect analysis
split_geojson.py	Splits a geojson into two random by a given ratio

Table 3: Lists some of the script files in the GitHub repository which were developed as part of the workflow.

6 CONCLUSION

When generating the point cloud data, we needed to have a sense of how to minimize gaps in the point cloud data. For a single vehicle this was a function of speed and how often to export a simulated point cloud snapshot. One way to do this was to slow the vehicle down and export a point cloud every 20 frames or so to generate a reasonably dense point cloud with few gaps that worked for our purpose and based on hardware limitations this was our best option. If not limited by hardware, the more realistic approach would be to have multiple vehicles moving around and concurrently exporting the LiDAR. Using either method still required the preliminary merging of the exported PLY files, which aggregates the point cloud, but removing duplicate points too minimize file size. When merged the point clouds for an area the size of our study area and with a good point density was around 850 MB. These two export methods reflect the difference between a dedicated and robust platform like the Pegasus 2 or a current automated vehicle sensor like a Velodyne[16] option. In practice, if the platform is moving fast and the sensor is low resolution then there will be a greater need to aggregate exported data from multiple vehicles to create a denser point cloud. However, research in this very competitive field means that improved[17] and cheaper[18] sensors producing denser point clouds could decrease the need to aggregate data.

The synthetic nature of our point clouds generated in a simulated environment could be a factor affecting the performance of both methods, which may favorably impact the results from the VB approach. For example, our synthetic point clouds contain minimal noise, with potential interference being limited to moving actors like vehicles and pedestrians. Although it is difficult to replicate the entropy of real systems in our simulated environment, we could introduce additional noise to the point clouds during post-processing by adding slight shifts, jittering of points, or introducing additional actors. It would be helpful to determine the exact impact this issue would have on the results, but the fact that realistic moderate noise would still be a relatively small number of points that impact may not have a huge effect on our current conclusions.

[16] https://velodynelidar.com/

[17] https://arstechnica.com/cars/2018/11/why-millions-of-lasers-on-a-chip-could-be-the-future-of-lidar/

[18] https://arstechnica.com/cars/2018/01/driving-around-without-a-driver-lidar-technology-explained/

One major advantage of our ultimate goal of using the change detection analysis to identify likely changes to prioritize further inspection of the assets is that any positive results are a gain. This means that even if only half or a quarter of the actual changes are detected, it is still helpful for prioritizing further confirmation. Nevertheless, increased accuracy of detection is ideal as it lowers the overall the effort needed to update the asset inventory data. The results were not perfect, but either method or a combination of both can help designate the bulk of the features as either likely or unlikely to have experienced a significant change. At the very least, it will help inform the decision on how to prioritize further manual inspection through other sensor data or field inspections. In this regard, the ML approach had results that were not ideal, but still helpful and it could potentially be improved through some workflow changes or customization. The results from the VB method were very encouraging and could be applied in a production environment if they still hold up to confirmation with non-synthetic point cloud data. In practice it could be applied by running point cloud data through the model workflow in combination with the GIS asset inventory. Once the geospatial features have fields populated with a likelihood of changes detected, an analyst would review the ones flagged, confirm it by corroborating it from aerial imagery or otherwise provide that priority data to a field crew for further inspection and follow their regular asset maintenance workflow. Fundamentally, our results suggest that leveraging point cloud data for this type of change detection is viable, but there is room for improvement.

This work could potentially be improved in several ways, and further research is required to more precisely determine the factors contributing to the apparent differences between the two methods. On the simulation side, the project used CARLA version 0.8, but it has been updated to version 0.9 which included a reworked API with new virtual LiDAR sensor system that may provide improvements to the quality of the generated point clouds. It also brings better support for using GIS data as a starting point to create new environment models. This would help in creating a study area environment based on real world locations. On the change detection side and starting with the ML method, we need to examine new ways to improve its performance. One potential improvement would be to use a more granular classification scheme by removing the asset class groupings and assign a class per asset feature instead of type, this will perhaps help the training model to save more specific information on each asset. The downside of this would be that now with 200 classes instead of 25 the training would use even more processing power, and this is for a method already inefficient when it comes to hardware resources, especially when compared to the VB method. We could also test if cleaning the reference condition point clouds and or adding more complete asset models as a reference has any positive impact on the results. Ultimately, if these additional efforts do not significantly improve the performance of the PointNet ML method, there are other ML algorithms available that could also be tested.
For the VB approach the next step would be introducing more variance in the synthetic point cloud by adding more distortions in post-processing to better simulate real world in noise in point clouds. This brings up to the next major task for any method to acquire real point cloud data for a small area with before and after conditions to test our workflow. Another possibility is to incorporate RGB color values into our comparison to be able to detect changes such as the writing on a sign and whether this has any advantage over just using an image.

The experience with the many open source tools used for this project was quite positive, whether the tool was familiar or new. The CARLA Simulator performed very well once compiled and its very active development keeps making it better with every new release and it should be well supported in the foreseeable future. The same applies to Open3D. It has great performance and its active development is ongoing although at a slower pace than CARLA. CloudCompare was also very useful for visualization and some point cloud calculations, but it takes a lot of memory with larger point clouds. The CloudCompare open source project is even older, more stable, and mature than either CARLA and Open3D and should

also be supported well into the future. The same can be said of QGIS, Visual Studio Code, and TensorFlow, which are solid tools with a bright and active future. PointNet is static, but the authors also have a PointNet++ project with additional capabilities, but the repositories are not regularly updated. Python, of course, is extremely popular with widespread use and will likely continue to attract developers that create even more modules. PyntCloud was the only tool used with a major role that is not actively developed since it is such as small and obscure project and the original developer is focused on other things.

Overall the project had many components and was perhaps too ambitious given the time constraints, unknowns, and their relative complexity. The project deliverables ended up having a somewhat reduced scope, but my initial goals may have been too broad. Some of the major factors contributing that were time-consuming technical challenges setting up unfamiliar technologies. These included some difficulty compiling the CARLA Simulator as needed for editing in the UE4 Editor, problems finding point cloud libraries that were able to perform the voxel intersect, finding and implementing a machine learning method, among other similar challenges. Despite that, the whole experience was positive, and still enjoyable and I learned some new things. I encountered new Python modules I had not used before like PyntCloud, and libraries like Open3D, managed to get an introduction to machine learning, and found many other tools that I may use in the future.

7 REFERENCES

Aijazi, A. K., et al., 2013. Detecting and Updating Changes in Lidar. Point Clouds for Automatic 3D Urban Cartography. In: ISPRS Annals of Photogrammetry, *Remote Sensing and Spatial Information Sciences. Vol. II-5/W2. pp. 7–12.*

Aspert, N. et al. **MESH**: **M**easuring **E**rror between **S**urfaces using the **H**ausdorff distance. *Proceedings of the IEEE International Conference on Multimedia and Expo 2002 (ICME), vol. I, pp. 705-708*

Dosovitskiy, A. et al. (2017) CARLA: An Open Urban Driving Simulator. *Proceedings of the 1st Annual Conference on Robot Learning, 1-16*

Girardeau-Montaut, D. eta al. (2005). Change detection on point cloud data acquired with a ground laser scanner. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences.* 36.

Guerra, E. (2016) Planning for Cars That Drive Themselves: Metropolitan Planning Organizations, Regional Transportation Plans, and Autonomous Vehicles. *Journal of Planning Education and Research, 36(2)*, 210-224.

Liu, K. et al. (2016) Change Detection of Mobile LiDAR Data Using Cloud Computing. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLI-B3. 309-313.*

Qi, C. et al. (2016) PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv preprint arXiv:1612.00593*

Vögtle, T. et al, 2004. Detection and Recognition of Changes in Building Geometry Derived from Multitemporal Laserscanning Data. *Proceedings of the XXth ISPRS Congress (Istanbul), Vol. 35, Part B2, p. 428-433.*

Zhou, et al. (2018) Open3D: A Modern Library for 3D Data Processing*.arXiv:1801.09847*