

Marius Udnæs

Semantic Dataset Generation and Detection For Industrial Environments

Masteroppgave i Kybernetikk og Robotikk

Veileder: Konstantinos Alexis

Medveileder: Nikhil Khedekar

Juni 2022

Marius Udnæs

Semantic Dataset Generation and Detection For Industrial Environments

Masteroppgave i Kybernetikk og Robotikk
Veileder: Konstantinos Alexis
Medveileder: Nikhil Khedekar
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk



Kunnskap for en bedre verden

Semantic Dataset Generation and Detection For Industrial Environments

Marius Udnæs

June, 2022

Abstract

Point cloud acquisition and semantic segmentation is an essential tool in the digitalization of industrial plants, yet much of this work remains manual. In collaboration with Cognite, who provides Cognite Data Fusion (CDF) to gather and process enormous volumes of data for industrial customers, this master thesis aims to investigate the use of semantic segmentation for autonomy in industrial zones. We focus our efforts on the lack of annotated data, and present a novel Synthetic Semantic Dataset Generator (SSDG) to be able to create valuable 3D point cloud datasets for training semantic segmentation networks. Particularly focus is placed on closing the reality gap between real-world and synthetically generated data by employing domain randomization, data augmentation, and increasing resemblance by modeling a real-world lidar with noise.

No ready-labeled large open-source industrial point cloud exists to date. To enable the usage and deployment of deep learning methods in the context of 3D semantic segmentation in industrial zones, as well as validating the proposed SSDG method, this thesis builds a high quality industrial LiDAR dataset containing 3.7 million data points collected from a set of four commonly found object categories. To validate the industrial dataset, we present point cloud segmentation results and a qualitative analysis from state-of-the-art segmentation networks and assess the quality and usability of the dataset. The results demonstrate that the best segmentation network achieves a good overall segmentation mIoU of 87.88%. Our method produces a useful industrial semantic point cloud dataset that can be employed in future perception pipelines for Simultaneous Localization and Mapping (SLAM), visual place recognition and the like. The Synthetic Semantic Dataset Generator (SSDG) is publicly available, enabling researchers to generate new labeled datasets tailored to their needs.

Sammen drag

Punktskyinnhenting og semantisk segmentering er et viktig verktøy i digitaltaliseringen av industrianlegg, men mye av dette arbeidet forblir idag manuelt. I samarbeid med Cognite, som leverer Cognite Data Fusion (CDF) for å samle og behandle enorme mengder data for industrielle kunder, har denne masteroppgaven som mål å undersøke bruken av semantisk segmentering for autonomi i industrielle soner. Vi fokuserer på mangelen av datasett, og presenterer en ny Syntetisk Semantisk Dataset Generator (SSDG) for å kunne lage verdifulle 3D-punktsky datasett for trening av semantiske segmenteringsnettverk. Det settes spesielt fokus på å lukke virkelighetsgapet mellom virkelige og syntetisk genererte data ved å bruke domenerandomisering og data augmentasjon. Det finnes idag ingen open-source industrielle punktskyer. For å muliggjøre bruk og distribusjon av dyplæringsmetoder for 3D semantisk segmentering i industrisoner, samt validering av det foreslåtte SSDG-metoden, bygger vi et høykvalitets industrielt LiDAR-datasett som inneholder 3,7 millioner datapunkter. For å validere det industrielle datasettet presenterer vi resultater og en kvalitativ analyse av state-of-the-art segmenteringsnettverk trent på datasettet. Resultatene viser det beste nettverket oppnår en god mIoU på 87,88%. Metoden vår produserer et nyttig industrielt semantisk punktskydatasett som kan brukes i fremtidige persepsjonsrørledninger for simultan lokalisering og kartlegging (SLAM), visuell stedsgjenkjenning og lignende. Den Syntetiske Semantisk Dataset Generatoren (SSDG) er offentlig tilgjengelig, noe som gjør det mulig for forskere generere nye datasett skreddersydd for deres behov.

Contents

Abstract	iii
Sammendrag	v
Contents	vii
Figures	ix
Tables	xiii
Acronyms	xv
1 Introduction	1
1.1 Background and Motivation	1
1.2 Aim and Scope of Thesis	3
1.2.1 Scope	3
1.2.2 Aim	3
1.2.3 Contributions	4
1.3 Thesis Outline	4
2 Background and Related Work	5
2.1 Light Detection and Ranging (LiDAR)	5
2.1.1 The LiDAR sensor	5
2.1.2 Point Cloud	6
2.1.3 LiDAR Image	7
2.2 Deep Learning	7
2.2.1 Neural Networks	7
2.2.2 Convolutional Neural Networks	9
2.2.3 Training Neural Networks	11
2.2.4 Improving neural networks	11
2.2.5 Loss functions	13
2.2.6 Regularization	15
2.2.7 Normalization	16
2.2.8 Metrics	17
2.2.9 Deep Learning Libraries	18
2.3 Semantic Segmentation	19
2.3.1 Projection-based methods	21
2.3.2 Point-based networks	26
3 Method and Implementation	29
3.1 Synthetic Semantic Dataset Generator (SSDG)	29
3.1.1 Pipeline Overview	30

3.1.2	Simulation Libraries and Tools	30
3.1.3	Gazebo Simulation	32
3.1.4	Labelling Node	39
3.1.5	Bridging the reality gap	42
3.2	Industrial Segmentation Dataset	47
3.2.1	Dataset format	48
3.2.2	CAD Models	49
3.2.3	Sequence creation	52
3.2.4	Overview of the resulting dataset	53
3.3	Segmentation Models	56
3.3.1	Network considerations	56
3.3.2	Implementation Details	58
4	Results	61
4.1	Experimental Setup	61
4.2	SqueezeSegV3	62
4.3	SalsaNext	62
4.4	SqueezeSegV3 and SalsaNext Comparison	63
4.5	Final model analysis	64
4.5.1	Quantitative Results	64
4.5.2	Qualitative Analysis	65
4.6	ROS Node	68
5	Discussion and Further Work	69
5.1	Summary	69
5.2	Synthetic Semantic Dataset Generator (SSDG)	70
5.2.1	Reality gap	70
5.2.2	Prospects for future research	71
5.3	Industrial Dataset	71
5.4	Segmentation Networks	72
5.5	Further work in semantic autonomy	73
6	Conclusion	75
	Bibliography	77

Figures

1.1	An Oil refinery model.	1
2.1	An example of the Ouster OS0 LiDAR with sample configurable parameters. α is vertical FOV, θ is vertical resolution, β is horizontal FOV and ϕ is horizontal resolution. ¹	5
2.2	Point cloud of a staircase, generated using method in Section 3.1. 3D CAD model from the ModelNet40 [5] dataset.	6
2.3	2D Range image of a an industrial scene. Colors stem from a semantic label, but can also be given by intensity or RGB channels from a camera.	7
2.4	The perceptron model, here consisting of 4 inputs, 4 weights and a single neuron.	8
2.5	The Multi-layer perceptron, with an input layer, hidden layer, and output layer.	9
2.6	An example of a CNN architecture, with an input layer, feature-extraction layer and classification layer. The feature-extraction layer in this particular example uses ReLU as activation and MaxPooling for downsampling.	9
2.7	Three Activation functions: Linear, Sigmoid and ReLU	12
2.8	The ReLU function and its counterpart LeakyReLU [9].	13
2.9	The Intersection over Union.	17
2.10	From top to bottom: input image, prediction, and ground truth. Input color is based on intensity, and grey represents tanks, blue represents stairs, red represents chimneys while green represents other objects. The neural networks predicts large parts of the tank wrong, but since most of the image is black the reported accuracy will still be high. This is the reason accuracy is not always a good evaluation metric.	18
2.11	A semantically labelled point cloud of a refinery, where each color represents a category. Created using SSDG.	19
2.12	One of the stairs from ModelNet40 [5] projected onto a spherical projection. The image is a combination of two projections: the top image presents the intensity-based image, while the bottom presents the label map.	21

- 2.13 The ResNet blocks used in the encoder layers. They consist of a 2D Convolution along with LeakyRELU activation function, and batch normalization. The double skip connection is added at the end of the block. 25
- 2.14 The novel combination of dilated convolutions introduced in SalsaNext. The dilated convolutions have a rate of 2, meaning that there is one space between each kernel element both row-wise and column-wise. 26
- 2.15 The PointNet architecture. Taken from R. Qi et al. [18] 27

- 3.1 Converting a raw point cloud (left) to a semantically labeled point cloud (right) using SSDG. The colors from the raw point cloud stem from z-value, while the colors on the semantic point cloud represent different object categories. 29
- 3.2 An example of using the segmentation camera included in Ignition Sensors, with the label map on the right hand side. 31
- 3.3 An overview of the simulation setup. 32
- 3.4 An overview of a refinery gazebo world with the pertaining point cloud gazebo generates on the left. 33
- 3.5 Ouster OS0-128 Specifications. ² 35
- 3.6 An example showing the labelling node colorizing the point cloud. The industrial tank on the left is labelled in red, and the two other objects are labelled in pink. Note that the original lidar point cloud is also displayed with red points. 39
- 3.7 On Callback 40
- 3.8 The gazebo simulation corresponding to the point cloud in Figure 3.6 41
- 3.9 An example of instance segmentation, with each object assigned a separate instance and color. 42
- 3.10 An example of a synthetically generated world comprising multiple parametric shapes. The black objects represent the domain randomization objects while the white object is a simple model of an industrial tank. 44
- 3.11 From left to right: pallet, cone, electrical box and pallet box. 46
- 3.12 The same industrial tank, with different scaling factors showing the data augmentation technique 47
- 3.13 Comparison of our dataset folder structure with SemanticKITTI. 49
- 3.14 A sample of the tanks, rendered in Ignition gazebo 50
- 3.15 A sample of the stairs, rendered in Ignition gazebo 50
- 3.16 A sample of the chimneys, rendered in Ignition gazebo 51
- 3.17 The stairs from ModelNet40 that were used in the industrial dataset. 51
- 3.18 A staircase from the industrial dataset. 54
- 3.19 An industrial chimney from the industrial dataset. 55
- 3.20 A tank from the industrial dataset. 55
- 3.21 Another tank from the industrial dataset. 56

3.22	Multi-object scan from the industrial dataset.	56
3.23	Reported runtimes of several SOTA models. Image taken from SalsaNext [28].	57
4.1	Comparison of mIoU, pixelwise accuracy and cross entropy loss for SSVG3-21 and SSVG3-53.	62
4.2	Comparison of mIoU, pixelwise accuracy and cross entropy loss for Salsanext with image sizes of 1024x128 and 512x128.	63
4.3	Comparison of mIoU, pixelwise accuracy and cross entropy loss for SalsaNext and SSVG3-53.	63
4.4	mIoU, pixelwise accuracy and cross entropy loss for final model. . .	64
4.5	Class-wise Intersection over Union (IoU).	65
4.6	A small cone found in the ground in front of the tank (in grey) is predicted incorrectly.	66
4.7	An object created by the domain randomization scheme occludes the tank (in grey) and is predicted incorrectly.	66
4.8	66
4.9	A correct prediction mask with occluded stairs (red).	66
4.10	A correct prediction mask containing industrial chimneys (blue). . .	66
4.11	66
4.12	A prediction on a tank (gray) with random objects surrounding. . .	67
4.13	A prediction on a staircase with random objects surrounding. . . .	67
4.14	67
4.15	Segmented point cloud from sequence 240 of the validation set . .	67
4.16	Segmented point cloud from sequence 241 of the validation set . .	67

Tables

3.1	Specifications for Collision Avoidance LiDAR	34
3.2	Specifications for the simulated Segmentation LiDAR, modelled from the Ouster OS0-128.	35
3.3	Camera Info	36
3.4	Specifications for Segmentation Cameras.	36
3.5	Topics shared between ROS Ignition Transport.	37
3.6	Initialization	40
3.7	Overview of point cloud datasets for semantic segmentation.	48
3.8	Overview of semantic classes in the industrial dataset. learning_map label is simply the one-hot encoding used by the SemanticKITTI dataset format	48
3.9	Frequency of the four different semantic labels in the industrial dataset	53
3.10	An overview of the sequences and their object categories, created for the industrial dataset.	54
4.1	Summary of hyperparameters used for training.	61
4.2	Comparison of Final IoU between SqueezeSegV3-53 and SalsaNext.	64
4.3	Reported inference times of SalsaNext-ROS on a CPU.	68

Acronyms

BEV	Bird-Eye-View. 24, 25
CAM	Context Aggregation Module. 23
CDF	Cognite Data Fusion. iii
CNN	Convolutional Neural Network. ix, 7, 9, 10, 21, 22
CRF	Conditional Random Field. 20, 22, 23, 57, 58, 65
CUDA	Compute Unified Device Architecture. 18, 19, 68
cuDNN	CUDA Deep Neural Network library. 19
FC	Fully-Connected Layer. 11
FOV	field-of-view. ix, 5, 6
FPS	Farthest Point Sampling. 28
GPU	Graphics Processing Unit. 18
IoU	Intersection over Union. xi, 14, 17, 53, 63–65, 73
kNN	k-nearest neighbors. 23, 28, 57–59, 65
LeakyReLU	Leaky Rectified Linear Unit. 13
LiDAR	Light Detection and Ranging. iii, vii, xiii, 1, 2, 4–7, 21, 23, 24, 30–35, 37, 42, 43, 57, 58, 62, 66, 70–73
LocSE	Local Spatial Encoding. 28
mIoU	mean Intersection over Union. iii, 17, 57, 62, 64, 73

MLP	Multilayer Perceptron. 8, 27, 28
ReLU	Rectified Linear Unit. 12
RNN	Recurrent Neural Network. 22
ROS	Robotic Operating System. 31, 32
RS	Random Sampling. 28
SAC	Spatially-Adaptive Convolution. 24
SDF	SDFFormat. 32
SFV	Spherical-Front-View. 24, 25
SGD	Stochastic Gradient Descent. 11, 23
SLAM	Simultaneous Localization and Mapping. iii, 1–3, 23, 59, 61, 63, 68, 69, 75
SOTA	state-of-the-art. iii, xi, 2, 7, 48, 57, 75
SSDG	Synthetic Semantic Dataset Generator. iii, ix, x, 3, 4, 19, 29, 30, 43, 69–73, 75
TSDF	Truncated Signed Distance Field. 57

Chapter 1

Introduction

1.1 Background and Motivation

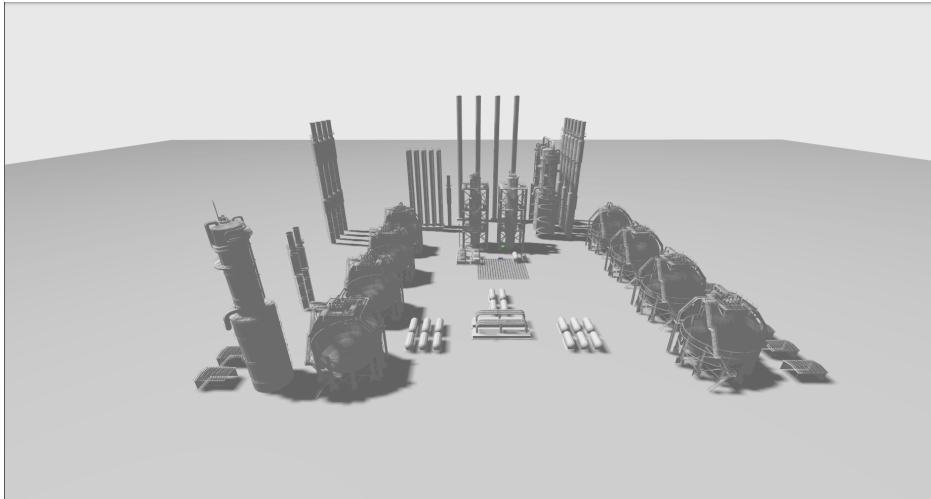


Figure 1.1: An Oil refinery model.

The emerging market opportunities in robotics have enabled the technological advances within 3D sensing to greatly improve in the last couple of years. Light Detection and Ranging (LiDAR) and RGB-D cameras have become increasingly available with more affordable prices, and a wide range of competitors allows for better and cheaper sensor units than before. Within LiDARs we find Ouster, Velodyne and Livox amongst others all competing on resolution, range, size and price. This has allowed 3D vision to flourish within topics such as Simultaneous Localization and Mapping (SLAM), 3D object reconstruction, and many more. The large point clouds generated by these 3D sensors can be interpreted further than their location in a 3-dimensional space, in order to understand the environment at a higher level. Semantic segmentation is the act of giving each point in the cloud a semantic label, giving the point cloud a new dimension. As this thesis is written

in collaboration with Cognite, we focus mainly on semantic segmentation in the industrial domain.

The motivation for providing 3D semantic segmentation in industrial areas is two-fold. Firstly, point cloud segmentation in the industrial sector can facilitate the creation of drawings and digital twins, an otherwise arduous and labor intensive process. By assigning semantic labels such as "pipe" or "valve" to each point in the cloud, digital twinning can be automated and manual labor cost drastically reduced.

Second to this, an environmental understanding and perception is key to realizing robotic autonomy without human intervention. A robot navigating and inspecting industrial zones needs to be able to reason on its environment to a certain extent for fully autonomous operations. For example, if the robot can semantically understand staircases, it knows it will be moving in an upwards or downwards trajectory. This understanding can be crucial for enabling robust localization and map-creation as it can be included as a valuable prior in the optimization step. The segmentation of stairs is also beneficial for motion-planning over complex environments, as it will allow the robot to plan a trajectory over multiple levels in a building. LiDAR-based solutions have proven to perform well in most environments, and are not affected by the same external influences that typically disturb camera sensors, such as obscurants, motion blur, and illumination changes. However, as most LiDAR-only SLAM algorithms rely on some form of scan-to-scan matching, their robustness to dynamical objects such as humans is often lacking. It must be able to filter out such data associations, and can be achieved by using a semantic understanding of the environment. Additionally, place recognition and loop closures can become more robust to seasonal changes by including semantics. For instance, snow or leaves falling from a tree will lead to entirely mismatched point clouds of the same spot, but by including the semantics of the objects in the place recognition framework, rather than the raw point cloud itself, the descriptiveness of the scene is substantially improved. We therefore establish that a semantic scene understanding can be used to give the robot an understanding beyond the spatial domain, in order to increase the robustness of the SLAM pipeline. The reasons for using semantics in a perception pipeline are clear, but semantics-based 3D autonomy is still a relatively new field. SemanticKITTI [1] has paved the way for semantics-based autonomous driving in urban scenarios by providing an open-source dataset containing millions of manually-labelled points consisting of categories such as car, road, pedestrian etc. This has paved the way for some of the state-of-the-art semantic SLAM algorithms such as SuMA [2], SA-LOAM [3] and SegMap [4] incorporating varying amounts of segmentation in their frameworks. However, no such open-source dataset exists for industrial zones, and few or no semantics-based algorithms tailored towards industrial autonomy is developed to date. The next step towards allowing robotic autonomy within industrial settings is creating high-quality large scale datasets for training and evaluation. As such, labelled industrial data creation becomes the focus of this thesis.

1.2 Aim and Scope of Thesis

1.2.1 Scope

This thesis is written as a collaboration between Cognite and the Autonomous Robots Lab¹, who aim to create resilient autonomous robotic systems capable of long-term operations in complex, and degraded environments. To give these robotic systems a higher level understanding of their environments in the form of semantic labels, real-time segmentation networks can be trained on domain-specific datasets. This thesis will focus on generating the semantic segmentation data for training these segmentation networks. The original intent was creating a synthetic semantic dataset for use within the industrial domain. However, as no sufficient method for the creation of synthetic semantically-labelled datasets exists to date, we instead expand our scope to creating a *general* framework for generating semantic datasets, which can be used in a multitude of domains and tasks. By focusing on the general framework rather than keeping it specific to our industrial domain, and open-sourcing our repository, we are able to further develop the field and contribute to the community.

1.2.2 Aim

The main objective of this work is to create a method for generating semantic point clouds, and additionally assess the capabilities of semantic segmentation networks on a dataset generated by the method, to verify that they can be used in real-world settings. We aim to create a general framework, named Synthetic Semantic Dataset Generator (SSDG), and demonstrate its usefulness by producing an industrial dataset. We further assess the semantic quality and use of the dataset by training and validating a set of segmentation networks. The general aim of this work can be summarized in the following objectives:

- *Objective 1:* Design a scalable method for generating synthetically labelled point cloud datasets where the label creation process is automated, Synthetic Semantic Dataset Generator (SSDG).
- *Objective 2:* Design a novel domain randomization scheme to reduce the disparity between real world and computer simulated data for 3D point cloud segmentation.
- *Objective 3:* Construct industrial 3D CAD objects and use them for generating a high-quality industrial-based dataset from the method described in objective 1.
- *Objective 4:* Investigate and train different segmentation networks on the proposed dataset to validate its use.
- *Objective 5:* Design a ROS Node for direct usage of the projection-based networks in a SLAM pipeline.

¹<https://www.autonomousrobotslab.com/>

1.2.3 Contributions

This thesis makes contributions to synthetic semantic segmentation dataset creation, as well as 3D semantics-based domain randomization. An industrial dataset is generated using the proposed method, and evaluated using two open-sourced segmentation networks.

The repository containing the source code of the Synthetic Semantic Dataset Generator (SSDG) implementation is made publicly available, and can be found under the following URL: https://github.com/ntnu-arl/semantic_simulator

1.3 Thesis Outline

Chapter 1 *Introduction*

Chapter 2 *Background* - Provides the theoretical background for the material presented, and related works. The chapter includes theory within the fields of LiDAR sensing, deep learning and semantic segmentation. Particular emphasis is placed on projection-based segmentation and related work.

Chapter 3 *Method and Implementation* - Presents the Synthetic Semantic Dataset Generator (SSDG) method, followed by an industrial dataset generated using SSDG. Implementation details for semantic segmentation of the industrial dataset with two segmentation networks is presented at the end of the chapter.

Chapter 4 *Results* - Presents the results obtained from the segmentation experiments on the industrial dataset.

Chapter 5 *Discussion and Further Work* - Summarizes and discusses the results observed, and provides suggestions for further works and potential extensions to the proposed synthetic dataset generator.

Chapter 6 *Conclusion*

Chapter 2

Background and Related Work

In this chapter, we present some background theory and related work. The chapter begins with a brief presentation of the LiDAR sensor and some of its data representations. This is followed by an overview of deep learning, before we move into deep learning-based segmentation networks. The chapter attempts to cover the key background theory in topics of deep learning and segmentation, but since these are hugely spanning fields we focus mainly around topics surrounding deep learning-based 3D point cloud segmentation.

2.1 Light Detection and Ranging (LiDAR)

In this section we present some of the core theoretical knowledge behind the LiDAR sensor.

2.1.1 The LiDAR sensor

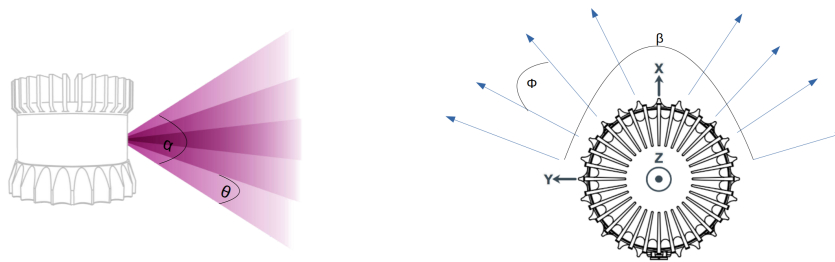


Figure 2.1: An example of the Ouster OS0 LiDAR with sample configurable parameters. α is vertical FOV, θ is vertical resolution, β is horizontal FOV and ϕ is horizontal resolution. ¹

The Light Detection and Ranging (LiDAR) sensor has been the subject of much research in the past decade, particularly due to their robustness to illumination

¹Images modified from <https://ouster.com/products/scanning-lidar/os0-sensor/>

changes and obscurants. A LiDAR works by emitting pulsed light waves into its surroundings and uses the time for each pulse to return to calculate the distance each pulse travels. By repeating this thousands of times per second, a LiDAR is able to create a precise, real-time 3D map of the environment. There are a number of configuration parameters that determine the size and accuracy of the map, including horizontal and vertical field-of-view (FOV), see Figure 2.1. Horizontal and vertical resolution is also configurable to some extent. A LiDAR is able to provide a 360° field-of-view by mechanically spinning a mirror around, and most LiDARs today produce a full horizontal FOV using this scanning effect. The configurations of LiDAR systems differ in suppliers and models, and for this thesis we base our work on the Ouster OS0-128 LiDAR.²

2.1.2 Point Cloud

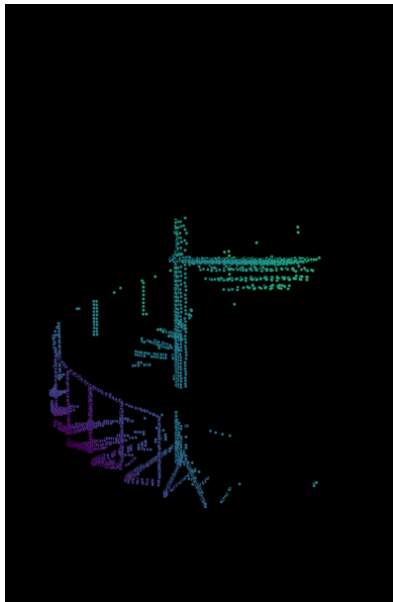


Figure 2.2: Point cloud of a staircase, generated using method in Section 3.1. 3D CAD model from the ModelNet40 [5] dataset.

The point cloud format is the common data representation for LiDAR data and is often used for localization and mapping. A point cloud is a set of points formed by the hit positions of the laser beams on a physical object. Every point in a point cloud consists of coordinates x , y , and z with additional information such as reflective intensity sometimes included.

²<https://ouster.com/products/scanning-lidar/os0-sensor/>

2.1.3 LiDAR Image



Figure 2.3: 2D Range image of an industrial scene. Colors stem from a semantic label, but can also be given by intensity or RGB channels from a camera.

Another LiDAR data representation is the LiDAR image, which can be constructed from the elevation and azimuth measurements generated by a LiDAR. It can be seen as a parametrized image representation of LiDAR data. Each vertical beam from the sensor will have a row, and the consecutive measurements from each beam comprise that particular row. As such, we can construct a 1024x128 image from an Ouster OS0-128 which has 128 vertical rays. A LiDAR image can be constructed from both intensity channels and range. 2D LiDAR images are a much more compact representation of the 3D geometrical world, and as such more efficient operations may be performed on it. Particularly for large-scale scenes, the 2D representations are much more computationally efficient than working directly on the 3D point cloud.

2.2 Deep Learning

In this chapter, deep learning theory will be presented to provide theoretical depth behind the neural networks used for semantic segmentation. Recently, deep learning networks and Convolutional Neural Networks have become more common in the semantic segmentation literature, and most state-of-the-art algorithms today use some form of deep learning. According to [6], their success in the field stems partly from their strong ability to yield hierarchies of features.

2.2.1 Neural Networks

Neural networks and deep learning is based around the ability to use large-scale datasets to extract patterns and structures to model nonlinear processes. Neural networks can be used for pattern recognition, 3D reconstruction, medical diagnosis and classification and more. The name and intuition behind them are inspired from biological neurons that we find in the human brain. A brain neuron can be excited from other cells by a electrical signal, or synaptic potential. Similarly, an artificial neuron will sum a set of weighted inputs x_i , representing synaptic potential, and pass it to an activation function f which will output a final value y .

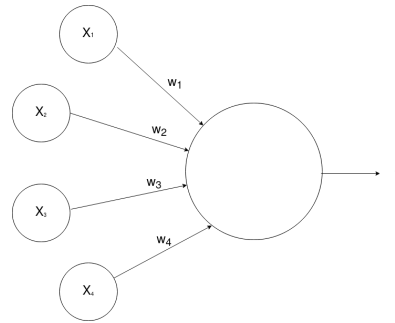


Figure 2.4: The perceptron model, here consisting of 4 inputs, 4 weights and a single neuron.

Figure 2.4 presents a fundamental unit of neural networks, the perceptron. It consists of a single neuron receiving a set of inputs from which it outputs a final value y . It adds a weight to each input value, and uses the weighted sum of all the inputs and a potential bias b which is passed to an activation function. During learning, the weight of both the input and bias will change. This is what allows the neuron to learn what to distinguish between, and connections between neurons to learn shapes and patterns. The output is calculated by taking the weighted sum of the n inputs and passing it through an activation function f , Equation (2.1). The perceptron in Figure 2.4 consists of 4 inputs, meaning it will have 4 trainable weights and 1 trainable bias, giving a total of 5 trainable values.

$$z = \sum_{i=1}^n x_i w_i + b \quad , \quad y = f(z) \quad (2.1)$$

By combining neurons in a layer-wise fashion, we can create deeper neural networks that are able to learn and generalize for more complex predictions. The Multilayer Perceptron (MLP) Figure 2.5 (also known as a *feed forward network*) consists of an input layer, one or more hidden layers and an output layer. Except for the input layer and output layer, these layers are built up by connections of neurons that input and output values from and to each other in the different layers. Each neuron will have its own set of trainable weights and biases allowing the network to "learn" by using a method called backpropagation, see Section 2.2.3.

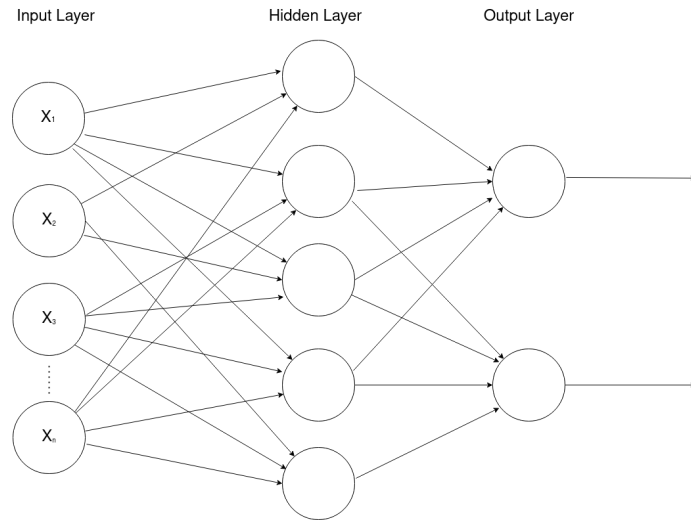


Figure 2.5: The Multi-layer perceptron, with an input layer, hidden layer, and output layer.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks are special type of neural network that employs convolution kernels or filters along with pooling and more traditional layers to extract features. They are designed for multi-dimensional data and their location invariance make them a popular choice for computer vision tasks [7]. They typically consist of three types of layers, the input layer, one or more feature-extraction layers and a classification layer, Figure 2.6.

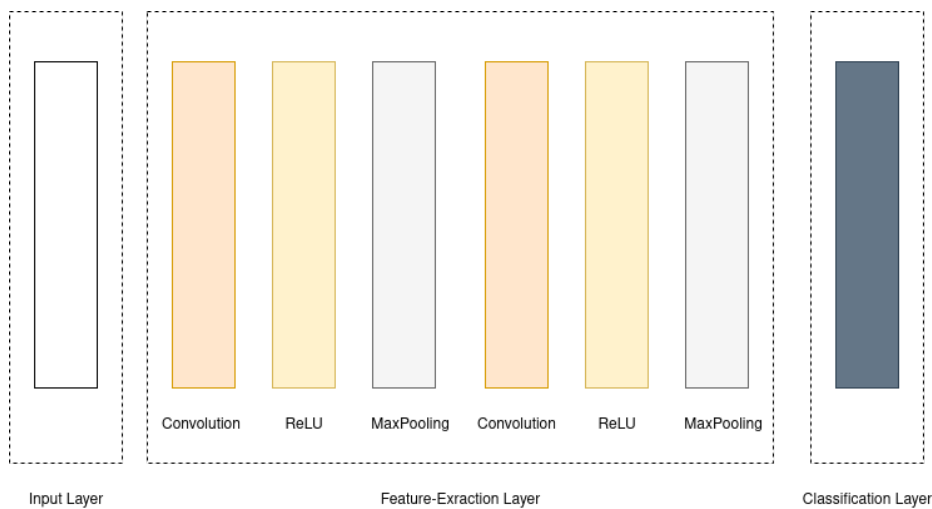


Figure 2.6: An example of a CNN architecture, with an input layer, feature-extraction layer and classification layer. The feature-extraction layer in this particular example uses ReLU as activation and MaxPooling for downsampling.

Convolution Layer

The convolutional layer consists of multiple convolution filters, *kernels* that each are iterated with a given *stride* across the width and height of the input. The filters have arbitrary sizes $M \times N$ but are typically square size in neural networks. At each position along the input data, the mathematical convolution is calculated. This is not a traditional matrix multiplication, but an expression for how the shape of a function is modified by another.

Each element of the kernel represent a separate learnable weight that is updated during training. The output of the convolution between the kernel and the input in the iterated location is an activation map which gives the response of the filter at that particular location. By backpropagating and updating the filter weights, the network will learn filters that activate when they see a particular visual feature or cue. By training on a large dataset these filters will correspond to features that activate when the feature maps correspond to the desired output.

Each convolutional layer will contain an entire set of such filters that each produce a separate activation map. By combining these the neural network constructs meaningful features that can be extracted to for example classify an image.

Pooling Layer

Most feature-extraction layers will at some level include a downsampling operation. This helps reduce the number of parameters, and prevents overfitting. The most common method to achieve this is using pooling layers. A pooling operation will downsample the input which causes information loss, but simultaneously allows for the network to extract features of higher levels. MaxPooling calculates the largest value in a patch of each feature map and dumps the other values. A 2x2 MaxPooling layer would reduce an equally sized 2x2 matrix as in Equation (2.2) to a single digit, and as with convolution kernels they can slide across an input matrix to downsample the whole matrix. The MaxPooling outputs the highest value in the input window the kernel is placed in, which in Equation (2.2) would correspond to the value 4.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow 4 \quad (2.2)$$

Equation 2.2: An example of the output from a 2x2 MaxPooling layer on a 2x2 matrix.

Fully-Connected Layer

The final classification layer is responsible for converting the features from the hidden layers into predictions that result in the output of the CNN. For classification and segmentation tasks, the output should contain the class scores and should

be a one-dimensional vector. To convert from a multi-dimensional input (such as an image), Fully-Connected Layers FC are used. Each neuron in a fully-connected layer is connected to all the neurons in the previous layer to flatten the output. Each neuron multiplies the input by a set of learnable weights and adds a bias. In most cases, the FC will be used in the last layer to convert the output from previous convolutional layers to a meaningful output vector, but they can be employed throughout the network.

2.2.3 Training Neural Networks

Training neural networks requires large datasets that contain training data with the correct output for each given input. By calculating the difference between the desired output and given output as given by a loss function (Section 2.2.5) for an iteration of the training, a learning algorithm can decide how to update the weights and biases of the hidden layers in the network. The optimization problem of minimizing the error function (loss function) is solved by the *backpropagation* algorithm. The backpropagation algorithm calculates the gradient of the error function with respect to the weights of the hidden layer and is a direct application of the chain rule for derivatives [7]. The calculation of the gradients is applied repeatedly starting from the output at the top all the way to the bottom where the input is fed. The gradients with respect to the weights can then be calculated for each module, and the weights be updated accordingly. The gradient vector for each weight indicates how the error would change if the weight was increased by a small amount. The backpropagation algorithm finally adjusts the weight vector in the negative direction of the gradient vector.

Stochastic Gradient Descent (SGD)

In practice, the de facto standard is not to use traditional backpropagation on a set (or *batch*) of the training data. Instead, a much faster method known as Stochastic Gradient Descent (SGD) is used. Whereas batch learning uses an entire batch to compute a gradient, SGD uses stochastic (online) learning on a *single* randomly chosen example at each iteration. The true gradient is then *estimated* based on the error of that example. The estimation may lead to noise in the model, but according to [7] this can be advantageous as it may lead to the optimization to jump over local minima basins. For more information on SGD, see [7].

2.2.4 Improving neural networks

Activation Functions

Each neuron in a network contains an activation function, and they are fundamental part of a neural network that help define the output of a node. Also known as transfer functions, their function and effect on learning have been researched in

depth and is still the subject of research. In biological neurons the activation functions represents the abstraction that makes the neuron fire or not. The activation function within a neural network is what enables the nonlinearity between the input and output values to the network.

The most simple form of an activation function is the identity function. This is known as a Linear activation function. Here the activation is simply proportional to the input, and no nonlinearity is introduced. Due to this, backpropagation will not yield results as the derivative is a constant with no relation to the input x . In other words, it reduces the network into a linear regression model. By introducing a nonlinear activation function such as the Sigmoid function, which provides a smooth gradient, the stacking of multiple layers of neurons will provide a nonlinear combination and create a complex mapping between the input and output from the network.

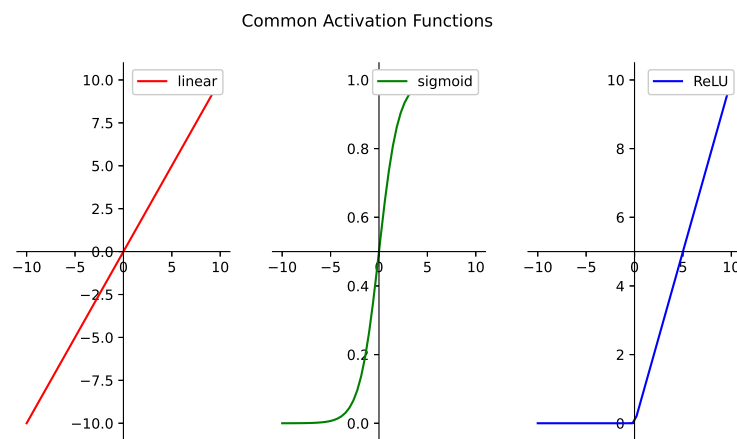


Figure 2.7: Three Activation functions: Linear, Sigmoid and ReLU

As we see in Figure 2.7 the output of the sigmoid function will approach either zero or one in the negative and positive directions respectively. This will yield a gradient of zero if you traverse far enough in one direction and as such the network will cease to learn if the input value is not in the correct range. This is what is called the *Vanishing gradient problem*.

LeakyReLU

A computationally cheaper and faster activation function is the Rectified Linear Unit (ReLU) [8]. Even though it may at first seem like a linear function in Figure 2.7, it is not. The function returns 0 if the input is negative and acts like a linear transformation if positive. It avoids and rectifies the vanishing gradient problem, and since it only acts upon positive values only a few neurons will be activated at a time. One issue caused by the rectification is that fragile gradients may die

resulting in dead neurons. To combat this, a modification of the ReLU aptly called Leaky Rectified Linear Unit (LeakyReLU) is introduced in [9]. Figure 2.8 illustrates the modification, where a small slope is introduced to keep the updates alive.

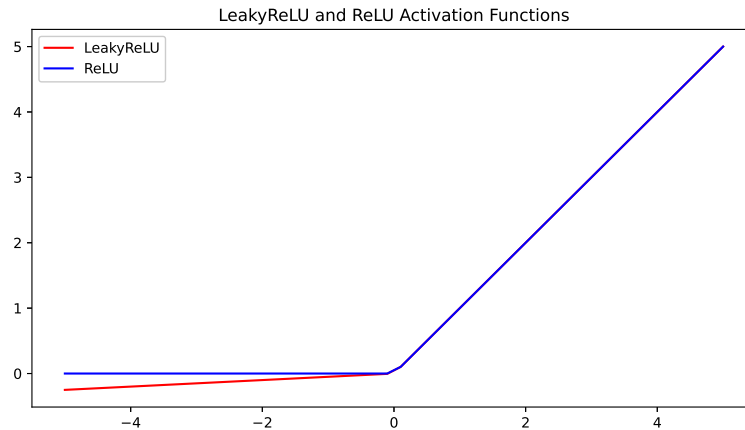


Figure 2.8: The ReLU function and its counterpart LeakyReLU [9].

Softmax

Often times the final layer of a neural network is expected to produce a probability distribution over the set of C classes, and as such a set of input numbers need to be converted to a set of probabilities. The *Softmax* function calculates the relative probabilities based on the normalized exponentials of the input numbers to determine a final output probability value.

$$\text{Softmax}(z_j) = \frac{e^{z_j}}{\sum_{c=1}^C e^{z_c}} = \hat{y}_j \quad (2.3)$$

Where z_j is the output from the j^{th} neuron. After softmax has been applied, each component will be in the interval $(0,1)$.

2.2.5 Loss functions

The loss functions are a way of describing how close or far away a network is from its ideal state. An ideal network will give the correct prediction given an input and ground truth set, and the loss functions explain how far away from the correct response the neural network is.

Softmax Cross-Entropy Loss

A common loss function is the Softmax Cross-entropy loss. This cross-entropy loss, also known as log loss, measures the performance of a classification model where

the output is a probability $p \in [0, 1]$. A perfect model will have a low log loss, while a bad prediction will result in a high loss value. The cross-entropy is defined as

$$\mathcal{L}(y, \hat{y}) = - \sum_{i=1}^n p(y_i) \log(p(\hat{y}_i)) \quad \text{for } n \text{ classes} \quad (2.4)$$

Where y_i is the ground truth label and \hat{y}_i the predicted label of the i^{th} class. For binary classification, the entropy would be defined as

$$\mathcal{L}(y, \hat{y}) = - \sum_{i=1}^n p(y_i) \log(p(\hat{y}_i)) \quad (2.5)$$

$$= -[y \log(p) + (1 - y) \log(1 - p)] \quad (2.6)$$

where the ground truth label y will be either 0 or 1, and p_i is the softmax probability for the class. Note that for binary classification the probability of the other class is simply $1 - p$.

Weighted Cross-Entropy Loss

Most datasets will have an imbalance between different classes. For instance, a large industrial zone will contain a lot more pipes than industrial tanks. This will make the network more biased towards the pipes and as such yield significantly poorer network performance. To address the imbalance in the training sets between classes, a weighted cross-entropy loss can be employed. Each class is given a weight based on how often the class appears in the set. The weighted cross-entropy with the inverse square root of the class frequency is defined as

$$\mathcal{L}(y, \hat{y}) = - \sum_{i=1}^n \alpha_i p(y_i) \log(p(\hat{y}_i)) \quad \text{for } n \text{ classes} \quad (2.7)$$

$$\alpha_i = 1/\sqrt{f_i} \quad (2.8)$$

Where y_i and \hat{y}_i are the true and predicted labels respectively, and f_i is the frequency of class. This loss function will reinforce the networks response for the less frequent appearing classes and improves performance on imbalanced datasets.

Lovász-Softmax Loss

In evaluation of image segmentation, the Intersection over Union (IoU) (See Section 2.2.8) is a commonly employed metric. The IoU metric is a discrete, non-derivable metric and as such cannot be directly employed as a loss. The cross-entropy loss on a validation set is often a poor indicator of the segmentation quality, and the Jaccard index (IoU) would be a better performance indicator for training the model. *Berman et al.* [10] presents the *Lovász-Softmax*, adopting the IoU metric using the *Lovász* extension of submodular losses. Based on submodular analysis of set functions, the Jaccard index is extended to provide a smooth

extension of the otherwise discrete loss. The *Lovász-Softmax* loss (\mathcal{L}_{ls}) can be formulated as

$$\mathcal{L}_{ls} = \frac{1}{|C|} \sum_{c \in C} \overline{\Delta_{J_c}}(\mathbf{m}(c)) \quad \text{and} \quad (2.9)$$

$$m_i(c) = \begin{cases} 1 - x_i(c), & \text{if } c = y_i(c) \\ x_i(c), & \text{otherwise} \end{cases} \quad (2.10)$$

where $|C|$ is the class number, $\overline{\Delta_{J_c}}$ defines the Lovász extension of the Jaccard index, and $x_i(c)$ and $y_i(c)$ respectively hold the class probability and ground truth label of pixel i for class c . $\mathbf{m}(c)$ represents the vector of errors constructed from each pixel i and class $c \in C$ in Equation (2.10). For further information, see [10].

2.2.6 Regularization

Regularization is an action of introducing additional information to prevent *Overfitting*, which is when a model fails to generalize the features or patterns in the training data. Often times, deep neural networks will have million of parameters and will try to learn any features or patterns it can find in the training set and can therefore be prone to overfitting, if for example the training set is too small. Overfitting is said to be caused by noise in the training being learned by the network as a feature or underlying concept. As such, when the model sees new data the performance will drop significantly. Regularization techniques aim to prevent this and improve the generalization ability of the network. Poor performance from machine learning models is often attributed to overfitting or underfitting. *Underfitting* refers to when a model is not able to properly model the data with poor performance both on the training set as well as the validation set. Typically this is a sign that the model is not a fit for the data, and new models should be explored. This section will cover some of the techniques used to deal with overfitting.

Data augmentation

Data augmentation is a method of enlarging a dataset by generating new data from the existing data by adding changes or perturbations. The most simple techniques of data augmentation for computer vision includes translation, whitening, scaling, flipping and rotation. It has a limited effect since completely new data is never introduced, but it is an easy and simple technique for enlarging a dataset without creating entirely new data.

L1 and L2 regularization

Another method of regularization is using Lasso Regression (L1) and Ridge Regression (L2). Both introduce a penalty on weights in the loss function, in order to reduce the absolute sum of the weights. By adding Equation (2.11) or Equation (2.12) to a loss function, the absolute value of the weights are penalized and

reduces the likelihood that noise in the training data will be learned by the model. By decreasing the number of parameters in the model, the L1 and L2 regression simplify the model and is an often used method for regularization.

$$L_1 = \lambda \sum_{i=1}^N |w_i| \quad (2.11)$$

In addition to the Lasso Regression (L1), we have Ridge regression, which squares the L2 norm of the weights and uses this as a penalty. The L2 term is defined as:

$$L_2 = \lambda \sum_{i=1}^N w_i^2 \quad (2.12)$$

Dropout

The key idea behind dropout is to randomly drop units and connections from a neural network during training. The technique was introduced in [11] as a simple way to prevent overfitting. By randomly dropping units in the hidden layers at different points in time, unique thinned versions of the network are being trained that are only slightly different versions of the others. The resulting final model will pick up only the key properties from each of the thinned networks created during training. The technique is an efficient method of reducing the overfitting of networks and has been prove to increase performance of neural networks on a number of tasks [11].

2.2.7 Normalization

Normalization is another technique against overffiting, that aims to bring the dataset to a common scale without distorting its shape. Normalizing the data ensures that the model can generalize appropriately. An independent and identically distributed dataset will allow the neural network to converge faster and stabilize the learning process.

Batch normalization works in two steps. The input is first normalized to ensure the data have the same mean and variance and thus fits the same distribution. The data is transformed to have a mean of zero and a standard deviation of one. During this normalization step, a smoothing term ϵ is added to ensure numerical stability. The final operation is re-scaling and offsetting of the input, by using two learnable parameters re-scaling (γ) and shifting (β) as in Equation (2.13). The normalized output of the batch normalization y_i is then passed onto subsequently layers of the network, while the normalized output \hat{x}_i remains internal to the current layer.

$$y_i = \gamma \hat{x}_i + \beta \quad (2.13)$$

2.2.8 Metrics

To report and compare classification between different methods and datasets, a number of assessment criteria have been developed. While loss functions provide a valuable tool for network training, evaluation metrics must be constructed to be able to validate the results.

IoU

The Intersection over Union (IoU) (also known as the Jaccard index) is the most commonly used metric to for evaluation of segmentation performance. The Intersection over Union metric describes the extent of which two bounding boxes overlap. In the world of segmentation, IoU determines how well an image is segmented by comparing the pixels in the target and prediction masks. A high IoU value is synonymous with a good overlap between target and prediction, while a low IoU value represents a low percentage of overlap.

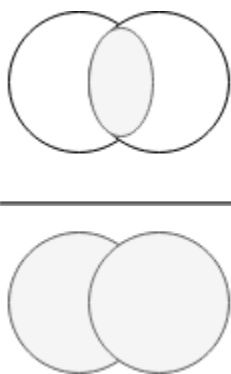
$$\text{IoU} = \frac{\text{Area of overlap}}{\text{Area of union}} = \frac{\text{Diagram 1}}{\text{Diagram 2}}$$


Figure 2.9: The Intersection over Union.

The IoU score is calculated for each class separately and averaged over all classes to provide a mean Intersection over Union (mIoU) of the semantic segmentation prediction.

Accuracy

As an alternative metric to IoU, the pixel accuracy is sometimes reported. To evaluate the semantic segmentation, the percentage of pixels correctly classified are reported for each class separately and combined to provide a global accuracy metric. The metric can sometimes be misleading if a class representation is small within an image, as this will result in the metric being biased mainly towards identifying negative cases.

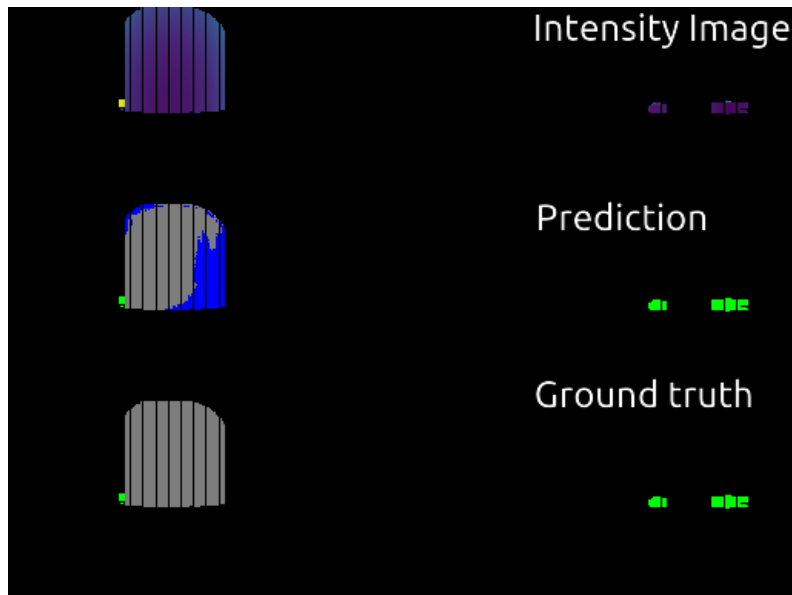


Figure 2.10: From top to bottom: input image, prediction, and ground truth. Input color is based on intensity, and grey represents tanks, blue represents stairs, red represents chimneys while green represents other objects. The neural networks predicts large parts of the tank wrong, but since most of the image is black the reported accuracy will still be high. This is the reason accuracy is not always a good evaluation metric.

Figure 2.10 illustrates the issue with the accuracy metric. The network predicts the objects in the scene wrong, but since most of the image is black, the reported accuracy will be upwards of 95%.

2.2.9 Deep Learning Libraries

Implementation of deep neural networks are centered around two main frameworks, PyTorch [12] and TensorFlow[13]. TensorFlow was released by Google Brain in 2015 and designed to replace Theano [14]. It is often times used in combination with Keras [15]. PyTorch is based on the Torch library, and is primarily developed by Facebook’s AI Research Lab. Both the frameworks are open-source and present an end-to-end platform for creating and training machine learning models.

In recent years, the usage of Graphics Processing Unit (GPU) originally intended for display devices have been used to accelerate deep learning networks. Their highly parallel structure make them perfect for DL implementations as it allows them to speed up computations that involve matrix operations. In addition, a number of accelerated libraries have been implemented by manufacturers to directly access the parallelism of GPUs. Compute Unified Device Architecture (CUDA) is a platform developed by NVIDIA [16] for general computing on GPUs. It allows the user to create high performance GPU-accelerated applications which

can be used for linear algebra and tensors, image and video processing and more. On top of CUDA, NVIDIA have developed CUDA Deep Neural Network library (cuDNN) [17] specifically designed for training deep neural networks. It provides low-level implementations for convolutions, pooling, normalization and activations with high efficiency. Both CUDA and cuDNN are both free to use and frequently used by DL practitioners.

2.3 Semantic Segmentation

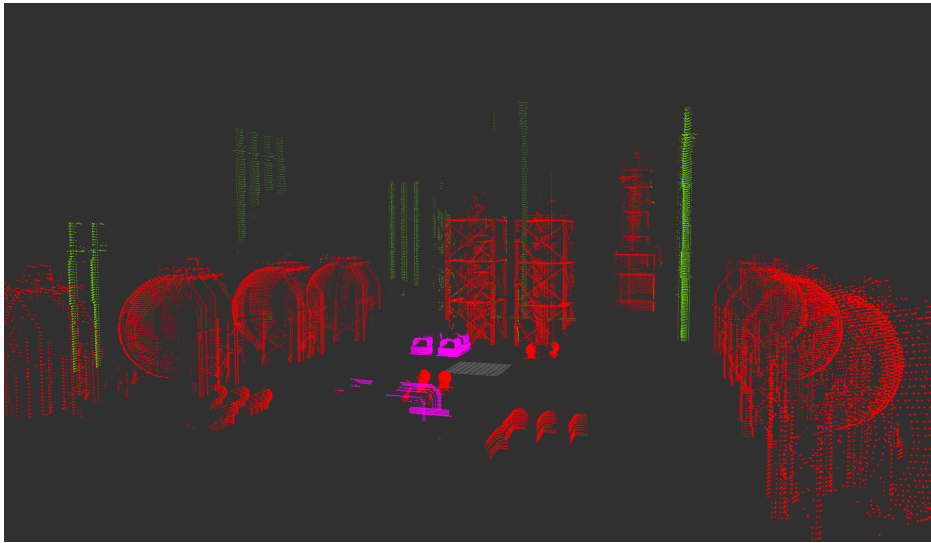


Figure 2.11: A semantically labelled point cloud of a refinery, where each color represents a category. Created using SSDG.

In computer vision, 2D segmentation is a well-known problem that has been studied extensively and refers to the process in which a pixel is assigned a semantic label (i.e. car, person, road). For a long time, advances within semantic segmentation were exclusively focused on images, and impressive results have been shown over a wide range of segmentation tasks including satellite imaging, medical imaging and robotic perception using traditional machine learning algorithms. Recent advances in the field of deep learning have enabled the image segmentation tasks to evolve and progress even further.

While deep learning has been used successfully in the past to solve 2D Vision problems, it is still relatively new in the 3D domain due to its processing challenges. Given an input point cloud, the goal of 3D semantic segmentation is to give each separate point in the cloud a label. Due to point clouds irregular format however, current 2D convolutional deep learning and other popular image segmentation techniques can not be exploited directly with point clouds. Point clouds unstructured and unordered form means it is not trivial to perform stan-

standard convolutions on them, and neural networks need to provide transformation and permutation invariance in order to successfully segment the 3-dimensional input data. Two main directions have been taken in a push to enable accurate 3D semantic segmentation. Point-wise methods directly process the point cloud, often by using shared multi-layer perceptrons and providing transformation and permutation invariance by exploiting certain network layers and operations. Today they are largely considered inefficient for large-scale point clouds and as such only used for offline segmentation. Faster methods are found within the projection-based networks, which instead transform the 3D point cloud into various other formats such as voxel cells or 2D range images. 2D rendered image representations are denser and as such computationally cheaper, and is therefore the most selected approach among real-time segmentation techniques.

This section on semantic segmentation is divided as follows. We first note some important properties of point clouds and give a general problem formulation for the segmentation of a 3D point cloud. This is followed by a presentation of several of the most efficient, open-sourced and accurate works within projection-based and point-based networks. Many of the projection-based methods foundations stem from previous generation models. For this reason, the SqueezeSeg, RangeNet and SalsaNet family of neural networks will all be presented, as the novelties each bring are collectively used by the others. A subsection is also dedicated to a brief review on Conditional Random Fields. The pioneering work within point-wise networks, PointNet, will be presented, along with a more lightweight and efficient neural architecture named RandLa-Net.

Properties of point clouds

Motivated by Qi *et al.* [18], we note the following important properties of point clouds in \mathbb{R}^n . They form the basis of architectural choices for point-wise methods, but do not afflict the projection-based methods.

1. The point cloud is unordered, meaning the set of points come in no specific order.
2. The points are not isolated and form local meaningful subsets.
3. invariance under transformations. The learned representation of the point cloud should be invariant to certain transformations.

Problem Formulation

Using the properties of a point cloud as a general foundation we can define a problem formulation for the point cloud segmentation task as follows.

Consider the unordered set S containing all the points defining a point cloud, where each point P_i is a vector containing its coordinates (x, y, z) :

$$S = \{P_0, P_1, \dots, P_i\} \quad \text{where} \quad P_i = (x, y, z) \quad (2.14)$$

For the segmentation task, each point from the input cloud is taken as input and the output is the classification scores for each label x_k in k number of classes. The output will therefore be a matrix of size $n \times k$ for each of the points n and semantic label k . Note that vector P_i may be extended to include features such as point intensity, color and ambience.

$$\text{Output} = [x_0, x_1, \dots, x_n] \quad n \in k \quad (2.15)$$

2.3.1 Projection-based methods

Projection-based networks generally work by transforming a point cloud into a 2D grid, inferring labels, and reprojecting the segmented 2D grid back onto the 3D point cloud. This allows them to take advantage of recent advances in state-of-the-art image segmentation by using techniques such as 2D Convolutional Neural Networks (CNN). These methods can be categorized into two general methodologies: multi-view projection and spherical projection. Multi-view representations suffer from occlusions and sensitivity to viewpoint selection. As such, the following subsections will only introduce works within spherical representations for point cloud segmentation.

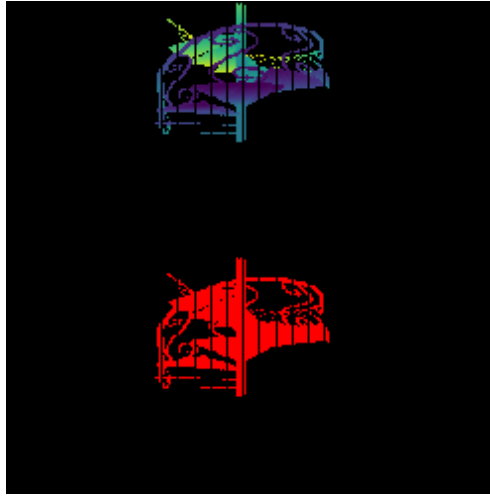


Figure 2.12: One of the stairs from ModelNet40 [5] projected onto a spherical projection. The image is a combination of two projections: the top image presents the intensity-based image, while the bottom presents the label map.

The spherical projection works by projecting the unstructured point cloud onto a spherical surface. This generates a range view image, and each raw LiDAR point (x, y, z) is mapped to an image coordinate (u, v) :

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[1 - \arctan(y, x)\pi^{-1}]w \\ [1 - (\arcsin(z, r^{-1} + f_{down}f^{-1})h] \end{pmatrix} \quad (2.16)$$

This represents the standard method of converting a point cloud for spherical-projection based methods.

SqueezeSeg

To achieve fast and accurate point cloud segmentation, Wu *et al.* introduced SqueezeSeg [19], an end-to-end approach based on a Convolutional Neural Network and Conditional Random Field (CRF). The proposed network is based on SqueezeNet [20], a lightweight CNN which achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. To achieve real-time inference speed and reduce computational complexity, the model is pruned by replacing convolution and deconvolutions with fireModules [20].

The authors behind SqueezeSeg noted that in line with image segmentation down-sampling operations, the label maps in spherical projections tended to be blurry using the SqueezeSeg network. To combat this, SqueezeSeg and SqueezeSegV2 uses CRFs to refine the label map generated by the CNN. The RNN formulation of the CRF is trained together with the CNN model in an end-to-end fashion as a single-stage pipeline, giving a fast and accurate model.

Conditional Random Field (CRF)

Down-sampling operations such as MaxPooling inevitably cause a loss of low-level details as described in Section 2.2.2. In image segmentation tasks, this causes segmentation masks to get blurry boundaries when using models that utilize down-sampling operations. To combat the blurry boundaries that tend to be predicted by CNN models, many segmentation models utilize Conditional Random Fields (CRF).

CRF arranges data points as nodes V and creates edges E between nodes to represent dependency between datapoints. For instance, two points in a point cloud with similar intensity and position measurements will likely belong to the same object and label. To ensure this, we can construct an energy function $E(\mathbf{c})$ that values a points likelihood of a class as well as its difference from neighbouring points as:

$$E(\mathbf{c}) = \sum_i \varphi(c_i) + \sum_i \phi(c_i, c_j) \quad (2.17)$$

Where c_i denotes the predicted label for point i . The unary potentials φ contains local information for a point i.e. the predicted probability from the classifier, and the pairwise potentials ϕ defines the "penalty" for assigning different labels to a point based on its neighbourhood. This enforces consistency within the neighbourhood of a point with the intention of removing blurry boundaries. By maximizing the energy function we ensure that both the labels with the highest probability are chosen and at the same time ensuring local consistency. The CRF energy function can not be minimized exactly, but a mean-field iteration algorithm proposed in [21] efficiently provides an adequate approximation. Furthermore, [22] reformulates the mean-field iteration as a Recurrent Neural Network (RNN).

SqueezeSegV2

To further improve the model structure of SqueezeSeg, combat *domain shift* and increase robustness against dropout noise, SqueezeSegV2 [23] was introduced. It presents an unsupervised domain-adaptation training pipeline to combat the domain shift observed on models trained with synthetic data tested on real-world data. The domain-adaptation training employs three significant strategies to deal with the domain shift problem: learned intensity rendering, a geodesic correlation alignment and a progressive domain calibration. The domain-adaptation pipeline nearly doubles the test accuracy on real-world data. Real-world LiDAR point cloud data will always contain some missing points, known as dropout noise. This can be caused by limited sensor range, reflections or jitter of the incident angle, and can corrupt the output of early convolution filters. SqueezeSegV2 introduces a Context Aggregation Module (CAM) to reduce the sensitivity of the network to dropout noise.

RangeNet

One of the most influential projection-based methods that achieves fast and accurate LiDAR segmentation is RangeNet [24]. It addresses several of the limitations of the SqueezeSeg (Section 2.3.1) framework and is the most frequently used segmentation network for real-time semantics-based SLAM. The following two novelties are presented:

- The projection is extended to include the full range of the LiDAR scan, in lieu of the frontal 90° approach used by SqueezeSeg.
- A novel kNN is presented to reduce discretization errors

While the CRF-based approach to infer semantics and reduce bleeding used by SqueezeSeg helps in 2D, it does not suffice for the re-projection to the 3D space. Two points stored in the same pixel from the range image will for instance be given the same semantic label. To solve this problem, *Milioto et al.* proposed a novel k-nearest neighbors (kNN) search operating on the point cloud. For each point in the semantic cloud, a point is labeled using a consensus vote based on the k closest points to it in 3D. A *cut-off* threshold sets the maximum allowed distance for a point to be considered a near neighbor using an absolute range distance in lieu of a Euclidean distance, due to computational efficiency. The novel kNN search improves the result by recovering important information around boundaries that is lost during the lossy discretization and back-projection of the laser scans. With the inclusion of the GPU-enabled kNN search, the name of RangeNet is extended to RangeNet++.

RangeNet is trained using SGD and weighted cross-entropy loss as described in Section 2.2. It presents a fast and accurate framework for semantic segmentation and can infer semantic segmentation using any CNN as backbone. Its most notable novelty is the introduction of a kNN search algorithm that is frequently put to use by later works.

SqueezeSegV3

SqueezeSegV3 [25] is the newest generation of squeeze networks, building upon the network architecture of RangeNet, introducing a novel Spatially-Adaptive Convolution (SAC). When LiDAR images are converted by spherical projection, the feature distribution varies drastically at different locations. Observing that the projection introduce strong spatial priors, Xu *et al.* proposes the Spatially-Adaptive Convolution (SAC) designed to be spatially-adaptive and content-aware.

Previous methods treat LiDAR images as RGB images, using the standard 2D image convolutions. The Spatially-Adaptive Convolution uses adaptive convolutions that change the weights according to the input as well as location in the image. The spatially-adaptive operator processes different parts of the image with different filters, and the filters adapt to feature variations. The Spatially-Adaptive Convolution (SAC) can be described as the following:

$$Y[m,p,q] = \sigma\left(\sum_{i,j,n} \mathbf{W}(\mathbf{X}_0)[m,n,p,q,i,j] \times \mathbf{X}[n,p+\hat{i},q+\hat{j}]\right) \quad (2.18)$$

Where $\mathbf{W}(\cdot)$ is the convolution weight. It is content-aware since it is a function of the raw input X_0 , and spatially-adaptive since it depends on the location (p, q) . The backbone of SqueezeSegV3 is built on RangeNet, with five stages containing several convolution blocks. Several of the convolutions in the original framework are replaced with versions of SAC, and the last two downsampling operations are removed. For more information, the reader is referred to [25]. SqueezeSegV3 is trained on a multi-layer cross-entropy loss where each of the five stages output are included in the loss function. The total loss is a sum of the weighted cross-entropy loss \mathcal{L}_{wce} of each of the five stages as following:

$$\mathcal{L} = \sum_{i=1}^5 \mathcal{L}_{wce} \quad (2.19)$$

SalsaNet

SalsaNet [26] is a simple and fast network that uses an encoder-decoder type architecture on either Bird-Eye-View (BEV) or Spherical-Front-View (SFV) projections. Since SalsaNext (the continued generation of SalsaNet) uses only the spherical (SFV) projection, only this approach will be described. For further information, the reader is referred to [26].

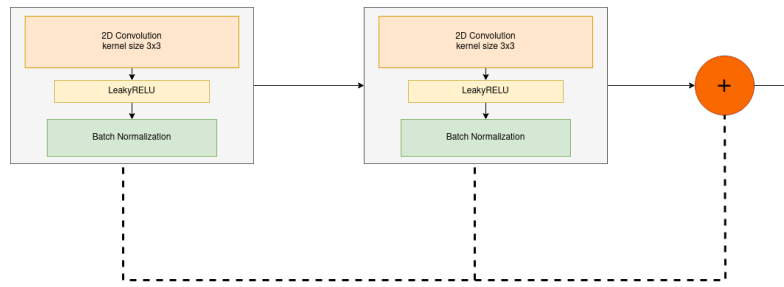


Figure 2.13: The ResNet blocks used in the encoder layers. They consist of a 2D Convolution along with LeakyReLU activation function, and batch normalization. The double skip connection is added at the end of the block.

The Encoder part aims to encapsulate the information in the input to provide meaning features for the decoder. The 2D projected image is directly used as the input and is fed into the encoder. The encoder structure is built up of multiple residual blocks as Figure 2.13 shows. Every block in the network is additionally combined with dropout and pooling layers, except for the very last layer. The max pooling layers employ a kernel size of 2, which brings the total downsampling factor to 16 at the end of the encoder. The feature channels of the convolutions within the blocks are equal. Their value increases along the architecture with values of 32, 64, 128, 256 and 256 used respectively. The double layer skip connection we see in Figure 2.13 reduces the impact of the vanishing gradient and allows the gradient to more freely move through the network. The ResNet blocks in SalsaNet are based on the work behind ResNets. For more information on residual networks, the reader is referred to [27].

The Decoder part is responsible for upsampling the feature maps and combining them with corresponding skip connections from the Encoder. This is done by using a sequence of de-convolution layers which is combined the corresponding layer from the Encoder, followed by a stack of three convolution layers. This was introduced to capture more precise spatial cues which can be used in the higher layers. The final layer applies a 1x1 convolution with channels equal to the number of semantic classes.

After the projected image has been fed through the Encoder-Decoder architecture, it is fed into a soft-max classifier, which returns a pixel-wise classification of the image. SalsaNet is trained with the weighted cross-entropy loss which the ablation study shows increases accuracy for the model. Additionally the ablation study showed that the SFV contains redundant information and the paper concludes BEV projection is more appropriate.

SalsaNext

SalsaNext [28] is the next generation of SalsaNet, ranked first on the semanticKITTI leaderboard when published in 2020. This state-of-the-art neural network introduces a novel architecture that performs uncertainty-aware semantic segmenta-

tion in real-time. The architecture of the network is improved in the following ways:

- A contextual module is introduced.
- The receptive field is increased by replacing the ResNet blocks with a set of novel dilated convolution blocks.
- A *pixel-shuffle* layer replaces the transposed convolution layer in the decoder
- A central encoder-decoder dropout approach is utilized, where dropout is inserted in every encoder-decoder layer except the first and last one
- Average Pooling is introduced in lieu of having stride convolutions in the encoder, to alleviate memory usage.

To gather contextual information throughout the network, a *contextual module* is introduced before the encoder part of the architecture. The context blocks consists of a 1x1 followed by a 3x3 convolution and a 3x3 dilated convolution. This allows for the fusion of the large receptive field of the first convolution with smaller ones to help capture the global context along with detailed spatial information. By placing three such residual dilated convolution stacks at the beginning of the network, global context information can be embedded in the network. As the author notes, this is important for learning complex correlations between classes.

The *pixel-shuffle* layer is introduced to avoid checkerboard artifacts in the upsampling process. It directly uses the feature maps to upsample the input by shuffling the pixels from the channel dimension to the spatial dimension, and is cheaper computationally than transpose convolutions.

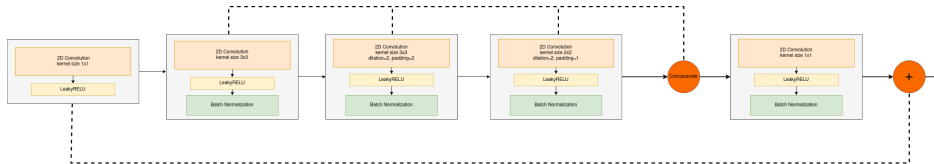


Figure 2.14: The novel combination of dilated convolutions introduced in SalSaNext. The dilated convolutions have a rate of 2, meaning that there is one space between each kernel element both row-wise and column-wise.

To further enhance the segmentation accuracy, the model is optimized using a combination of weighted cross-entropy loss \mathcal{L}_{wce} and *Lovász-Softmax* loss \mathcal{L}_{ls} as described in Section 2.2.5. The total loss will be

$$\mathcal{L} = \mathcal{L}_{wce} + \mathcal{L}_{ls} \quad (2.20)$$

2.3.2 Point-based networks

Point-based networks work directly on the irregular 3-dimensional point cloud. PointNet [18] introduced in 2016, is considered the pioneering neural network within 3D Classification and Segmentation as it was the first neural network to

perform convolutions directly on 3D point clouds. It was later extended to PointNet++ [29] and forms the basis for many of the consequent deep-learning based segmentation methods.

PointNet

The PointNet algorithm works in two parts, a classification part and a segmentation part, as we see in Figure 2.15. Its three main modules are a max-pooling layer, a local and global information combination structure, and two joint alignment networks.

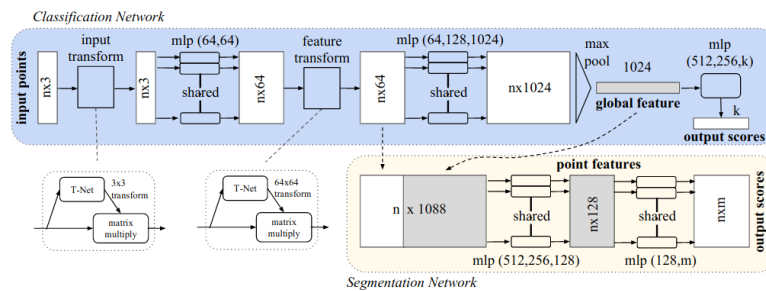


Figure 2.15: The PointNet architecture. Taken from R. Qi et al. [18]

The classification network computes global feature descriptors by convolution kernels, shared multilayer perceptrons (MLP) and a max-pooling layer. After the global point cloud feature vector has been computed, it is concatenated with the point features and based on the combined point features a new per point feature is computed, based on both local and global information. This is what the segmentation network in Figure 2.15 depicts.

To provide transformation invariance, the PointNet network predicts an affine transformation matrix which is then applied directly. Because the predicted transformation is now applied to the neural network, the learnt representation will be invariant to geometric transformations. This joint alignment network is performed on both the input points as well as the point features, to enable invariance throughout. To reduce the complexity of the optimization, the feature transformation matrix is constrained by adding a regularization term in the training loss.

A final max-pooling layer enables permutation invariance and combines all the features to provide a global feature. Permutation invariance implies that if the point cloud was to be rearranged such that the order of the points in the set changed, it would give the same result. This is an important remark as a main impediment of 3D semantic segmentation is the unordered set property.

PointNet++ [29] extended PointNet with a hierarchical feature learning architecture. Instead of the single max pooling layer being applied to the whole point set as in PointNet, PointNet++ groups points and incrementally abstracts larger and larger local regions along the hierarchy. Each abstraction level is composed of a sampling layer, a grouping layer and a PointNet layer. The sampling

layer consists of the iterative Farthest Point Sampling (FPS) to select a subset of points, and compared to random sampling this allows for a better coverage of the entire point set given the same number of centroids. The grouping layer takes in the point set along with the coordinate of the centroids and will output groups of point sets based on their region. Instead of the traditional k-nearest neighbors (kNN) algorithm for grouping, ball query is used to find points within a radius of the queried point. Ball query is particularly better suited at higher dimensions due to it guaranteeing a fixed region scale. This will in turn make local region feature more generalizable, and make ball query more preferable for semantic point labeling. The PointNet layer consists of the network described in [18] which is used to learn local features.

RandLA-Net

To efficiently segment large-scale 3D point clouds, expensive sampling techniques and processing steps must be replaced with more efficient techniques. RandLA-Net [30] is the to-date most efficient and accurate point cloud semantic segmentation that works directly on the 3D point cloud. RandLA-Net distinguishes itself from other point-based networks in the following ways:

- The network only relies on random sampling
- A proposed local feature aggregator obtains larger receptive fields by considering explicitly the local spatial relationship and point features
- The network consists only of shared MLPs and no kernelisation or graph construction, enabling efficient segmentation

Random Sampling (RS) is the key enabler for the efficiency of the network. It uniformly selects K points from the original set of N , and with a computational complexity of $\mathcal{O}(1)$ is both scalable and extremely efficient regardless of input size. Due to this, the RS technique is the most suitable approach for efficient processing of large-scale point clouds. While the Random Sampling is an efficient tool it may also discard key information, particularly on objects with few points. To counteract this, a local feature aggregation module is proposed. The module consists of three neural units: a Local Spatial Encoding block (LocSE), an Attentive Pooling block, and a dilated residual block. The dilated residual block contains a stack of LocSE and Attentive Pooling units with a skip connection, inspired by ResNets [27]. By combining Random Sampling (RS) with the local feature aggregation module, RandLA-Net is able to efficiently and accurately segment a point cloud directly, citing a speed improvement of 200x over existing approaches.

Chapter 3

Method and Implementation

3.1 Synthetic Semantic Dataset Generator (SSDG)

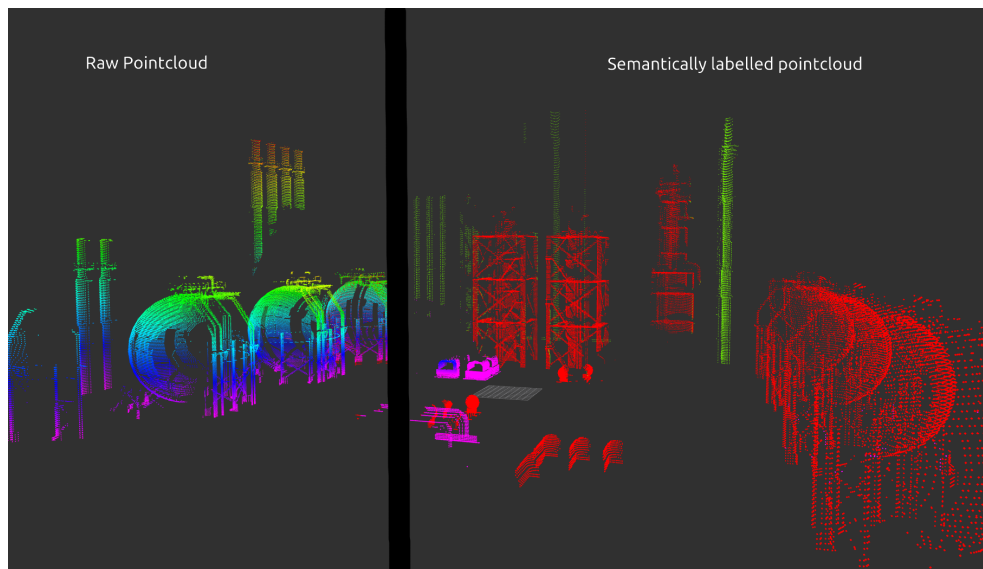


Figure 3.1: Converting a raw point cloud (left) to a semantically labeled point cloud (right) using SSDG. The colors from the raw point cloud stem from z-value, while the colors on the semantic point cloud represent different object categories.

The impressive results of deep learning networks in tasks such as semantic segmentation are in large attributed to the availability of massive datasets with high-quality precise labels. Impressive work has been done on labelling tools and ready-labelled datasets such as SemanticKITTI, but most of the existing benchmarks today are tailored towards autonomous driving. To the best of the authors knowledge, very few or no open-source ready-labelled large industrial point cloud exists to date. It could be suspected that this is due to privacy concerns and data protection, or simply because of the complexity of the data collection process in heavy

industry areas. Some previous work has focused on creating synthetically labelled datasets to deal with this. [31] presents a plugin for the GTA-V computer game that rapidly creates point clouds with accurate per-points labels. Notably, Squeeze-SegV2 [23] mentioned in Section 2.3.1 exploits this plugin for data augmentation. The method is however meant for usage within urban scenes and as such is not suitable for our industrial use-cases. As no sufficient method for semantic dataset generation exists to date, we expand our original intent of creating an industrial dataset and instead create a novel method for general synthetic semantic datasets. The method, Synthetic Semantic Dataset Generator (SSDG), is presented in the following chapter.

3.1.1 Pipeline Overview

SSDG creates a simulated world in which a robot travels around, mounted with LiDAR and segmentation cameras. Using these sensors, the method creates a realistic labelled point clouds of the 3D models in the simulation viewed from multiple different angles. By using the segmentation cameras mounted on the robot, each point generated by the LiDAR is given a semantic label by projecting the 3D cloud onto the 2D image plane. The 2D image from the segmentation cameras contains a label map which is used to give each pixel a semantic value, and can thus be used to give corresponding points from the cloud its correct semantic label.

In this way, SSDG generates a dataset that closely resembles what a real robot would produce in a similar environment to the simulation. The method can be used for a broad range of use-cases, and is not meant for merely autonomous driving or industrial areas. Furthermore, the method may be extended to include an unlimited set of categories and 3D models, and tailored to the use of the practitioner.

At its core, the software consists of the following modules: A Gazebo world, a collision avoidance node, a labelling node for point cloud segmentation, and finally a ROS IGN Bridge to connect the labelling node with the Gazebo world. First, some of the tools and libraries required is presented in Section 3.1.2. Following this, implementation details of the nodes are presented. Our efforts focuses on closing the reality gap and creating realistic data, notably by using a LiDAR sensor that is directly modelled from the specifications of an Ouster OS0-128 as described in Section 3.1.3. The reality gap is further investigated in Section 3.1.5, using techniques within data augmentation and domain randomization.

3.1.2 Simulation Libraries and Tools

This section presents some of the libraries and tools used by SSDG to create the semantic point clouds.

Ignition Gazebo

For our simulations we employ Ignition Gazebo. It offers an efficient simulation environment for robots with a built-in physics engine and graphical interface. Gazebo remains one of the most common simulation environments in robotic research. One of its main advantages is its ease of integration of simulated robots and sensors in experiments. For more information, see [32].

Ignition Sensors

Within the Ignition Gazebo world, the Ignition sensors component provides sensor models that can be used to generate realistic data from the simulation environment. A noise modelling tool is also provided to introduce noise models in the sensor data. This thesis builds on the Ignition LiDAR sensor and Segmentation Camera included in the package. The Segmentation Camera creates a label map where each pixel contains the label of the object in that pixel, and a colored map which contains either instance or label of object depending on whether or not panoptic segmentation is chosen.

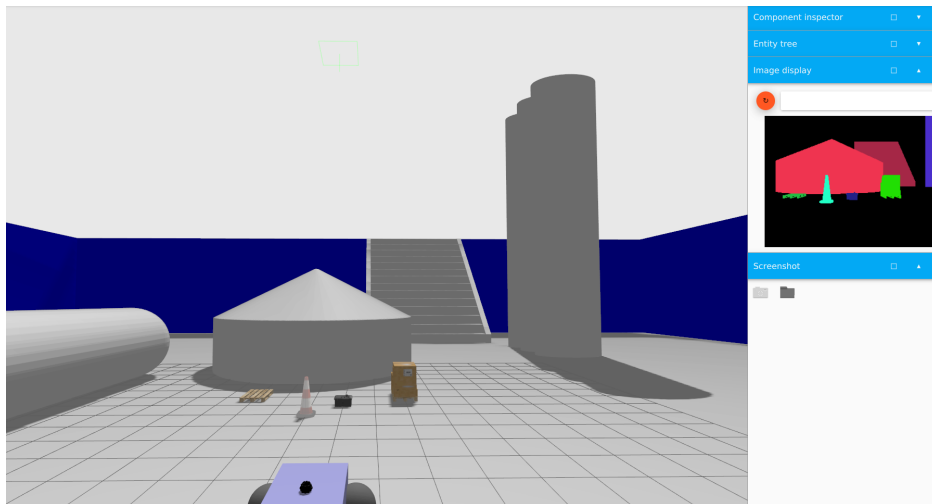


Figure 3.2: An example of using the segmentation camera included in Ignition Sensors, with the label map on the right hand side.

ROS

The Robotic Operating System (ROS) is an increasingly popular robot middleware platform that allows for the systemization and ease of communication between a robot and its software applications. It presents an open-source standardized software platform that allows users to share packages that can be cross-compiled and distributed across systems. This has allowed ROS to become the de-facto operating system for robotic systems, and a global community that helps develop and support the platform. For more information on ROS, see [33].

TF Transform

To keep track of the different coordinate frames and their relative transformations over time, ROS uses the tf package. The package keeps the relationships between the coordinate frames in a tree-like structure, and can be used to transform points between coordinate frames such as lidar-to-camera transformations.

RViz

RViz is a 3D visualization tool for allowing users to visualize information like sensor data, robot pose and point cloud data. In this thesis it is mainly used to visualize the point cloud ported from Gazebo, as well as the labelled point cloud stemming from the labelling node in Section 3.1.4.

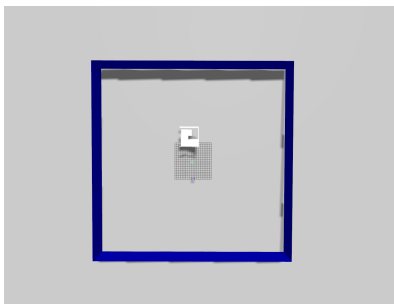
Ros IGN bridge

To be able to port the simulation environment from Gazebo to ROS, we use the `ros_ign_bridge`. It holds packages that provides integration between ROS and Gazebo, namely for transportation of images, point clouds and transformations.

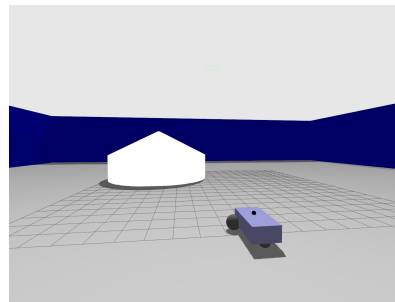
SDFFormat

To describe a simulation to be loaded, Both Gazebo Classic and Ignition Gazebo uses the SDFFormat (SDF) [34]. An SDF file loads the plugins chosen for the simulation, as well as defining the world environment and the robot.

3.1.3 Gazebo Simulation



(a) Birds-eye view of the simulation. The white object in the center is the CAD model and the blue borders are walls.



(b) First-person view of the robot traversing the environment. The white object is an industrial tank, and the blue vehicle is the robot with mounted LiDARs and segmentation cameras.

Figure 3.3: An overview of the simulation setup.

At the main core of the simulation, a gazebo world with containing elements is created from a ROS launch file. The launch file sets up three nodes: the gazebo node, the ROS bridge, and the labelling node. By using a launch file, roslaunch is able to set up all nodes in a single setup making it a seamless task to create pipelines for running and closing simulations. The simulation setup is shown in Figure 3.3 containing a 3D CAD object in the model, and surrounding walls to keep the robot from moving too far away for segmentation of the object.

Gazebo Node

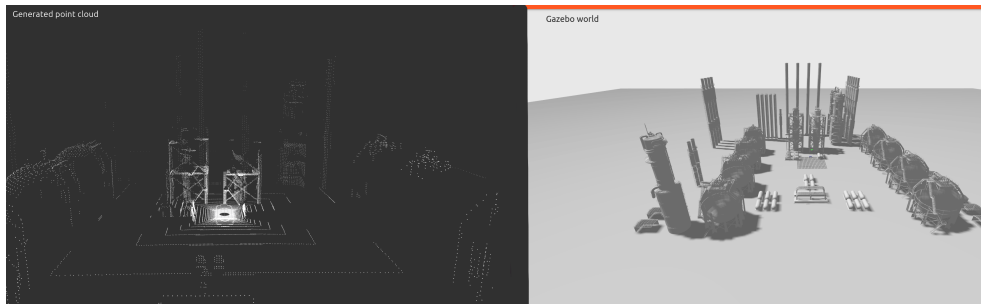


Figure 3.4: An overview of a refinery gazebo world with the pertaining point cloud gazebo generates on the left.

The Gazebo world is set up from the SDF file containing the elements for the simulation. It contains a simple ground plane model and a sun. It additionally loads plugins needed for the simulation, such as ogre2 for rendering, a physics system, ignition sensors, and the label system plugin used by the segmentation camera. To create a realistic dataset that simulates a robot autonomously navigating in an industrial environment, a robot with mounted lidar and segmentation camera sensors is spawned in gazebo. It consists of a square chassis and two separate wheels on each side, that can be controlled to drive and turn the robot by sending an `ignition::msgs::Twist` message over the `/cmd_vel` topic.

The robot is mounted with two LiDAR sensors, one for creating the segmentation point cloud, and one for collision avoidance. The collision avoidance lidar only publishes points that are in front of the robot, and is used to keep the robot from hitting obstacles. The specifications for this LiDAR can be found in Table 3.1. Note that the minimum range of the collision avoidance lidar is 1.7 meters so it does not risk confusing the chassis of the robot with an obstacle.


Collision Avoidance LiDAR	
Update rate	60 Hz
Max range	10.0 m
Min range	1.7 m
Range resolution	0.03
Horizontal samples	100
Horizontal ray max angle	$\frac{\pi}{2}$ rad
Horizontal ray min angle	$-\frac{\pi}{2}$ rad
Horizontal ray resolution	1
Vertical samples	100
Vertical max angle	1 rad
Vertical min angle	0 rad
Vertical resolution	0.1

Table 3.1: Specifications for Collision Avoidance LiDAR

The other LiDAR is responsible for creating the point cloud that is assigned labels by the labelling node and used as dataset. To create a segmentation network that can work on real-world data, the LiDAR is configured as the Ouster OS0-128 LiDAR. The OS0-128 has a 90° vertical Field-of-view, a vertical resolution of 128 and a configurable horizontal resolution of 512, 1024 or 2048. The specifications for the LiDAR can be found in Table 3.2.

Segmentation LiDAR	
Update rate	10 Hz
Max range	50.0 m
Min range	1.7 m
Range resolution	0.03
Horizontal samples	512
Horizontal ray max angle	π rad
Horizontal ray min angle	$-\pi$ rad
Horizontal ray resolution	1
Vertical samples	128
Vertical ray max angle	$\frac{\pi}{4}$ rad
Vertical ray min angle	$-\frac{\pi}{4}$ rad
Vertical resolution	0.1
Noise	$\sim \mathcal{N}(0, 0.008)$

Table 3.2: Specifications for the simulated Segmentation LiDAR, modelled from the Ouster OSO-128.



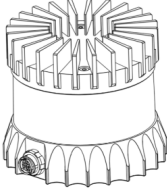
OUSTER

Ouster, Inc.
350 Treat Ave.
San Francisco, CA 94110

OSO
Ultra-Wide View High-Resolution Imaging Lidar

Revision: 12/7/2020
FIRMWARE VERSION: v2.0
HARDWARE VERSION: Rev C

SUMMARY
The OSO offers an ultra-wide 90° vertical field-of-view with an industry-leading combination of price, performance, reliability, size, weight, and power. It is designed for indoor/outdoor all-weather environments and long lifetime. As the smallest high performance lidar on the market, the OSO can be easily integrated into autonomous vehicles, heavy machinery, robots, drones, and mapping solutions.



HIGHLIGHTS

- Fixed resolution per frame
- Camera-grade near-infrared and intensity data
- Multi-sensor crosstalk immunity
- Fixed intrinsic calibration
- Open source drivers

OPTICAL PERFORMANCE	
Range (80% Lambertian Reflectivity)	45 m @ 100 kix sunlight, >90% detection probability 50 m @ 100 kix sunlight, >50% detection probability
Range (10% Lambertian Reflectivity)	15 m @ 100 kix sunlight, >90% detection probability 20 m @ 100 kix sunlight, >50% detection probability
Minimum Range	0.3 m for point cloud data 0 m - 0.5 m blockage detection (flag to indicate object within 0.3 m)
Range Accuracy	±3 cm for lambertian targets, ±10 cm for retroreflectors
Precision (10% Lambertian Reflectivity, 1 standard deviation)	0.3 - 1 m: ±2 cm 1 - 10 m: ±1 cm 10 - 15 m: ±1.5 cm >15 m: ± 5 cm
Range Resolution	0.3 cm
Vertical Resolution	32, 64, or 128 channels
Horizontal Resolution	512, 1024, or 2048 (configurable)
Field of View	Vertical: 90° (+45° to -45°) Horizontal: 360°
Angular Sampling Accuracy	Vertical: ±0.01° / Horizontal: ±0.01°
False Positive Rate	1/10,000

Figure 3.5: Ouster OSO-128 Specifications.¹

Along with the two LiDARs a set of 16 segmentation cameras are mounted on the robot. The cameras are responsible for creating label maps that are used by the labelling node (3.1.4) to give each point in the point cloud a label. The cameras are mounted in the same spot as the LiDAR, but pointing in different directions. This is done to semantically label every point in the full 360° cloud. The specifications for the segmentation cameras are found in Table 3.4 and Table 3.3.

¹Retrieved from <http://www.oust.com/wp-content/uploads/2021/01/Ouster-datasheet-revc-v2p0-os0.pdf>

Camera Settings
frame_id: "vehicle_blue/semantic_camera_link_1/semantic_segmentation_camera_1"
height: 240
width: 320
distortion_model: "plumb_bob"
D: [0.0, 0.0, 0.0, 0.0, 0.0]
K: [277.0, 0.0, 160.0, 0.0, 277.0, 120.0, 0.0, 0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [277.0, 0.0, 160.0, -0.0, 0.0, 277.0, 120.0, 0.0, 0.0, 0.0, 1.0, 0.0]
binning_x: 0
binning_y: 0
roi:
x_offset: 0
y_offset: 0
height: 0
width: 0
do_rectify: False

Table 3.3: Camera Info

Segmentation Camera	
Update rate	30 Hz
Max range	50.0 m
Min range	1.7 m
Image width	320
Image height	240

Table 3.4: Specifications for Segmentation Cameras.

In addition to the robot and sensors, the Gazebo world will contain the objects to be labelled. The code snippet below presents the model for a tank. On line 6-8 the label system is used to give a semantic label to the object, in this case 150. The SDF model also contains the *uri* for the object file (Collada/OBJ/STL) on line 12, and the scaling factor on line 11.

```

1  <model name="tank">
2  <pose>0 0 0 0 0 0</pose>
3  <static>true</static>
4  <link name="body">
5    <visual name="visual">
6      <plugin filename="ignition-gazebo-label-system" name="
7        ignition::gazebo::systems::Label">
8        <label>150</label>
9      </plugin>
10     <geometry>
11       <mesh>
12         <scale>0.112334 0.198570 0.169766</scale>
13         <uri>tanks/tank_20.STL</uri>
14       </mesh>

```



```

14     </geometry>
15   </visual>
16 </link>
17 </model>

```

ROS Bridge

To port the gazebo simulation to a platform that can be used for inference and segmentation, the ROS IGN bridge is employed. The package bridges the network between a ROS core and Ignition Transport from the Gazebo world. We employ a unidirectional bridge, where ROS is the subscriber and Ignition the publisher. The bridge is responsible for transporting two central components for the labelling node, the LiDAR point cloud and the label map. To correctly overlay the pointcloud to the label maps, the TF transformations are also included, as well as the camera info used by the cameras. The `gpu_lidar_collision_avoidance/points` topic is used by the collision avoidance node (Section 3.1.3) algorithm for robot movement.

Topic	ROS type	Ignition Transport type
lidar	<code>sensor_msgs/LaserScan</code>	<code>ignition::msgs::LaserScan</code>
lidar/points	<code>sensor_msgs/PointCloud2</code>	<code>ignition::msgs::PointCloudPacked</code>
gpu_lidar_collision_avoidance/points	<code>sensor_msgs/PointCloud2</code>	<code>ignition::msgs::PointCloudPacked</code>
panoptic/camera_info	<code>sensor_msgs/CameraInfo</code>	<code>ignition::msgs::CameraInfo</code>
model/vehicle_blue/pose	<code>tf2_msgs/TFMessage</code>	<code>ignition::msgs::Pose_V</code>
panoptic/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic2/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic3/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic4/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic5/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic6/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic7/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic8/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic9/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic10/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic11/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic12/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic13/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic14/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic15/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>
panoptic16/labels_map	<code>sensor_msgs/Image</code>	<code>ignition::msgs::Image</code>

Table 3.5: Topics shared between ROS Ignition Transport.

Collision Avoidance Node

To keep the robot within segmentation distance of the 3D object, a set of walls surround the object. The robot movement is controlled from a collision avoidance node which gives the robot random trajectories to follow, and keeps the robot from hitting the object and walls. The node subscribes to the `gpu_lidar_collision_avoidance` topic and on callbacks checks the range of all rays to decide whether the robot is

too close to an object or not. If the object is within 4 meters of the object, the node publishes a message to rotate the robot, otherwise it publishes a message to move forward with a random slight rotation. A random number generator is used to decide the direction of the robot, with the aim of letting it see the object from as many random angles as possible. The algorithm for the collision avoidance is described in Algorithm 1. x and z is the forward and rotational speed respectively, and once the state of the robot has been determined, a `ignition::msgs::Twist` message containing the new angular and linear speed is published to the `"/cmd_vel"` topic.

Algorithm 1 Robot movement algorithm

```

if all_ranges < 4.0 then           ▷ Rotates the robot until no object is in front
   $x \leftarrow 0.0$ 
   $z \leftarrow 0.5$ 
else                               ▷ Robot moves forward with random direction
   $x \leftarrow 1.5$ 
   $z \leftarrow \text{random}(-0.3, 0.3)$ 
end if

```

3.1.4 Labelling Node

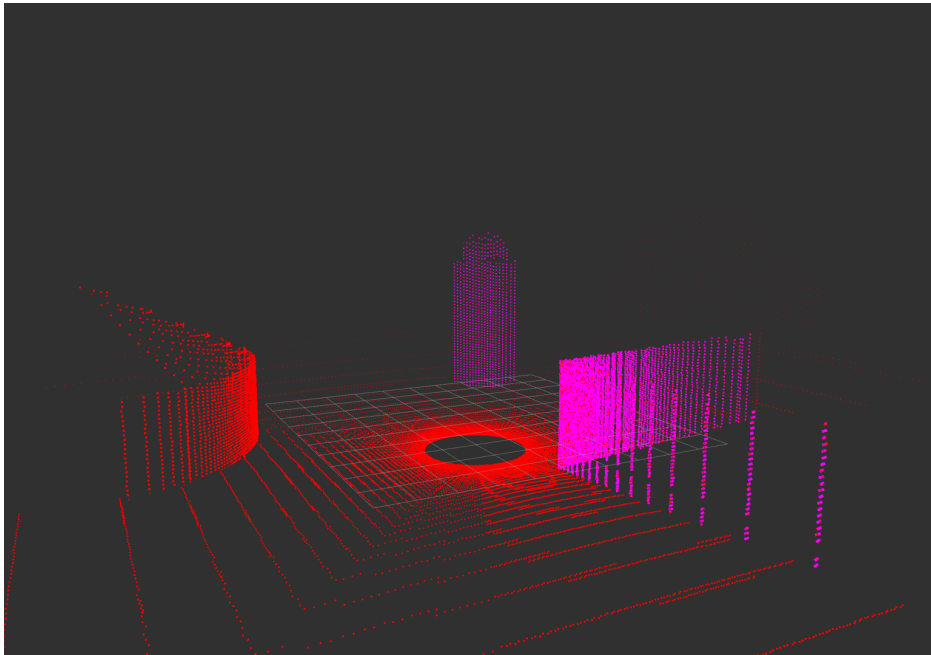


Figure 3.6: An example showing the labelling node colorizing the point cloud. The industrial tank on the left is labelled in red, and the two other objects are labelled in pink. Note that the original lidar point cloud is also displayed with red points.

The labelling node is responsible for assigning labels to each point in the generated point cloud. It subscribes to the topics described in Table 3.5 and uses the label map to give each point in the pointcloud a semantic label. The method can be split into two steps: initialization and callback handling. For initialization the following steps are carried out:

Table 3.6: Initialization

1. Initialize the ROS node: "labelling_node".
2. Set up a pinhole camera model for projections.
3. Subscribe once to "camera_info" topic to update the pinhole camera model based on the segmentation camera configurations.
4. Subscribe to image and lidar topics described in Table 3.5.
5. Set up tf2 transform listener and pointcloud publisher.
6. register a callback based on an approximate time synchronizer. The *approximate time synchronizer* module synchronizes multiple messages over the ROS topics and triggers callbacks when all have arrived. This guarantees that the point cloud and label maps are from the same point in time, within a given limit of 0.1 secs.

After initialization is successful, the approximate time synchronizer callback triggers the following:

Figure 3.7: On Callback

1. Read point cloud using fields ("x","y","z","intensity") and filter out points with NaN values.
2. For all 16 images, convert type sensor_msgs/Image to numpy array for fast indexing.
3. For all 16 images, look up segmentation camera transformation.
4. For the $x \in [1, 16]$ sets of [point cloud, transform_link_x, image_x]:
 - a. Transform the point cloud to the camera frame using the tf transformation.
 - b. Map each point p from the 3D point cloud to the 2D image plane applying the pinhole camera model
 - c. Retrieve label from image_x for all projected points that are within the image frame i.e. ($0 < x < width$ and $0 < y < height$)
 - d. For all points with a valid label within the image, add the original point p and its corresponding label
5. Publish the colored pointcloud to ROS, as well as saving it to file. Note that multiprocessing is used to process all 16 sets [point cloud, transform_link_x, image_x] simultaneously.

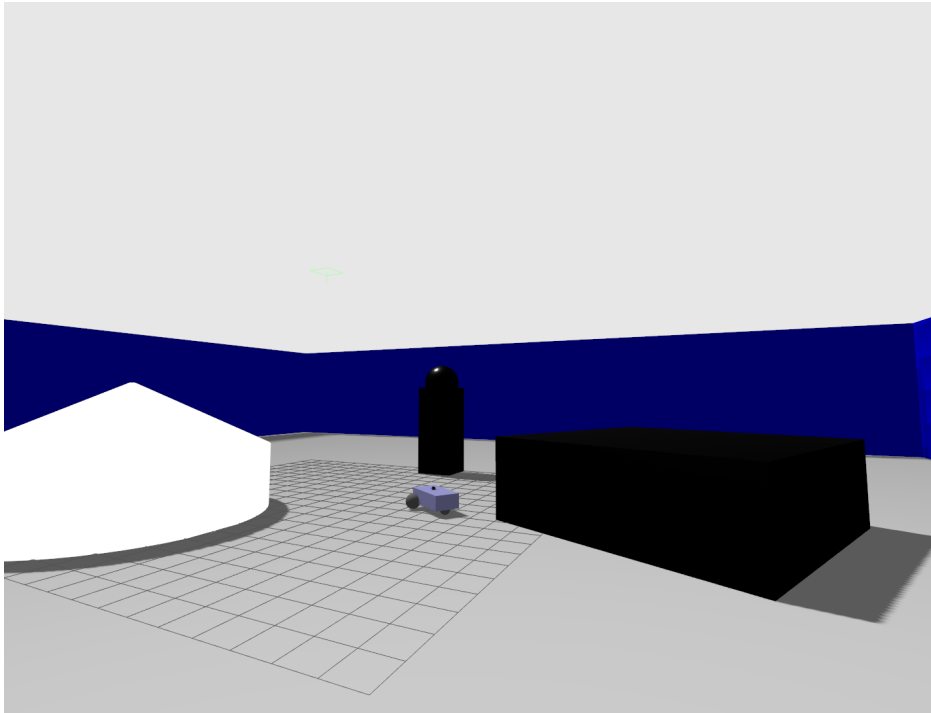


Figure 3.8: The gazebo simulation corresponding to the point cloud in Figure 3.6

In Figure 3.6 we see the colored pointcloud overlaid on the original lidar pointcloud. In this example, the industrial tank is labelled with the color red, and the other objects are labelled with the color pink. The corresponding gazebo simulation is shown in Figure 3.8. The segmentation cameras are panoptic, and can deliver instance segmentation in addition to label segmentation. An example of instance segmentation can be seen in Figure 3.9, and can be developed further to provide a panoptic segmentation dataset.

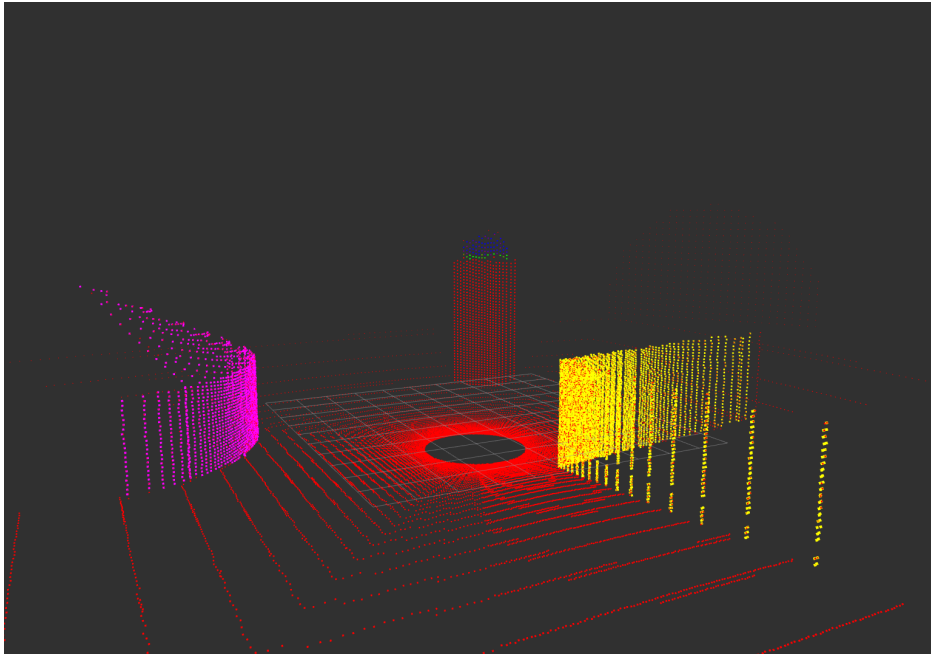


Figure 3.9: An example of instance segmentation, with each object assigned a separate instance and color.

3.1.5 Bridging the reality gap

An important step in generating synthetic data that can be used in real-world situations is reducing the disparity between the computer generated environment and the real world environment. This discrepancy between the real world and simulated world is often referred to as the reality gap. To bridge this disparity, this thesis focuses on three main approaches: increasing resemblance to real-world data, domain randomization and data augmentation.

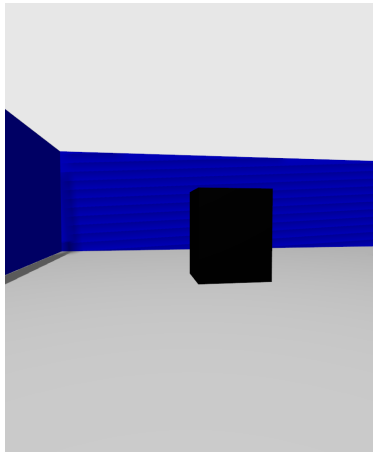
Increasing resemblance to real-world data

Some efforts on increasing the resemblance between real-world and synthetic data is carried out to reduce the reality gap. By using the Ignition Gazebo simulation world with a physics engine that closely emulates the physical world, our method generates highly realistic data. We simulate a Ouster OS0-128 LiDAR sensor directly adapted from the specifications from Ouster, increasing the likeness to the real-world domain. Moreover, the sensor is infused with gaussian noise according to Table 3.2 to increase the likeness to the physical world, as LiDAR sensors are not perfect and inevitably contain both fragments and sensor noise.

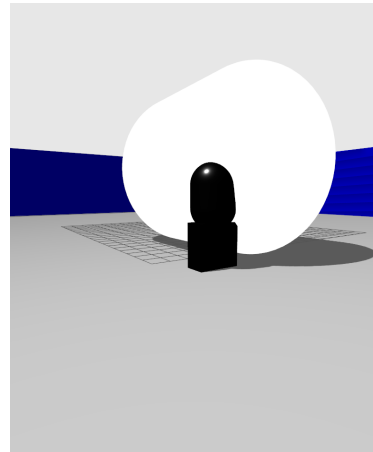
Mesh Sampling

Another matter that can alter the resemblance between a physically collected dataset and a synthetic dataset is how the synthetic objects are converted to point cloud objects. A naive approach would be to convert the 3D CAD models to point cloud meshes directly, without a simulated world. This presents several issues. Firstly, a mesh sampling is too perfect: a LiDAR will never provide a fully precise point cloud of a real-world object like mesh sampling will, but rather create a viewpoint-specific noisy point cloud with outliers and artifacts. Additionally, the mesh-converted point cloud will not capture the object from multiple viewpoints. To create a complete dataset that captures what a LiDAR would see in the real world, the object needs to be captured from as many point of views as possible so the segmentation network can correctly label the object no matter which side of the object it is viewing. The object will likely not be found alone, and as such all contextual information is also lost by doing a mesh sampling of a single model. This is one of the reasons why our method SSDG is of a more hybrid domain than what a mesh conversion would be. Note also that the LiDAR does not see through objects, and as such natural effects such as occlusions occur, and only the parts of an object within line-of-sight are captured.

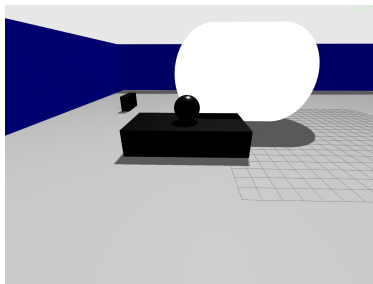
Domain Randomization



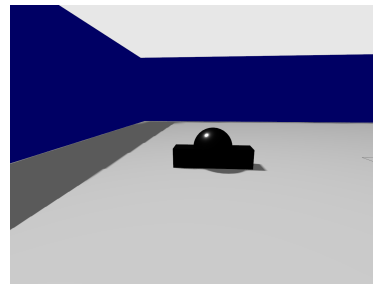
(a) A cube



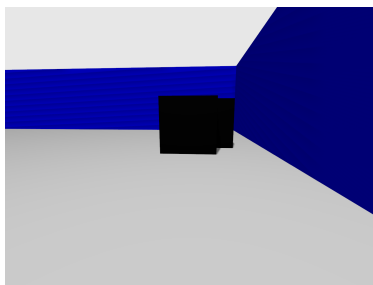
(b) Two spheres on top of an oblong cube.



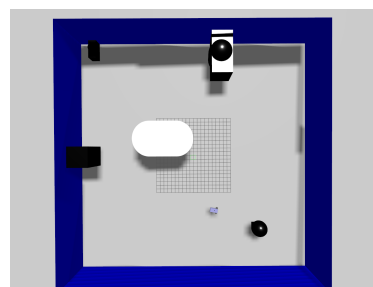
(c) A small sphere on top of a rectangular cube.



(d) A large sphere in the middle of a rectangular cube.



(e) Two cubes in close proximity.



(f) Birds view of the simulation setup, with domain randomization objects in random areas.

Figure 3.10: An example of a synthetically generated world comprising multiple parametric shapes. The black objects represent the domain randomization objects while the white object is a simple model of an industrial tank.

Few synthetic 3D domain randomization schemes for segmentation exists to date. This is perhaps no surprise as there exists very few synthetic 3D point cloud segmentation datasets. In the following section, we introduce our attempt at designing a domain randomization scheme to generate robust 3D segmentation models and improve the results and resilience for future real-world use.

We follow the work by Borrego *et al.* [35] on applying domain randomization to synthetic data, and introduce a novel method that randomly generates a new set of objects to the simulation. The goal behind the method is to synthetically induce random data into the dataset to introduce *non-essential* variances in the environment. This high variability of the environment will allow the network to become robust towards disturbances. The end goal of the domain randomization scheme is that a model trained on the simulated domain perceives the variation between the simulated and a real domain as mere disturbance.

The method works in the following fashion. In each sequence containing a new unseen world and 3D CAD model of a labelled object, we introduce a set of extra objects based on different combinations of parametric shapes. The objects are created based on two parametric shapes: a sphere, and a cube. Their shapes are varied in both radius, form and their location relative to each other, and these variables are randomly generated for each new object. In Figure 3.10 we see a selection of random objects that has been generated using this method. There exists a multitude of combinations that creates vastly different objects. In Figure 3.10a we see a simple cube, in Figure 3.10b two spheres on an oblong cube, while Figure 3.10e is instead a small sphere on top of a rectangular cube. We consequently imply that the model is able to generate high variation within the shapes of the generated objects.

The implementation is based around the entity creation found within the Ignition Gazebo framework². This service allows creating entities in the scene such as cubes, spheres, lights, etc. We generate a string for each object, for example *sphereStr* for a sphere, that contains an SDF-style spherical model with randomized location and radius. This is then parsed through the entity creation module embedded in the Gazebo framework, and we can create an array of randomized shapes in this way. To generate the complete objects, consisting of several parametric shapes, we construct several of the parametric shapes and deviate their poses slightly from each other. In this way, an object consisting of for example two spheres and one cube will be formed by constructing a pose for the first parametric shape, and deviating this pose for the next two.

The parametric shapes create a set of new dissimilar objects in each instance. Between each of the sequences in the dataset, the scene is destroyed and a new created with new randomized objects. As such, no sequences will have the same parametric objects. Our method for domain randomization reduces the amount of non-essential and unrealistic data which the robot will believe to be essential, but is in fact not.

²Documentation can be found at https://gazebo.org/api/gazebo/4.3/entity_creation.html

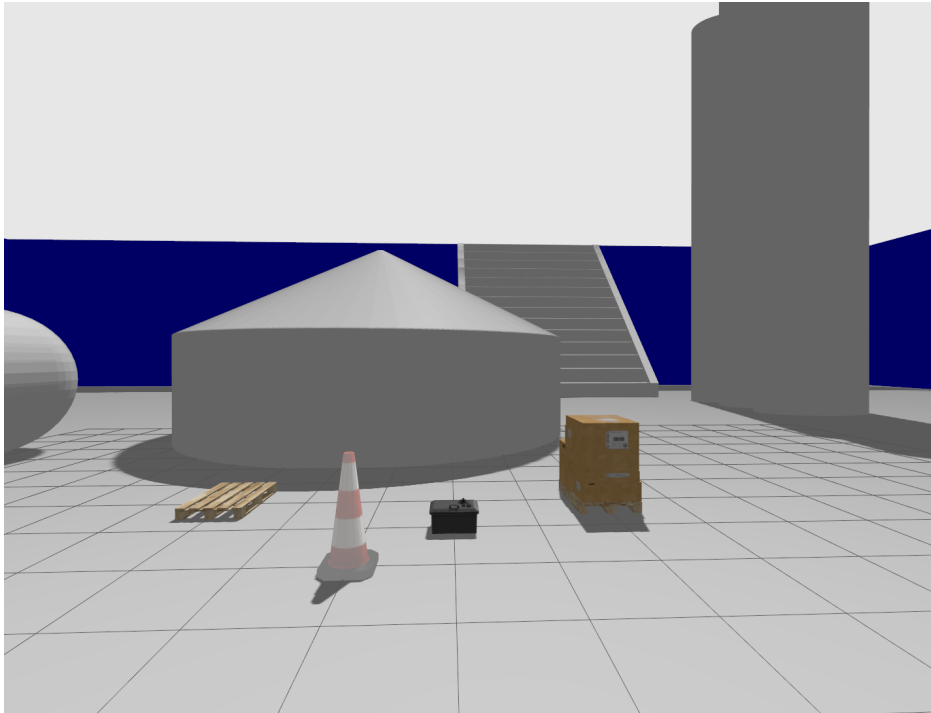


Figure 3.11: From left to right: pallet, cone, electrical box and pallet box.

In addition to the random objects, we introduce 4 industrial objects to increase the variation in the dataset. The cone [36], pallet box [37], pallet [38], and electrical box [39] were retrieved directly from the Ignition Fuel³ web application. They represent random objects that the robot could come across in industrial areas and introduce both variation and contextualization to the point cloud. Since we only include four objects, the variation this induces will be relatively small. Nonetheless, we provide an easily extendable method for creating a highly varying environment in which a robot can traverse and generate point cloud data. The main intention behind these objects is showing a proof-of-concept and presenting the potential possibilities for further use of the method, more than just usage for this particular industrial dataset.

Data Augmentation

An alternative to manually creating more CAD models to increase dataset size is to generate additional synthetic data from copies of already existing CAD models. In this thesis, the set of CAD models for the object categories *tanks* and *chimneys* were augmented since only 20 tanks and 20 chimneys were created.

The data augmentation was done in the following fashion. Each set of 20 tanks and 20 chimneys were augmented in two different fashions, creating a total of 60 unique tanks and 60 unique chimneys. By sampling a random distribution

³<https://app.gazebosim.org/dashboard>

transform in a pre-determined range, each object is given a new scaling factor in x, y, z directions. For the first data augmentation, each object is given a scaling of $x = y = \mathcal{U}(4 - 7)$ and $z = \mathcal{U}(4 - 7)$. For the second augmentation transform, the scaling values are $x = y = \mathcal{U}(1 - 4)$ $z = \mathcal{U}(1 - 4)$. Note that the x and y values have corresponding values to not warp the general object form in a manner where e.g. a chimney could look like a tank.

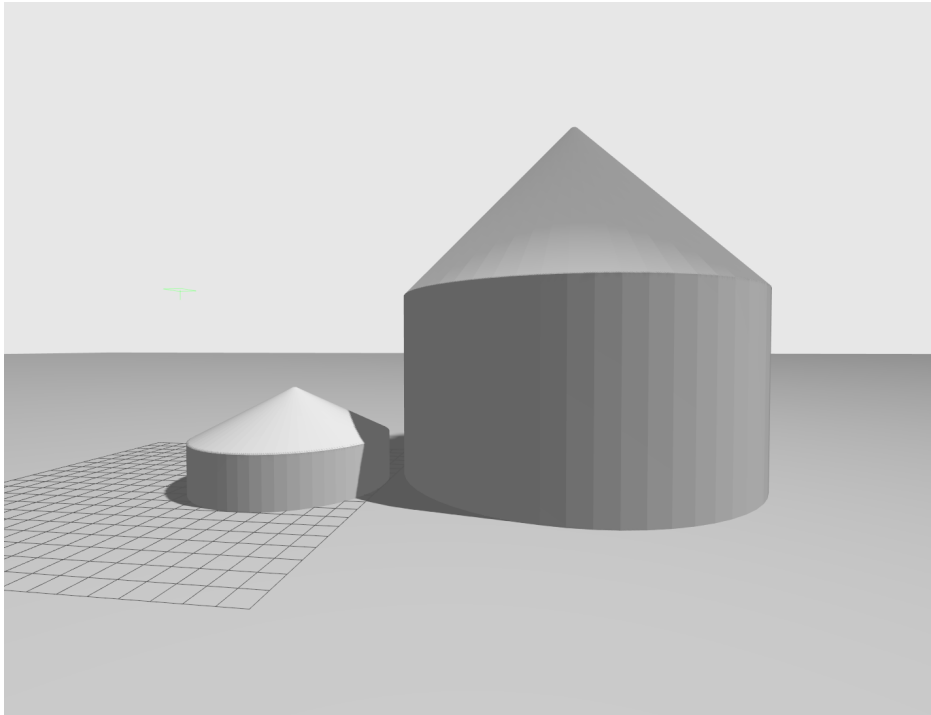


Figure 3.12: The same industrial tank, with different scaling factors showing the data augmentation technique

3.2 Industrial Segmentation Dataset

Only recently has the semantic annotation of LiDAR scans taken a big leap due to the releases of annotated large-scale datasets such as SemanticKITTI, Paris-Lille-3D [40] and Oakland3D [41]. The otherwise hugely extensive task of annotating thousands of points in a point cloud means these public datasets have facilitated the development of numerous LiDAR-based semantic segmentation techniques. However, most of them cater only to the task of autonomous driving in urban scenes and many are in relative small spatial scale and have limited semantic annotations. Due to the expensive cost of data annotation and acquisition, the development of semantic understanding in 3D point clouds is mainly limited by a lack of data.

	#scans	#points (in millions)	#classes	sensor	Use
Paris-Lille-3D [40]	3	143	9	Velodyne HDL-32E	Urban scenes
Oakland3D [41]	17	1.6	44	SICK LMS	Urban Scenes
SemanticKITTI [1]	43552	4549	25	Velodyne HDL-64E	Urban scenes
Industrial (ours)	164851	3.7	4	Simulated Ouster OS0-128	Industrial scenes

Table 3.7: Overview of point cloud datasets for semantic segmentation.

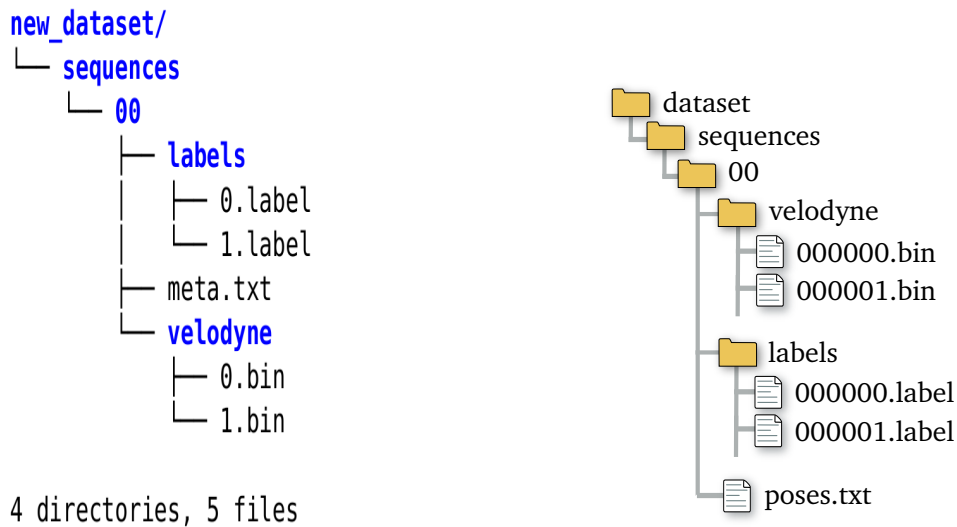
In this paper, we present a industrial point cloud dataset with over 3.7 million annotated points. Our dataset consists of 180 objects pertaining to 4 semantic classes we consider essential in industrial facilities. In the dataset, each 3D point is labeled as one of the four semantic classes based on the label map from the segmentation cameras. We extensively evaluate the performance of two selected state-of-the-art (SOTA) deep learning segmentation networks on our dataset and provide an analysis of the results in Chapter 4. The four classes, their annotation labels and colors is presented in Table 3.8. Note that the colors persist throughout the visualizations for the whole thesis.

class	Annotation label	learning_map label	BGR color map	color (for visualization)
other-objects	10	1	[0, 255, 0]	green
stairs	50	2	[0, 0, 255]	red
chimneys	100	3	[255, 0, 0]	blue
tanks	150	4	[125,125,125]	gray

Table 3.8: Overview of semantic classes in the industrial dataset. learning_map label is simply the one-hot encoding used by the SemanticKITTI dataset format

3.2.1 Dataset format

SemanticKITTI is per today the de-facto standard for benchmarking point cloud semantic segmentation in autonomous driving scenarios. As such, to easily train several segmentation model, we mirror the data structure used by semanticKITTI.



(a) The folder structure for the industrial dataset.

(b) The SemanticKITTI folder structure.⁴

Figure 3.13: Comparison of our dataset folder structure with SemanticKITTI.

The industrial and semanticKITTI dataset structures are seen in Figure 3.13. It contains a sequence folder, in which each sequence will contain a set of corresponding labels and laser scans. The labels and laser scans are given subfolders, and numbered sequentially. Two minor changes have been made from the original KITTI structure:

- The label files do not include instance label, only a semantic label.
- A text file with object name and scaling factors are included in each sequence for traceability.
- The text file with poses is omitted.

Each 3D object is given a separate sequence, and thus the amount of sequences will amount to the number of 3D objects provided.

3.2.2 CAD Models

For the industrial dataset, a total of 120 stairs were collected from ModelNet [5]. The industrial chimney and tank CAD models were created by the author, using SolidWorks⁵. A total of 20 tanks and 20 chimneys were created in different sizes and variations, closely mimicking models from real-world industrial facilities. The resulting files were converted to STL files and directly imported to Gazebo.

⁴Retrieved from <http://semantic-kitti.org>

⁵<https://www.solidworks.com/>

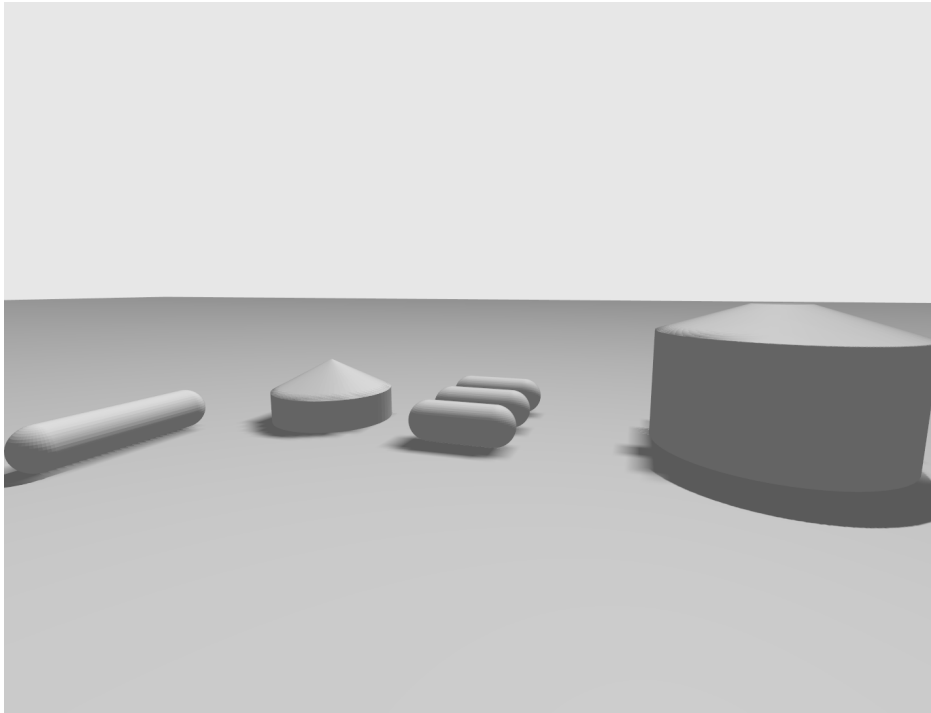


Figure 3.14: A sample of the tanks, rendered in Ignition gazebo

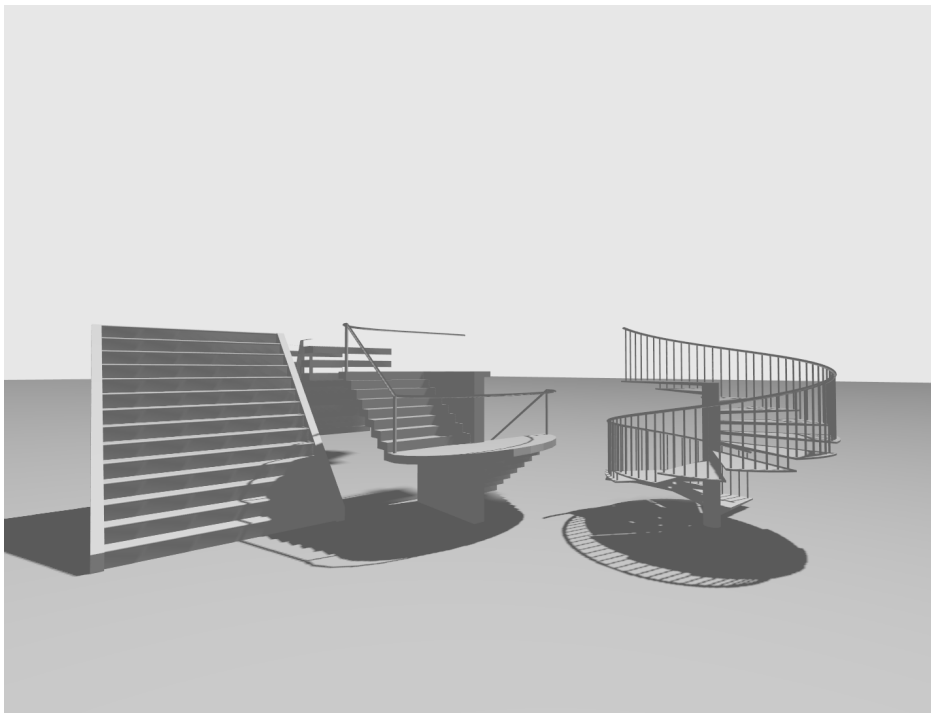


Figure 3.15: A sample of the stairs, rendered in Ignition gazebo

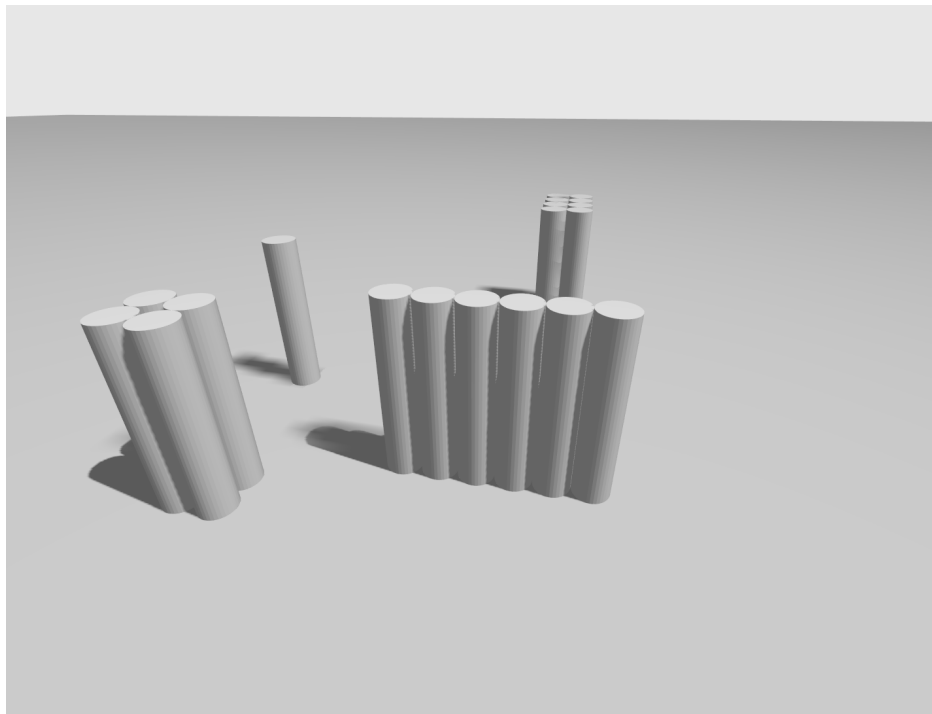


Figure 3.16: A sample of the chimneys, rendered in Ignition gazebo

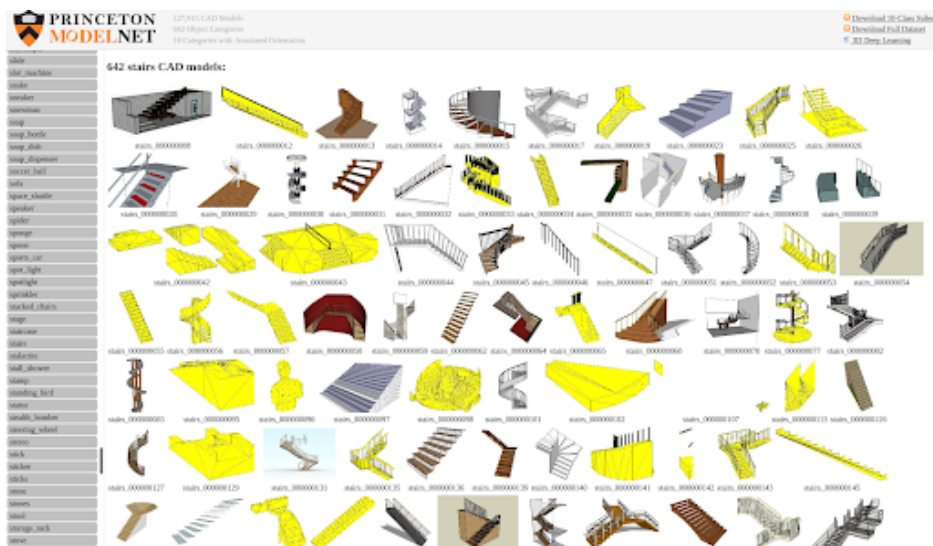


Figure 3.17: The stairs from ModelNet40 that were used in the industrial dataset.

For the stairs stemming from ModelNet40 (3.17) a few extra steps were required to include the models in the simulation. ModelNet40 delivers model in the Object File Format (OFF) which cannot directly be employed by Ignition Gazebo. Therefore, a blender script was created to convert the files from OFF to Collada

(DAE) files. Additionally, a scaling factor is introduced to correct convert the sizes. The script for the conversion is included below.

```

1 import os
2 import sys
3 import glob
4 import bpy
5 import time
6
7 scale = 0.01
8 if len(sys.argv) != 7:
9     print("Must provide input and output path")
10 else:
11     for infile in glob.glob(os.path.join(sys.argv[5], '*.off')):
12         override = bpy.context.copy()
13         override['selected_objects'] = list(bpy.context.scene.objects)
14         bpy.ops.object.delete(override)
15         obj = bpy.ops.import_mesh.off(filepath=infile)
16         bpy.ops.transform.resize(value=(scale, scale, scale))
17         bpy.context.view_layer.update()
18
19         for ob in bpy.data.objects:
20             x,y,z = ob.dimensions
21             ob.scale = (10/x,10/y,10/z)
22             bpy.context.view_layer.update()
23         outfilename = os.path.splitext(os.path.split(infile)[1])[0] + ".dae"
24         bpy.ops.wm.collada_export(filepath=os.path.join(sys.argv[6],
25             outfilename))
26         time.sleep(0.1)

```

3.2.3 Sequence creation

To ease the process of data generation, scripts that run through a set of objects in a folder and create a dataset were created. In this way, the otherwise manual work of going through a set of files and running the method described in Section 3.1 is automated. The script runs through the folder and generates the dataset following the structure described in Section 3.2.1. The script can be summarized as follows:

1. Iterate through the folder and find all objects with the given format (Collada/OBJ/STL).
2. Create the necessary folder structure.
3. For each object:
 - a. Create the 'meta.txt' file included in each sequence, describing scaling factor and object filename.
 - b. Edit the SDF file employed by the gazebo world to spawn the current object being iterated.
 - c. Include a scaling factor for the object, based on the method described in Section 3.1.5.

- d. Run the method described in Section 3.1. This is done by starting the roslaunch file with the updated SDF file, and the domain randomization creator described in Section 3.1.5.
 - e. Run the collision avoidance node to move the robot around the simulation.
4. Run simulation for 30 minutes, close all processes and begin with the next object.

3.2.4 Overview of the resulting dataset

In this subsection, the resulting industrial dataset is presented, together with detailed statistics. The dataset consists of around 3.7 million points, split up into four semantic categories. Since a domain randomization scheme was included, a large portion of the dataset is labelled as "other objects" which correspond to objects from the domain randomization. The frequency of each category is presented in Table 3.9. Class-balanced metrics in the form of weighted IoU or similar should be employed by neural networks on the detection task for the dataset. The dataset is well suited for applications in industrial facilities such as oil refineries.

For contextualization purposes, five multi-object worlds were created to augment the dataset. This is due to the fact that a model trained only on simulations where a single semantic object of importance is present will get a poor prediction accuracy on real-world datasets containing a multitude of objects bundled together. We created five worlds that contain one staircase, one industrial tank, and one industrial chimney, in addition to the domain randomization objects. Below we present details and sample semantic point clouds from the dataset.

Type	Class frequency
Other objects	54.15%
Stairs	24.99%
Industrial chimneys	9.67%
Industrial tank	11.19%

Table 3.9: Frequency of the four different semantic labels in the industrial dataset

The following table represents the generated sequences and their respective object categories and whether or not they include data augmentation and domain randomization schemes.

sequence number	object	data augmentation	domain randomization
0-119	stairs	no	yes
120-139	chimneys	no	yes, but no industrial objects
140-160	chimneys	yes	yes, but no industrial objects
160-179	chimneys	yes	ye, but no industrial objects
180-199	tanks	no	yes
200-219	tanks	yes	yes
220-239	tanks	yes	yes
240-245	multi-object	no	yes

Table 3.10: An overview of the sequences and their object categories, created for the industrial dataset.

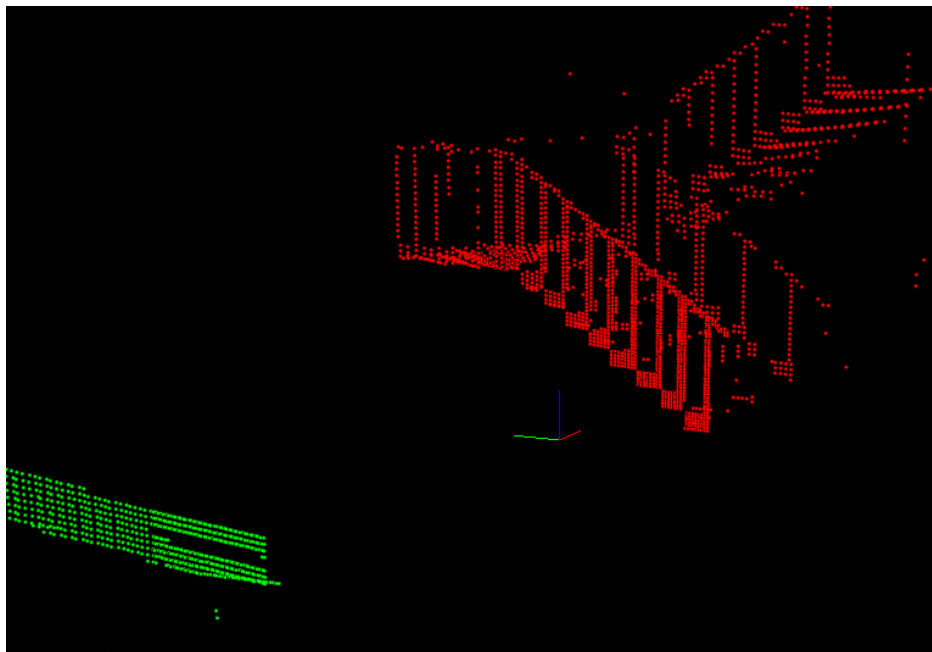


Figure 3.18: A staircase from the industrial dataset.

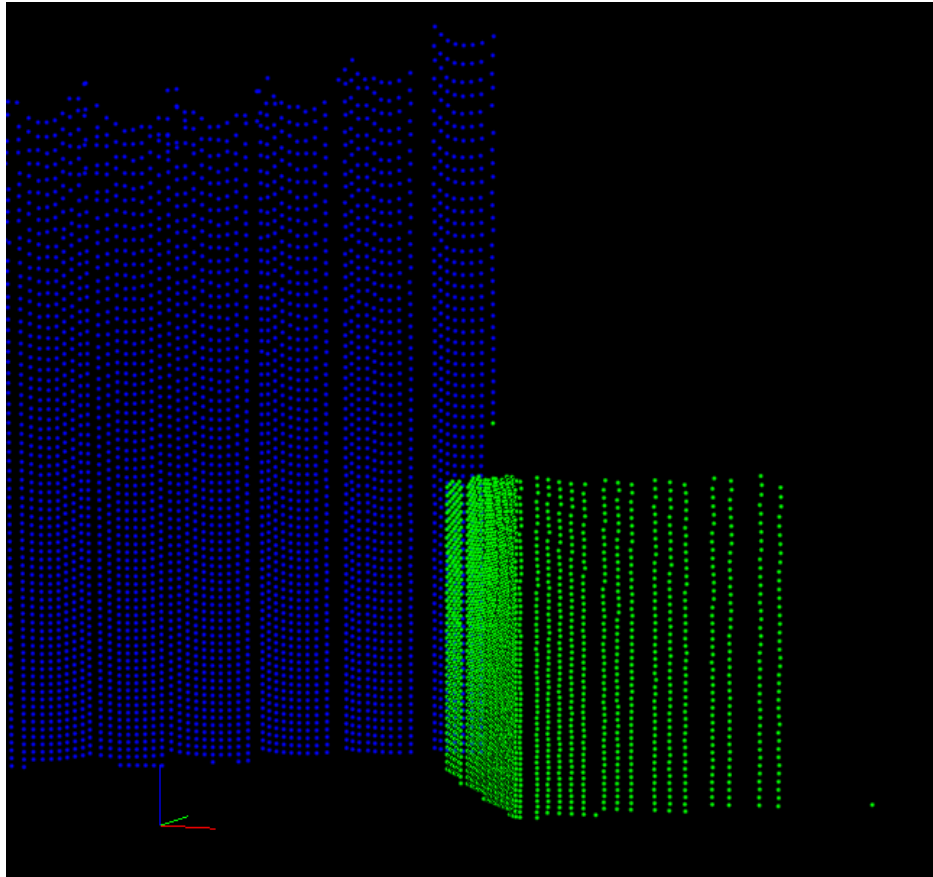


Figure 3.19: An industrial chimney from the industrial dataset.

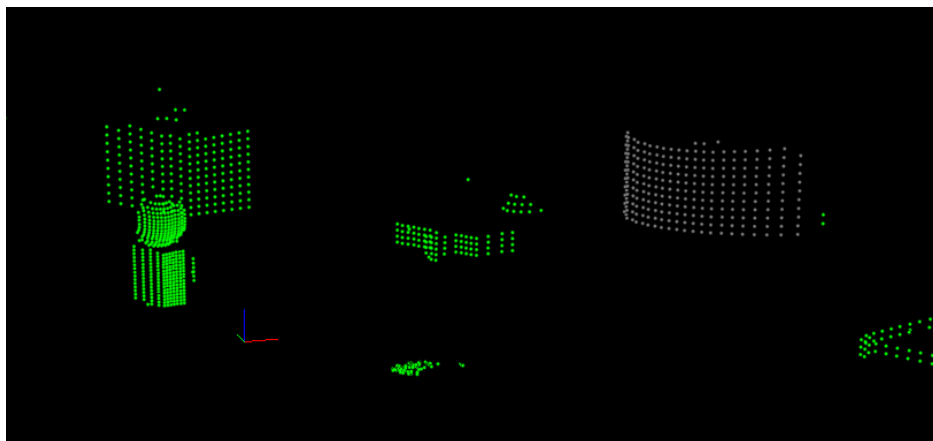


Figure 3.20: A tank from the industrial dataset.

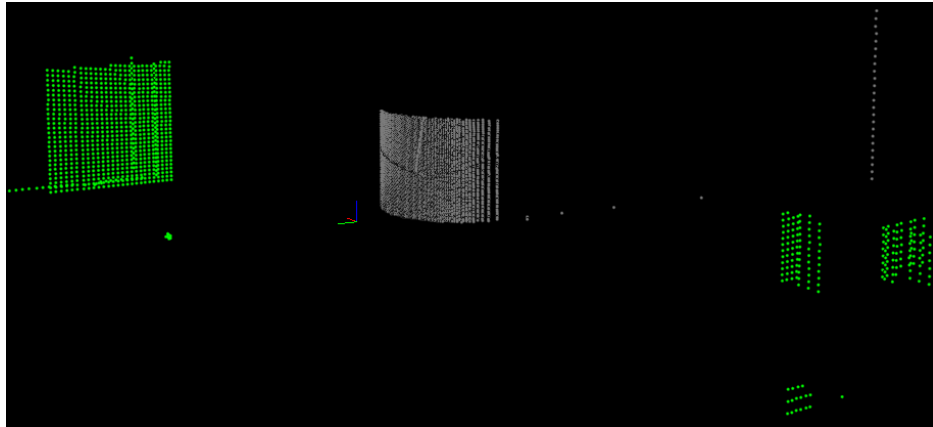


Figure 3.21: Another tank from the industrial dataset.

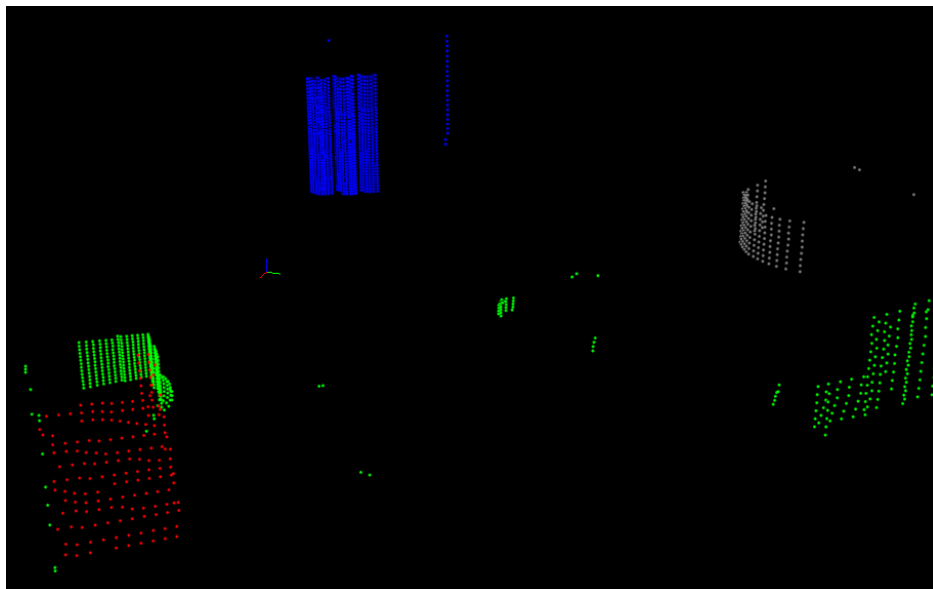


Figure 3.22: Multi-object scan from the industrial dataset.

3.3 Segmentation Models

In order to perform evaluation of the proposed industrial dataset, a segmentation architecture is required. This section provides a short discussion of model considerations made, followed by short presentations of implementation details.

3.3.1 Network considerations

A myriad of different segmentation network architectures have been proposed in the literature, all with a different set of features and applications. The main fac-

tors for selecting a network for this thesis were **accuracy**, **inference speed** and **open-source availability**. Due to the short time-span of this thesis, creating a new implementation from scratch was considered too time-consuming and thus only open-sourced models were considered. The segmentation networks were chosen based on inference times and mIoU as reported on the SemanticKITTI dataset, but only projection-based methods were explored. We briefly presents our reflections behind the reasoning of excluding point-wise methods in the next paragraph.

A case could be made for including point-wise models in the evaluation as some spatial information is inevitably lost in the 2D domain. There are also a number of other shortcomings in projection-based methods, stemming mainly from the data transformation. The projection-based methods use a 3D-2D projection which can cause quantization artifacts, and reprojection errors occur when projecting pixel labels back onto the point cloud. The point-wise methods evades these errors entirely by working directly on the point cloud. Newer projection-based models are able to mitigate these types of errors using the Conditional Random Field (CRF) and kNN methods, but some error will always be present.

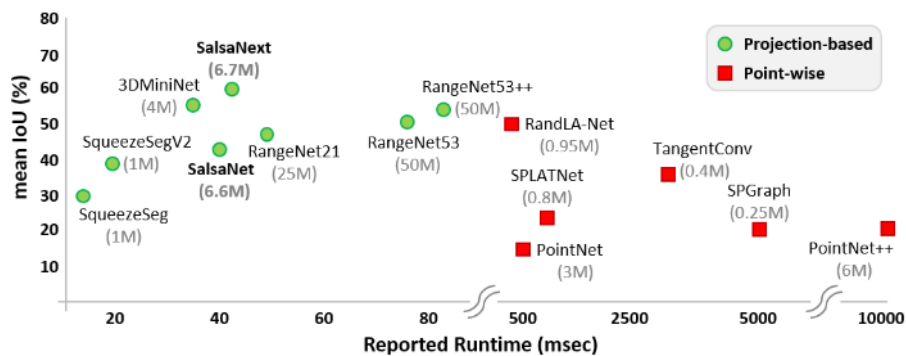


Figure 3.23: Reported runtimes of several SOTA models. Image taken from SalsaNext [28].

However, we argue there are still a number of challenges in point-wise methods. The most glaring fault is the inference times of these models. Figure 3.23 clearly presents the substantial variation in speed for some SOTA point-wise and projection-based methods. Point-wise methods are therefore not suitable for real-time semantic segmentation. In addition, point clouds can be sparse and inefficient and there will be a lot of unused space around objects, particularly for the outdoors and large industrial areas. As such, we consider the computational cost of working directly with LiDAR point clouds in outdoor areas too big. Further research on point-wise methods may be able to reduce the computational cost, perhaps by working on alternative representations such as voxels, surfels or Truncated Signed Distance Field (TSDF).

The two projection-based models **SqueezeSegV3** and **SalsaNext** were selected

due to their efficient point cloud segmentation and impressive accuracy and inference speed on the semanticKITTI dataset. As they are additionally open-sourced, these two networks were chosen for our evaluation studies. Some modifications were done to the networks to be able to train on our generated industrial dataset.

3.3.2 Implementation Details

For the sake of reference and reproducibility, we report implementation details as well as the system specifications used for generating the results in Chapter 4.

Data processing. We pre-process all the points by spherical projection, and the 2D LiDAR images are then input to the models to get a 2D prediction label map. This is then restored back to the 3D space in a post-processing step involving CRF and kNN if this is employed by the model. This post-processing step is not included in the training. During training, the SGD optimizer is employed with a warm-up strategy for the learning rate.

Training. The training pipeline follows the proposed input pipeline by semanticKITTI and RangeNet++, and uses the PyTorch Dataset extension⁶ for readability and modularity of dataset loading. This enables us to easily modify the code for our industrial dataset. RangeNet++ provides a easy-to-use open-source training and evaluation scripts for semantic segmentation, mainly based around PyTorch, OpenCV, and Tensorflow. This forms the basis of the training pipeline.

Multi-GPU implementation. To be able to efficiently train and use two separate GPUs for training and evaluation, data parallelism is employed. Using `torch.nn.DataParallel` the models are wrapped so batches of samples can be split into multiple smaller batches and computed separately on multiple GPUs.

Class-balanced evaluation metrics. The industrial dataset contains around twice as many stairs as tanks and chimneys. A number of domain randomization objects are in each scene that contains only one of the stair/chimney/tank objects. As such, the dataset will contain an uneven number of objects from the different semantic classes. To cope with the imbalanced class problem, both the models incorporate a weighted softmax cross-entropy loss that considers the class frequency.

Custom Loss Functions. While both methods employ the class-balanced metric, their implementation is different. SqueezeSegV3 uses a multi-layer cross-entropy loss which calculates the loss at each of the five layers of the network. SalsaNext uses a combination of weighted cross-entropy loss \mathcal{L}_{wce} and Lovász-Softmax loss \mathcal{L}_{ls} .

More details behind implementation details and custom parameters are spared for the reader and can be found at our open-sourced repositories customized to fit our industrial dataset:

- SqueezeSegV3: <https://github.com/mariusud/SqueezeSegV3>
- SalsaNext: <https://github.com/mariusud/SalsaNext>

⁶https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

ROS Node

For ease of use and inclusion in SLAM pipelines and perception tasks, a ROS node consisting of a projection-based semantic segmentation pipeline was developed. It is an extension on SalsaNext [28] and directly subscribes and converts a lidar topic in ROS. By projecting the point cloud into a 2D spherical image and inferring with a chosen model, we can achieve fast runtime and integration with ROS-based methods and SLAM pipelines. The method reprojects the generated semantic point cloud and publishes to a new topic `"/colored_points"` with inference times as presented in Section 4.6. We use the SalsaNext model, but note that any spherical-projection based model can be used. kNN post processing is also included with reported runtimes.

The ROS Node is open sourced and can be found at:
<https://github.com/mariusud/SalsaNext-ROS>

Hardware

All experiments and training have been completed on a workstation with the following technical specifications:

Processor: AMD Ryzen Threadripper PRO 3975WX 32-Cores

Graphics Card: 3x Nvidia GeForce RTX 3090

Chapter 4

Results

This chapter presents the results of the conducted experiments. We present the results of training both the SalsaNext and SqueezeSegV3 networks on the industrial dataset, as well as the results of varying image sizes on SalsaNext and backbones on SqueezeSegV3. This was done to study the effect of image size and backbones on the overall performance. A final results section of the final chosen model is provided, with a qualitative presentation of visual examples that has been generated by the model. Finally, we present inference times and the ROS Node developed for future SLAM pipelines. Note that color guidelines for the figures follow Table 3.8.

4.1 Experimental Setup

This section presents the experimental setup. The models were trained on a training set with validation and test performed on separate validation/test data. The train/test/validation split was set to 80/10/10, where the training set contains 80% of the data. The validation is done after each epoch, while the test set is used to inform us about performance on new data.

Hyperparameter	Value (SalsaNext / SqueezeSegV3)	Explanation
Epochs	25	Varied between experiments, see plots
Optimizer	SGD	
Learning rate	0.01 / 0.005	SGD learning rate
Warm-up	1	Warmup during first XX epochs
Momentum	0.9	SGD Momentum
Learning rate decay	0.99	Learning rate decay per epoch after initial cycle
Weight decay	0.0001	
Batch size	4	
ϵ	0.001	Class weight $w = \frac{1}{content+\epsilon}$
Workers	12	Number of threads to get data

Table 4.1: Summary of hyperparameters used for training.

4.2 SqueezeSegV3

Figure 4.1 presents the results from the two different backbone configurations we tested for SqueezeSegV3. We followed the proposed backbones by the original paper [25] and test two backbones that are adapted versions of Darknet53 and Darknet21. The two backbones form the respective networks SqueezeSegV3-53 (SSVG3-53) and SqueezeSegV3-21 (SSVG3-21). As can be seen from these figures, the models perform moderately well, and SSVG3-53 narrowly outperforms SSVG3-21. The reported pixelwise accuracy is high, but this is not a good metric for evaluation as sparse images will naturally have higher accuracies.

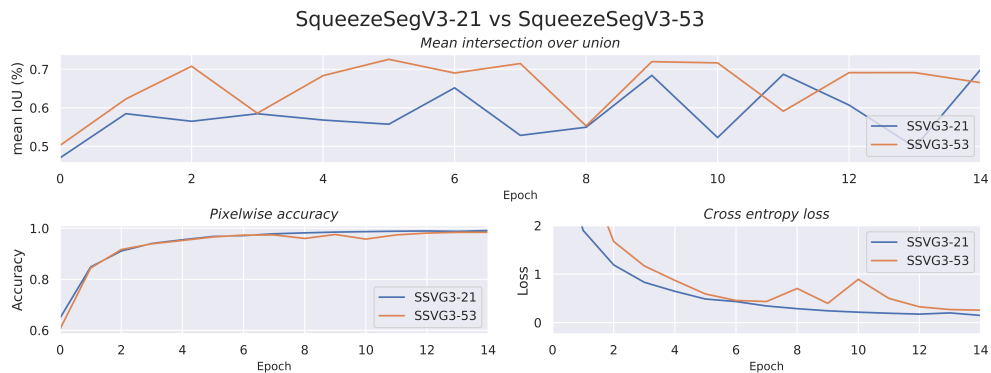


Figure 4.1: Comparison of mIoU, pixelwise accuracy and cross entropy loss for SSVG3-21 and SSVG3-53.

4.3 SalsaNext

Figure 4.2 presents the results of training SalsaNext on the industrial dataset. The time-consuming process of training multiple configurations of deep-learning models and continued hyperparameter tuning was not the focus of this thesis. We therefore only experimented with varying the 2D image sizes for the SalsaNext segmentation network. We chose to focus on image size of the LiDAR image as it influences the inference speed, an important factor for real-time segmentation networks. The larger image size performs slightly better on mean Intersection over Union while the pixelwise accuracy is nearly matching. The cross entropy loss landscape looks similar for both networks.

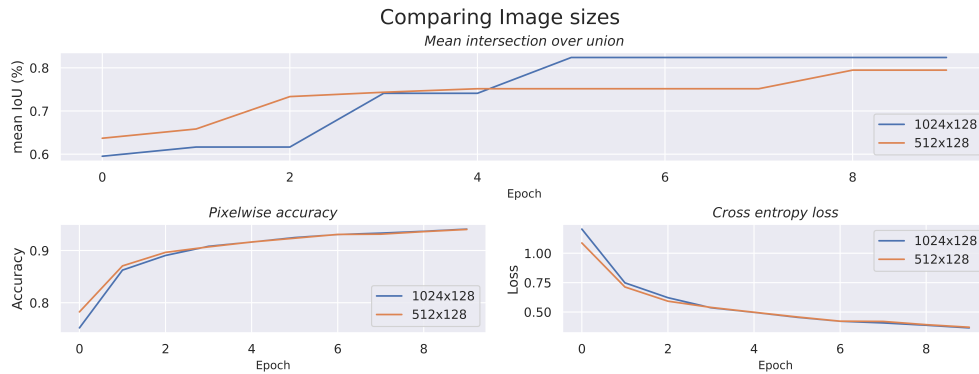


Figure 4.2: Comparison of mIoU, pixelwise accuracy and cross entropy loss for Salsanext with image sizes of 1024x128 and 512x128.

4.4 SqueezeSegV3 and SalsaNext Comparison

To determine a model for final evaluation and further use in a SLAM pipeline in the form of a ROS Node, we compare the two segmentation networks. Figure 4.3 presents the model comparison between SalsaNext and SqueezeSegV3-53. Compared to SqueezeSegV3-53, SalsaNext performs distinctly better on mean IoU but both models perform satisfactorily. Interestingly, SqueezeSegV3 performs better on pixelwise accuracy than SalsaNext and has higher spikes in the loss curve. Table 4.2 presents the final IoU value for both models, and we see that Salsanext clearly outperforms SqueezeSegV3. The final model analysis is therefore conducted using the SalsaNext model.

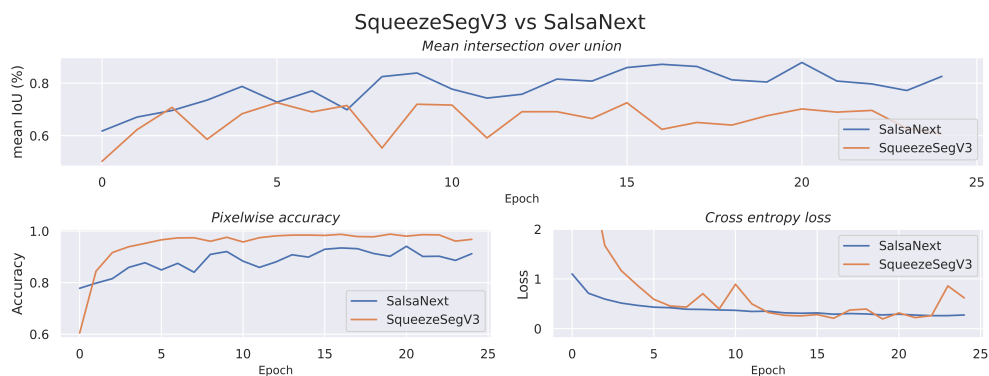


Figure 4.3: Comparison of mIoU, pixelwise accuracy and cross entropy loss for SalsaNext and SqueezeSegV3.

Model	Final IoU (%)
SalsaNext	87.88%
SqueezeSegV3	72.53%

Table 4.2: Comparison of Final IoU between SqueezeSegV3-53 and SalsaNext.

4.5 Final model analysis

For the final model analysis, SalsaNext with image size of 512x128 was selected, as it had negligible accuracy difference and a lower inference speed than the 1024x128 input sized model. This section presents some quantitative and qualitative results of the model over 25 epochs. We first present the quantitative results, with mean Intersection over Union, loss and accuracy over the course of training and at test time. We present the per-class IoU to inspect which of the four classes are most challenging to the network and present some example predictions produced by the trained model for visual inspection and qualitative verification, both in the 2D projection image as well as the backprojected 3D point cloud.

4.5.1 Quantitative Results

Figure 4.4 shows the performance of the model over the course of training. We note that the loss curve is steadily falling in an exponential fashion, and that the mean IoU improves only moderately after around 15 epochs. The pixelwise accuracy has a steady performance improvement over the course of the training.

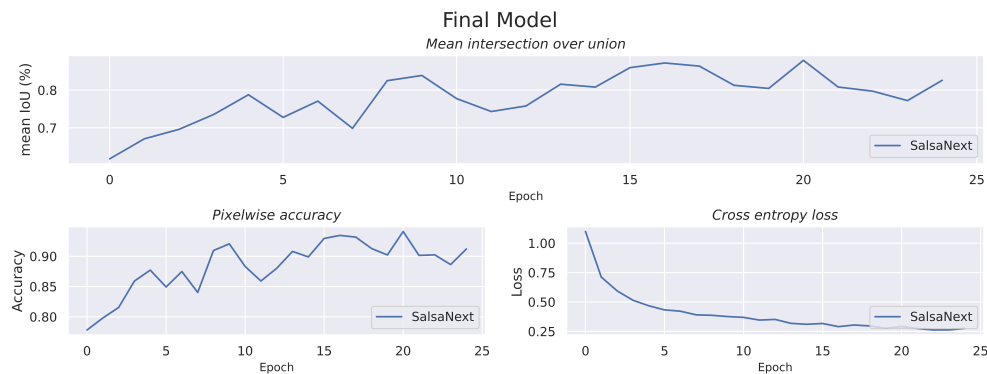


Figure 4.4: mIoU, pixelwise accuracy and cross entropy loss for final model.

To investigate the network’s performance on individual classes, Figure 4.5 shows the individual IoU scores of each class during training. We see that the variation is relatively high between epochs, but that the mean IoU is high for all object categories. No object category stands out significantly even though the dataset is class-imbalanced.

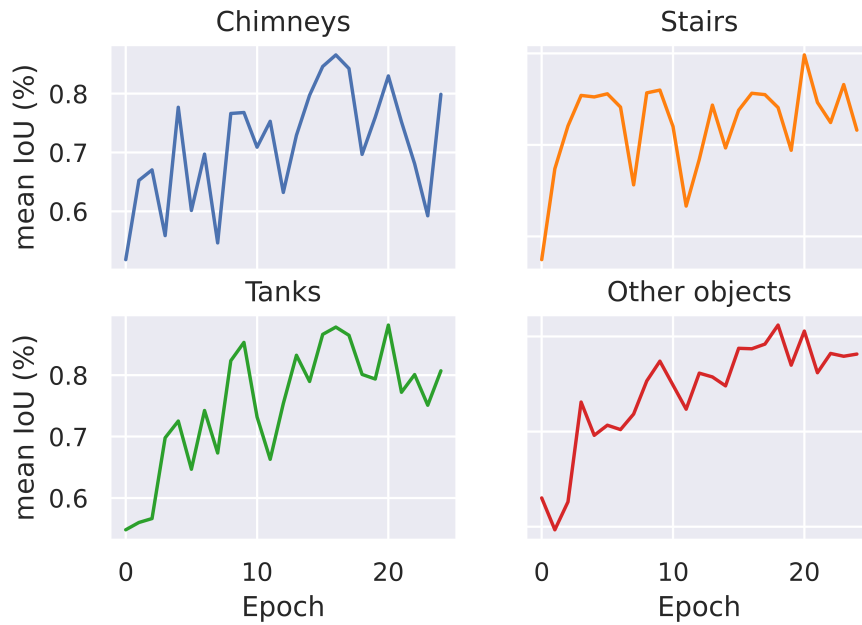


Figure 4.5: Class-wise Intersection over Union (IoU).

4.5.2 Qualitative Analysis

Lastly, we provide images of 2D image segmentation masks, as well as the segmentation masks reprojected back onto the point cloud. This step includes the post-processing methods included in the model, namely the Conditional Random Field (CRF) and k-nearest neighbors (kNN). Figure 4.8 shows two segmentation masks with errors stemming from occlusion. We see that the segmentation model struggles when objects are occluded. When two occluded objects are in close proximity, the intensity channel will not be able to provide enough information for the model to determine that there are in fact two separate objects there. Additional information from RGB channels or similar could provide enough information for the model to distinguish such close-proximity and occluded objects.

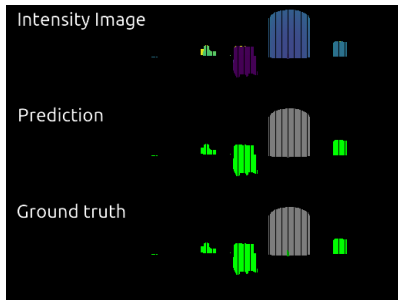


Figure 4.6: A small cone found in the ground in front of the tank (in grey) is predicted incorrectly.

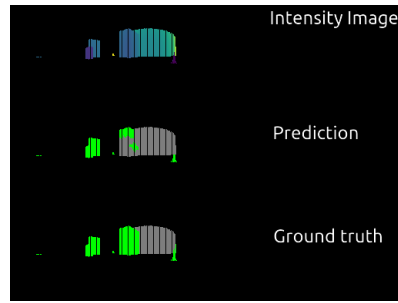


Figure 4.7: An object created by the domain randomization scheme occludes the tank (in grey) and is predicted incorrectly.

Figure 4.8

The visual inspection of the predicted segmentation mask we see in Figure 4.11 and Figure 4.14 shows samples displaying cases where the model performs well. There is enough information in the 2D LiDAR image for the model to separately and correctly predict the stairs, industrial chimney and tanks as well as the domain randomization objects. Figure 4.11 exhibits that occlusion does not always cause a bad prediction, and in this particular example the stairs are correctly predicted and distinguished from the random objects contained in the scene.

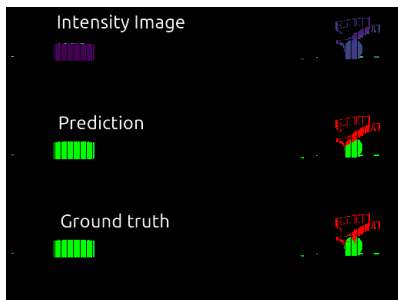


Figure 4.9: A correct prediction mask with occluded stairs (red).

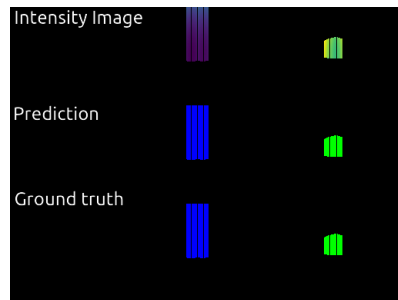


Figure 4.10: A correct prediction mask containing industrial chimneys (blue).

Figure 4.11

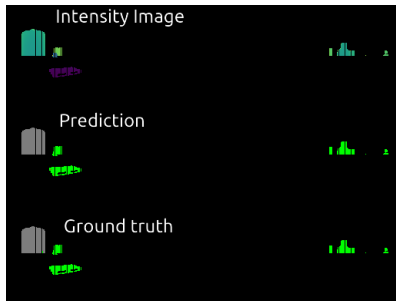


Figure 4.12: A prediction on a tank (gray) with random objects surrounding.

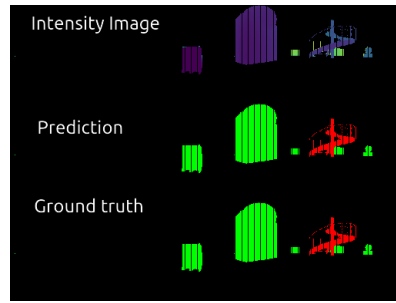


Figure 4.13: A prediction on a staircase with random objects surrounding.

Figure 4.14

The final figures, Figure 4.15 and Figure 4.16 provide an example of what the semantic point cloud will look like. These particular samples are taken from prediction on the multi-object simulations found in the test set, and as such contain multiple labelled objects in the scene. We see that the model performs accurate prediction in 3D on a multi-object dataset it has never seen before, indicating that the model will be able to generalize well to new industrial zones and can be used in a perception pipeline.

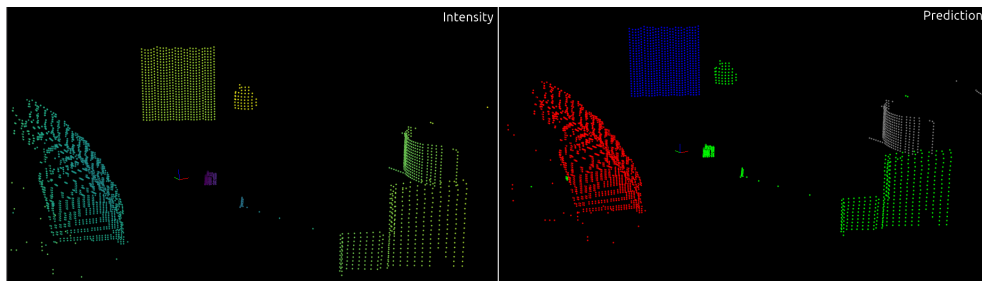


Figure 4.15: Segmented point cloud from sequence 240 of the validation set

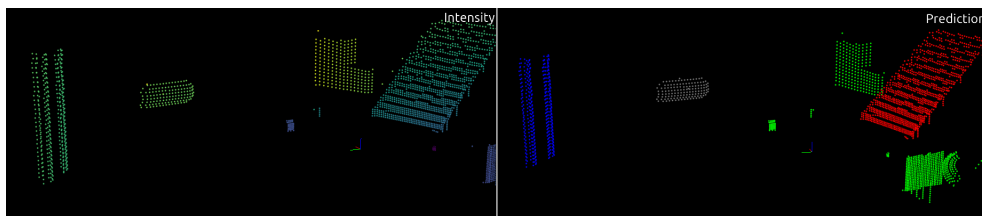


Figure 4.16: Segmented point cloud from sequence 241 of the validation set

4.6 ROS Node

Our final section covers the ROS Node and the results obtained from running the SalsaNext network with this method. The network subscribes to the "/lidar/-points" topic and publishes segmented point clouds to "/colored_points", and can as such easily be included in a SLAM framework. We report the inference time of running the network in a ROS Node, with post-processing steps included. To get a picture of what the network will be able to infer on a lightweight hardware setup, we report the speed on a CPU instead of GPU. Further inference speed improvements could be induced in the future by running the model on a single or multiple GPUs with CUDA. The inference times are presented in Table 4.3, and we deduce that the model is able to perform real-time semantic segmentation of point clouds on synthetic data.

Parameter	Value
Avg Network inference	~ 0.74sec
Avg kNN inference	~ 0.24sec
CPU Processor	Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz

Table 4.3: Reported inference times of SalsaNext-ROS on a CPU.

Chapter 5

Discussion and Further Work

The thesis is wrapped up with a summary of the experiments and results obtained on the industrial dataset. Following this, a discussion of further work and extensions to the SSDG method and the industrial dataset are proposed. We shortly comment on the choice of segmentation networks, before we present a discussion on further work within semantic autonomy.

5.1 Summary

Motivated by the lack of industrial semantic point cloud datasets, the goal of this thesis was to investigate the usage of a synthetic annotated segmentation pipeline for generating new segmentation datasets. The novel synthetic dataset generator (SSDG) was proposed, utilizing the ROS and Gazebo frameworks. An industrial dataset was created using the method, and evaluated using two real-time segmentation networks. All experiments performed in this thesis were evaluated on the industrial dataset. As our results in Chapter 4 indicate, the SSDG method can successfully be used to generate a large high-quality dataset based on 3D CAD objects that can be used to train segmentation networks.

Some effort was spent on minimizing the discrepancy between the simulation and real-world domain. We attempted to diminish the reality gap by as closely possible emulating a simulation resembling reality, and additionally implemented a domain randomization scheme to achieve robustness to high variability in the environment. Some data augmentation was also included in the dataset creation.

To be able to provide real-time segmentation and a higher level understanding for Simultaneous Localization and Mapping (SLAM) algorithms, we chose two efficient segmentation networks, SqueezeSegV3 and SalsaNext, to conduct experiments on the industrial dataset. We experimented with varying the backbone of SqueezeSegV3 and input image size for SalsaNext. A final qualitative analysis of SalsaNext was presented, to give the reader an impression of the quality of segmentation.

To include the trained model in a future SLAM pipeline and generally provide an easy-to-use semantic network, we developed a ROS Node built on SalsaNext.

This gave us a general indication of the inference time that can be expected for projection-based methods.

5.2 Synthetic Semantic Dataset Generator (SSDG)

This section will discuss the SSDG methodology, a brand new framework we presented for building synthetic semantic datasets. In order to generate datasets that produces robust, generalizable and domain-adaptive segmentation networks in the future, we present a discussion on the validity of the proposed method, before proposing new directions for future works.

5.2.1 Reality gap

One of the main focuses of the thesis has been to emulate the real world as best possible. In addition to focusing on perfecting the emulation of the physical world, we explore methods to train the models in more generic domains by introducing variance in different forms. We note that while some newer researcher focuses on the effects on 2D image segmentation [35] or real-world urban point clouds [42], the applicability to the synthetic 3D domain remains undeveloped. Yi *et al.* [43] presents a novel method within the 3D domain adaptation field, but attempts to combat the real-world LiDAR domain discrepancies between different LiDAR sensors instead of the synthetic-real domain discrepancy we focus on. By recovering the underlying 3D surface and filling the point cloud from two different LiDARs with more points, the proposed approach is able to increase transferability between the LiDAR sensors used by the different datasets the experiments were conducted on. A natural question to ask is whether a similar approach could be applied to decrease the domain gap between synthetic and real-world data by supplying the real-world data with more points from a completion network. We leave it as a task for further work to test the validity of this approach on synthetic SSDG-generated and real-world data from an Ouster OS0-128.

Our domain randomization scheme was based around *Applying Domain Randomization to Synthetic Data for Object Category Detection* [35]. The paper presents a randomization scheme for object category detection based on the Gazebo framework, and note that a study on the applicability of domain randomization on segmentation should be conducted. Our method SSDG does exactly this, and applies domain randomization on the segmentation task in the same framework. However, we note that an in-depth study of the effects of the domain randomization scheme we designed was not conducted due to time constraints. We propose a study to be conducted in order to quantify the robustness our method induces to the model, and verify its suitability for real-world 3D semantic segmentation tasks in the future.

5.2.2 Prospects for future research

The constrained time frame of the project limited the scope of the Synthetic Semantic Dataset Generator (SSDG). In this section, we summarize some of the work that time did not permit, and present directions for further research. We believe that future work should focus on two main focus areas: domain discrepancy and generalizability.

Domain discrepancy We believe more work should be carried out for assessing the discrepancy between real-world data and the synthetically generated data presented in Section 3.2. The evaluation should consider the differences in point cloud density, class distributions and point intensities, amongst other variables. If there exists distribution discrepancies between the synthetic and real data, the trained model will in most cases fail to generalize to real data. Domain calibration similar to [23] will help improve the model if this is the case. Extensive experiments should be able to uncover discrepancies and should form the basis of future method improvements and novelties. A key question that remains unanswered due to lack of real data, is to what extent there will be a distribution discrepancy between the real data and the synthetically generated LiDAR data generated by SSDG.

Generalizability of the SSDG method. To verify the generalization ability of networks trained on synthetic data, they should be tested on a real world dataset such as semanticKITTI. Note however that semanticKITTI contains no shared labels with the industrial dataset we created, and our models therefore cannot be validated on it. It is therefore difficult to conclude on the generalizability of our models in a real-world setting as there to-date exists no real-world industrial segmentation dataset we can use to validate it. We have presented the first steps towards a general goal of providing robust semantic autonomy in industrial use-cases, but remark that there is still some work to be done. Particularly, a real-world dataset should be constructed for validation purposes. This will allow for validation and confirmation of the dependability of our method and give a broader view of the transferability of the trained models to a real-world setting. We therefore propose that a real-world industrial dataset should be constructed for future research and employed as a validation of our method.

5.3 Industrial Dataset

The constrained time frame of this project affected several key decisions for the industrial dataset. In this section, we summarize some of the work that we would have liked to improve given the time, and directions to further develop semantic autonomy for industrial areas.

Quality of dataset. We argue that the quality of the dataset is good, but with room for improvement. We note that there exists fragments in the scans, likely due to bordering issues in the matching of the 3D-2D projected point cloud and label map. This induces artificial artifacts and fragments to the data, but means

the simulation is in actual fact closer to what a real LiDAR would produce as real data naturally contains artifacts and dropout noise caused by the lidar sensor. Nonetheless, this should be investigated further and either retained in a controlled fashion, or eliminated from the method. Apart from these fragments, the data looks convincing from a qualitative standpoint, and the induced sensor noise gives the data a realistic look. We argue that our method presents an adequate way of producing realistic synthetic semantic data.

Generalizability of models trained on the industrial dataset. We now discuss the generalizability of models trained specifically on the industrial dataset, rather than the generalizability of the SSDG method discussed in the previous section. We expect the generalizability of models trained on the industrial dataset will be mediocre on real-world data. The reason for this is two-fold. Some fault should be contributed to the SSDG method, but as discussed earlier this cannot be validated due to the lack of real data to compare against. It is therefore difficult to reason on the generalizability of the SSDG method. The domain randomization scheme should increase the generalizability somewhat, but in the context of shapes in industrial facilities, we believe generalization ability will in large be attributed to the likeness between the simulated and real-world objects. Due to the limited time of this thesis, very simple models were created of the industrial tanks and chimneys. These models will not likely be close to what these objects look like in a real-life industrial zone, but to some extent follow the general shape. For instance, while industrial chimneys generally are understood as tall circular objects, there can be a lot of smaller features embedded in their form that can be captured by a segmentation network. The industrial chimneys the author created in SolidWorks were very simple shapes, and could consist of just a single cylindrical shape. The extent of the industrial dataset is therefore very small. This was due to time limitations, but our dataset is nonetheless a solid step in the future goal of achieving industrial point cloud segmentation. For future works, we propose generating high-quality 3D CAD models. We believe the availability of high-quality industrial objects may already exist at a large scale due to the recent developments in digital twinning, but that they are not open-sourced. The usage of these models in our SSDG framework will undoubtedly greatly increase the applicability and generalization ability of the generated segmentation model. In particular, if a digital twin exists of an industrial facility, the domain discrepancy between the simulation and real-world will be drastically reduced. A segmentation network trained on a digital twin of an oil refinery will undoubtedly perform very well in segmenting the same oil refinery in the real-world domain.

5.4 Segmentation Networks

In this section, an analysis of the segmentation models is performed along with a discussion on their general architecture. The results in Chapter 4 form the basis for the discussion in this section.

For the SqueezeSegV3 models, the deeper layered SqueezeSegV3-53 has a

higher accuracy than SqueezeSegV3-21. This can be considered expected behaviour, as generally more layers favours better accuracy at the loss of computational speed. The pixelwise accuracy is higher for SqueezeSegV3 than SalsaNext despite lower mIoU, and this will no doubt stem from the fact that SqueezeSegV3 does not compute a form of IoU in its metric. The pixelwise accuracy is not a good evaluation metric for deciding 2D segmentation precision, as large parts of the image will be black. We therefore have used miou as our evaluation metric of choice. The increased mIoU in the SalsaNext network likely stems from the inclusion of *Lovász-Softmax* in the loss function as it provides a method of including IoU in the loss, directly affecting the network. Future works could try to include the Lovász-Softmax in a SqueezeSegV3 network in an attempt to improve the mIoU of the model. This could perhaps also reduce the spiking in the loss curve perceived on the SqueezeSegV3 network, since the model may be able to learn better features once IoU is incorporated in the loss function.

5.5 Further work in semantic autonomy

We wrap up the chapter by discussing prospects for further research in semantic segmentation autonomy, and whether LiDAR-only semantic segmentation can be considered satisfactory or multiple sensing modalities should be fused together. Previously, it has been shown that multi-modal systems are successful in perception tasks, providing resilient and robust solutions such as [44]. While the semantic segmentation task has predominantly been solved using RGB cameras in the past, the task of transporting it to the 3-dimensional domain has been introduced in line with the increased availability of 3D sensors such as RGB-D cameras and LiDAR sensors.

Camera-LiDAR fusion. Compared with point clouds, camera images have richer semantic information and contain more regular and dense pixels and considering the complementary of the two sensors, their fusion is a desired option to explore further. Two main issues hinder the performance of such a fusion: how to precisely align the camera image and point cloud in a spatiotemporal domain, and how to effectively fuse them together. LIF-Seg [45] proposed in 2021, present a coarse-to-fine fusion-based network that uses an offset rectification approach to align the two-modality features. The method successfully achieves better performance than projection-based methods such as SalsaNext on the nuScenes dataset, and considering that SalsaNext was our top performer on the industrial dataset, we regard it as likely that the LIF-Seg network would improve the benchmark on our dataset. Particularly for instances like Figure 4.8 the added RGB channels will provide valuable information for the network, as it is easy to discern a bright orange cone from a dark-colored tank.

Extending SSDG with RGB data. Considering that cameras are readily available in the Gazebo simulation, we propose further work using SSDG incorporate RGB cameras into the method, to be able to generate datasets for multi-modal pipelines. As many real-world resilient autonomy pipelines already contain both

sensing modalities, further work within the generation of synthetic semantic data should involve both sensors to enable the fusion of the two modalities in real-world settings.

Symbiotic relationship between robots and digital twins. We end this discussion by noting that semantic segmentation networks are used for both digital twinning and semantic autonomy in the industrial domain, but interestingly no literature can be found on bridging the gap between the two tasks. One could envision further work focusing on using an autonomous robot running a semantics-based autonomy pipeline that is able to work in a symbiotic relationship with a digital twin of the industrial facility in which it is operating. In this way, both the digital twin model and the autonomous robot benefit from their shared semantic understanding of the facility.

Chapter 6

Conclusion

The goal of this thesis was to design and implement a framework that can be used to create semantic datasets from 3D CAD models, and use the method to create an industrial dataset for applications within robotic autonomy. The Synthetic Semantic Dataset Generator (SSDG) method was presented, a system capable of generating annotated point clouds from a Gazebo simulation. To close the reality gap, we introduced a novel domain adaptation approach designed to overcome the domain gap specifically in a semantic 3-Dimensional point cloud setting. We show that by introducing a randomized set of 3D objects from parametric shapes, we are able to introduce *non-essential* variation to the task, increasing the robustness of the trained segmentation models.

By making a set of industrial 3D CAD models by hand, we generated a large 3D semantic point cloud industrial dataset using SSDG, containing 3.7 million points from four semantic categories. Extensive experiments with two segmentation networks demonstrated that state-of-the-art models achieve good accuracy on the dataset, and the pre-trained models could be used for semantic segmentation of industrial facilities. Detection quality was overall good with a reported accuracy of 87.88% for the best model.

3D semantic segmentation has a broad range of use in industrial settings, as it can improve digital twinning processes and increase resiliency and robustness of SLAM algorithms by introducing a semantic dimension. Our method SSDG fills a gaping hole in the literature, as industrial scene data are scarcely available to the public, and few existing method exists for automating the creating of synthetic labelled point cloud datasets. Seeing as 3D models are available in large quantities online, our method will allow for the development of large-scale datasets for a broad range of settings. SSDG opens the door for the development of more advanced methods in new areas, but we also provide plentiful data to investigate research within industrial point cloud segmentation. We hope that this thesis provides a foundation of which future semantic autonomy can be built upon, and note that the work is developed with the Autonomous Robots lab, who will it as a foundation for a future SLAM pipeline.

Bibliography

- [1] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences,” in *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.
- [2] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss, “Suma++: Efficient lidar-based semantic SLAM,” *CoRR*, vol. abs/2105.11320, 2021. arXiv: 2105.11320. [Online]. Available: <https://arxiv.org/abs/2105.11320>.
- [3] L. Li, X. Kong, X. Zhao, W. Li, F. Wen, H. Zhang, and Y. Liu, “SA-LOAM: semantic-aided lidar SLAM with loop closure,” *CoRR*, vol. abs/2106.11516, 2021. arXiv: 2106.11516. [Online]. Available: <https://arxiv.org/abs/2106.11516>.
- [4] R. Dubé, A. Cramariuc, D. Dugas, H. Sommer, M. Dymczyk, J. Nieto, R. Siegwart, and C. Cadena, “Segmap: Segment-based mapping and localization using data-driven descriptors,” *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 339–355, 2020. DOI: 10.1177/0278364919863090.
- [5] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao, “3d shapenets for 2.5d object recognition and next-best-view prediction,” *CoRR*, vol. abs/1406.5670, 2014. arXiv: 1406.5670. [Online]. Available: <http://arxiv.org/abs/1406.5670>.
- [6] E. Shelhamer, J. Long, and T. Darrell, *Fully convolutional networks for semantic segmentation*, 2016. DOI: 10.48550/ARXIV.1605.06211. [Online]. Available: <https://arxiv.org/abs/1605.06211>.
- [7] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–44, May 2015. DOI: 10.1038/nature14539.
- [8] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077.
- [9] A. L. Maas, “Rectifier nonlinearities improve neural network acoustic models,” 2013.

- [10] M. Berman and M. B. Blaschko, “Optimization of the jaccard index for image segmentation with the lovász hinge,” *CoRR*, vol. abs/1705.08790, 2017. arXiv: 1705.08790. [Online]. Available: <http://arxiv.org/abs/1705.08790>.
- [11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, Dec. 2019.
- [13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: <https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>.
- [14] R. Al-Rfou, G. Alain, A. Almahairi, *et al.*, “Theano: A python framework for fast computation of mathematical expressions,” *CoRR*, vol. abs/1605.02688, 2016. arXiv: 1605.02688. [Online]. Available: <http://arxiv.org/abs/1605.02688>.
- [15] F. Chollet *et al.* “Keras.” (2015), [Online]. Available: <https://github.com/fchollet/keras>.
- [16] NVIDIA, P. Vingelmann, and F. H. Fitzek, *Cuda, release: 10.2.89*, 2020. [Online]. Available: <https://developer.nvidia.com/cuda-toolkit>.
- [17] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, “Cudnn: Efficient primitives for deep learning,” *CoRR*, vol. abs/1410.0759, 2014. arXiv: 1410.0759. [Online]. Available: <http://arxiv.org/abs/1410.0759>.
- [18] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.
- [19] B. Wu, A. Wan, X. Yue, and K. Keutzer, *Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud*, 2017. DOI: 10.48550/ARXIV.1710.07368. [Online]. Available: <https://arxiv.org/abs/1710.07368>.

- [20] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and lt;0.5mb model size*, 2016. DOI: 10.48550/ARXIV.1602.07360. [Online]. Available: <https://arxiv.org/abs/1602.07360>.
- [21] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," 2012. DOI: 10.48550/ARXIV.1210.5644. [Online]. Available: <https://arxiv.org/abs/1210.5644>.
- [22] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr, "Conditional random fields as recurrent neural networks," *CoRR*, vol. abs/1502.03240, 2015. arXiv: 1502.03240. [Online]. Available: <http://arxiv.org/abs/1502.03240>.
- [23] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer, *Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud*, 2018. DOI: 10.48550/ARXIV.1809.08495. [Online]. Available: <https://arxiv.org/abs/1809.08495>.
- [24] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "RangeNet++: Fast and Accurate LiDAR Semantic Segmentation," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.
- [25] C. Xu, B. Wu, Z. Wang, W. Zhan, P. Vajda, K. Keutzer, and M. Tomizuka, "Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation," in *European Conference on Computer Vision*, Springer, 2020, pp. 1–19.
- [26] E. E. Aksoy, S. Baci, and S. Cavdar, "Salsanet: Fast road and vehicle segmentation in lidar point clouds for autonomous driving," *CoRR*, vol. abs/1909.08291, 2019. arXiv: 1909.08291. [Online]. Available: <http://arxiv.org/abs/1909.08291>.
- [27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016. DOI: 10.1109/cvpr.2016.90. [Online]. Available: <http://dx.doi.org/10.1109/cvpr.2016.90>.
- [28] T. Cortinhal, G. Tzelepis, and E. E. Aksoy, "Salsanext: Fast semantic segmentation of lidar point clouds for autonomous driving," *CoRR*, vol. abs/2003.03653, 2020. arXiv: 2003.03653. [Online]. Available: <https://arxiv.org/abs/2003.03653>.
- [29] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, *Pointnet++: Deep hierarchical feature learning on point sets in a metric space*, 2017. arXiv: 1706.02413 [cs.CV].
- [30] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, *Randla-net: Efficient semantic segmentation of large-scale point clouds*, 2019. DOI: 10.48550/ARXIV.1911.11236. [Online]. Available: <https://arxiv.org/abs/1911.11236>.

- [31] X. Yue, B. Wu, S. A. Seshia, K. Keutzer, and A. L. Sangiovanni-Vincentelli, “A lidar point cloud generator: From a virtual world to autonomous driving,” in *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, ser. ICMR ’18, Yokohama, Japan: Association for Computing Machinery, 2018, pp. 458–464, ISBN: 9781450350464. DOI: 10.1145/3206025.3206080. [Online]. Available: <https://doi.org/10.1145/3206025.3206080>.
- [32] Open Robotics, *Ignition gazebo*, version Ignition Fortress, Sep. 2021. [Online]. Available: <https://ignitionrobotics.org>.
- [33] Open Robotics, *Robotic operating system*, version ROS Noetic Ninjemys, May 23, 2020. [Online]. Available: <https://www.ros.org>.
- [34] Open Source Robotics Foundation, *Sdformat*, version 12.0.0, Sep. 30, 2021. [Online]. Available: <http://sdformat.org/>.
- [35] J. Borrego, R. Figueiredo, A. Dehban, P. Moreno, A. Bernardino, and J. Santos-Victor, “A generic visual perception domain randomisation framework for gazebo,” in *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Apr. 2018, pp. 237–242. DOI: 10.1109/ICARSC.2018.8374189.
- [36] adlarkin. “Construction cone label test,” Open Robotics. (Jun. 2021), [Online]. Available: <https://fuel.gazebosim.org/1.0/adlarkin/models/Construction%20Cone%20Label%20Test>.
- [37] MovAi. “Pallet_{box}mobile,” Open Robotics. (Mar. 2022), [Online]. Available: https://fuel.gazebosim.org/1.0/MovAi/models/pallet_box_mobile.
- [38] MovAi. “Pallet,” Open Robotics. (Mar. 2022), [Online]. Available: <https://fuel.gazebosim.org/1.0/MovAi/models/pallet>.
- [39] OpenRobotics. “Electrical box,” Open Robotics. (Mar. 2021), [Online]. Available: <https://fuel.gazebosim.org/1.0/OpenRobotics/models/Electrical%20Box>.
- [40] X. Roynard, J. Deschaud, and F. Goulette, “Paris-lille-3d: A large and high-quality ground truth urban point cloud dataset for automatic segmentation and classification,” *CoRR*, vol. abs/1712.00032, 2017. arXiv: 1712.00032. [Online]. Available: <http://arxiv.org/abs/1712.00032>.
- [41] D. Munoz, J. A. (Bagnell, N. Vandapel, and M. Hebert, “Contextual classification with functional max-margin markov networks,” in *Proceedings of (CVPR) Computer Vision and Pattern Recognition*, Jun. 2009, pp. 975–982.
- [42] Y. Zhang, P. David, and B. Gong, “Curriculum domain adaptation for semantic segmentation of urban scenes,” *CoRR*, vol. abs/1707.09465, 2017. arXiv: 1707.09465. [Online]. Available: <http://arxiv.org/abs/1707.09465>.

- [43] L. Yi, B. Gong, and T. Funkhouser, “Complete label: A domain adaptation approach to semantic segmentation of lidar point clouds,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 15 363–15 373.
- [44] M. Tranzatto, F. Mascarich, L. Bernreiter, C. Godinho, M. Camurri, S. Khat-tak, T. Dang, V. Reijgwart, J. Loeje, D. Wisth, S. Zimmermann, H. Nguyen, M. Fehr, L. Solanka, R. Buchanan, M. Bjelonic, N. Khedekar, M. Valceschini, F. Jenelten, M. Dharmadhikari, T. Homberger, P. D. Petris, L. Wellhausen, M. Kulkarni, T. Miki, S. Hirsch, M. Montenegro, C. Papachristos, F. Tresoldi, J. Carius, G. Valsecchi, J. Lee, K. Meyer, X. Wu, J. I. Nieto, A. Smith, M. Hut-ter, R. Siegwart, M. W. Mueller, M. F. Fallon, and K. Alexis, “CERBERUS: autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the DARPA subterranean challenge,” *CoRR*, vol. abs/2201.07067, 2022. arXiv: 2201.07067. [Online]. Available: <https://arxiv.org/abs/2201.07067>.
- [45] L. Zhao, H. Zhou, X. Zhu, X. Song, H. Li, and W. Tao, “Lif-seg: Lidar and cam-era image fusion for 3d lidar semantic segmentation,” *CoRR*, vol. abs/2108.07511, 2021. arXiv: 2108.07511. [Online]. Available: <https://arxiv.org/abs/2108.07511>.

