Automated Bridge Inspection for Concrete Surface Defect Detection Using Deep Neural Network Based on LiDAR Scanning

Majid Nasrollahi

A Thesis

in

The Department

of

Building, Civil, and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Building Engineering) at

Concordia University

Montreal, Quebec, Canada

December 2019

© Majid Nasrollahi, 2019

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Majid Nasrollahi

Entitled: Automated Bridge Inspection for Concrete Surface Defect Detection Using Deep Neural Network Based on LiDAR Scanning

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Building Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining committee:

Dr. Sang Hyeok Han Chair

Dr. Ashutosh Bagchi Examiner

Dr. Farnoosh Naderkhani Examiner

Dr. Amin Hammad Supervisor

Approved by Dr. Michelle Nokken, Graduate Program Director

December 2019 Dr. Amir Asif, Dean of Gina Cody School of Engineering and Computer Science

ABSTRACT

Automated Bridge Inspection for Concrete Surface Defect Detection Using Deep Neural Network Based on LiDAR Scanning

Majid Nasrollahi

Structural inspection and maintenance of bridges are essential to improve the safety and sustainability of the infrastructure systems. Visual inspection using non-equipped eyes is the principal method of detecting surface defects of bridges, which is time-consuming, unsafe, and encounters inspectors falling risks. Therefore, there is a need for automated bridge inspection. Recently, Light Detection and Ranging (LiDAR) scanners are used for detecting surface defects. LiDAR scanners can collect high-quality 3D point cloud datasets. In order to automate the process of structural inspection, it is important to collect proper datasets and use an efficient approach to analyze them and find the defects. Deep Neural Networks (DNNs) have been recently used for detecting 3D objects within 3D point clouds. PointNet and PointNet++ are deep neural networks for classification, part segmentation, and semantic segmentation of point clouds that are modified and adapted in this work to detect surface concrete defects. The research contributions are: (1) Designing a LiDAR-equipped UAV platform for structural inspection using an affordable 2D scanner for data collection, and (2) Proposing a method for detecting concrete surface defects using deep neural networks based on LiDAR generated point clouds. Training and testing datasets are collected from four concrete bridges in Montréal and annotated manually. The point cloud dataset prepared in five areas, which contain more than 51 million points and 2,572 annotated defects in four classes of crack, light spalling, medium spalling, and severe spalling. The accuracies of 75% (adapted PointNet) and 79% (adapted PointNet++) in detecting defects are achieved in binary semantic segmentation.

ACKNOWLEDGMENT

My greatest appreciation goes to my supervisor, Dr. Amin Hammad for his intellectual and personal support, encouragement and patience. His guidance, advice, and criticism were my most valuable asset during my studies. Overall, I feel very fortunate having the opportunity to know him and work with him.

I would like to thank Mr. Raymond Bruton and Mr. Dmitry Rozhdestvenskiy for their help in the hardware integration for designing the LiDAR-equipped UAV platform. I would, also, like to thank my colleague, Mr. Yusheng Huang, for his help in piloting the UAV for the outdoor test of the platform.

Furthermore, I would like to use this opportunity to thank my colleagues, Ms. Neshat Bolourian and Mr. Fardin Bahraini, for their help in data collection and manual annotation.

A heartfelt thank you goes to my wife, Mahsa, my mother, Azam, and my dear brother, Hamid. They supported me with their encouragement, love, and support.

DEDICATION

I would like to dedicate this thesis to my loving wife, Mahsa, for her endless support, many encouragements, and unconditional love that made it all possible.

I would also like to dedicate this thesis to my dear mother, Azam, and my dear brother, Hamid. They always believed in me, even when I did not believe in myself. They are always in my thoughts.

TABLE OF CONTENTS

LIST OF FIGURES ix
LIST OF TABLES xi
LIST OF ABBREVIATIONS xii
CHAPTER 1 Introduction 1
1.1 General Information 1
1.2 Problem Statement 2
1.3 Research Objectives 2
1.4 Thesis Organization 2
CHAPTER 2 Literature Review 4
2.1 Introduction 4
2.2 Bridge Structural Inspection 4
2.3 LiDAR Scanning Technology 7
2.3.1 SLAM 8
2.3.2 LOAM 8
2.4 Using UAV for Data Collection 8
2.5 Deep Neural Networks and Applications in Semantic Segmentation 9
2.5.1 Convolutional Neural Networks 10
2.5.2 Object detection in 3D point cloud dataset 14
2.5.3 PointNet 15
2.5.4 PointNet++ 16
2.5.5 Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) 17
2.6 Summary 17
CHAPTER 3 Designing LiDAR-equipped UAV Platform for Structural Inspection 18
3.1 Introduction 18
3.2 Design Considerations 20
3.2.1 Objectives 20
3.2.2 Requirements 20
3.2.3 Constraints 22
3.2.4 Other Considerations 23
3.3 Platform Design 23
3.3.1 Integration 25
3.4 Case Study 31

3.4.1 Indoor Test 31
3.4.2 Outdoor Test 33
3.5 Summary and Conclusions 36
CHAPTER 4 Concrete Surface Defect Detection Using Deep Neural Network Based on LiDAR Scanning 37
4.1 Introduction 37
4.2 Methodology 38
4.2.1 Data Collection 39
4.2.2 Annotation 40
4.2.3 Data Pre-Processing 40
4.2.4 Training and evaluation 41
4.2.5 Testing 44
4.3 Case study 44
4.3.1 Data Collection 45
4.3.2 Annotation 50
4.3.3 Data Pre-Processing 52
4.3.4 Validation of Binary Classes Semantic Segmentation 54
4.3.5 Validation on Detecting Types of Defects Using Adapted PointNet++ Model 81
4.3.6 Validation on Detecting Severity Levels of Defects Using Adapted PointNet++ Model 86
4.4 Summary and Conclusions 90
CHAPTER 5 Summary, Conclusions and Future Work 91
5.1 Summary of Research 91
5.2 Research Conclusion and Contributions 91
5.3 Limitations and Future Work 93
REFERENCES 94
Appendices 99
Appendix A – Robot Operating System (ROS) 99
A.1 ROS workspace (catkin workspace) 105
A.2 Hector Slam 105
A.3 Hokuyo node 106
Appendix B - Hector Slam Example 106
B.1 Creating a Map Using Hokuyo UTM-30LX 109
B.2 Recording the Data 112
B.3 Mapping a Hallway Hsing Hokuyo 113

B.4 Converting bag Files to Point Cloud 115
Appendix C – Compute Canada........ 118
C.1 Submitting a Job 119
C.1 Installing TensorFlow........ 120
C.2 SBATCH Command Options 123
Appendix D – Python Code for Flipping the Dataset 125

LIST OF FIGURES

Figure 2-1 ReLu activation function (Goodfellow et al., 2016) 12
Figure 2-2 The effect of the momentum optimization method in learning (Goodfellow et al., 2016) 13
Figure 2-3 PointNet (Qi et al. 2017a) 16
Figure 4-1. Scanner position on top of UAV in (a) and (b), and under UAV in (c) 21
Figure 4-2 Some specifications of the LiDAR 23
Figure 4-3 Design of interfacing parts for 3D printing, (a) Part for holding the servo, (b) Part for holding the scanner, (c) Part for attaching the servo vertically, and (d) Part for attaching the servo inclined 26
Figure 4-4 Hardware integration 27
Figure 4-5 LiDAR-equipped UAV platform 29
Figure 4-6 The initial test's results: (a) The generated point cloud by the designed platform, (b) A segment of the point cloud, (c) The reference segment using FARO, (d) The comparison distance distribution histogram (m), and (e) The compared segment colored based on distances from the reference segment 33
Figure 4-7 The location of the outdoor test in Montreal, QC, Canada 34
Figure 4-8 Outdoor test 34
Figure 4-9 The outdoor test's results: (a) The overlapped point clouds of the designed platform, and FARO, (b) A segment of the overlapped point clouds, and (c) The comparison distance distribution histogram (m) 35
Figure 5-1. The proposed method 39
Figure 5-2 Location of the bridges in, Montreal, QC, Canada: (1) Guy Street, (2) Lucian L'Allier Street, (3) Atwater Avenue, and (4) Pierre-Dupuy Avenue 47
Figure 5-3 Scanning positions on the western side of Bridge 1 (Guy Street) 48
Figure 5-4 Scanning positions on the eastern side of Bridge 1 (Guy Street) 49
Figure 5-5 (a) An original point cloud sample, (b) An annotated point cloud sample. No-defect is yellow, crack is blue, light spalling is pink, medium spalling is green, severe spalling is red 50
Figure 5-6 The structure of preparing the dataset 51
Figure 5-7 (a) Sample of annotated original points segment, (b) Flipped segment 53
Figure 5-8 Evaluation mean loss of the models 58
Figure 5-9 Evaluation accuracies of defect class, no-defect class, and overall 58
Figure 5-10. Modified PointNet network 59
Figure 5-11 The chart of output results of the detecting defects (Adapted PointNet). 63

Figure 5-12 Adapted PointNet++ network.. 67
Figure 5-13 The chart of output results of the detecting defects (Adapted PointNet++)............................ 69
Figure 5-14 Accuracies of the studied cases based on block sizes 77
Figure 5-15 Accuracies of the studied cases based on the number of points in each block........................ 77
Figure 5-16 Accuracy of detecting defects compared to the sizes of defects 79
Figure 5-17 Effect of changing density on the accuracy of defects 80
Figure 5-18 Number of points of annotated classes... 82
Figure 5-19 Chart of the output results of the testing on detecting types of defects (Adapted PointNet++)
.... 85
Figure 5-20 Number of points of the defect classes.. 87
Figure 5-21 Chart of the output results of the testing on detecting types and severity levels of defects
(Adapted PointNet++)... 89

LIST OF TABLES

Table 2-1 Scaling severity levels based on the depth of loss of surface mortar (OSIM, 2008) 5

Table 2-2 Delamination severity levels based on the dimensions of the delaminated area (OSIM, 2008) .. 5

Table 2-3 Spalling severity levels based on the dimensions of the spalled area and depth (OSIM, 2008)... 6

Table 2-4 Cracking severity levels based on the width of the crack (OSIM, 2008). 6

Table 4-1 UAV specifications 24

Table 4-2 Scanner specifications 24

Table 4-3 Summary table of hardware components 28

Table 5-1 Defects severity based on the depth of loss (d), width and height of the affected area (w, h) (Ministry of Transportation 2008) 38

Table 5-2 FARO Focus3D LiDAR scanner specifications (Faro 2012) 46

Table 5-3 Scanning information 46

Table 5-4 Statistics datasets used in training, evaluation, and testing processes 52

Table 5-5 Details on the trained and evaluated models for optimization.. 57

Table 5-6 PointNet model's hyperparameters 60

Table 5-7 Results of training and evaluation of the selected model 61

Table 5-8 The output results of the detecting defects (Adapted PointNet).. 62

Table 5-9 Results of training and evaluation of the selected model 64

Table 5-10 PointNet++ model's hyperparameters 65

Table 5-11. The output results of the detecting defects (Adapted PointNet++). 68

Table 5-12 Binary semantic segmentation results 70

Table 5-13 Density ratio of the raw dataset 75

Table 5-14 Annotated defects dimensions 75

Table 5-15 Results of the sixteen studied cases 76

Table 5-16 Accuracy of defect detection based on the depth of defects 79

Table 5-17 Cost weight of the tree classes 82

Table 5-18 Results of training and evaluation on detecting types of defects (adapted PointNet++) 83

Table 5-19 Output results of the testing on detecting types of defects (Adapted PointNet++) 84

Table 5-20 Cost weight of the four classes 87

Table 5-21 Results of training and evaluation on detecting severity levels of defects (PointNet++) 87

Table 5-22 Output results of the testing on detecting types and severity levels of defects (Adapted PointNet++) 88

LIST OF ABBREVIATIONS

Abbreviation	Description
2D	Two-Dimensional
3D	Three-Dimensional
4D	Four-Dimensional
ADAM	Adaptive Momentum
CAR	Canadian Aviation Regulations
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
DOF	Degrees of Freedom
FOV	Field of View
GPS	Global Position Systems
GPU	Graphical Processing Unit
HDF	Hierarchical Data Format
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
LOAM	LiDAR Odometry and Mapping
MLP	Multi-Layer Perceptron
OSIM	Ontario Structure Inspection Manual
PCD	Point Cloud Data
PCL	Point Cloud Library

PS	Phase Shift
ReLu	Rectified Linear unit
RGB	Red Green Blue
ROS	Robot Operating System
S3DIS	Stanford Large-Scale 3D Indoor Spaces Dataset
SFOC	Special Flight Operation Certification
SGD	Stochastic Gradient Descent
SLAM	Simultaneous Localization and Mapping
TLS	Terrestrial Laser Scanning
TOF	Time-of-Flight
UAV	Unmanned Aerial Vehicle
VO	Visual Odometry

CHAPTER 1 Introduction

1.1 General Information

Efficient inspection and maintenance of bridges and other structures are vital for improving the safety and sustainability of infrastructure systems, such as bridges. Traditionally, visual inspection using non-equipped eyes and manual measurements are used for detecting surface defects, which may lead to subjective results. This approach is time-consuming and unsafe, especially for inspecting the inaccessible elements of a bridge (Kim et al., 2014; Guldur et al., 2015).

Recently, 3D Light Detection and Ranging (LiDAR) scanners (Liu et al., 2011) and cameras (Adhikari et al., 2014) are used for detecting surface defects (e.g. cracks) using computer vision methods. LiDAR scanners can collect high-quality 3D point cloud datasets. In order to automate the process of structural inspection, it is important to collect proper datasets and use an efficient approach to analyze them and find the defects. The LiDAR scanner can be mounted on a tripod (Kim et al., 2014) (i.e. terrestrial scanning) or on an Unmanned Aerial Vehicle (UAV) (Freimuth et al., 2017; Yoder & Sebastian, 2016) (i.e. aerial scanning). Although terrestrial scanning provides high stability for the scanner and less vibration, it is not time efficient. The aerial scanning provides easier access to most parts of the structure.

Deep Neural Networks (DNNs) have been recently used for detecting 3D objects within 3D point clouds. In order to automate the process of structural inspection, a DNN is used to detect defects in the point clouds of concrete bridges. In this research, PointNet (Qi et al. 2017a) and

PointNet++ (Qi et al. 2017b) are adapted to detect surface defects using point cloud datasets from scanning bridge surfaces.

1.2 Problem Statement

Visual inspection using non-equipped eyes is the principal method of inspecting surface defects of bridges. This approach is time-consuming, unsafe, and encounters inspectors falling risks in the cases of inspecting the inaccessible elements of bridges. Therefore, there is a need for an automated data collection for bridge inspection. In addition, an automated method for detecting bridge's surface defects from the collected datasets is inevitable to fulfill the requirements of having a unified framework.

1.3 Research Objectives

The main objectives of this research are defined as:

1) Designing a platform for LiDAR-equipped UAV for structural inspection using an affordable 2D scanner.
2) Developing a proper 3D annotated dataset of bridge's surface defects for analyzing.
3) Developing a method for detecting concrete surface defects using a Deep Neural Network (DNN) based on LiDAR scanning.

1.4 Thesis Organization

This research is organized as follows:

CHAPTER 2 Literature Review: In this chapter, the new methods in bridge inspection and the current stage of using LiDARs and DNNs are discussed.

CHAPTER 3 Designing LiDAR-equipped UAV Platform for Structural Inspection: This chapter explains the design process, requirements and other design considerations of a LiDAR-equipped UAV platform for structural inspection using an affordable 2D scanner. The initial test results of the platform are also discussed in this chapter.

CHAPTER 4 Concrete Surface Defect Detection Using Deep Neural Network Based on LiDAR Scanning: This chapter proposes a method for detecting surface defects of concrete bridges using LiDAR generated point clouds and DNN. The process of collecting and annotating the datasets and the promising initial results are explained in this chapter.

CHAPTER 5 Summary, Conclusions and Future Work: This chapter summarizes the present research and concludes the findings. Moreover, the limitations of the current study are investigated followed by the suggestions for future work.

CHAPTER 2 Literature Review

2.1 Introduction

This chapter presents the literature review of various subjects, including bridge inspection methods, LiDAR scanning technologies, using UAVs for data collection and Deep Neural Networks and their applications in point cloud semantic segmentation. First, the state of the bridge structural inspection is explained, and then the available related state of the art technologies are studied. The technologies and methods that are discussed in this chapter support this research's ideas.

2.2 Bridge Structural Inspection

The traditional visual inspection is the most common method of inspecting structural infrastructures, such as bridges. For visual inspection, it may be necessary to stop the traffic on the bridge fully or partially to allow the inspection to access different parts of the bridge. This process is unsafe, time-consuming, expensive and subjective to human errors (Guldur et al., 2015). Manual inspection is dangerous, especially in the case of inspecting the hard-to-access elements of infrastructure (e.g. underneath the surface of a bridge superstructure) (Guldur et al., 2015).

Using the collected data of discrete sensors attached to a bridge is another way of assessing the health of a structure. Guldur et al. (2015) explained that the collected data cannot represent the behavior of all the parts of a structure due to discrete locations of sensors. Also, the sensors' accuracy and method of data collection have challenges.

Defects related to reinforced concrete bridge elements are spalling, exposed rebars, rust staining, cracking, distortions, and settlements (Koch et al., 2016). Much of the research in defect detection for reinforced concrete bridges have focused on cracks (Koch et al., 2015). The size of the bridge's defect is a vital factor in deciding if it is necessary to go further than the visual method (Koch et al., 2015).

Regular concrete surface defects on bridges are cracks, spalling, scaling, and delamination. Scaling is the flaking or peeling of the surface of the concrete as a result of exposure to freezing and thawing (Ontario Structure Inspection Manual (OSIM), 2008). Table 2-1 classifies the scaling defect severities based on the depth of loss of the surface of the concrete. Delamination is separating the surface layer of concrete, which is not completely detached (Table 2-2). Continuing of delamination until detaching causes spalling and spall is a detached part of a concrete mass (OSIM, 2008) (Table 2-3). Fracture in concrete happens as a result of tensile stresses in the concrete member (OSIM, 2008). The severity of cracks is classified based on the width of the cracks (Table 2-4).

Table 2-1 Scaling severity levels based on the depth of loss of surface mortar (OSIM, 2008)

Severity	**Depth (d) of loss of surface mortar**
Light	$d \leq 5mm$
Medium	$6mm \leq d \leq 10mm$
Severe	$11mm \leq d \leq 20mm$
Very Severe	$20mm < d$

Table 2-2 Delamination severity levels based on the dimensions of the delaminated area (OSIM, 2008)

Severity	**Dimensions of delaminated area (a, b)**
Light	$a, b < 150mm$
Medium	$150mm \leq a, b < 300mm$
Severe	$300mm \leq a, b < 600mm$
Very Severe	$600mm \leq a, b$

Table 2-3 Spalling severity levels based on the dimensions of the spalled area and depth (OSIM, 2008)

Severity	Dimensions of palled area (a, b)	Depth (d)
Light	a, b < 150mm	d < 25mm
Medium	150mm ≤ a, b < 300mm	25mm ≤ d < 50mm
Severe	300mm ≤ a, b < 600mm	50mm ≤ d < 100mm
Very Severe	600mm ≤ a, b	100mm ≤ d

Table 2-4 Cracking severity levels based on the width of the crack (OSIM, 2008).

Severity	Wide (w)
Hairline cracks	w < 0.1 mm
Narrow cracks	0.1 mm ≤ w < 0.3 mm
Medium cracks	0.3 mm ≤ w < 1.0 mm
Wide cracks	1.0 mm ≤ w

There are two main approaches to detect concert surface defects. The first is based on geometry analysis and the second is based on artificial intelligence methods. In the geometry analysis approach, volume losses are calculated using Gaussian curvature distribution (Teza et al., 2009) and crossing section method (Olsen et al., 2009). Armesto-Gonzalez et al. (2010) proposed an automated classification algorithm in order to detect moisture-based defects. Girardeau-Montaut et al. (2005) detected volume changes in excavation using an octree-based comparison between average and Hausdorff distance. Liu et al. (2011) presented an algorithm to detect defects based on distance and gradient criteria of concrete bridge surface. Tsai et al. (2012) detected cracks in asphalt paving based on dynamic optimization and linear buffered Hausdorff scoring. Laefer et al. (2014) contributed a mathematical basis for using Terrestrial Laser Scanning (TLS) to detect cracks in unit-block masonry (i.e. stone, brick, or concrete masonry units). A new and automated technique that can simultaneously localize and quantify spalling defects on the concrete surface using TLS was proposed by Kim et al. (2014). Guldur et al. (2015) developed a strategy to define an appropriate threshold considering Red, Green, and Blue (RGB) color values of point clouds or intensity values to detect defect objects. Once the objects are detected their geometrical features are extracted from point clouds. The geometrical features of point clouds, which are the most valuable information of point clouds, are not used in the object detection process of their work. The artificial intelligence methods are discussed in Section 2.5.

2.3 LiDAR Scanning Technology

Light detection and ranging (LiDAR) scanning technology is a remote sensing method to measure distances. The two main approaches to detecting distances that a LiDAR can use are time-of-flight technique and phase-based technique. In the time-of-flight technique, a LiDAR emits a light beam with a known traveling speed. It measures the distance by multiplying the time spent by the light pulse to reach a rigid surface, reflect, and come back to the scanner, by the traveling velocity of the pulse. In phase-shift technology, a LiDAR emits a light beam and by comparing the reflected beam phase with the reference phase, it measures the distance from the scanner to the pointed surface (Liu et al., 2011).

The output of a LiDAR is point cloud, a 3D dataset that contains geometrical information of the sparse captured points from a 3D space. The geometrical information (x, y, z) of points is calculated based on the measured distances of the scanned objects or surfaces from the LiDAR's location. To generate point clouds while a scanner is moving in an environment, it is important to have continuous LiDAR's location.

2.3.1 SLAM

For scanning in GPS-denied environments, localizing the LiDAR’s position is an important issue. A solution of simultaneous localization and mapping (SLAM) problem helps overcome this issue. It gives the ability to an autonomous system to start from an unknown position in an unknown environment, map the environment by scanning, and simultaneously use the generated map to localize itself. A SLAM solution converges the relative observed maps and finds the error in the locations of the landmarks. The landmarks are considered static, so the convergence error is about the scanner’s movement and it is possible to localize the scanner simultaneously (Dissanayake et al., 2001).

2.3.2 LOAM

LiDAR odometry and mapping (LOAM) is a real-time solution to localize a 2-axis moving LiDAR. LOAM tries to solve the SLAM problem in a 3D environment by using two algorithms. One algorithm performs odometry to evaluate the LiDAR’s velocity and the other algorithm registers the scanned 2D maps to generate 3D point clouds (Zhang & Singh, 2014).

2.4 Using UAV for Data Collection

The LiDAR scanner can be mounted on a tripod (i.e. terrestrial scanning) (Kim et al., 2014) or on an Unmanned Aerial Vehicle (UAV) (i.e. aerial scanning) (Freimuth et al., 2016). Terrestrial scanning provides high stability for the scanner and less vibration, but it is not time efficient to scan large areas using terrestrial scanners from various points of view. The aerial scanning provides easier access to most parts of a large structure and can fly close to the structure.

Consequently, higher coverage of the inspected surfaces and more accurate results can be achieved.

In addition, a LiDAR-equipped UAV eliminates the inspectors' falling risks encountered in the traditional inspection method. The risk of damages caused by the UAV is low because of its size, weight, and controllability (Zink & Barritt, 2015). LiDAR-equipped UAVs are used in different applications such as surveying (Wallace et al., 2012), inspection (Guldur et al., 2015), navigation (Bachrach et al., 2011), and agriculture (Honkavaara et al., 2012). There are two types of mobile LiDAR scanners: two dimensional (2D) and three dimensional (3D). A 2D scanner is more affordable, but it can only scan points on a plane, while a 3D scanner can capture the point cloud of the surrounding space, which makes the data more accurate. However, a 2D scanner can be transferred into a 3D scanner by rotating the scanner using a servo motor (Zhang & Singh, 2014) or by moving the scanner on a robot/UAV while collecting the point cloud (Winkvist & Rushforth, 2013).

2.5 Deep Neural Networks and Applications in Semantic Segmentation

Deep neural networks, also known as multi-layer perceptron's (mlps), or deep feedforward networks, are deep learning models with the ultimate goal of approximating a function f based on the input data for the use cases (e.g. classifying data). A deep neural network tries to map $y=f(x;\theta)$ by learning the value of the parameters θ from the best function approximation f over the input values of x. A deep neural network is created by combining many functions as links of a chain, and the length of the chain is the depth of the network and the links are the hidden layers of the network (Goodfellow et al., 2016).

An epoch is a process of going through all the training samples once. The training samples are distributed in mini-batches and training over a mini-batch is a step (iteration). So, the number of steps of an epoch is equal to the number of all the data samples (data size) divided by the number of data samples in a mini-batch (batch size).

2.5.1 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a class of deep forward networks, which is inspired by biological processes in the connectivity patterns between neurons (LeCun & Bengio, 1995). A CNN contains input, convolutional, subsampling, and output layers. The input layer receives size-normalized and centered images. Each layer receives inputs by means of local receptive fields from the previous layer. Using local receptive fields, neurons extract elementary geometric features such as edges, boundaries, and corners. Local receptive fields are known as filters or kernels. A simple feature detector, which is suitable on a part of an input image, is expected to be useful for the whole image. By convolving a kernel using a unique weight vector over the entire image, a feature map would be generated. Typically, a convolutional layer generates several feature maps in order to extract several features from the input image (LeCun & Bengio, 1995). Each feature map has an identical weight vector, which will improve during the backpropagation process.

In the traditional neural networks, matrix multiplication is used on the whole input data in one step to create an output unit, but a CNN uses a kernel, which is smaller than the input, and it causes fewer operations (Goodfellow et al., 2016). In the layers of a CNN, a weight vector is used repeatedly while computing the output of a layer, this process is parameter sharing. It does not affect the forward propagation time but has large effects on memory requirements and efficiency of the model (Goodfellow et al., 2016). The idea of parameter sharing minimizes the number of free parameters and it directly improves the generalization ability of a model (LeCun, 1989). Convolutional is dramatically more efficient than the traditional neural networks in the volume of computations (Goodfellow et al., 2016).

The quality of data plays an important role in finding a proper fit function for any neural network. Therefore, collecting adequate datasets is necessary to achieve an accurate model. The datasets should represent the appropriate parameters and include different cases based on the requirements.

The process of multiplying a weight vector to the input data by convolving a kernel and creating linear activations in a feature map is the first stage of a convolutional neural network (Goodfellow et al., 2016). Nonlinear activation and subsampling feature map by applying pooling functions are the other two stages.

2.5.1.1 Activation functions

As mentioned above, a deep neural network is a combination of functions derived from the hidden layers. Equation 2-1 refers to a simple network with two hidden layers. If the first derived function (f (1)) is a linear function, then the whole model would remain linear to the input x. However, in most cases, the answer of a problem is not a linear predictive function (Goodfellow et al., 2016).

$$\mathbf{y} = \mathbf{f}(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x})) \qquad \textbf{2-1}$$

Therefore, a nonlinear function has to help the model describe the features of the input data. This nonlinear function is called the activation function. Rectified linear unit (ReLu) function is the most common activation function used in deep neural networks (Figure 2-1). ReLu is a nonlinear function that is so close to linear (containing two linear pieces). It is helpful for both linear and nonlinear models (Goodfellow et al., 2016).

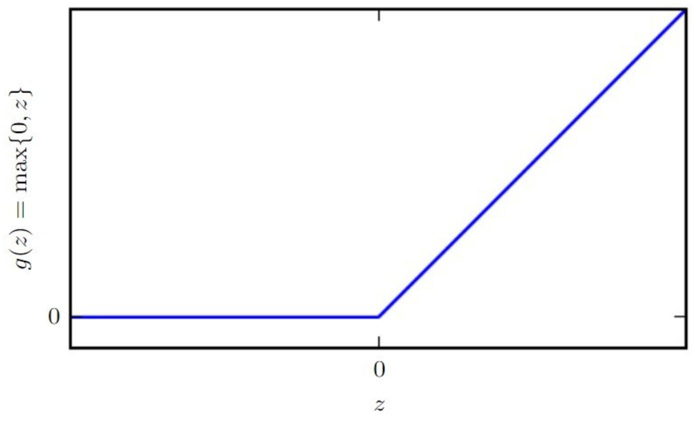


Figure 2-1 ReLu activation function (Goodfellow et al., 2016)

2.5.1.2 Pooling

The pooling process summarizes the elements of a feature map over a specified neighborhood (e.g. 2 by 2 or 3 by 3). Pooling (subsampling) makes a model invariance with small geometric transformations or distortions of objected features in the input (LeCun & Bengio, 1995), which significantly improves the efficiency of the network (Goodfellow et al., 2016). Max pooling and mean pooling are two popular pooling functions. Max pooling extracts the maximum element of a rectangular neighborhood of feature map elements; and means pooling computes the mean of the elements in the neighborhood (Goodfellow et al., 2016). Scherer et al. (2010) evaluated that max-pooling works better than mean pooling in CNN for object detection.

2.5.1.3 Optimization

A deep neural network needs an optimization algorithm to train the predictor weight vectors. In order to reach the best approximation of function f in $y=f(x;\ \theta)$ (Section 2.5), a cost function $J(\theta)$ is defined based on measuring the performance of the function f in each iteration. An optimization algorithm is responsible for finding the parameters θ, which significantly reduces the value of cost function $J(\theta)$ (Goodfellow et al., 2016).

Optimization algorithms that process all the training examples at the same time are named *batch gradient* methods and optimization algorithms that use only a part of the dataset at a time are *minibatch stochastic* methods. The stochastic gradient descent (SGD) method is the most basic and regular method of optimization. SGD works by setting an initial parameter θ, computes gradient estimate of minibatch of m, and updates parameter θ using a learning rate parameter in each step. The learning rate is the most important parameter in SGD. Momentum (Polyak, 1964) is another optimization method, which defined a new parameter of momentum (α) to accelerate the process of learning by exponentially decaying α over steps (Figure 2-2). Adam (adaptive moments) (Kingma & Ba, 2014) is an adaptive learning rate optimization algorithm. It has the features of a momentum optimizer plus a decay rate (ρ) parameter to adapt the learning rate parameter over the training steps (Goodfellow et al., 2016; Polyak, 1964).

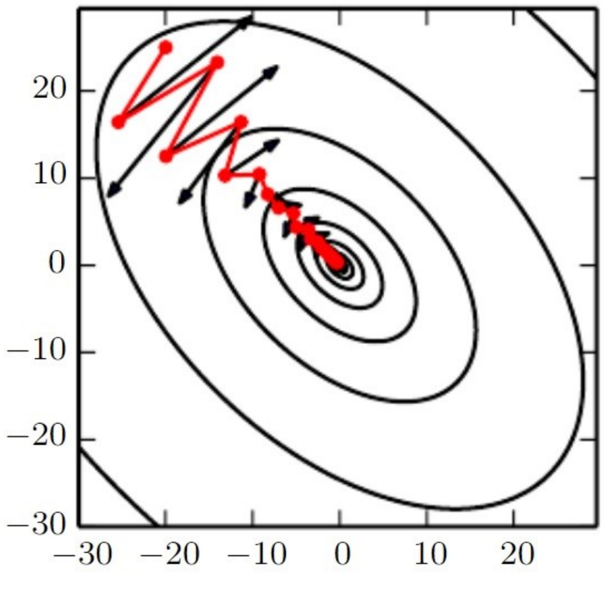


Figure 2-2 The effect of the momentum optimization method in learning (Goodfellow et al., 2016)

2.5.1.4 Batch normalization

In processing DNNs, every hidden layer takes input data from the output of the previously hidden layer and applies its own computation to output a vector for feeding the next sequential layer. In this process, some internal covariant shift appears in the data and makes the distribution of nonlinearity unstable for different layers (Ioffe & Szegedy, 2015). Each layer in the sequential system of DNNs is not aware of other previous layers, but the previous one. Normalizing the values of parameters, by initializing to zero means and unit variance, smooth the process of learning in a DNN (Ruder, 2016).

Batch normalization is the process of normalizing parameters in each mini-batch (Ioffe & Szegedy, 2015). It affects the activator's output and by normalizing the distribution, preserves the representation abilities.

2.5.1.5 Dropout

Overfitting is one of the serious issues in large deep neural networks. Dropout (Srivastava et al., 2014) addresses this issue by randomly dropping units from the network during the training process. This technique avoids the model from co-adopting too much to the training data. It helps the model overcome the issue of overfitting and improve the generalization performance of the model.

2.5.2 Object detection in 3D point cloud dataset

A point cloud is a 3D dataset that contains geometric information of sparse points captured from a 3D space. Point cloud datasets may also contain RGB and density information. Based on the 3D data representation, CNN approaches are classified into three main groups: pixel-based approaches, voxel-based approaches, and 3D point-based approaches. In pixel-based approaches (Su et al. 2015), 3D data are converted to 2D projection. Voxels are generated from the 3D points (Brock et al. 2016). In 3D point-based approaches, Qi et al. (2016) presented the very first novel approach to point cloud processing using 3D CNN and applying 3D recognition tasks containing object classification, part segmentation, and semantic segmentation.

Due to the popularity of the pixel- and voxel-based methods, 3D point cloud datasets are typically transformed into images or 3D-voxel grids in order to be used in deep learning. The transformation results in voluminous data with unclear invariances in some cases. Moreover, learning from point cloud sets is easier than meshes due to their simplicity and unified structure. Meshes are complex and have combinatorial irregularities (Qi et al. 2017a).

2.5.3 PointNet

PointNet is a state of the art CNN model for point cloud analysis which can be applied in classification, part segmentation, and semantic segmentation directly based on 4-Dimensional (4D) tensor data (Figure 2-3). This network was proposed in 2017 in order to overcome the problems related to voxelization and rendering point clouds (Qi et al. 2017a). The input of PointNet is a subset of points (point clouds), which has three main characteristics. First, these points are unordered. Interaction among neighbored points is the second important characteristic of these datasets. The points are not isolated and the local structure of the combination of neighboring points affects the semantic information of the point sets. The third characteristic of point clouds is being invariant under transformation. These three characteristics are considered in designing the architecture of PointNet. Qi et al. (2017a) used Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) to validate the semantic scene segmentation part of the PointNet algorithm (Armeni et al. 2016).

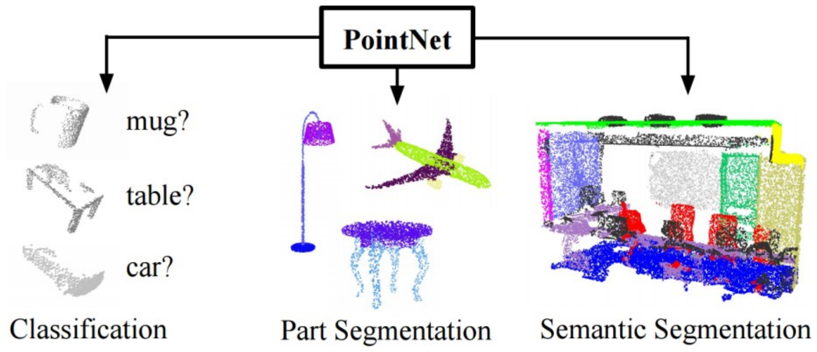


Figure 2-3 PointNet (Qi et al. 2017a)

2.5.4 PointNet++

PointNet normalizes the number of points in pre-defined geometrical blocks in order to feed the neural network. Points have various densities in different parts of a point cloud; Although this process may affect the segmentation process and loses potential information. PointNet++ feeds the neural network with a combination of non-uniform density samples of the input point clouds. To reach the ideal goal of capturing the fine details of every single class in point sets, it is necessary to look at the smallest possible portions of point sets. However, in the low-density parts of the point clouds, the small portions do not give useful information. Therefore, looking at larger portions is necessary too. PointNet++ extracts local features from small portions of the point set and group them together in larger portions and continues this process to reach the local features of the whole point set. This multi-scale grouping method is very effective and important for semantic segmentation with multiple labels, and especially for segmenting objects with small sizes (Qi et al., 2017b).

2.5.5 Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS)

Armeni et al. (2016) proposed a new method in parsing large-scale point clouds and prepared Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS). S3DIS covers 6 large-scale indoor areas from three different buildings, which are collected using RGB-D technology with Matterport Camera.

S3DIS used a canonical coordinate system for all the spaces to facilitate utilizing the recurrent structure of the indoor spaces in the detected point cloud of the building. In the canonical coordinate system, (Z) axis is aligned to the gravitational axis, (X) axis is aligned along the entrance surface of the room, and (Y) axis is aligned perpendicular to the entrance wall (Armeni, et al., 2016).

2.6 Summary

This chapter reviewed the concepts, methods, and technologies that are used in the current research. Based on the literature, the LiDAR and UAV technologies can solve the data collection issues for bridge inspection, especially for the inaccessible elements of bridges. The integration of these two technologies aims to define an automated method for the bridge's data collection. Moreover, point cloud analysis using deep learning methods can semantically segment LiDAR generated point clouds, which is the core method of this research for concrete surface defect detection.

CHAPTER 3 Designing LiDAR-equipped UAV Platform for Structural Inspection

3.1 Introduction

Efficient inspection and maintenance of bridges and other structures are vital for improving the safety and sustainability of infrastructure systems. Traditionally, visual inspection using non-equipped eyes and manual measurements are used for detecting surface defects, which may lead to subjective results. This approach is time-consuming and unsafe, especially for inspecting the inaccessible elements of a bridge (Kim et al., 2014).

Recently, 3D Light Detection and Ranging (LiDAR) scanners (Liu et al., 2011) and cameras (Adhikari et al., 2014) are used for detecting surface defects (e.g. cracks) using computer vision methods. In general, LiDAR technology is more expensive than cameras, and defect detection results may miss some edge points. However, it is a promising method, not only for detecting the location and size of the defects but also for computing their depth and volume (Teza et al., 2009; Olsen et al., 2009). Also, unlike digital images, the generated point clouds are not affected by lighting, and their analysis does not require supplementary information (Laefer et al., 2014).

The LiDAR scanner can be mounted on a tripod (Kim et al., 2014) (i.e. terrestrial scanning) or on an Unmanned Aerial Vehicle (UAV) (Freimuth et al., 2017; Yoder & Sebastian, 2016) (i.e. aerial scanning). Although terrestrial scanning provides high stability for the scanner and less vibration, it is not time efficient. The aerial scanning provides easier access to most parts of the structure and can fly close to the structure. Consequently, higher coverage of the inspected surfaces and more accurate results can be achieved.

In addition, the LiDAR-equipped UAV eliminates the inspectors' falling risks encountered in the traditional inspection method. The risk of damages caused by the UAV is low because of its size, weight, and controllability (Zink & Barritt, 2015). The Special Flight Operation Certification (SFOC), required by the Canadian Aviation Regulations, includes the plan of operation respecting specific safety rules, such as the distance between the operators and UAV, keeping people away from the flight site, and flying the UAV in the Line of View (LoV) (Canadian Aviation Regulations (CARs) and standards, 2017).

LiDAR-equipped UAVs are used in different applications such as surveying (Wallace et al., 2012), inspection (Guldur et al., 2015), navigation (Bachrach et al., 2011), and agriculture (Honkavaara et al., 2012). There are two types of mobile LiDAR scanners: two dimensional (2D) and three dimensional (3D). A 2D scanner is more affordable, but it can only scan points on a plane, while a 3D scanner can capture the point cloud of the surrounding space, which makes the data more accurate. However, a 2D scanner can be transferred into a 3D scanner by rotating the scanner using a servo motor (Zhang & Singh, 2014) or by moving the scanner on a robot/UAV while collecting the point cloud (Winkvist & Rushforth, 2013). The accurate rotational positions of the servo or the Simultaneous Localization And Mapping (SLAM) algorithm can be used to register the collected data. For example, in the research of Winkvist et al. (Winkvist & Rushforth, 2013) and Bachrach et al. (Bachrach et al., 2011), a 2D LiDAR is mounted on the top of a UAV, and SLAM is used for generating the point cloud taking advantage of the vertical movement of the UAV. In order to increase the Field of View (FoV) of the scanner, in the ARIA project (Zhang & Singh, 2014) a servo mounted on the UAV is used to rotate the scanner while the UAV is flying, which leads to collecting 3D point clouds for inspection and navigation purposes. However, the details of the platform design for integrating the UAV, the servo and the scanner are not available in the literature.

The objective of this part of the research is to design a platform for LiDAR-equipped UAV for structural inspection using an affordable 2D scanner. The remaining sections of this chapter are as follows: First, the requirements and other design considerations are introduced in Section 3.2. Section 3.3 provides the design details and the hardware and software integration steps. Section 3.4 provides an initial test of the platform. Section 3.5 provides a conclusion and summary of future work.

3.2 Design Considerations

In order to have a successful design of the LiDAR-equipped UAV, the following should be considered:

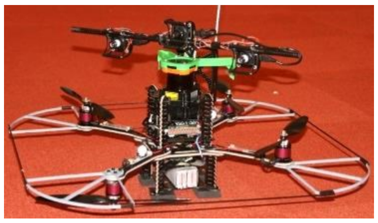
3.2.1 Objectives

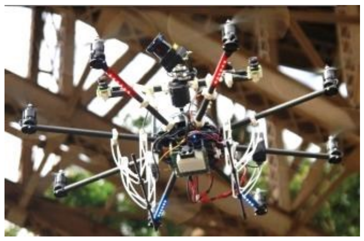
The maximum coverage of the surface of the inspected structure and the minimum cost are the two main objectives of an efficient inspection using the LiDAR-equipped UAV. The main costs of this method are the equipment cost and the flight cost. The flight cost depends on the time of the flight. The full coverage may not be achieved because of the obstacle near the inspected structure, which can limit the visibility of the LiDAR. So, the path planning goal is finding a collision-free path with minimum time-of-flight and maximum coverage.

3.2.2 Requirements

In order to choose the most appropriate LiDAR-equipped UAV, all the following requirements should be considered with respect to the budget.

(1) Mounting location: Most of the commercially available solutions have the scanner mounted under the UAV because they are designed for surveying purposes (Figure 3-1(c)). However, for structural inspection purposes, the LiDAR can be mounted either on top (Figure 3-1(a) and (b)) or under the UAV depending on the location of the inspected area of the structure.


(a) MIT RANGE (Bachrach et al., 2011)


(b) CMU ARIA (Huber, 2014)

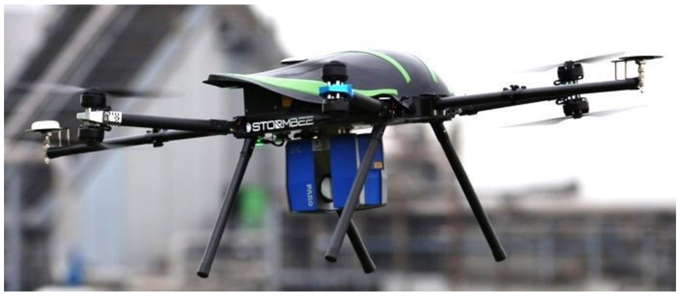

(c) Stormbee (Stormbee demo days, 2017)

Figure 3-1. Scanner position on top of UAV in (a) and (b), and under UAV in (c)

(2) Metrology method: There are two types of metrology methods for LiDAR: Time-of-Flight (ToF) and Phase Shift (PS). ToF is used for a long-range of measurement with an accuracy of 4-10 mm at 100 m. Unlike ToF, PS is practical for short-range measurement with 2-4 mm at 20 m (Kim et al., 2014).

(3) Maximum payload: The maximum payload is the weight that the UAV is capable to carry. Therefore, the weight of all carried devices (e.g. scanner, minicomputer, batteries, and GPS) should not exceed this threshold. The payload affects the UAV time of flight because carrying a heavier payload consumes more energy. As the weight of the scanner is one of the major weights in the payload of the UAV, it should be carefully considered. Providing a lightweight and accurate scanner is expensive, and choosing the best option depends on the available budget. Moreover, extra batteries are needed to supply the power for the scanner and other electronic parts attached to the UAV (e.g. servo, microcomputer, etc.).

(4) Size of UAV: The UAV should be big enough to carry the scanner and other equipment, and small enough to fly safely as close as possible to the inspected surface.

3.2.3 Constraints

There are several constraints that should be considered during planning.

(1) Minimum and maximum distances: A specific distance range should be considered during inspection based on safety and the characteristics of the scanner. The density of the scanned point cloud decreases with longer distances.

(2) Battery capacity: The battery capacity has effects on the time of flight. As mentioned above, although adding more batteries helps the UAV to fly further, it increases the weight of the system.

(3) Vibration: The vibration of the LiDAR-equipped UAV during inspection causes errors. A suitable design of a LiDAR-equipped UAV, which includes designing an appropriate engine and body shape, installing dampers, etc., can decrease the vibration (Li et al., 2015).

(4) Degrees of Freedom (DoFs): Each UAV has six DoFs: three displacements (x, y, and z) and three rotations (roll, pitch, and yaw). In general, the pitch and roll of the UAV are constrained to keep the UAV in a horizontal position.

(5) LiDAR parameters: The 2D and 3D scanners have one and two FoVs, respectively. The FoV is an important parameter in the visibility analysis. Furthermore, other important parameters of the scanner are the angular resolution ($\Delta\theta$), incidence angle (θ), and beam diameter (Figure 3-2). The LiDAR light beam reflects in two ways of specular and diffuse reflection after hitting an object (Jiang et al., 2017). The accuracy of a point cloud is mainly related to the measurement resolution, angular resolution, and scanning speed. In the case of the 2D scanner integrated with a servo, the angular resolution and the speed of the servo affect the accuracy of the generated point cloud.

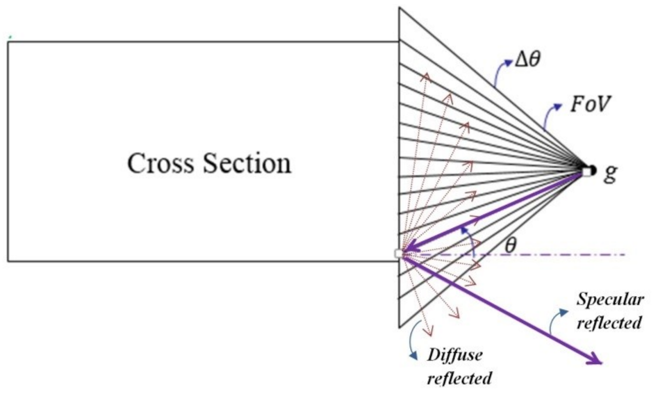


Figure 3-2 Some specifications of the LiDAR

3.2.4 Other Considerations

(1) Safe operation: Mounting additional devices should not change the center of gravity of the UAV because it affects the stability of the UAV. Also, the additional devices should not interrupt the GPS signals.

(2) Real-time: The LiDAR-equipped UAV platform has to collect a large amount of point cloud data to be used in real-time for path planning and obstacle detection.

3.3 Platform Design

UAV selection

DJI Matrice 100 (MATRICE 100, 2016) is used in this platform because it is customizable and has expansion bays to mount the scanner and other devices on top or below the UAV. Also, its radius is less than one meter, which makes it agile and able to enter narrow spaces near the inspection surfaces. The specifications of this UAV are shown in Table 3-1. The maximum payload is about 1.2 Kg.

Table 3-1 UAV specifications

Specification	**Value**
Max Takeoff Weight	3.60 Kg
Net weight	2.43 Kg
Battery	5700 mAh – 22.8V
Diameter	996 mm
Hovering Time (no payload)	28 minutes

LiDAR selection

Hokuyo UTM-30LX 2D laser range finder is used for data collection because of its lightweight and affordable price. The specifications of this scanner are shown in Table 3-2 (HOKUYO AUTOMATIC, 2014).

Table 3-2 Scanner specifications

Specification	**Value**
Detection range	0.1 ~ 30m
Accuracy	±30mm (under 10m)
Horizontal FoV	270º
Angular resolution	0.25º
Scan speed	43,200 points per second
Weight	210 g (without cables)

Servo selection

In order to enable the scanner to generate a 3D point cloud, a servo is used to rotate it. Dynamixel MX-28T is a robotic actuator servo that can control the movement of the scanner with a minimum step of 0.088º, which means the angular resolution of the platform is 0.088º. By rotating the scanner 180º, the vertical FoV of the scanning becomes 360º.

The servo uses an adapter (USB2Dynamixel) to connect to the microcomputer and another adapter (SMPS2Dynamixel) to connect to the battery. Both the servo and the scanner need a 12V power supply.

Microcomputer selection

To control the scanner and the servo, and to collect data from them in real-time, the DJI MANIFOLD microcomputer is selected in this platform because it is compatible with the UAV (MANIFOLD). The power for MANIFOLD is supplied directly from the UAV.

Electronic connectors

An isolated voltage regulating board is designed and built to convert the 25V power of UAV's port into 12V. This board makes it possible to run the scanner and servo without adding an extra battery. The weight of the voltage regulating board is 72 g.

Interfacing parts using 3D printing

Although previous research exists about interfacing a servo with 2D LiDAR (Bogosian et al., 2016), the integration with the UAV requires additional interfaces to control the direction of the scanning. Three different interfacing parts are designed and 3D printed to attach the scanner and the servo to each other and to the UAV. The designed parts are shown in Figure 3-3. Figure 3-3 (c) and (d) show the parts for attaching the servo to the UAV in vertical or inclined positions, respectively.

3.3.1 Integration

To integrate the components, hardware and software integrations are required. These are discussed in the following sections.

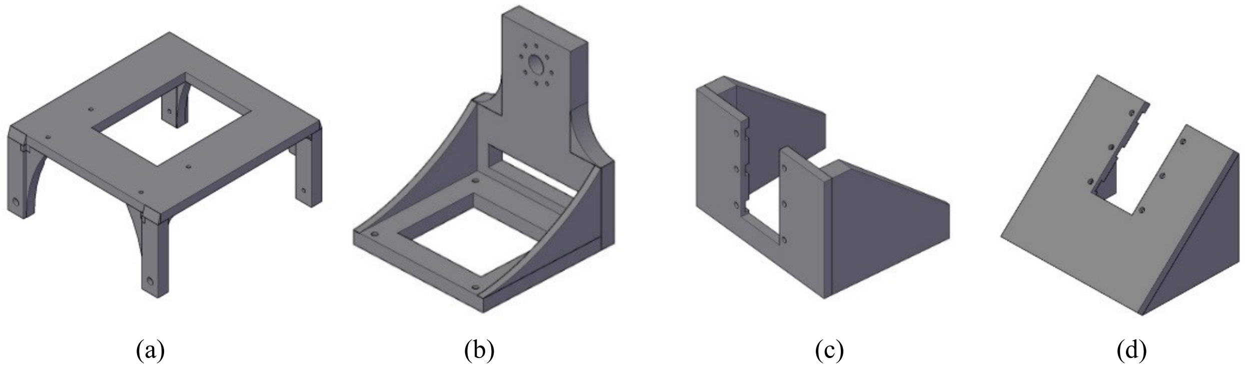


Figure 3-3 Design of interfacing parts for 3D printing, (a) Part for holding the servo, (b) Part for holding the scanner, (c) Part for attaching the servo vertically, and (d) Part for attaching the servo inclined

3.3.1.1 Hardware Integration

Figure 3-4 shows the integration of the hardware components of the platform. Table 3-3 shows the summary of hardware specifications and their connectivity to each other. The total weight of the integrated platform is 3.13 Kg, which is less than the maximum takeoff weight of the UAV.

In this platform, the scanner rotates 180º (from -90º to 90º). It stops for 0.1 s when changing direction from clockwise to counterclockwise. This stop causes some errors in the registration process of the point cloud. It is possible to rotate the scanner continuously in one direction by adding a slip-ring between the scanner and the servo to eliminate the rotation of the cables of the scanner. Figure 3-5 shows the interfacing part for mounting the servo on the UAV vertically or with inclination, and the corresponding configurations of the LiDAR-equipped UAV platform.

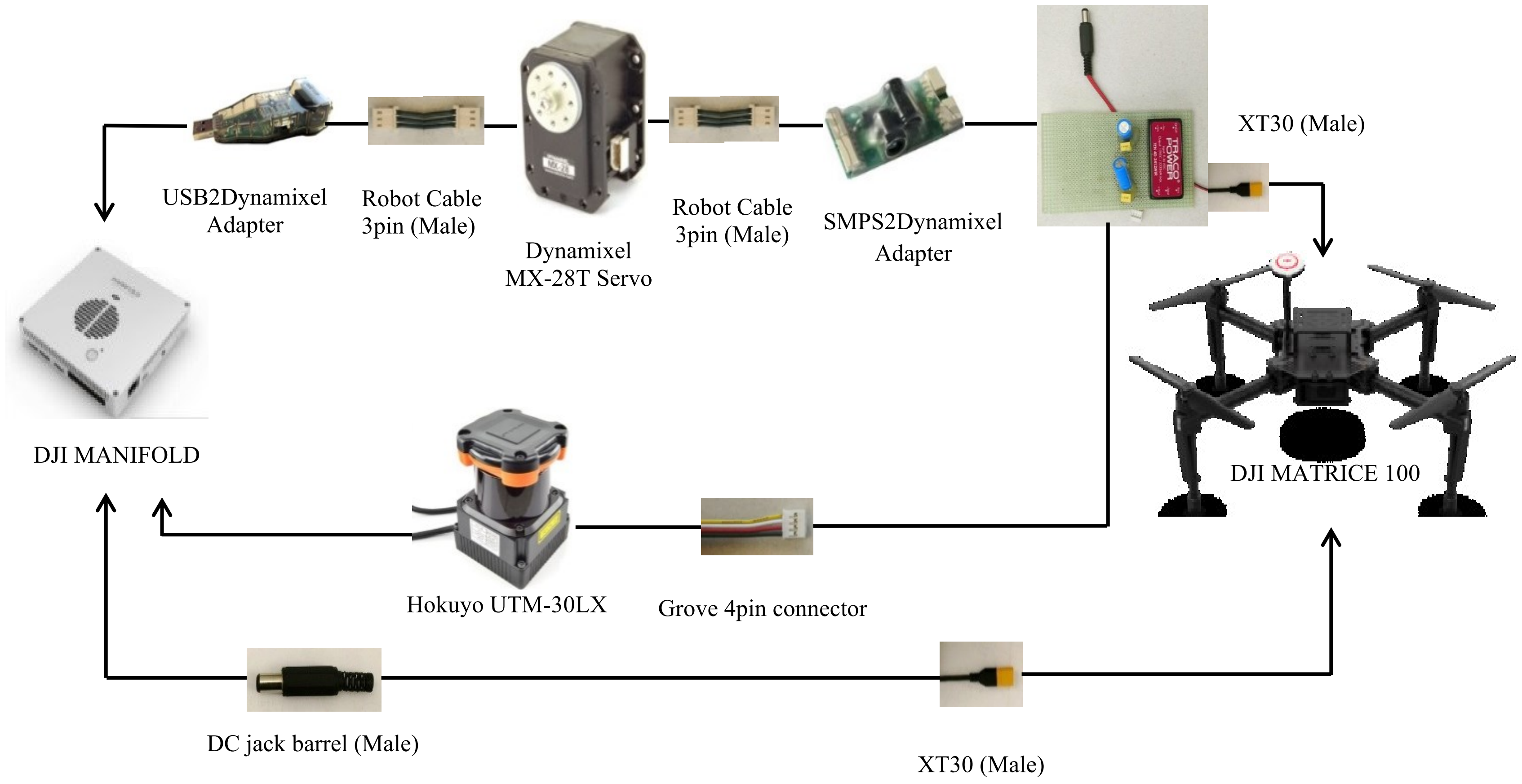
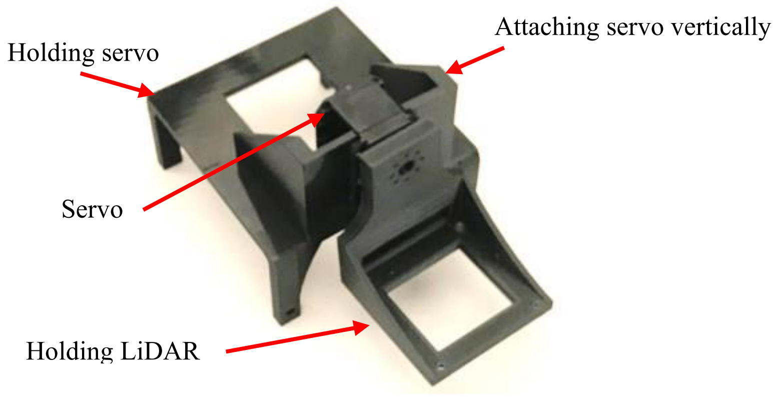


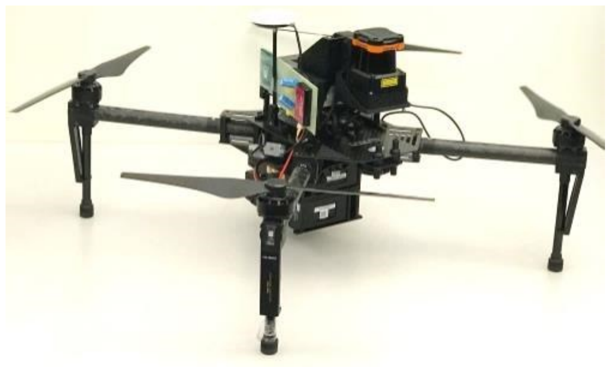
Figure 3-4 Hardware integration

Table 3-3 Summary table of hardware components

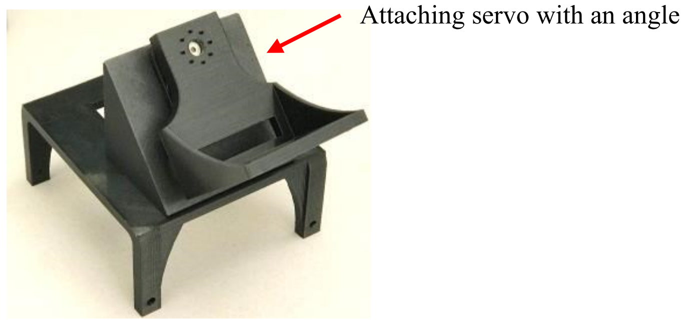
Component	Function	Voltage (V)	Current (A)	Weight (g)	Attached to
Hokuyo UTM-30LX	Laser Scanning	12	1.0	210	MANIFOLD, Voltage Regulator Board
MANIFOLD	Controlling scanner and Servo	14-26	Up to 10	197	UAV, USB2Dynamixel Adapter, scanner
Dynamixel MX28T	Rotating scanner	12	1.4	72	USB2Dynamixel Adapter, SMPS2Dynamixel Adapter
USB2Dynamixel Adapter	Connect servo to MANIFOLD	N/A	N/A	28	Servo, MANIFOLD
SMPS2Dynamixel Adapter	Power supply for servo	12	1.4	14	Servo, Voltage Regulator Board
Voltage Regulator Board	Regulate the voltage to 12V	9-36	3.3	72	SMPS2Dynamixel Adapter, scanner, UAV
Part for holding the servo	Connect servo and scanner to UAV	N/A	N/A	45	UAV, 3D printed servo part
Part for attaching servo vertically	Connect the servo to the table part	N/A	N/A	26	Servo, 3D printed table part
Part for attaching servo inclined	Connect the servo to the table part	N/A	N/A	29	Servo, 3D printed table part
Part for holding the scanner	Connect the scanner to the servo	N/A	N/A	31	Servo, scanner



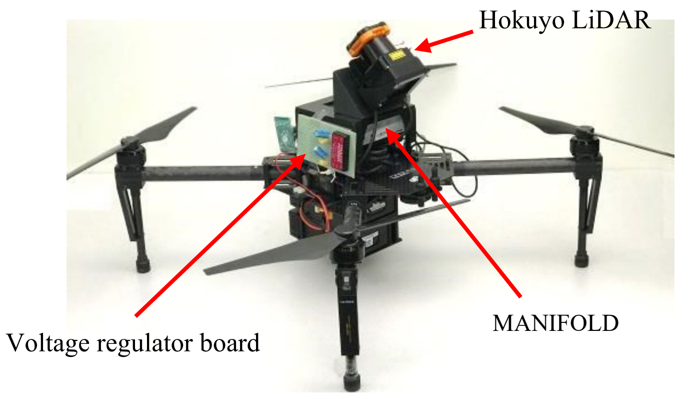
(a) Interfacing part for vertically mounting



(b) Platform with vertial LiDAR



(c) Interfacing part for inclined mounting



(d) Platform with inclined LiDAR

Figure 3-5 LiDAR-equipped UAV platform

3.3.1.2 Software Integration

Spin Hokuyo Robot Operating System (ROS) software package is installed on the microcomputer to create 3D point clouds in real-time (Bertussi et al., 2017). This software works under Ubuntu operating system and contains the code to control the servo and the scanner to generate a 3D point cloud. Spin Hokuyo has five nodes for the following purposes: (1) two nodes (tilt motor and tilt transform) for controlling the servo and assembling point cloud messages; (2) one node (Hokuyo robot filter) to remove unnecessary points that are related to the body of the operating robot. It eliminates all the points inside the radius of 50 centimeters; (3) one node (scan to PCL (Point Cloud Library)) to convert the scanned data into point cloud messages; and (4) one node (PCL assembler client) to combine all the published point cloud messages into one point cloud message. Spin Hokuyo can adjust the initial, start and end positions of the servo and its rotation speed. The SLAM algorithm can be used in the software package to enable the platform to scan during the UAV flight.

After scanning, the point cloud that is generated by Spin Hokuyo can be visualized in Rviz (ROS 3D visualization tool) (Hershberger et al., 2018). Rosbag package records the output messages of spin Hokuyo and saves them as Bag file (Field et al., 2010). There is a node named *Bag to PCL* in PCL-ROS package, which reads the Bag file and converts ROS point cloud messages to PCD (Point Cloud Data) files (FARO, 2012) CloudCompare software can open and visualize PCD point cloud files, and convert them to other point clouds file formats, such as LAS, LAZ, and E57 (Venator, 2015).

In this work, Spin Hokuyo is used in a stationary mode for initial testing as explained in Section 4. When integrated with In the case of flying on a UAV, because the GPS signals may not be available when the UAV is flying under a bridge., Therefore, other localization methods can be investigated, such as integrating an onboard Inertial Measurement Unit (IMU) with Visual Odometry (VO), Simultaneous Localization and Mapping (SLAM) algorithms, or Lidar Odometry and Mapping (LOAM) (Zhang & Singh, 2014).

3.4 Case Study

In order to validate the performance of the designed platform, two tests are implemented. The first test is performed in an indoor environment to realize the LiDAR system. The second test is performed in an outdoor environment to investigate the LiDAR-equipped UAV platform's performance when it is flying.

3.4.1 Indoor Test

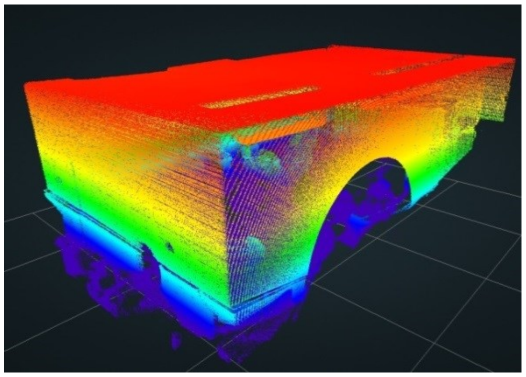
Before moving to outdoor flying tests of the designed platform, the initial test is performed in an indoor environment, and it is limited to testing the LiDAR system (LiDAR, servo, microcomputer, battery, interface elements, and connectors) when the drone is stationary. The width, length, and height of the room are 4, 7 and 3 m, respectively. The generated point cloud is shown in Figure 3-6 (a). The color distribution of the point cloud is based on elevation. The horizontal and vertical FoVs of the LiDAR system in this test are 270º and 360º, respectively. The LiDAR scans the environment in 2D lines, and each line contains 1,080 points. The number of lines of scanning in a sweep is dependent on the rotation speed of the servo. In this test, the rotation speed was set on 0.5 radians per second. Each sweep is π radians, so, one sweep was taken about 6.28 s to complete. The LiDAR scans 40 lines in a second, so, one sweep should contain about 251 lines and almost 270,000 points. The point cloud is shown in Figure 3-6 (a) is generated by 13 sweeps and contains 3.2 million points. Each sweep has about 250,000 points. Some points are eliminated by the filtering node.

In order to check the accuracy of the collected point cloud, the same space was scanned with a higher accuracy 3D LiDAR scanner (FARO Focus3D) to generate a ground truth point cloud. FARO Focus3D is a 3D laser scanner with an accuracy of 2 mm, which can scan about one million points per second (FARO, 2012; Bogosian et al., 2016).

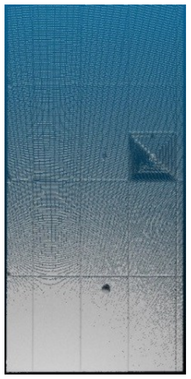
A segment of the ceiling with the dimensions of 2.3 m × 4.7 m is selected (Figure 3-6 (b)) and compared as a cloud-to-cloud distance comparison by CloudCompare software, where the distance of each point of the compared cloud is measured to its nearest neighbor in the reference cloud.

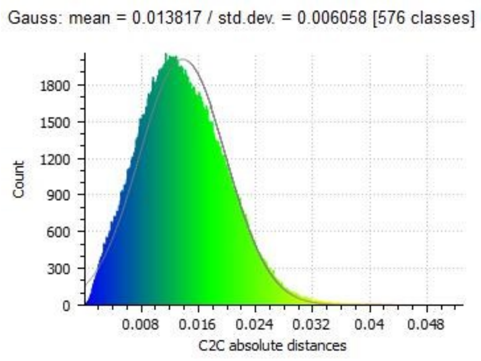
The two point clouds were aligned to each other manually in CloudCompare by picking five equivalent point pairs. Each segment contains about 250,000 points. The segment point cloud from FARO (Figure 3-6 (c)) is considered as the reference cloud. The distance distribution histogram of the comparison is shown in Figure 3-6 (d). Gaussian distribution is used in this computation, and the mean of the distribution is 1.4 cm with a standard deviation of 0.6 cm.

As shown in Figure 3-6 (e), the gaps between the drop ceilings tiles of the room are visible. The width of these gaps is about 2 cm, which is greater than the calculated error. Assuming that these gaps are similar in size to some large cracks that could be detected on the actual structure during an inspection, the accuracy of the point cloud collected by the platform can be considered enough to detect this size of cracks.


(a)

(b)


(c)



(d)

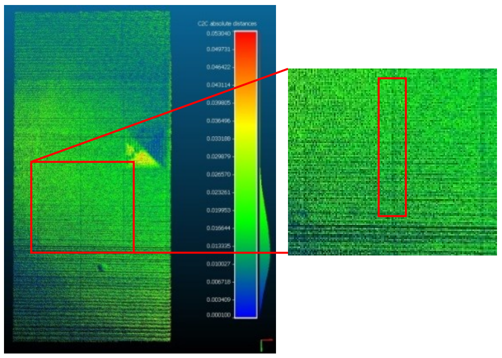

(e)

Figure 3-6 The initial test's results: (a) The generated point cloud by the designed platform, (b) A segment of the point cloud, (c) The reference segment using FARO, (d) The comparison distance distribution histogram (m), and (e) The compared segment colored based on distances from the reference segment

3.4.2 Outdoor Test

After performing the stationary indoor test, an outdoor test is executed to validate the designed platform when it is flying. The test is executed on the 6th of March, 2019, in a park in Nuns' Island, Montreal (Figure 3-7). This location is selected due to the UAV flying restrictions in an outdoor public environment. The LiDAR-equipped UAV scanned a small building during flying using the LOAM method. Also, in order to check the accuracy of the collected point cloud, the same building is scanned using FARO Focus3D (a higher accuracy 3D LiDAR scanner) (Figure 3-8).

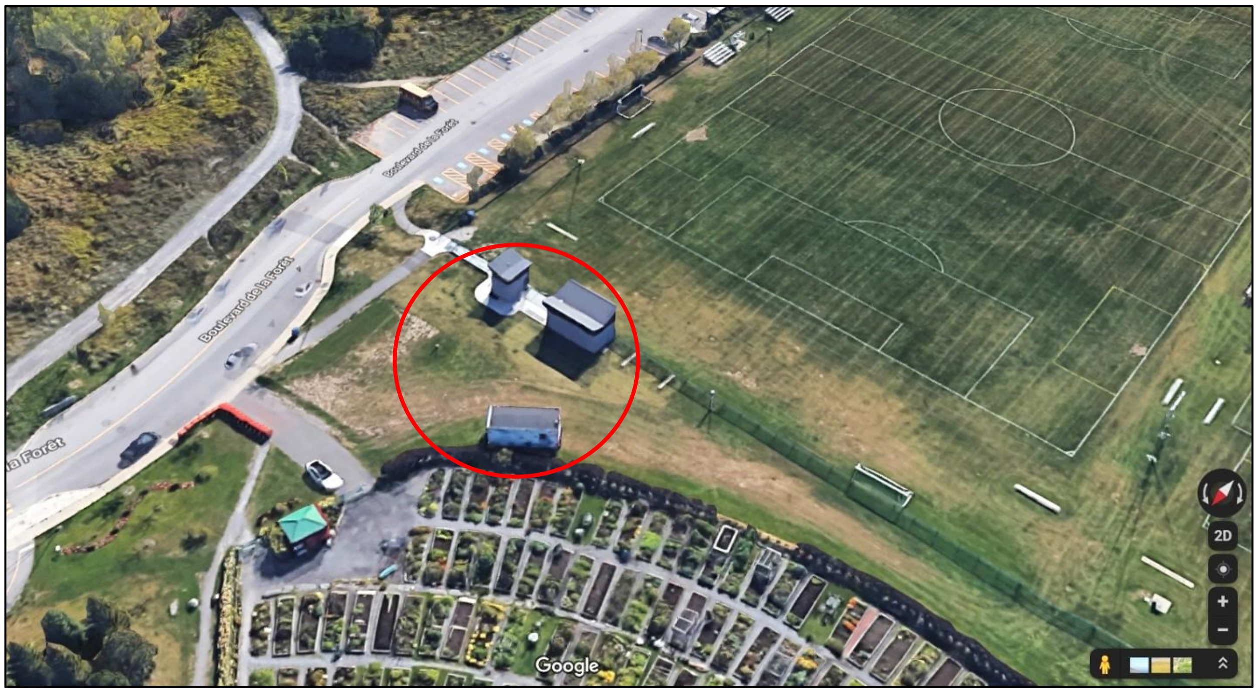


Figure 3-7 The location of the outdoor test in Montreal, QC, Canada

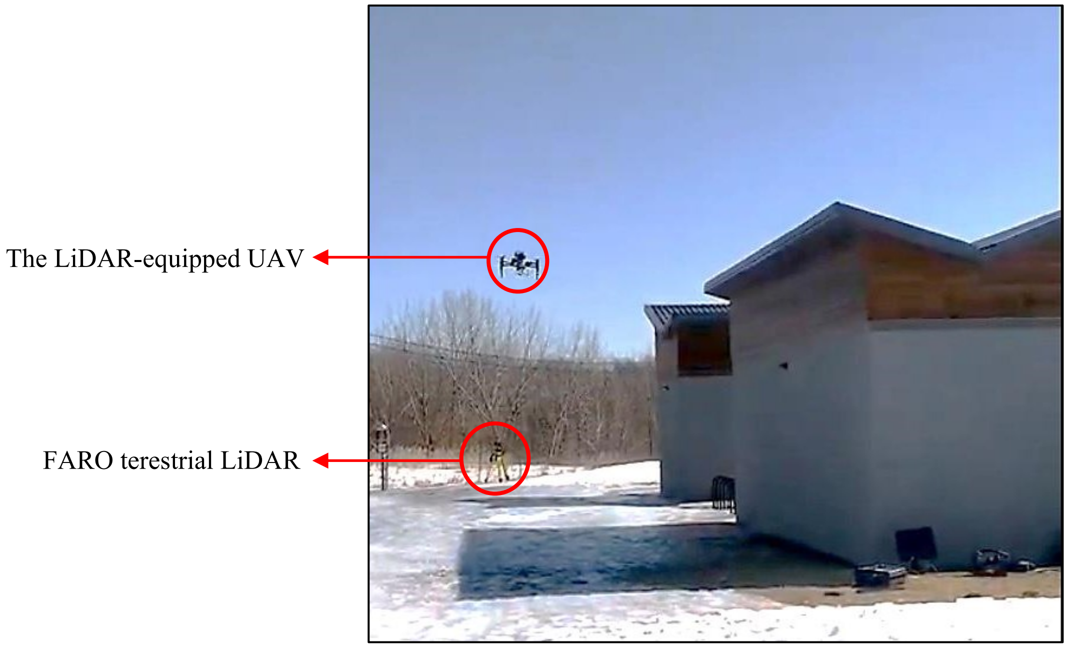
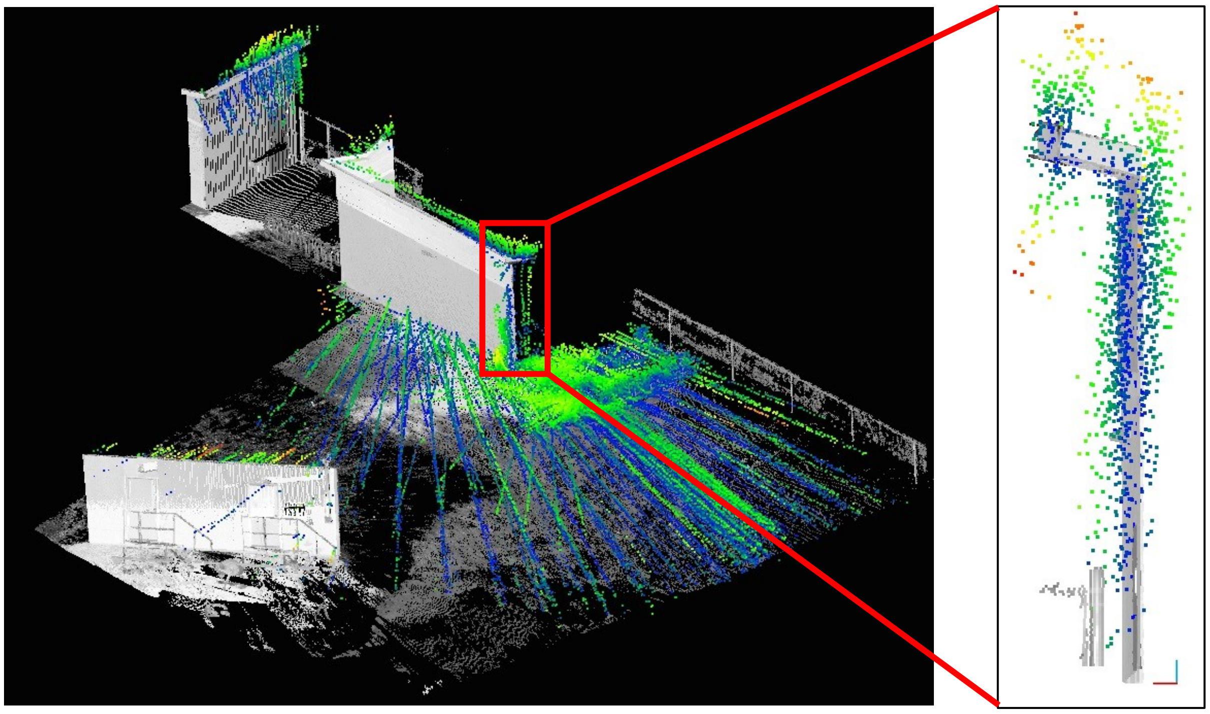
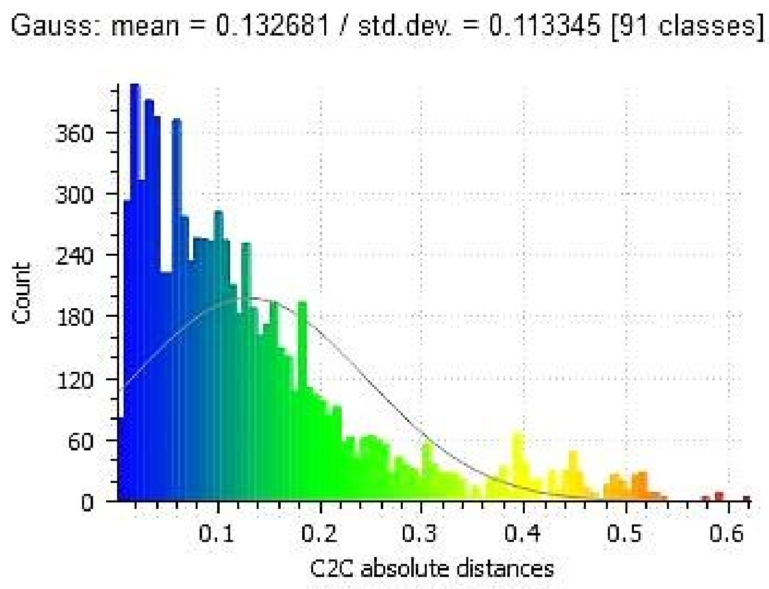


Figure 3-8 Outdoor test



(a) (b)



(c)

Figure 3-9 The outdoor test's results: (a) The overlapped point clouds of the designed platform, and FARO, (b) A segment of the overlapped point clouds, and (c) The comparison distance distribution histogram (m)

The two point clouds are aligned with each other manually in Figure 3-9 (a). Figure 3-9 (b) shows a part of the aligned point clouds. This part is a section of a wall of the scanned building. Two point clouds segments in the part are compared as a cloud-to-cloud distance comparison. The distance distribution histogram of the comparison is shown in Figure 3-9 (c). The mean of the distribution is 13.3 cm with a standard deviation of 11.3 cm. The accuracy of the point cloud collected by the platform is not enough to detect surface defects smaller than 10cm. The reasons of platform's low accuracy when it is flying are: (1) The low accuracy of the 2D scanner, (2) The inaccuracy of matching the scanned lines during rotation of the 2D scanner that multiples by the inaccuracy value of the 2D scanner, (3) The inaccuracy of matching the scanned point clouds during flying, which also multiples by the inaccuracy value of the 2D scanner and matching scanned lines, and (4) The weather condition (wind).

3.5 Summary and Conclusions

A LiDAR-equipped UAV platform is designed to collect 3D point cloud data using a 2D LiDAR scanner. The design satisfies the main identified requirements and constraints for structural inspection and the platform is realized and tested in an indoor environment in a stationary mode. The results of an outdoor test in flying mode show that the point clouds' accuracy is not enough to detect surface defects smaller than 10 cm. The platform can generate point clouds with higher accuracy using an accurate light-weighted 3D LiDAR but this solution is very expensive at the current time. Therefore, our future work includes validating the designed LiDAR-equipped UAV platform using a more accurate light-weighted 3D LiDAR.

CHAPTER 4 Concrete Surface Defect Detection Using Deep Neural Network Based on LiDAR Scanning

4.1 Introduction

As mentioned in CHAPTER 1, the structural inspection of bridges is essential to improve the safety of infrastructure systems, such as bridges. Visual inspection is the traditional method of detecting surface defects of bridges. This process can be unsafe, time-consuming, expensive and subjective to human errors (Guldur et al. 2015). In order to automate the process of structural inspection, it is important to collect proper datasets and use an efficient approach to analyze them and find the defects. Recently, 3D Light Detection and Ranging (LiDAR) scanners (Liu et al. 2011) are used for detecting surface defects (e.g. cracks) using computer vision methods. LiDAR scanners can collect high-quality 3D point cloud datasets.

The objective of this chapter is to develop a method for detecting concrete surface defects using a DNN (Section 2.5) based on LiDAR scanning. In the developed method, PointNet (Qi et al., 2017a) is adapted to detect surface defects using point cloud datasets from scanning bridge surfaces. The reason for selecting PointNet is that it is robust to missing and corrupted data. PointNet++ (Qi et al., 2017b) is also adapted to detect concrete surface defects. It is applied to detect types and severity levels of the structural defects in point cloud datasets. PointNet++ uses PointNet units in its hierarchical network and takes advantage of PointNets robustness. It also extracts point's features from various sample scales, which makes it more accurate on segmenting adjacent or relatively small objects.

4.2 Methodology

As explained in Section 2.2, regular concrete surface defects on bridges are cracks, spalling, scaling, and delamination. Scaling is the flaking or peeling of the surface of the concrete as a result of exposure to freezing and thawing. Delamination is separating the surface layer of concrete, which is not completely detached. The prolonged delamination until detaching causes spalling and spall is a detached part of a concrete mass (Ministry of Transportation, 2008). Table 4-1 shows the severity of defects, which is categorized into four levels of light, medium, severe, and very severe, based on the depth and area of the defects.

Table 4-1 Defects severity based on the depth of loss (d), width and height of the affected area (w, h) (Ministry of Transportation 2008)

Type of defect	Light*	Medium*	Severe*	Very Severe*
Scaling	$d < 5$	$5 \leq d < 10$	$10 \leq d < 20$	$20 \leq d$
Delamination	$w, h < 150$	$150 \leq w, h < 300$	$300 \leq w, h < 600$	$600 \leq w, h$
Spalling	$w, h < 150$ $d < 25$	$150 \leq w, h < 300$ $25 \leq d < 50$	$300 \leq w, h < 600$ $50 \leq d < 100$	$600 \leq w, h$ $100 \leq d$

* All of the dimensions are in mm

This section explains the steps of applying the CNN approach on 3D point cloud datasets to recognize concrete surface defects. Two networks of PointNet and Pointnet++ are used in this study, which are point cloud analysis solutions using CNN. These networks are adapted to solve the structural defect detection problem. The performance of the networks is improved by modifying their algorithms (e.g. modifying the network topology, changing the loss function, adding the dropout layers) and by improving the dataset (e.g. collecting more data, annotating in multi-classes, adding data by flipping). Moreover, the details of the CNN architecture are explained in Section 4.3. There are five main steps in this method: (1) data collection, (2) manual annotation, (3) data pre-processing, (4) training and evaluation, and (5) testing (Figure 4-1).

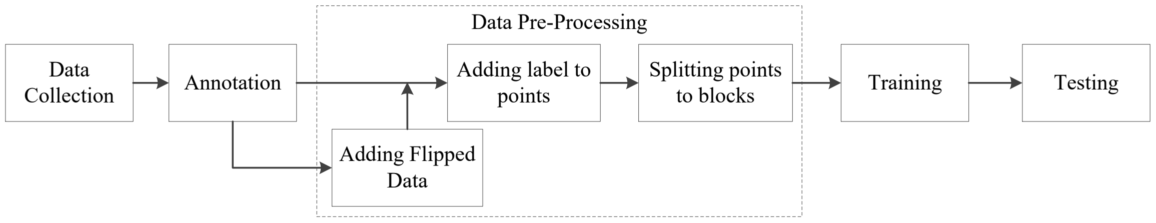


Figure 4-1. The proposed method

The adapted semantic segmentation parts of PointNet and PointNet++ are used for binary classes' semantic segmentation. These networks are originally designed to detect indoor building elements. In this chapter, they are adapted to detect surface defects using point cloud datasets from scanning bridge surfaces. The results of applying adapted PointNet and PointNet++ are compared in the binary classes' semantic segmentation.

The adapted semantic segmentation model of PointNet++ is also used in a part of the method for multi-classes' semantic segmentation. This network is used in this work because it has PointNet's advantages of being robust to missing and corrupted data and directly applying on point datasets. Also, it is more efficient in segmenting classes with small geometries.

4.2.1 Data Collection

The process of accurate data collection is an important aspect of reaching valuable output results of a CNN algorithm. The geometric features of defects, especially the depth, play an important role in extracting useful features using the neural network. Terrestrial laser scanning provides high stability and accuracy compared to other methods of laser scanning (Nasrollahi et al. 2018). The position of the LiDAR scanner affects the visibility of defects in the collected point clouds. In addition, the fine registration process results in having reliable datasets to feed CNN.

4.2.2 Annotation

In addition to collecting accurate datasets, the annotation process is vital for reaching a satisfactory CNN trained model. The annotation process is manual and is based on the structural definitions of concrete surface defects as shown in Table 4-1. The datasets are annotated into five main classes of crack, light spalling, medium spalling, severe spalling, and no-defect.

4.2.3 Data Pre-Processing

PointNet and PointNet++ semantic segmentation are based on feeding the input data with the structure of S3DIS; so the training and testing datasets of this work are prepared in the same structure. Point clouds are distributed manually into different areas and each area has different parts. The *Z*-axis of the canonical coordinate system is set in the vertical direction. The *X*-axis is along the concrete surface of the bridge, and the *Y*-axis is perpendicular to the concrete surface (inside direction). This setting makes the depths of the defects' *Y* values positive. The convolutional process is 2D and performs on the *XZ* surface of the dataset; so the 3D blocks of points are selected on this surface with the specified dimensions in the *X* and the *Z* directions. The depth of the dataset is the third dimension *Y*. By choosing the dimensions and number of points of the 3D blocks, points are downsampled or upsampled as required, based on the density of the input datasets. The effect of changing the density on the results is discussed in Section 4.3.4.4.

In the data pre-processing step, the annotated point clouds are converted to Hierarchical Data Format (HDF). HDF is an abstract data managing and storing model (HDF-Group, 2018). Points are wrapped and normalized inside the blocks and saved in HDF5 format. In this process, a channel adds to each point's information. This channel is the normalized location value of the points in dimension *Y* and with respect to the point cloud. For each part, the point with the minimum coordinate value is set to the origin of a local coordinate system and the normalized value of all the points is calculated as follows (Qi et al., 2017a):

$$\mathbf{N_{y_i}} = \frac{\mathbf{y_i}}{\mathbf{Y}} \qquad \mathbf{4\text{-}1}$$

where *Y* is the maximum value in the specific part. The value of N_y is distributed from 0 to 1. By adding N_y, each point is represented as a 7-dimensional vector of *XYZ*, *RGB*, and N_y. In the original PointNet network, three channels of the normalized location values for *XYZ* dimensions are calculated and added to each point's information, but the normalized location values on *XZ* surface are not useful for defect detection. A point location value in *Y*-axis is a piece of valuable information and helps the network to learn how to detect defects points.

4.2.4 Training and evaluation

In this part, PointNet and PointNet++ are adapted to detect the surface concrete defects. The numbers of points in the defect classes of the dataset of this study are much less than the number of no-defect points, which is known as the issue of "imbalanced dataset". There are two main categories of methods for addressing the imbalance issues of datasets; data-level methods and classifier-level methods (Buda et al., 2018). Data-level methods are oversampling the small-sized classes or undersampling the large-sized classes. Thresholding and cost-sensitive learning are classifier-level methods. In this study, cost-sensitive learning is selected to overcome the dataset unbalanced issue. In a regular loss function, the effect of miss-predicting a point on the learning predictor is equal, regardless of the size of the class. By using a weighted loss function, the effective weight of points of each class on the correcting process of backpropagation can be adjusted. Weighted sparse softmax cross-entropy loss function is used in this study, which is discussed in Section 4.3.

4.2.4.1 Adapted PointNet

The training model has two main parts: (1) classification network and (2) segmentation network, which are explained in the following sections. The generated blocks of points in data pre-processing step are fed into the CNN as input data. The number of points in each block in the original PointNet is 1,024 with a block size of 1 m × 1 m on the *XY* surface for rooms with a height of 3 m. This is assumed as a very low density of points for detecting most types of defects (e.g. Medium-sized spalls). Therefore, the selected block sizes are less than 50 cm × 50 cm on the *XZ* surface, with the depth of the defects as the third dimension, which is less than 10 cm. Choosing the same number of points (i.e. 1,024) with the new block size can increase the density of points by more than 100 times. A weighted loss function is used to adapt the model to our dataset. The original PointNet receives 9-dimensional input vectors that contain normalized location values over *X*, *Y*, and *Z* directions ($N_xN_yN_z$). As mentioned in Section 4.2.3, just the N_y is calculated for our dataset because the relative location values of defects' points over *X* and *Z* direction are not valuable in detecting them and mislead the learning process.

Since the number of the defect points are less than the points belonging to the non-defect parts of the point cloud, which is known as the issue of "Imbalanced Datasets", a weighted softmax loss function is used and the corresponding weight vector is defined based on the points distribution on the classes.

4.2.4.1.1 Classification network

The classification network has two sets of MLP. The first set of MLP accepts blocks of points as input and every layer extracts detailed features of points by convolving on the blocks. Every hidden layer includes batch normalization and the ReLU activation function. The main goal of this set of MLP is to extract the local features per point from the 7-dimensional input vectors of points. The output of the first set of MLP is a vector of all input points, where every point has a weight. This vector represents the extracted local features of points. A max-pooling layer is applied to the feature vector to down-sample the features, followed by the second set of MLP in order to extract the global features of each point.

4.2.4.1.2 Segmentation network

This part of the network has a set of MLP that is fed by concatenation of the extracted local and global features. Each convolutional layer in this set of MLP is followed by a dropout layer, except the last one. Every hidden layer includes batch normalization and the ReLU activation function. The output of this network is a vector of predicted probabilities of belonging to each class for every point.

4.2.4.2 Adapted PointNet++

The training model has a hierarchical feature learning architecture and uses mini-PointNet units in every set of feature abstraction. The first hidden layer of PointNet++ has sub-layers of sampling and extracting features by applying PointNet CNN on them. The sub-layers work recursively. The second hidden layer of PointNet++ has sub-layers of grouping and applying PointNet CNN recursively. In each sub-layer, two output feature vectors of the first hidden layers' sub-layers are concatenated. The last set of layers are fully connected with a dropout layer in the middle. Every hidden layer (and sub-layer) includes batch normalization and the ReLU activation function.

A weighted loss function is also used in PointNet++ to overcome the imbalanced dataset issue. In PointNet++ the weight vector of classes is calculated for every set of batches separately. It enables the model to learn as robustly as possible from various classes because every set of batches may have different weight distribution.

4.2.5 Testing

The testing process uses a part of the datasets, which is not seen by the model in the training and the evaluation processes to validate the accuracy of the model.

4.3 Case study

In the case study, several datasets are collected using a terrestrial LiDAR and annotated manually into the five main classes of cracks, light spalling, medium spalling, severe spalling, and no-defect. In the first part of the case study, all the defects, regardless of the types and severity levels, are categorized as one class (defect), in order to get the initial validation of the proposed CNN approaches. The adapted PointNet and PointNet++ models are applied, and the results are compared. Also, the adapted PointNet++ model is applied to the multi-classes dataset in two steps. In the first step, it is evaluated on detecting types of defects (crack and spalling) and in the second step on detecting the severity of the spalling defects.

4.3.1 Data Collection

In CNN, a large dataset is required to train an accurate model. However, in this study, four reinforced concrete bridges in Montréal are scanned using a FARO Focus3D scanner (FARO, 2012) in order to collect accurate point cloud datasets. The specifications of this LiDAR are presented in Table 4-2. The locations of the scanned bridges are shown in Figure 4-2. In order to cover all the lower parts of the bridges, they are scanned from various stations. The positions of the stations in the first scanning of bridge one are shown in Figure 4-3 and Figure 4-4. In all the scanning stations the LiDAR is positioned in a way the points on the scanning surfaces are not further than 10m.

The scanning parameters, such as the fields of view (FoVs), resolution, quality, and the number of scanned points are in Table 4-3. Quality setting can be set between 1x and 8x, which affects the quality of the point cloud by reducing noises in the scan data. Ranging noise has two effects: (1) The higher the ranging noise is, the thicker the scan point cloud on a flat object will be, and (2) The higher the ranging noise is, the fewer scan points will remain on far distance objects (FARO, 2017).

The major contributor to the ranging noise is the sensor electronics, where the incoming signal is evaluated to determine the distance. One way to increase the signal strength is to increase the observation time. The quality factor 1x has the smallest observation time of 1 μs per scan point and 4x has 8 μs per scan point (FARO, 2017). In other words, by increasing the quality factor, the observation time will be increased, the incoming signals will be stronger, and the effective number of scanned points will be higher.

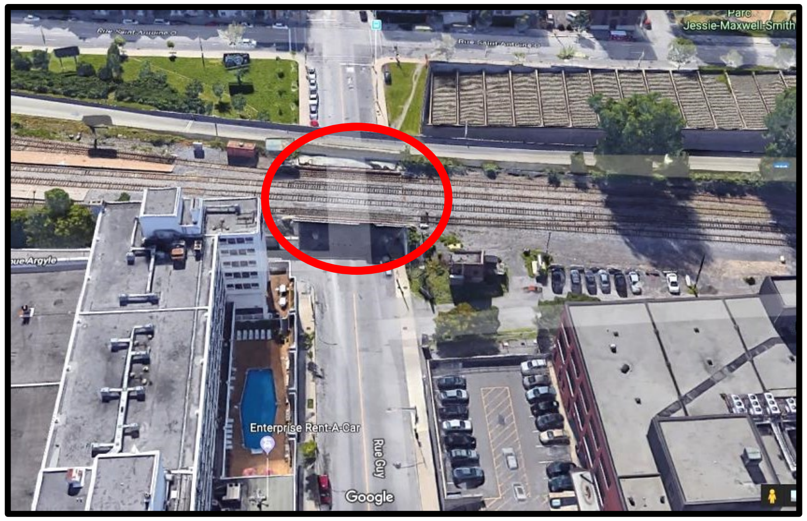
The resolution parameter can be set between 1 and 1/32. It declares the fraction of the number of points that will be detected over the number that can be detected. Resolution 1 means 710.7 million points in a 360º scan and 1/8 means 11.1 million points in a full scan (FARO, 2017). In other words, the resolution factor directly affects the distances between scanned points.

Table 4-2 FARO Focus3D LiDAR scanner specifications (Faro 2012)

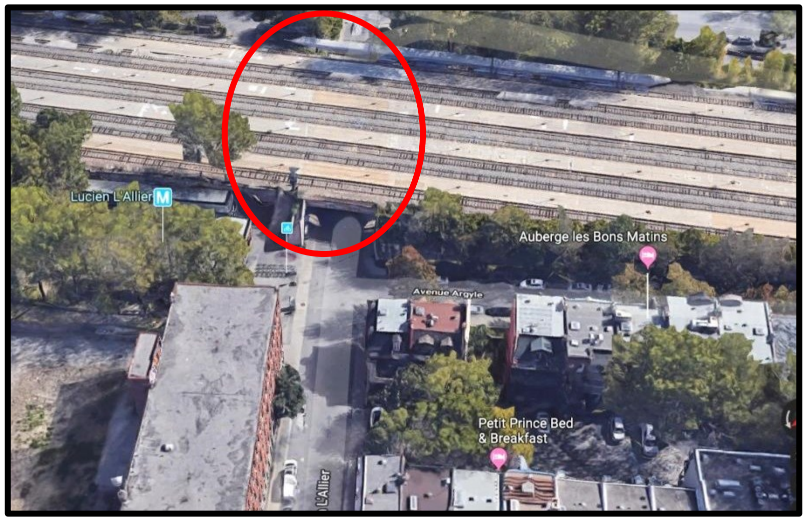
LiDAR	Points per Second	Field of View		Angular Resolution	Accuracy	Measurement Range
		Vertical	Horizontal			
FARO Focus 3D	976,000	305°	360°	0.009°	±2mm	1.5m - 120m

Table 4-3 Scanning information

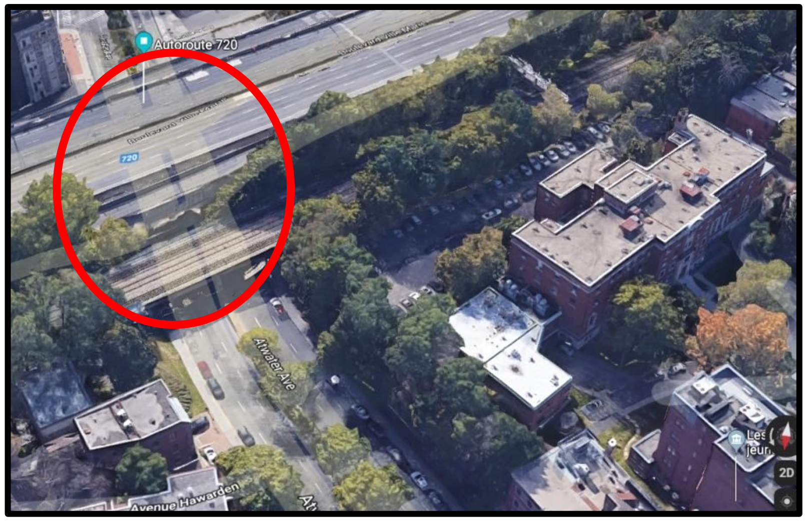
	Scan 1	Scan 2	Scan 3	Scan 4	Scan 5
Bridge	1	1	2	3	4
Stations	8	4	6	4	2
Horizontal FoV	23° to 259°	23° to 259 °	0° to 360°	0° to 360°	0° to 360°
Vertical FoV	-42.5° to 71°	-42.5° to 71°	-60° to 90°	-45° to 71°	-60° to 90°
Resolution	1/4	1/4	1/1	1/2	1/2
Quality	6x	6x	2x	4x	4x
Number of Points (Mpts)	25.5	25.5	710.7	134.5	177.7



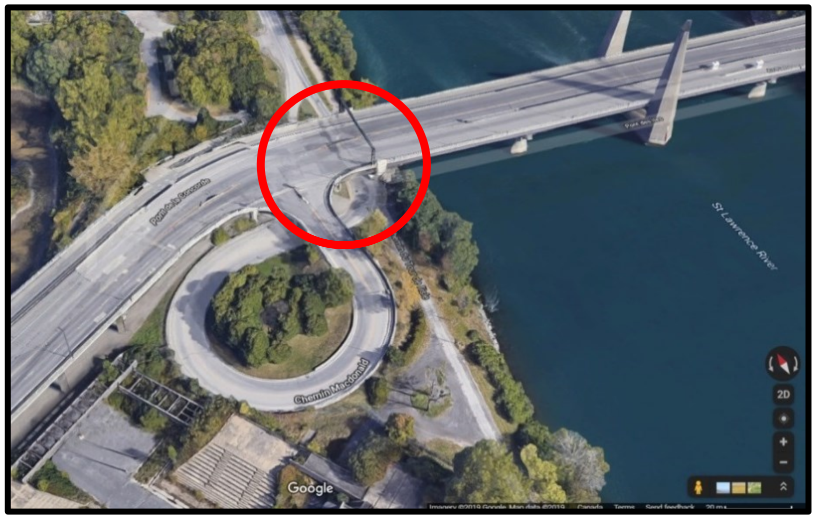
(1)



(2)



(3)



(4)

Figure 4-2 Location of the bridges in, Montreal, QC, Canada: (1) Guy Street, (2) Lucian L'Allier Street, (3) Atwater Avenue, and (4) Pierre-Dupuy Avenue

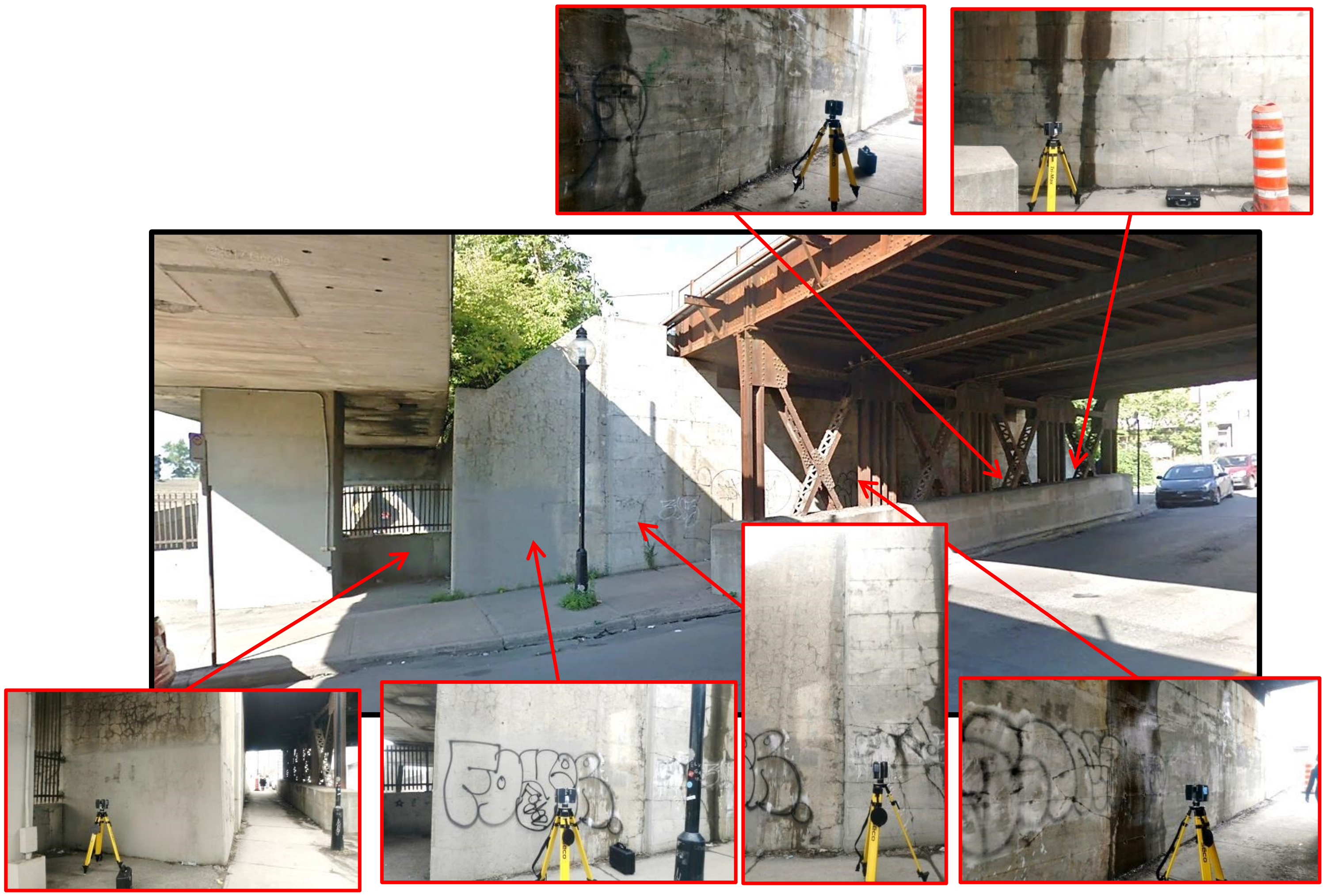


Figure 4-3 Scanning positions on the western side of Bridge 1 (Guy Street)

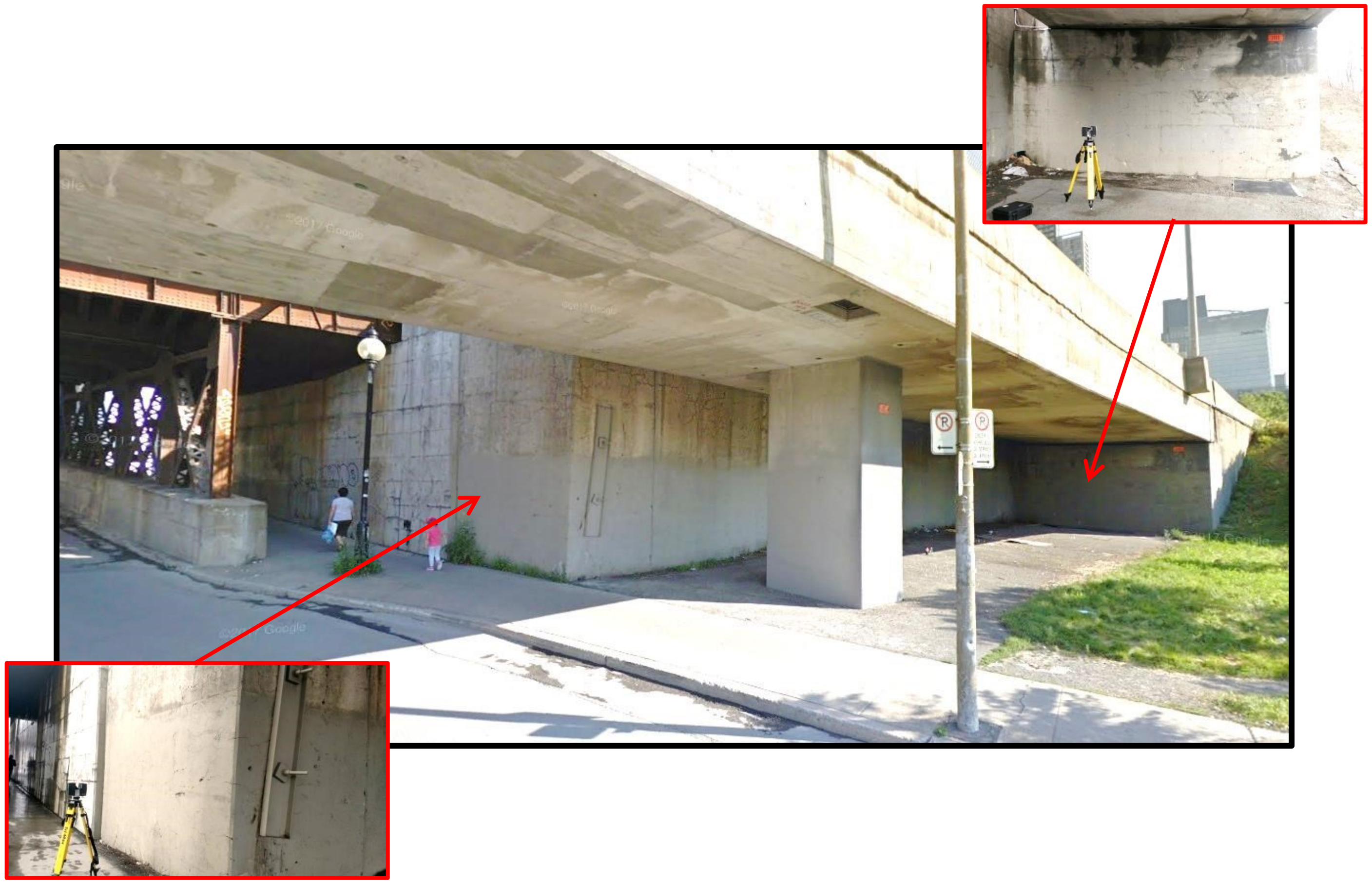


Figure 4-4 Scanning positions on the eastern side of Bridge 1 (Guy Street)

4.3.2 Annotation

The point clouds are split into several parts considering the following specified rules: (1) the number of points in each part must be in a specific range (between 150,000 pts to 400,000 pts); (2) since the blocks have a box shape, the scanned surfaces are divided into rectangular parts; and (3) the size of each part should be big enough to contain different sizes of defects and small enough not to contain more points than the maximum number, especially in the areas with a higher density of points. Moreover, annotating the defects based on their expected patterns leads to more effective learning. Annotation is done in CloudCompare software (Girardeau-Montaut 2015). The prepared dataset contains 193 parts (with the flipped data that will be discussed in Section 4.3.3.1) from the scanned bridges that are categorized into 5 areas and contain 51 million points. 2,572 defects are annotated in the dataset. These defects contain more than 4.1 million points. Figure 4-5 shows a sample of an annotated segment. Figure 4-6 shows the structure of the prepared dataset and the statistical information is given in Table 4-4.

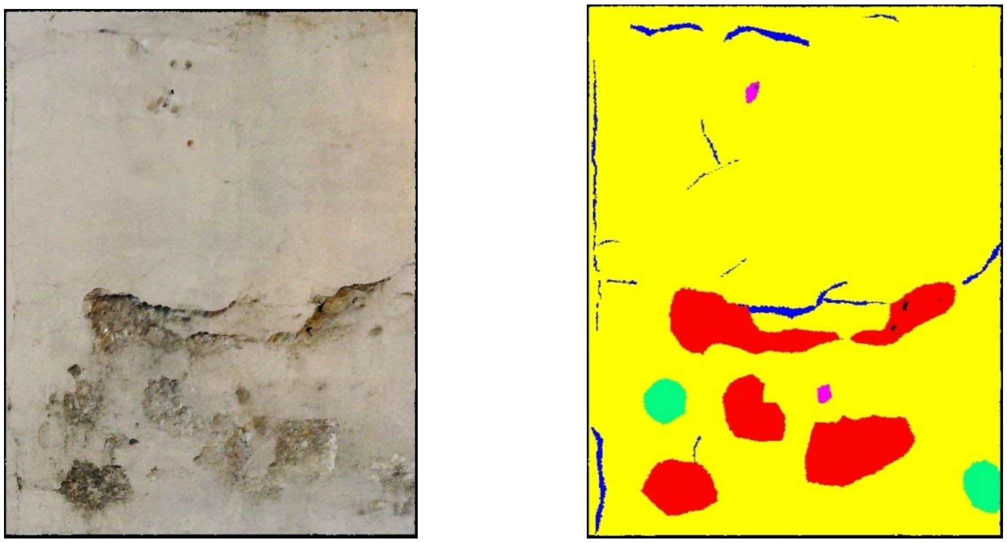

(a) (b)

Figure 4-5 (a) An original point cloud sample, (b) An annotated point cloud sample. No-defect is yellow, crack is blue, light spalling is pink, medium spalling is green, severe spalling is red

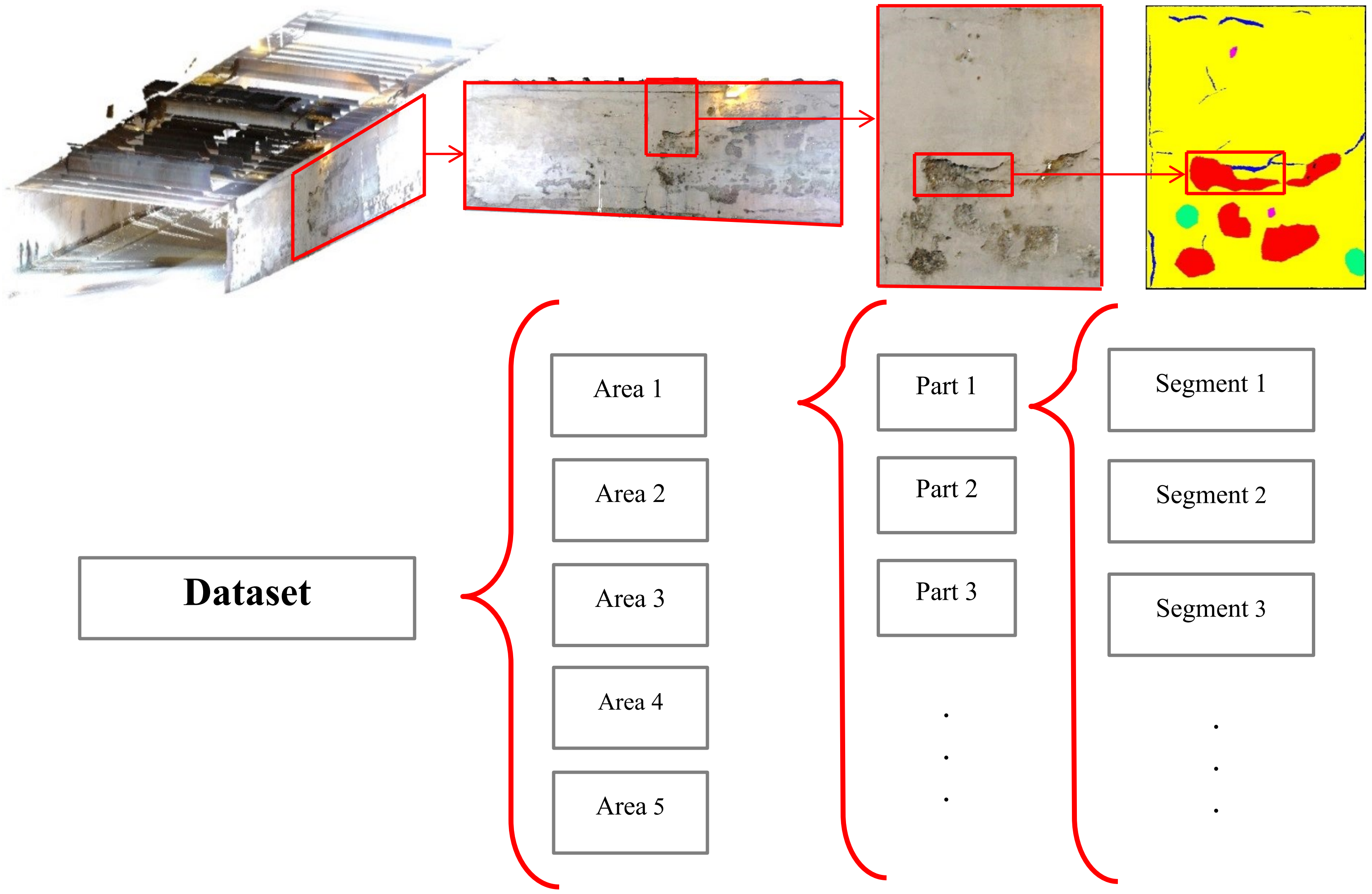


Figure 4-6 The structure of preparing the dataset

Table 4-4 Statistics datasets used in training, evaluation, and testing processes

Datasets (%)	Area	Number of Parts	Number of points	Defects		
				Number	Number of points	Percentage
Training (70.0)	1	19	6,221,356	281	496,351	8.0%
	1-Flipped	19	6,221,356	281	496,351	8.0%
	2	23	5,889,032	304	266,470	4.5%
	2-Flipped	23	5,889,032	304	266,470	4.5%
	3	24	5,796,043	291	511,606	8.8%
	3-Flipped	24	5,796,043	291	511,606	8.8%
Evaluation (24.3)	4	25	6,200,503	330	687,986	11.1%
	4-Flipped	25	6,200,503	330	687,986	11.1%
Testing (5.6)	5	10	2,878,367	160	262,847	9.1%
Total (100)		193	51,092,235	2,572	4,187,673	8.2%

4.3.3 Data Pre-Processing

The annotated dataset is distributed in three categories for training, evaluation, and testing as shown in Table 4-4. As discussed before, CNN needs a large dataset to train an accurate model. In order to enlarge the dataset, an oversampling method of flipping the point cloud data is used in this case study.

4.3.3.1 Oversampling Data by Flipping

The point clouds are flipped with respect to the YZ surface (Figure 4-7). Python code is written for this case study (Appendix D – Python Code for Flipping the Dataset) to flip the training and evaluation parts of the dataset. The flipped data is not used in the testing process. The flipped point clouds of Areas 1 to 4 are added to double the size of the dataset. In this research, 70% of the dataset is set as the training part, 24.4% for evaluation and 5.6% of the dataset is kept unseen to validate the trained model.

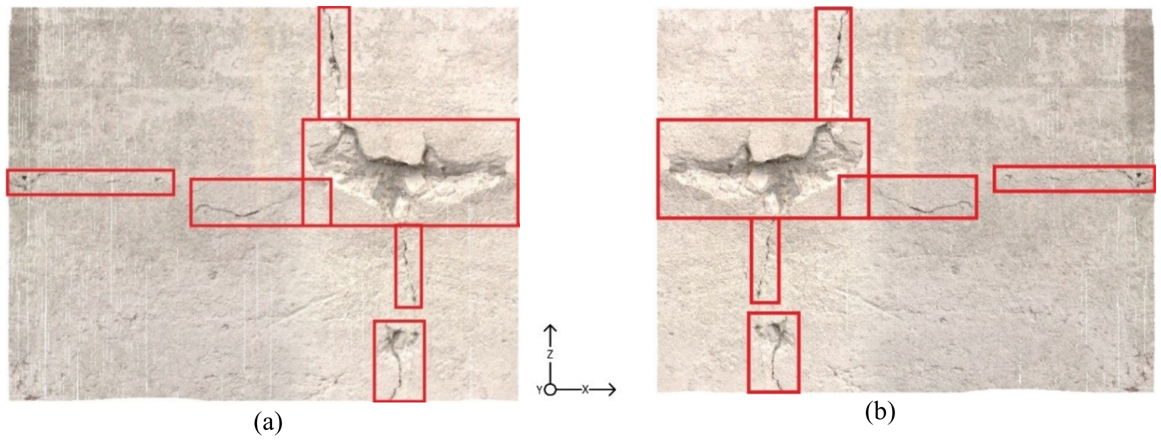


Figure 4-7 (a) Sample of annotated original points segment, (b) Flipped segment

The next steps of data pre-processing are adding the annotated labels (*L*) to the point's arrays and splitting them into blocks. After adding labels to points, the output files are 2D matrices, with each row having seven parameters (*XYZRGBL*) for each point (Qi et al. 2017a). Every part is split into blocks (saved in HDF5 files) and is prepared for the CNN training process. The sizes of blocks of points are defined based on the sizes of the structural defects discussed in Section 4.3.2 (Table 4-1).

The surface density of points varies in different parts of point clouds because of the change in the angle of incidence. This issue has two effects on the processing methods. The first issue is that CNN needs equal-sized input data. PointNet normalizes the number of points in each block to a unique number. This process requires downsampling or upsampling. The second issue is about the difficulty of extracting features from small portions of the dataset using CNN. PointNet++ solves this issue by extracting features from different scales of the input point cloud.

4.3.4 Validation of Binary Classes Semantic Segmentation

In this part of the case study, all the points of the dataset are split into two classes of defect and no-defect, regardless of the types and severity levels of defects. The goal of this section is to validate the proposed method of detecting the defected areas of concrete surfaces of bridges.

In all of the validation parts of this study the accuracy is calculated using Equation 4-2. This equation is selected because the main goal of this study is detecting defect points, which are true positive points (*TP*) and to minimize the number of points that are labeled as defect and detected as no-defect, which are false negative points (*FN*). The summation of *TP* and *FN* points is the number of points that labeled as defect.

$$\boldsymbol{Accuracy = \frac{TP}{TP+FN}} \qquad \textbf{4-2}$$

4.3.4.1 Adapted PointNet

4.3.4.1.1 Training and Evaluation

In order to find an appropriate model for detecting defects in point clouds, several models with different network topologies are trained and evaluated (Table 4-5). Modifying the network's topology is experimental and based on the results and is explained in this section. The evaluation mean loss, the accuracies of defect detection, no-defect detection, and overall accuracy of the models are shown in Figure 4-8 and Figure 4-9.

Model 1 is applied to PointNet with the initial network topology. The overall accuracy is 88.2% and the defect accuracy is 41.1%. As mentioned in Section 4.2.4, since the percentage of the defect points is almost 14% of the whole point cloud and is much less than the no-defect points (86%), a weighted softmax loss function is used in Model 2. The corresponding weight vector is defined based on the points distribution of the classes, which is [0.86, 0.14]. In Model 2, if the evaluation model predicts a point as no-defect by mistake, the weight of its effect is 0.14 on the process of minimizing the calculated loss values and correcting the predictors in the backpropagation process. The defect accuracy in model 2 increased to 64.2%, but the no-defect accuracy decreased from 98.2% to 91.9%. In Model 3, a dropout layer is added and the no-defect accuracy increased by 1.4%. Different combinations of hidden layers, number of neurons, and number of dropout layers are used in Models 4 to 9, but the results did not show any significant improvements. In Model 10, a pyramid shape network topology is used by increasing the number of neurons in the first hidden layer of the third set of MLP to be equal to two times the number of neurons in the last hidden layer of the first set of MLP. This modification increased the accuracy of detecting defects to 70.3%. In Model 11, the input points variables are changed from a 9-dimensional vector to a 7-dimensional vector (the normalized location values of the *x*-axis and *z*-axis are removed as explained in section 4.2.3). This modification improved the accuracy of detecting defects to 75.4%, with an overall accuracy of 86.7%. Model 11 is selected as the best model.

In the best model (Figure 4-10), the first MLP in the classification part has six hidden layers with numbers of neurons 64, 64, 64, 128, 256, and 512. The second MLP has two fully connected layers with numbers of neurons 128, and 64. The local features vector has 512 elements and the global features vector has 64 elements. By concatenating these vectors, an n×576 vector is fed to the segmentation network. The segmentation network is changed to a pyramid neural network and the feature map of dimension n×576 is expanded to n×1024 to extract more features. The segmentation network has four convolutional hidden layers and three dropout layers. The testing results are in Section 4.3.4.1.2. The hyper-parameters of the training network include epoch number, size of blocks, number of points in each block, number of batches for computing at the same time, filter stride size, and learning rate. The size of blocks changed from 1m to 0.1m because the size of the defects is smaller than the size of the building elements. The number of points remained 1,024 but the density increased because of the change in the size of the blocks. The size of the network increased from 8 hidden layers to 11 hidden layers. The number of elements of the local features decreased from 1,024 to 512 and the number of elements of the global features decreased from 128 to 64. The number of batches indicates how many blocks of points are analyzed for learning at the same time. A higher number of batches improves the performance of the model, but the available memory on the hardware constraints the possible volume of computation. A Compute Canada cluster is used to implement this case study. Two NVIDIA P100 Pascal GPUs, 32 CPUs, and 120 GB of memory are used for the training process. The output results of training and evaluation are in Table 4-7.

Table 4-5 Details on the trained and evaluated models for optimization

Model	Loss function	Weight for the loss function	Number of input variables	Classification Network		Segmentation Network	
				First MLP	Second MLP	Third MLP	Number of Dropouts
1	Softmax	N/A	9	5 (64,64,64,128,1024)	2 (256,128)	3 (512,256,2)	N/A
2	Weighted Softmax	[0.86, 0.14]	9	5 (64,64,64,128,1024)	2 (256,128)	3 (512,256,2)	N/A
3	Weighted Softmax	[0.86, 0.14]	9	5 (64,64,64,128,1024)	2 (256,128)	3 (512,256,2)	1
4	Weighted Softmax	[0.86, 0.14]	9	6 (64,64,128,256,512,1024)	2 (256,128)	5 (512,256,64,16,2)	1
5	Weighted Softmax	[0.86, 0.14]	9	6 (64,64,128,256,512,1024)	2 (256,128)	5 (512,256,64,16,2)	3
6	Weighted Softmax	[0.86, 0.14]	9	5 (64,64,64,128,512)	2 (128,64)	3 (256,128,2)	1
7	Weighted Softmax	[0.86, 0.14]	9	5 (64,64,64,128,512)	2 (128,64)	3 (256,128,2)	2
8	Weighted Softmax	[0.86, 0.14]	9	6 (64,64,64,128,256,512)	2 (128,64)	4 (256,128,32,2)	3
9	Weighted Softmax	[0.86, 0.14]	9	4 (64,64,128,512)	2 (128,64)	3 (256,128,2)	1
10	Weighted Softmax	[0.86, 0.14]	9	6 (64,64,64,128,256,512)	2 (128,64)	4 (1024,512,128,2)	3
11	Weighted Softmax	[0.86, 0.14]	7	6 (64,64,64,128,256,512)	2 (128,64)	4 (1024,512,128,2)	3

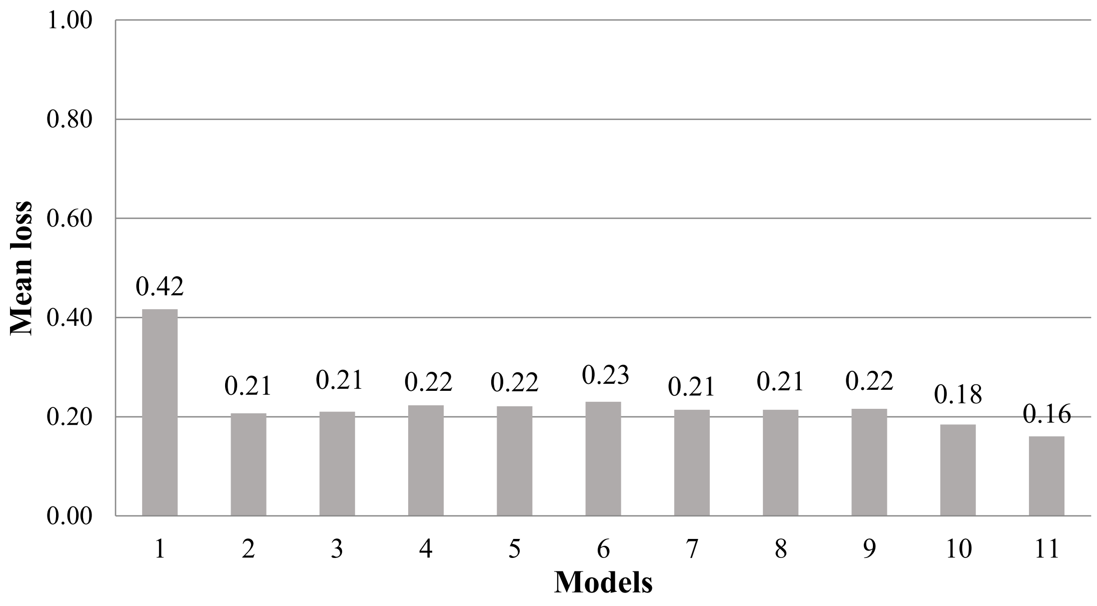


Figure 4-8 Evaluation mean loss of the models

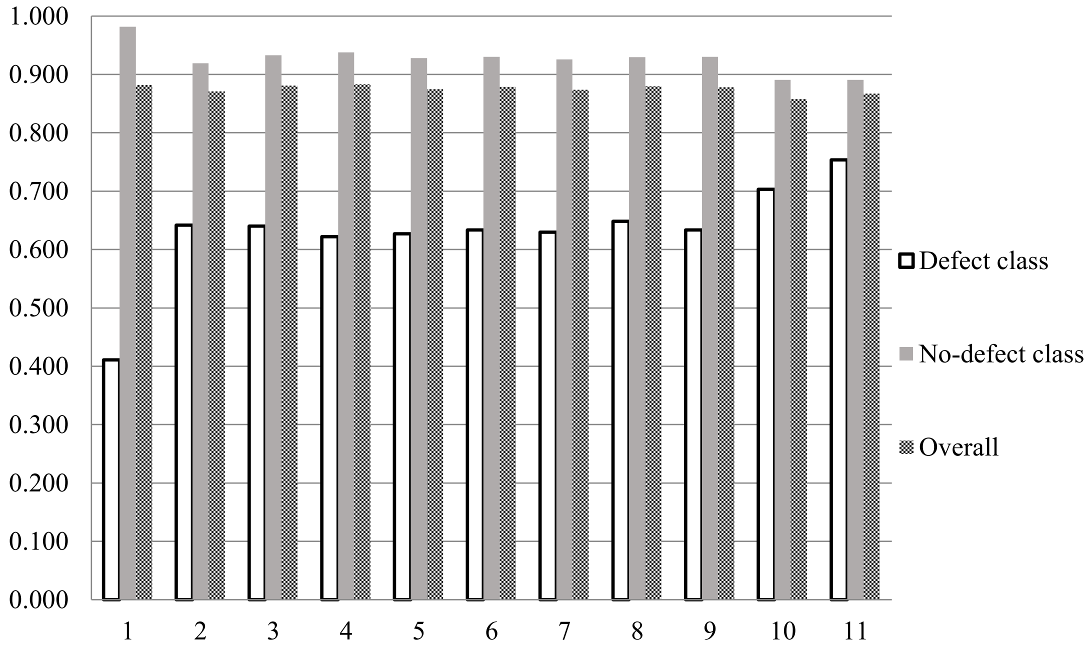


Figure 4-9 Evaluation accuracies of defect class, no-defect class, and overall

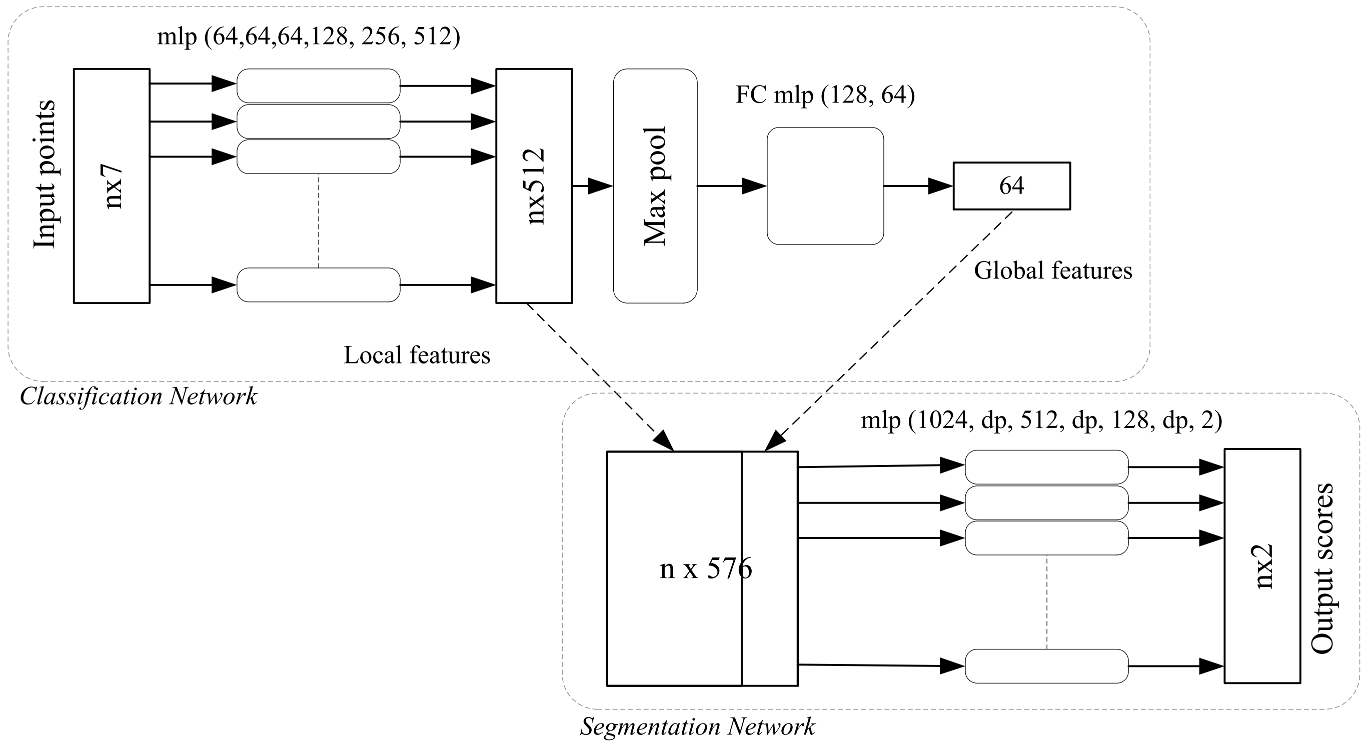


Figure 4-10. Modified PointNet network

Table 4-6 PointNet model's hyperparameters

Parameter	PointNet	Modified Network
Convolving direction	XY surface	XZ surface
Size of blocks (X,Y,Z)	1m , 1m , Z_{max}	0.1m , Y_{max} , 0.1m
Number of points in each block	1,024	1,024
Size of the network	8 layers (1 dropout)	11 layers (3 dropouts)
Local / Global features	1024 / 128	512 / 64
Number of epochs	50	50
Learning rate	1e-3 (decays exponentially to minimum of 1e-5)	1e-3 (decays exponentially to minimum of 1e-5)
Optimizer	Adam	Adam
Weight vector for loss function	Softmax cross entropy	Weighted Softmax cross entropy [0.86, 0.14]
Flipping training dataset	N/A	Yes

Table 4-7 Results of training and evaluation of the selected model

Process		**Value**
Training	Mean loss	0.051
	Overall accuracy	0.911
Evaluation	Mean loss	0.160
	Overall accuracy	0.867
	Accuracy of defect	0.754

4.3.4.1.2 Testing

The number of points in each block and the block size of the testing process must be the same as those of the training process. The dataset of Area 5 is used for testing the trained model.

The statistical results of the testing are shown in Table 4-8 and Figure 4-11. Ten parts are used as the testing area. Each part has defects with different depths. The depth of the part's bounding box is assumed as the maximum depth of the segment's defects. The accuracies of the detected defects of the ten parts are shown in Table 4-8. The parts are sorted based on the depth of the defects. As expected, the detection results are better in the case of deeper defects. The accuracies of detecting defects in the first two parts, which have defects with the depth of less than 2 cm, are less than 20%. For the parts with deeper defects, the accuracy gets larger than 50% and reaches the value of 89.2% for the part with deepest defects. The overall trend show the value accuracy increase by increasing the depth of the defects (as it will be discussed in Section 4.3.4.4) but there are some fluctuations in different parts (e.g. from part 4 to part 5). The average accuracy of detecting defects is 74.9%. The results of the test are visualized in Table 4-12.

Table 4-8 The output results of the detecting defects (Adapted PointNet)

Part	Number of defects	Number of points	Area (cm^2)	Depth (cm)	Mean loss	Accuracy	Accuracy of defects	Accuracy of non-defects	IOU of defects	Precision of defects	F1 score of defects
1	30	327,777	6,248	1.5	0.162	0.884	0.132	0.935	0.067	0.122	0.126
2	16	195,140	5,957	1.6	0.328	0.863	0.195	0.958	0.150	0.395	0.261
3	5	153,353	18,490	2.6	0.035	0.949	0.523	0.961	0.223	0.280	0.365
4	24	486,062	6,607	2.8	0.120	0.749	0.661	0.758	0.199	0.221	0.332
5	10	166,446	19,338	3.5	0.168	0.904	0.537	0.940	0.337	0.475	0.504
6	15	166,667	16,152	3.7	0.139	0.790	0.596	0.819	0.268	0.328	0.423
7	19	184,671	20,104	4.1	0.228	0.773	0.735	0.789	0.494	0.602	0.662
8	5	291,776	13,756	5.1	0.050	0.943	0.667	0.955	0.325	0.387	0.490
9	13	259,111	42,174	6.0	0.080	0.858	0.869	0.856	0.462	0.496	0.632
10	16	278,342	47,597	8.1	0.104	0.852	0.892	0.846	0.466	0.494	0.636
Weighted average						0.862	0.749	0.875	0.408	0.473	0.580

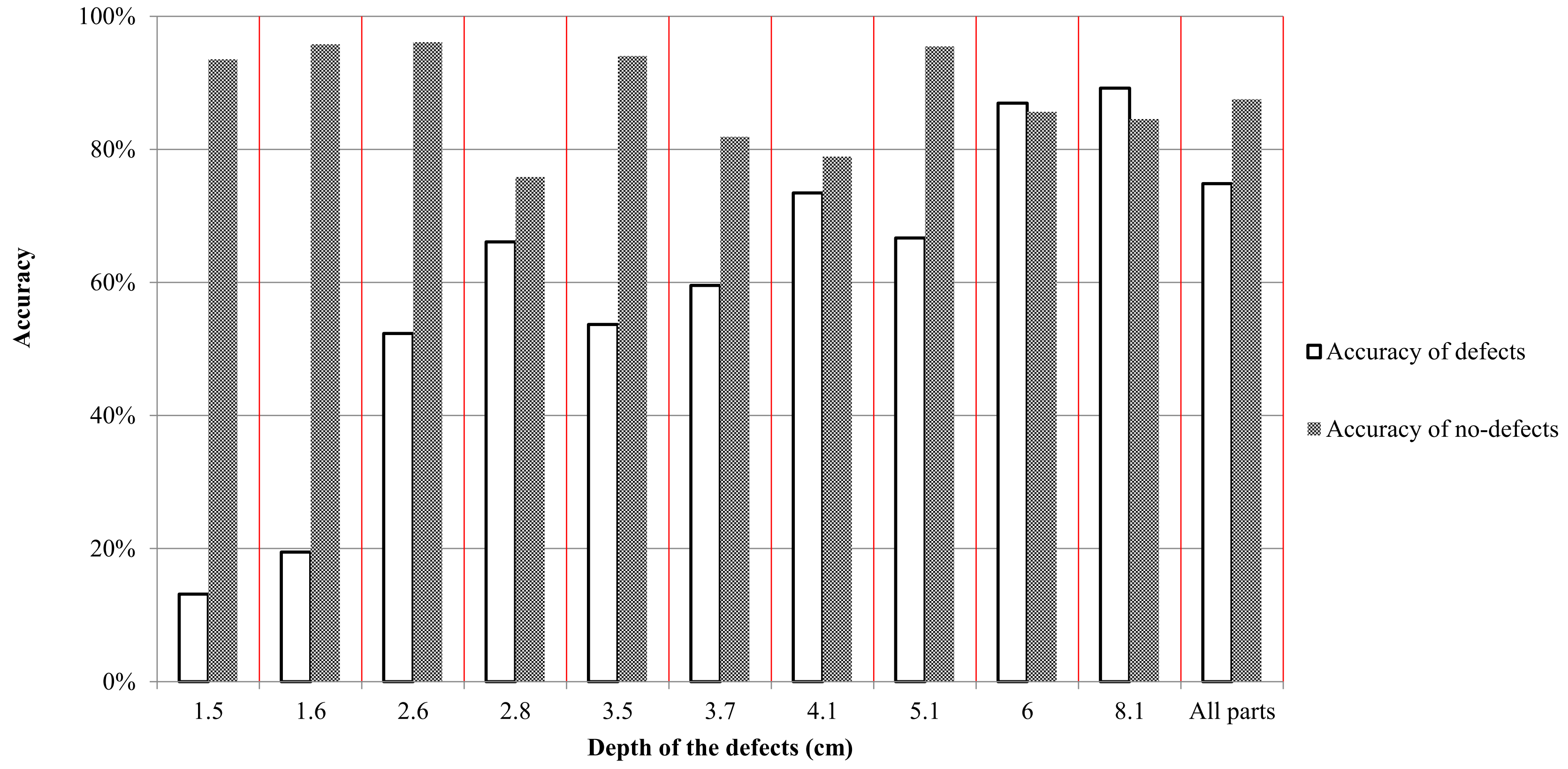


Figure 4-11 **The chart of output results of the detecting defects (Adapted PointNet).**

4.3.4.2 Adapted PointNet++

4.3.4.2.1 Training and Evaluation

PointNet++ is also adapted to detect defects in point clouds. The neural network topology of PointNet++ is kept as-is because it is working properly with the same adapted data-preprocessing used for PointNet (4.3.3). The network is fed by 7-dimensional input data. The adapted PointNet++ (Figure 4-12) has two main sets of hidden layers. The first hidden layer has four sub-layers; each applies a sampling method and a mini-PointNet unit recursively. The sampling sizes of 5, 10, 20, and 30 centimeters were used. Each one is followed by three-layer PointNet to extract the feature vectors. The second hidden layer contains four sets of grouping algorithms; each one is followed by a two-layer PointNet unit. The grouping units concatenate the outputs of the sampling units. The network is followed by two fully connected layers with a dropout in the middle and extracts the kernel values for each class. The training and evaluation results are in Table 4-9. As shown, by using PointNet++ the training overall accuracy, the evaluation overall accuracy, and the accuracy of detecting defects increase respectively 7.7%, 10%, and 6.4% compared to the adapted PointNet results. The hyperparameters of the PointNet++ model are in Table 4-10.

Table 4-9 Results of training and evaluation of the selected model

Process		**Value**
Training	Mean loss	0.052
	Overall accuracy	0.988
Evaluation	Mean loss	0.114
	Overall accuracy	0.967
	Accuracy of defect	0.818

Table 4-10 PointNet++ model's hyperparameters

Parameter	PointNet++	Modified Network
Convolving direction	XY surface	XZ surface
Size of blocks (X,Y,Z)	1.5m , 1.5m , Z_{max}	0.4m , Y_{max} , 0.4m
Input variables	9-dim ($XYZRGBN_xN_yN_z$)	7-dim ($XYZRGBN_y$)
Number of points in each block	8,192	12,288
Sampling sizes (m)	0.1, 0.2, 0.4, 0.8	0.5, 0.1, 0.2, 0.3
Number of epochs	200	50
Learning rate	1e-3 (decays exponentially to minimum of 1e-5)	1e-3 (decays exponentially to minimum of 1e-5)
Optimizer	Adam	Adam
Weight vector for loss function	Weighted Softmax cross entropy	Weighted Softmax cross entropy
Flipping training dataset	N/A	Yes

4.3.4.2.2 Testing

The accuracies of the detected defects of the ten parts are shown in Table 4-11 and Figure 4-13. Each part has defects with different depths. The results of this model also show the expected sensitivity to the depth of the defects. The accuracies of detecting defects on the first four parts, which have defects with the depth of less than 3 cm, are less than 20%. The accuracies increase to more than 60% for parts with deeper defects up to 93.4%. There is a jump in the accuracy of detecting defects in Figure 4-13 after Part 4. The results compared to the adapted PointNet model (Figure 4-11) show good improvement in the average accuracy and accuracies of the deepest six parts but for the first four parts adapted PointNet is performing better. The average accuracy of detecting defects on all the ten parts is 78.8%, which is 3.9% more than the adapted PointNet model. The average accuracy of detecting defects on the six deepest parts is 86.2% and 7.4% more than the average of all the ten parts.

The results of the test are visualized in Table 4-12. As shown the results of the adapted PointNet++ network are more precise than the adapted PointNet network in detecting the edges of the defects (last six parts). PointNet++ uses the point's information from four different scales and extracts local features from these different scales. It enables the model to learn geometrical features precisely, even for small size defects. Also, as shown in Table 4-12, the results of the first four parts are not good. It is expected that by increasing the size of the dataset it may work better on detecting defects with a depth of less than 3 cm too.

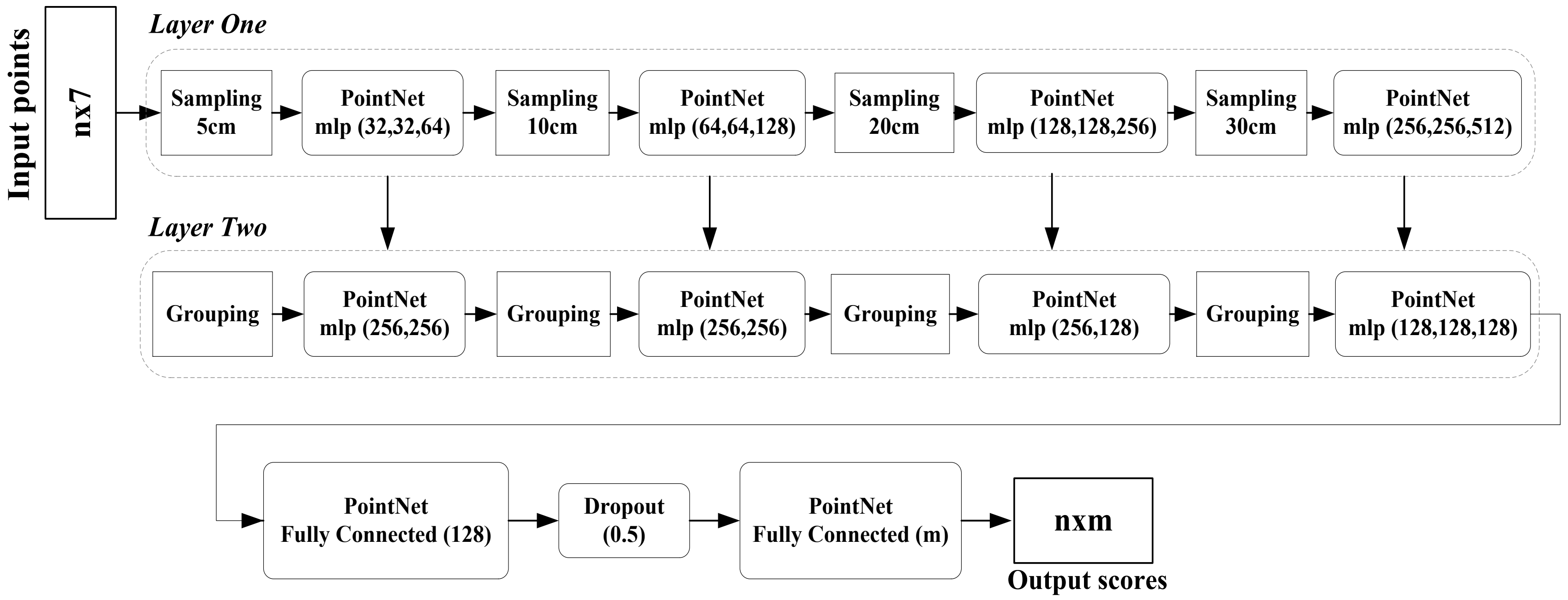


Figure 4-12 Adapted PointNet++ network

Table 4-11. The output results of the detecting defects (Adapted PointNet++).

Part	Number of defects	Number of points	Area (cm^2)	Depth (cm)	Mean loss	Accuracy	Accuracy of defects	Accuracy of non-defects	IOU of defects	Precision of defects	F1 score of defects
1	30	327,777	6,248	1.5	0.072	0.964	0.071	0.980	0.020	0.028	0.040
2	16	195,140	5,957	1.6	0.766	0.910	0.024	1.000	0.024	0.856	0.047
3	5	153,353	18,490	2.6	0.266	0.950	0.192	0.986	0.148	0.388	0.257
4	24	486,062	6,607	2.8	0.358	0.932	0.122	0.998	0.119	0.801	0.212
5	10	166,446	19,338	3.5	0.291	0.903	0.639	0.930	0.376	0.478	0.547
6	15	166,667	16,152	3.7	0.239	0.857	0.691	0.869	0.252	0.284	0.403
7	19	184,671	20,104	4.1	0.335	0.798	0.888	0.765	0.544	0.585	0.705
8	5	291,776	13,756	5.1	0.060	0.970	0.798	0.983	0.560	0.652	0.718
9	13	259,111	42,174	6.0	0.140	0.917	0.881	0.923	0.583	0.633	0.737
10	16	278,342	47,597	8.1	0.172	0.915	0.934	0.912	0.577	0.601	0.732
Weighted average						0.917	0.788	0.932	0.482	0.563	0.656
Weighted average of the six deepest parts						0.900	0.862	0.905	0.567	0.862	0.684

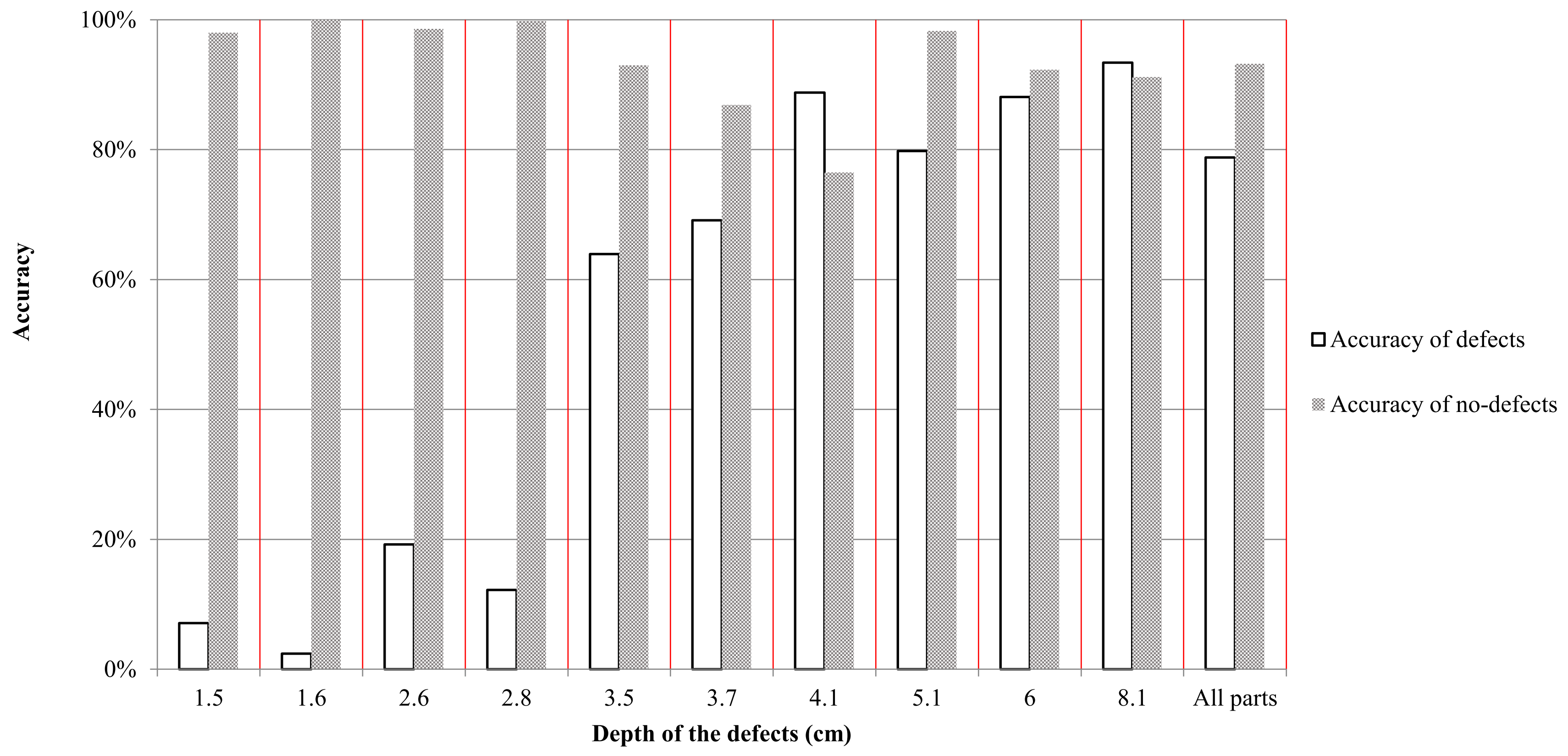


Figure 4-13 **The chart of output results of the detecting defects (Adapted PointNet++).**

Table 4-12 Binary semantic segmentation results

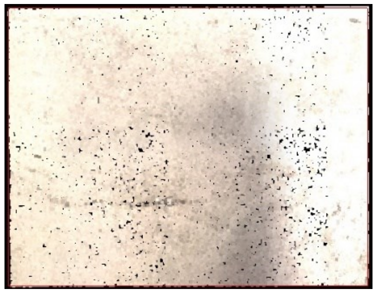
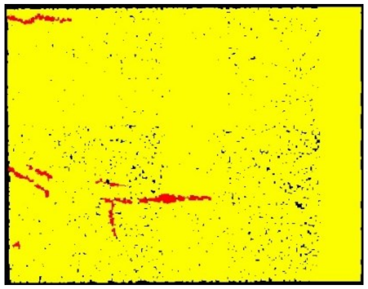
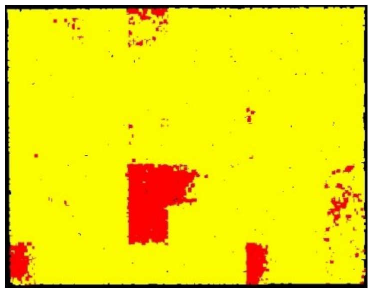
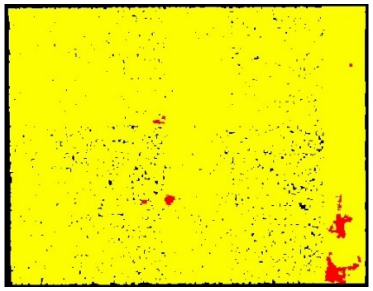
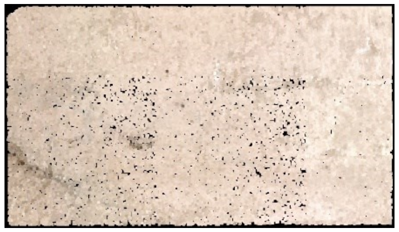
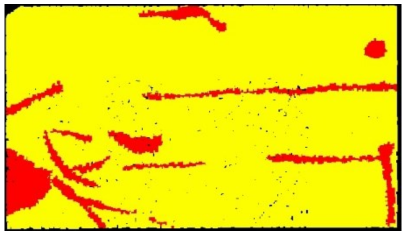
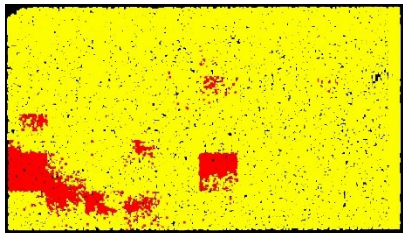
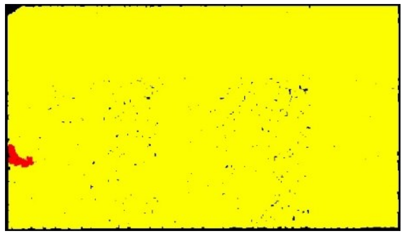
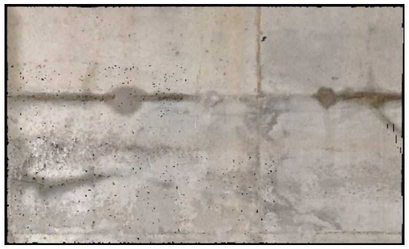
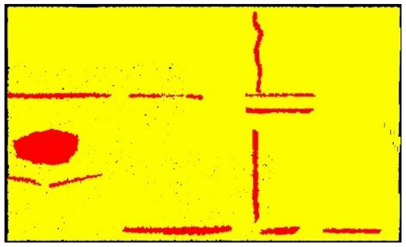
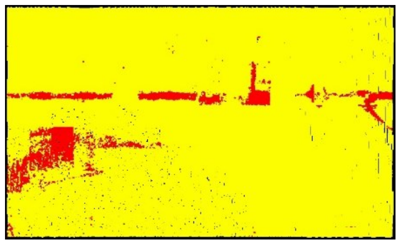
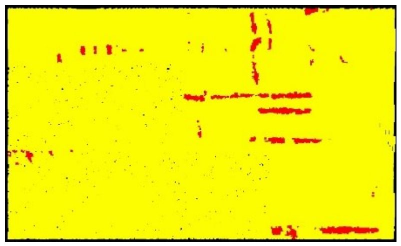
Part	Original point cloud	Manual annotation	Adapted PointNet	Adapted PointNet++
1				
2				
3				

Table 5-13 Binary semantic segmentation results (continue)

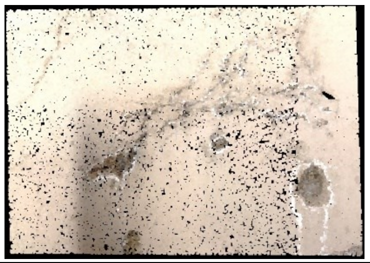
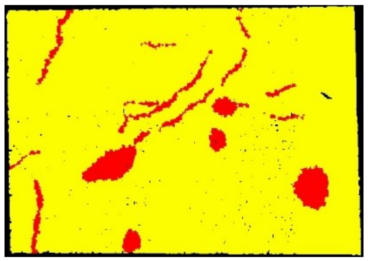
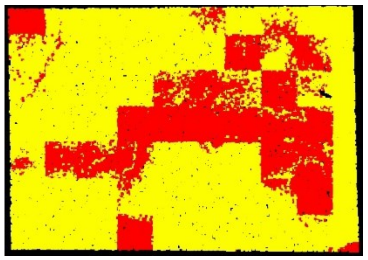
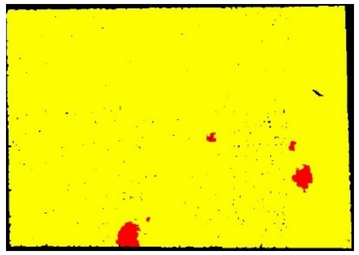
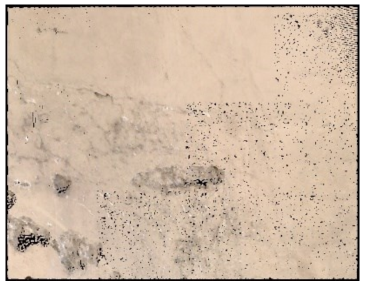
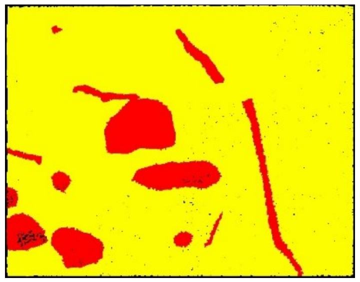
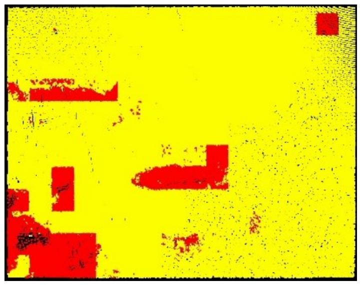
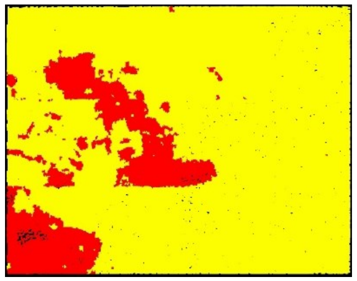
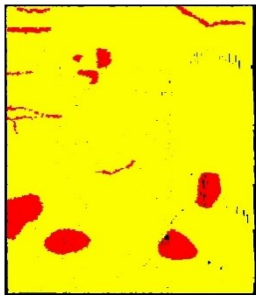
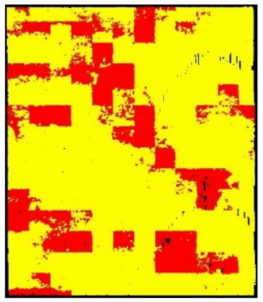
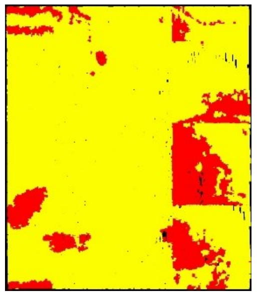
Part	Original point cloud	Manual annotation	Adapted PointNet	Adapted PointNet++
4				
5				
6				

Table 5-13 Binary semantic segmentation results (continue)

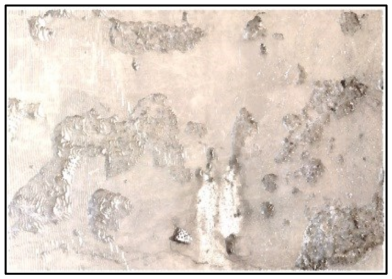
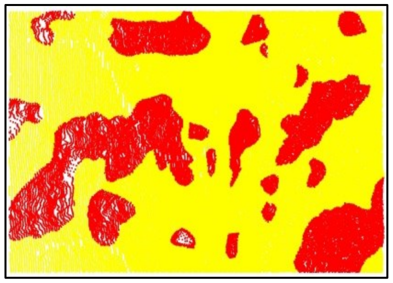
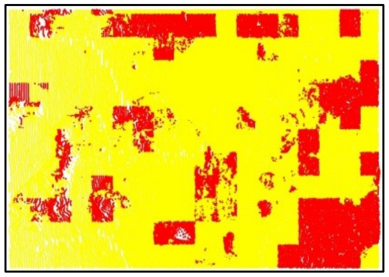
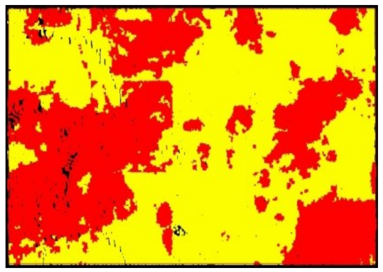
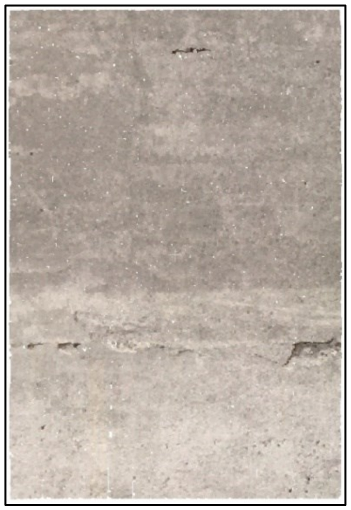
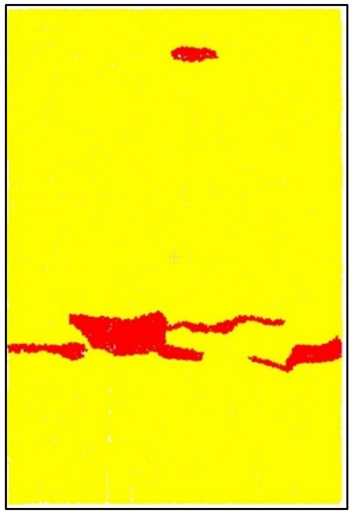
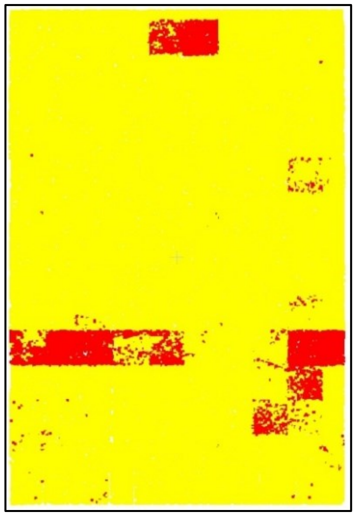
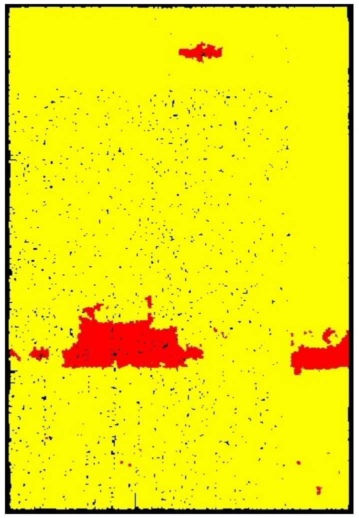
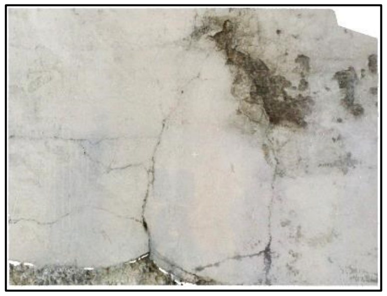
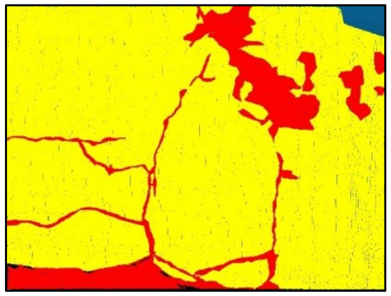
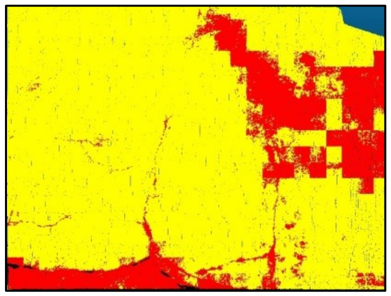
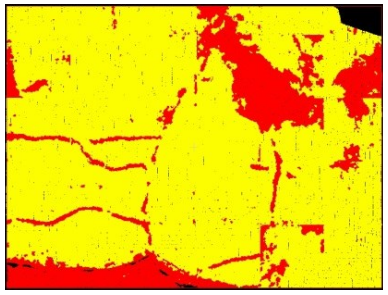
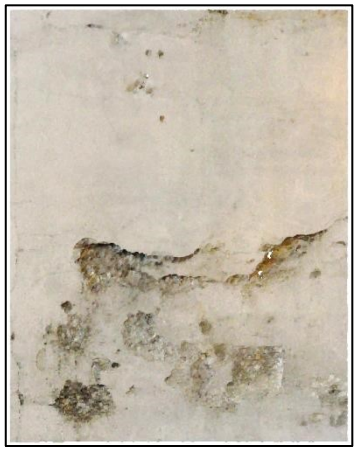
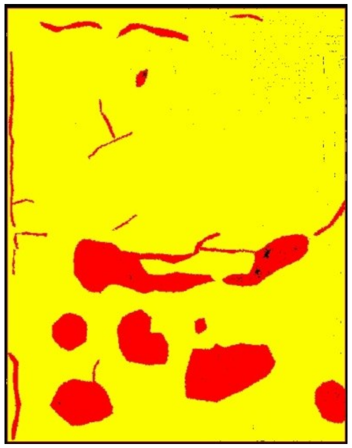
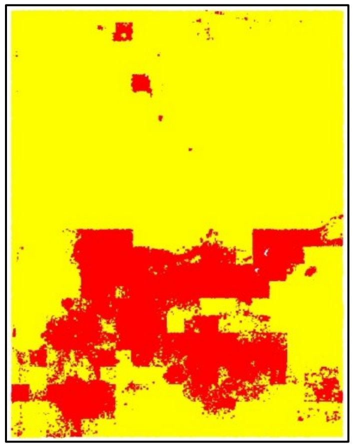
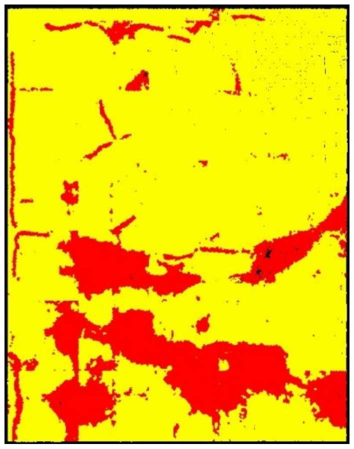
Part	Original point cloud	Manual annotation	Adapted PointNet	Adapted PointNet++
7				
8				

Table 5-13 Binary semantic segmentation results (continue)

Part	Original point cloud	Manual annotation	Adapted PointNet	Adapted PointNet++
9				
10				

4.3.4.3 PointNet Sensitivity Analysis

The main goal of the sensitivity analysis in this study is to determine the impact of different input variables on the results based on the quantitative information. A 3D point cloud dataset has various volume density and surface density rates in every specific segment. The overall density of a point cloud depends on the LiDAR scanner resolution. For every segment, the density of the points also depends on the angle of view of the LiDAR. To apply the convolutional network on point clouds, PointNet equalizes the surface density rates of blocks, by defining the size of blocks and the number of points in each block. In the sensitivity analysis of the proposed model, hyper-parameters of the number of points and block size are studied. Also, the effect of changes in point density ratio is studied. This ratio is calculated by dividing the number of points in each block over the surface area of the block (Equation 4-3).

$$\boldsymbol{Density\ Ratio} = \frac{\boldsymbol{Number\ of\ points}}{(\boldsymbol{Block\ size})^2} \qquad \textbf{4-3}$$

Table 4-13 shows the areas, number of points, and the density ratios of all the areas of the raw dataset. Mean density ratio is 11.425 pts/cm^2. The “number of points” parameter in the sensitivity analysis has to be chosen based on the density ratio of the raw datasets to decrease the probability of upsampling in the data pre-processing step (4.2.3). In order to estimate the proper values for the block size parameter, the average size of the annotated defects bounding boxes of the first four areas (used for training and evaluation) are calculated (Table 4-14). The last column of Table 4-14 is showing the square root of the related area. It shows the regulated dimension of the defect if we assume it as a square. The average width, height and regulated dimensions of the defects are 23.3, 18.7, and 22.4 centimeters respectively.

Table 4-13 Density ratio of the raw dataset

Dataset	Surface area (cm2)	Number of points (pts)	Density ratio (pts/cm2)
Area1	251,168	2,991,923	11.912
Area2	313,029	3,077,769	9.832
Area3	154,294	2,239,013	14.511
Area4	263,598	2,522,791	9.571
Area5	217,881	2,878,641	13.212
Total	1,199,971	13,710,137	11.425

Table 4-14 Annotated defects dimensions

Dataset	Width(cm)	Height(cm)	Area(cm2)	Dimension(cm)
Area1	33.6	14.8	561.4	23.7
Area2	12.3	20.9	555.0	23.6
Area3	8.0	6.0	74.8	8.6
Area4	39.4	33.1	819.6	28.6
Average	23.3	18.7	502.7	22.4

Sixteen cases are defined for four different numbers of points (1024, 2048, 4096, 6144) and four block sizes (10, 20, 30, 40 centimeters). Selected values for these two parameters are related to the values of the raw dataset for the sensitivity analysis. In these cases, the stride size is equal to the block size, which means there is no overlapping in the training datasets and convolutions. The number of batches is 24 and the initial learning rate is 0.001. The learning rate decays 50% every 8 epochs until it reaches the minimum value of 1e-5. The calculated accuracies and mean losses for training, validation, and testing processes are shown in Table 4-15. As shown before (Table 4-6) there are several hyperparameters in the model. The size of the block and the number of points in each block are the two hyperparameters that used in this sensitivity analysis. All the other hyperparameters are not changed from the best modified model (Figure 4-10). The average accuracies are compared to each other based on the block sizes and the numbers of points in Figure 4-14 and Figure 4-15 respectively.

Table 4-15 Results of the sixteen studied cases

Case	Parameters			Train		Evaluation				Test (Weighted average)					
	Block size (cm)	**Number of points**	**Density ratio (pts/cm2)**	**Accuracy**	**Mean Loss**	**Accuracy**	**Mean Loss**	**Defect Acc.**	**Non-Defect Acc.**	**Accuracy**	**Defect Acc.**	**Non-Defect Acc.**	**Defect IoU**	**Defect Precision**	**Defect F1**
A	10	1,024	10.2	0.911	0.051	0.867	0.160	0.754	0.891	0.862	0.744	0.878	0.393	0.455	0.564
B	10	2,048	20.5	0.908	0.054	0.869	0.148	0.756	0.893	0.857	0.740	0.873	0.383	0.443	0.554
C	10	4,096	41.0	0.910	0.053	0.867	0.166	0.722	0.899	0.848	0.723	0.865	0.363	0.422	0.533
D	10	6,144	61.4	0.908	0.053	0.863	0.157	0.739	0.889	0.850	0.741	0.865	0.373	0.429	0.543
E	20	1,024	2.6	0.914	0.048	0.839	0.162	0.725	0.863	0.857	0.672	0.883	0.361	0.438	0.531
F	20	2,048	5.1	0.914	0.049	0.824	0.169	0.706	0.850	0.859	0.648	0.887	0.355	0.439	0.524
G	20	4,096	10.2	0.914	0.050	0.839	0.162	0.749	0.858	0.849	0.713	0.867	0.361	0.422	0.530
H	20	6,144	15.4	0.918	0.047	0.822	0.176	0.747	0.840	0.845	0.718	0.866	0.347	0.422	0.525
I	30	1,024	1.1	0.910	0.054	0.813	0.168	0.699	0.834	0.859	0.568	0.897	0.314	0.413	0.478
J	30	2,048	2.3	0.914	0.052	0.834	0.147	0.728	0.855	0.855	0.630	0.884	0.330	0.409	0.496
K	30	4,096	4.6	0.911	0.050	0.818	0.152	0.752	0.830	0.864	0.617	0.896	0.339	0.430	0.507
L	30	6,144	6.8	0.908	0.053	0.810	0.158	0.733	0.824	0.842	0.628	0.870	0.311	0.381	0.474
M	40	1,024	0.6	0.910	0.053	0.782	0.173	0.703	0.796	0.849	0.650	0.875	0.334	0.408	0.501
N	40	2,048	1.3	0.908	0.054	0.777	0.155	0.669	0.797	0.861	0.584	0.898	0.330	0.431	0.496
O	40	4,096	2.6	0.902	0.054	0.732	0.161	0.723	0.734	0.845	0.610	0.876	0.314	0.393	0.478
P	40	6,144	3.8	0.907	0.729	0.777	0.157	0.700	0.791	0.862	0.614	0.894	0.340	0.432	0.507

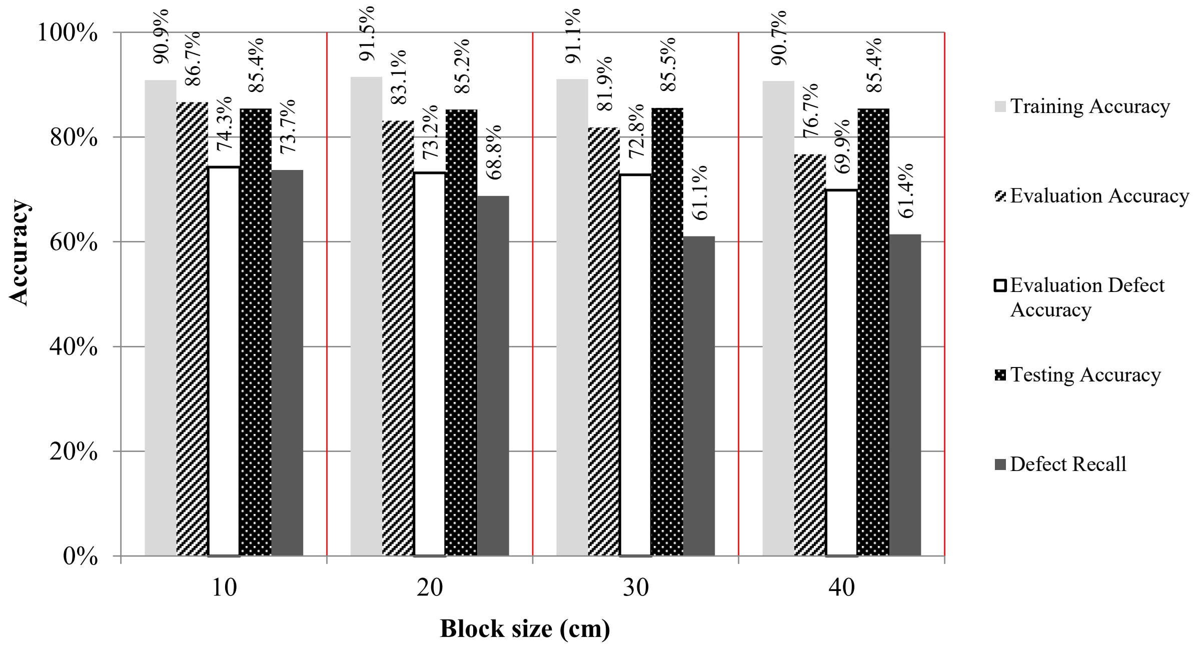


Figure 4-14 Accuracies of the studied cases based on block sizes

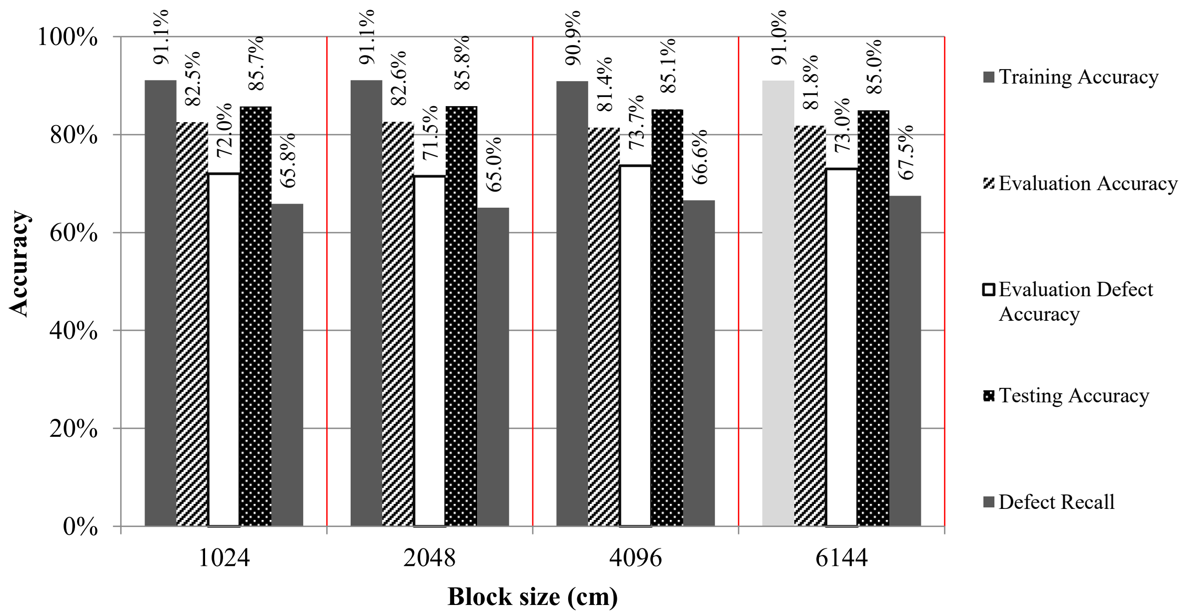


Figure 4-15 Accuracies of the studied cases based on the number of points in each block

4.3.4.4 Effect of the depth of defects

Ten parts are used as the testing area of the dataset. Each part has defects with different depths. The depth of the segment's bounding box is assumed as the maximum depth of the part's defects. Accuracies of the detected defects of all the 16 cases are shown in Table 4-16. Average accuracies of defect detection are compared based on the sizes of defects in Figure 4-16. As shown in the figure, the detection is better in large-sized defects. This is expected because the main feature of the defects is geometry and it is learned by the model in the training step. In this sensitivity analysis, all kinds of defects are categorized into one class, which has different dimensions. As an example, the size and depth of a crack are much smaller than a spalling defect. By adding more classes to the training datasets, this method would perform better in detecting the specified defects.

Table 4-16 Accuracy of defect detection based on the depth of defects

		Case																Average
		A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	
Depth (cm)	**D>6**	0.788	0.660	0.616	0.688	0.487	0.671	0.696	0.659	0.764	0.819	0.815	0.831	0.892	0.902	0.903	0.912	0.756
	6≥D>5	0.674	0.565	0.579	0.624	0.795	0.857	0.755	0.802	0.731	0.677	0.774	0.796	0.768	0.762	0.784	0.778	0.733
	5≥D>4	0.608	0.494	0.509	0.583	0.493	0.540	0.548	0.551	0.535	0.490	0.638	0.481	0.634	0.600	0.539	0.556	0.550
	4≥D>3	0.403	0.400	0.425	0.423	0.568	0.537	0.548	0.622	0.531	0.482	0.581	0.599	0.566	0.541	0.531	0.533	0.518
	3≥D	0.477	0.578	0.417	0.534	0.449	0.404	0.298	0.396	0.504	0.477	0.490	0.529	0.378	0.403	0.415	0.454	0.450

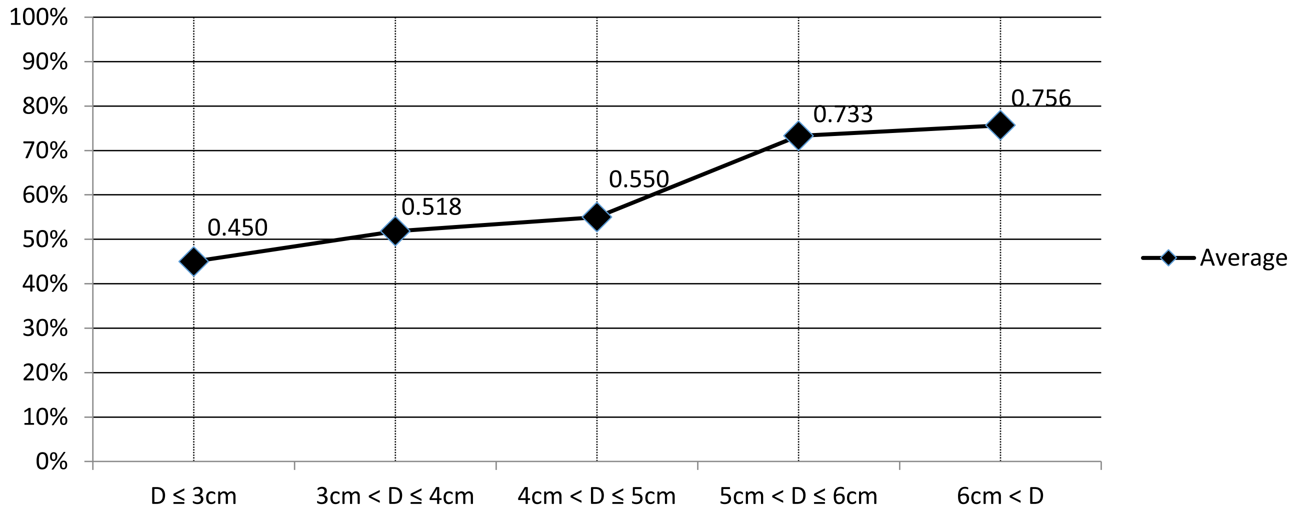


Figure 4-16 Accuracy of detecting defects compared to the sizes of defects

To find out the impact of density changes on the results, the 16 cases with different settings of block's point densities (between 0.6 and 61.4 pts/cm^2) are studied. Figure 4-17 shows the impact of the changes in the block densities on the accuracy of defects with respect to the minimum density. By increasing the density, first, the accuracy of the defects extremely fluctuates. After reaching about 10 pts/cm^2, a steady trend appears within the range of 10 to 15% higher accuracy. Since the average point density of the raw dataset is 11.4 pts/cm^2, it can be concluded that increasing the block's density more than the density of the actual dataset (upsampling) does not have a considerable effect on the results. The maximum accuracy in detecting defects happens in Case G that has a point density of 10.2 pts/cm^2.

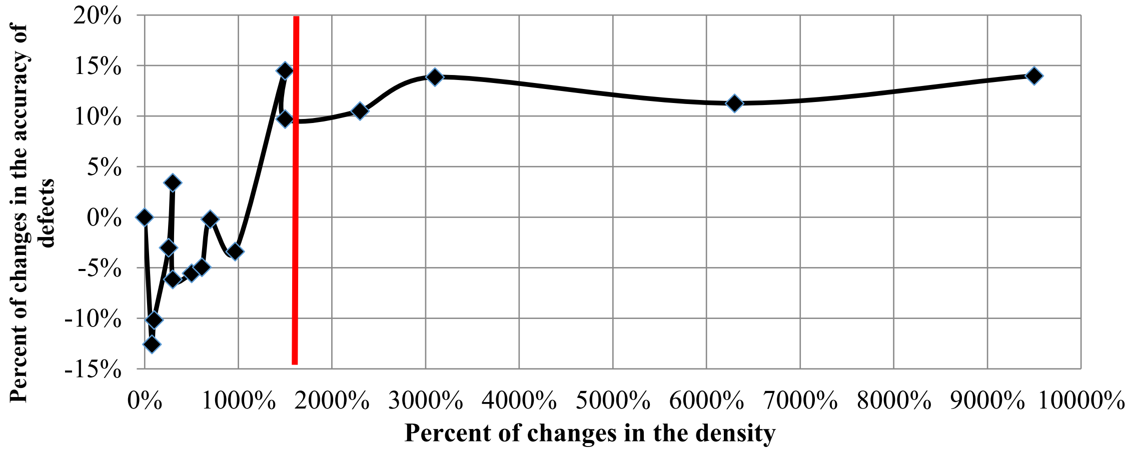


Figure 4-17 **Effect of changing density on the accuracy of defects**

4.3.5 Validation on Detecting Types of Defects Using Adapted PointNet++ Model

The dataset is annotated into five categories of crack, light spalling, medium spalling, severe spalling, and no-defect. In this part of the case study, the points of the dataset are split into three classes of crack, spalling, and no-defect. It means that the severity level of the spalling defects is not considered and all of them are considered as one class. The goal of this section is to validate the proposed method of detecting the type of defects of concrete surfaces of bridges. As shown in Figure 4-18, the number of cracks points is much less than the other two classes, especially the no-defect class. Cracks, spalling, and no-defect classes have 0.8%, 7.4%, and 91.8% of all the points. The method of using a cost-sensitive loss function is chosen to deal with this imbalanced class issue in PointNet++ method. Equation 4-4 indicates the function that is used for calculating the weight of classes in evaluation. In the equation, w_i is the weight of class *i*. The resulted cost weights of the classes are tabulated in Table 4-17. It means that the calculated loss value of each class is multiplied by this number, so the effect of the smaller class is more than the larger one. This process makes the model learn from each class equally, despite the number of points of the classes.

$$\boldsymbol{Cost\ Weight_i = \frac{1}{\log(1.05 + w_i)}} \qquad \textbf{4-4}$$

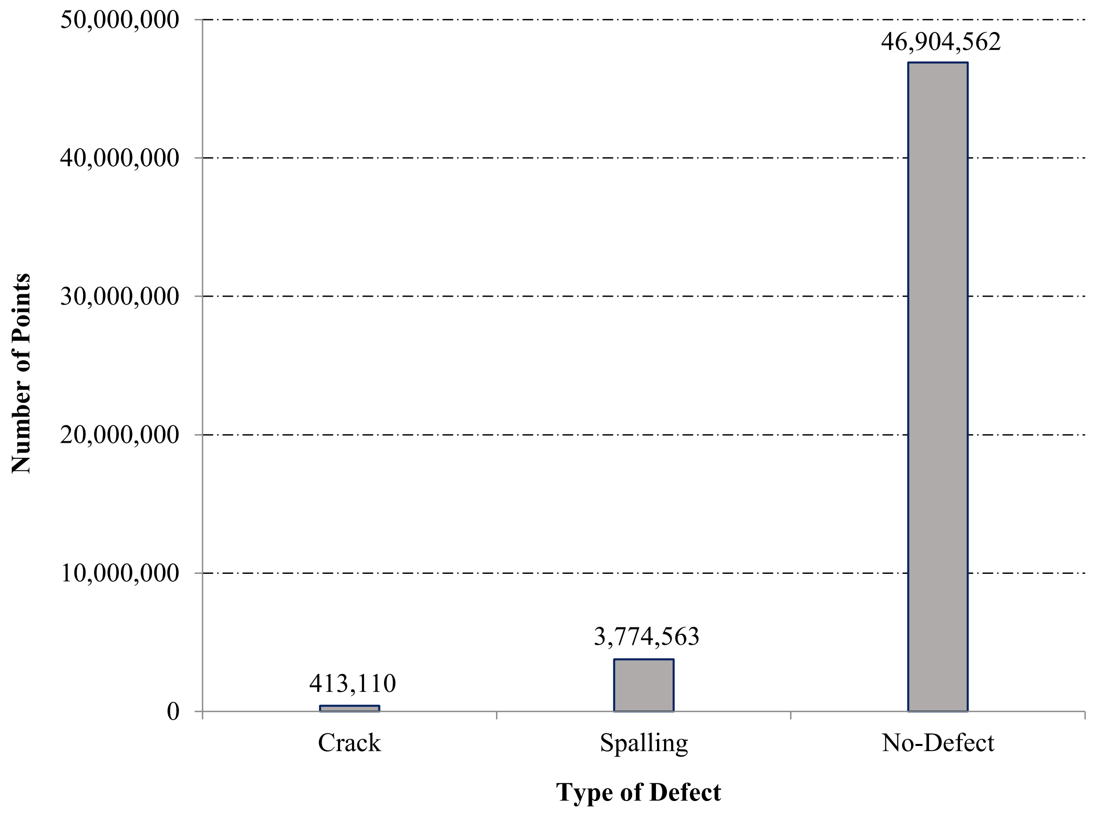


Figure 4-18 Number of points of annotated classes.

Table 4-17 Cost weight of the tree classes

Class	**Crack**	**Spalling**	**No-Defect**	**Summation**
Number of points	413,110	3,774,563	46,904,562	51,092,235
Number of points weight	0.008	0.074	0.918	1.0
Cost weight	40.8	19.7	3.4	NA

The statistics of the output results of the training and evaluation are in Table 4-18 and the testing are in Table 4-19 and Figure 4-19. The performance of the model is not good for the first four parts. The parts are sorted by the maximum depth of the parts' defects. So, the model does not perform well on the parts with defects with a depth of fewer than three centimeters. The accuracy of detecting cracks on the parts deeper than three centimeters is 47%. The numbers of points of cracks data are very small in the training, evaluation, and even testing area. So, it was expected to not have satisfying results in this class. Accuracy of detecting cracks on parts number 9 and 10 are higher than 65%. It shows the model is potentially able to detect cracks and differentiate between crack and spalling. With a larger dataset or larger data in the class of cracks, the results may be better in detecting cracks.

Results show the accuracy of detecting spalling increases in parts with deeper defects up to 93.2%. The average accuracy is 87.9% and it shows the model is working well on detecting spalling on parts with defects deeper than three centimeters.

Table 4-18 Results of training and evaluation on detecting types of defects (adapted PointNet++)

Process		**Value**
Training	Mean loss	0.083
	Overall accuracy	0.988
Evaluation	Mean loss	0.099
	Overall accuracy	0.970
	Accuracy of cracks	0.522
	Accuracy of spalling	0.704

Table 4-19 Output results of the testing on detecting types of defects (Adapted PointNet++)

Part	Number of defects	Number of points	Area (cm^2)	Depth (cm)	Mean loss	Accuracy	Accuracy of cracks	Accuracy of spallings	Accuracy of no-defects
1	30	327,777	6,248	1.5	0.059	0.992	0.000	0.000	0.999
2	16	195,140	5,957	1.6	0.534	0.913	0.000	0.108	0.999
3	5	153,353	18,490	2.6	0.281	0.946	0.265	0.000	0.982
4	24	486,062	6,607	2.8	0.475	0.931	0.001	0.075	0.997
5	10	166,446	19,338	3.5	0.280	0.911	0.092	0.616	0.953
6	15	166,667	16,152	3.7	0.227	0.908	0.381	0.458	0.943
7	19	184,671	20,104	4.1	0.313	0.805	0.000	0.930	0.762
8	5	291,776	13,756	5.1	0.225	0.948	0.000	0.836	0.973
9	13	259,111	42,174	6.0	0.148	0.928	0.662	0.927	0.937
10	16	278,342	47,597	8.1	0.207	0.909	0.663	0.932	0.911
Weighted average						0.914	0.357	0.838	0.935
Weighted average of the six deepest parts						0.914	0.460	0.879	0.918

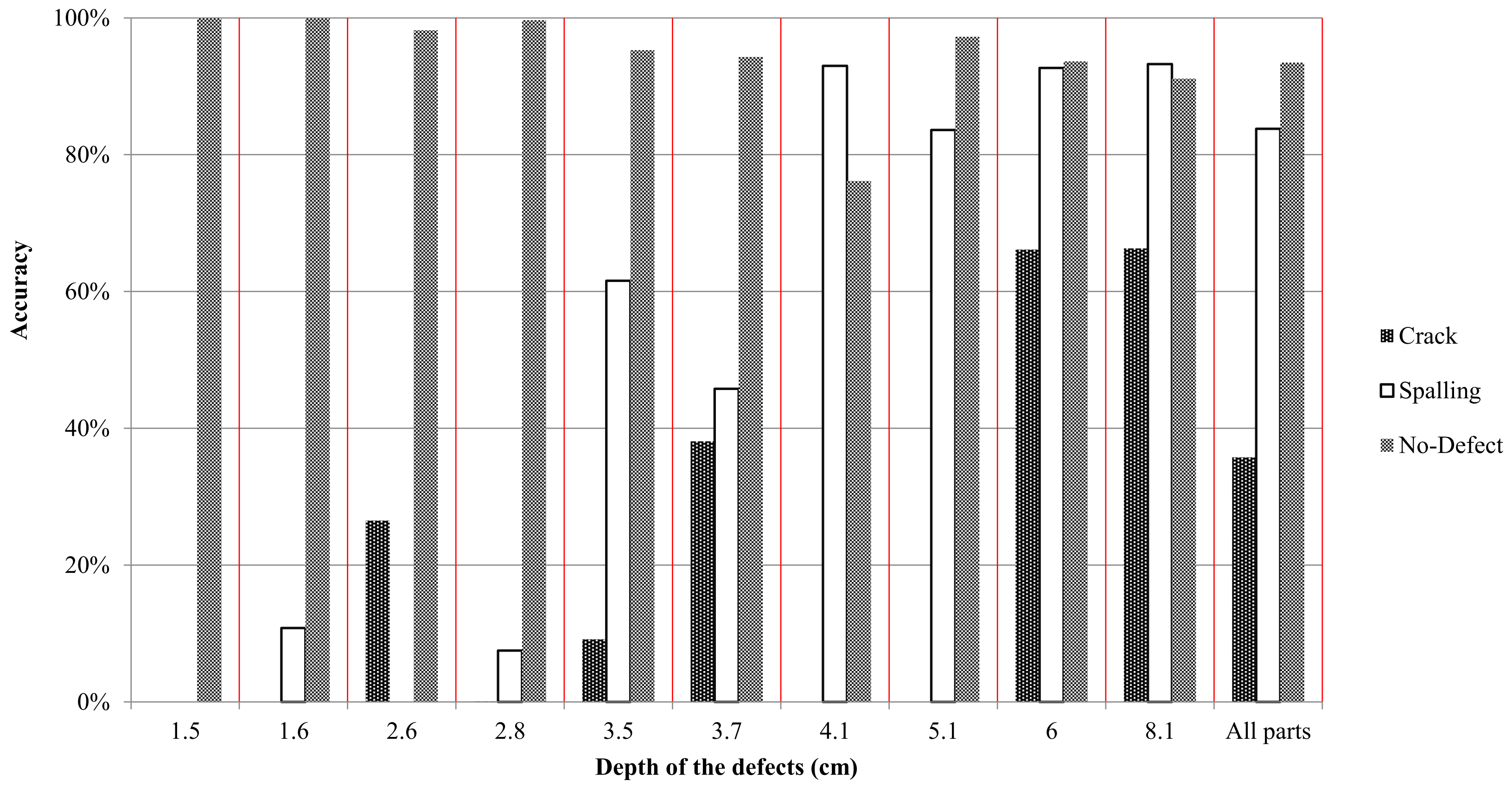


Figure 4-19 **Chart of the output results of the testing on detecting types of defects (Adapted PointNet++)**

4.3.6 Validation on Detecting Severity Levels of Defects Using Adapted PointNet++ Model

In this part of the case study, the points of the dataset are split into four classes. They are crack, medium spalling, severe spalling, and no-defect. It means that the severity level of the spalling defects is considered in two groups. The light and medium spalling annotations are considered as the medium spalling class. The goal of this section is to see the performance of the proposed method of detecting the severity levels of concrete surface defects.

As shown in Figure 4-20, the defect classes of cracks, spalling, and no-defect classes have respectively 0.8%, 3.3%, 4.1%, and 91.8% of all the points. The same method of applying a cost-sensitive loss function is used here and the overall resulted cost weights of the classes are tabulated in Table 4-20. In this case effect of imbalanced classes would be more than the previous one because an imbalanced class is added and two classes (medium and severe spalling) have the same texture and it makes the learning process more complex than before.

The statistics of the output results of the training and the evaluation are in Table 4-21 and the testing are in Table 4-22 and Figure 4-21. As expected, the performance of the model is not acceptable. The three defect classes are not detected together in the first four parts, which have defects with a depth of fewer than three centimeters. From part five, the model started to detect points of defect classes and the accuracies rise by increasing the depth value of the parts' defects. The maximum accuracy of detecting cracks and severe spalling belongs to part 10 with a depth of 8.1 centimeters and they are 69.4% and 80.6% respectively. The accuracies of detecting medium spalling were less than 40% in all parts. The average accuracies of detecting cracks, medium spallings, severe spallings, and no-defects are respectively 37%, 16.5%, 59.6%, and

95.3%, and 48.1%, 19.7%, 59.6%, 94.1% for the parts with defects deeper than three centimeters. It is expected to reach better results by increasing the size of the training dataset.

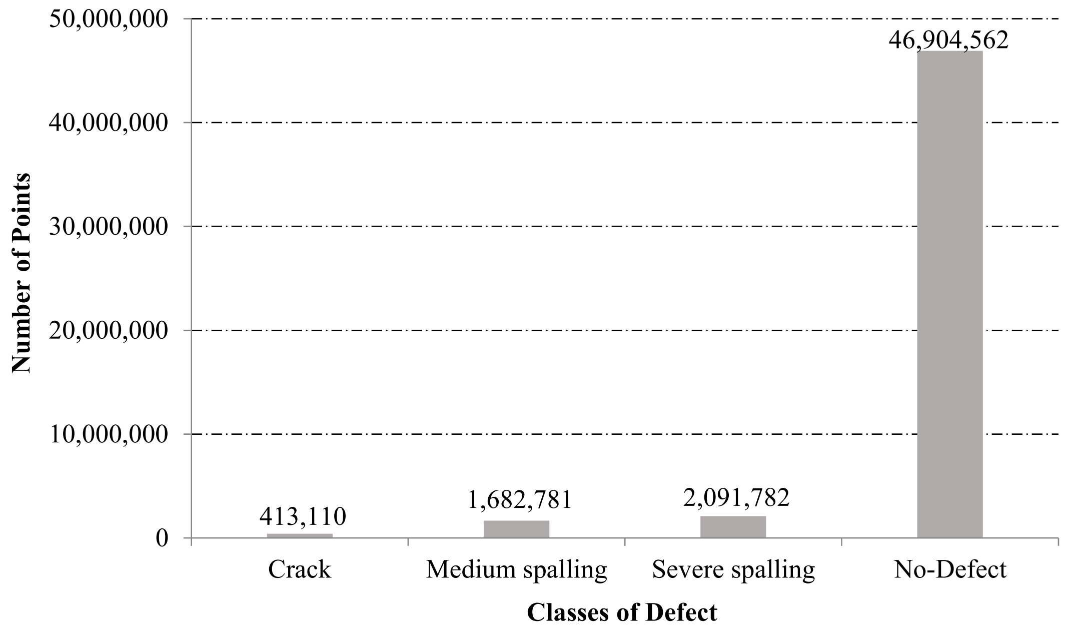


Figure 4-20 Number of points of the defect classes.

Table 4-20 Cost weight of the four classes

Class	**Crack**	**Medium spalling**	**Large spalling**	**No-Defect**	**Summation**
Number of points	413,110	1,682,781	2,091,782	46,904,562	51,092,235
Number of points weight	0.008	0.033	0.041	0.918	1.0
Cost weihgt	40.8	28.9	26.4	3.4	NA

Table 4-21 Results of training and evaluation on detecting severity levels of defects (PointNet++)

Process		**Value**
Training	Mean loss	0.097
	Overall accuracy	0.987
Evaluation	Mean loss	0.177
	Overall accuracy	0.958
	Accuracy of cracks	0.539
	Accuracy of medium spalling	0.172
	Accuracy of severe spalling	0.680

Table 4-22 Output results of the testing on detecting types and severity levels of defects (Adapted PointNet++)

Part	Number of defects	Number of points	Area (cm^2)	Depth (cm)	Mean loss	Accuracy	Accuracy of cracks	Accuracy of medium spallings	Accuracy of severe spallings	Accuracy of no-defects
1	30	327,777	6,248	1.5	0.055	0.993	0.036	0.000	N.A.	1.000
2	16	195,140	5,957	1.6	0.694	0.909	0.000	0.000	N.A.	1.000
3	5	153,353	18,490	2.6	0.271	0.948	0.240	0.000	N.A.	0.984
4	24	486,062	6,607	2.8	0.563	0.930	0.000	0.082	N.A.	0.996
5	10	166,446	19,338	3.5	0.523	0.875	0.073	0.058	0.101	0.955
6	15	166,667	16,152	3.7	0.318	0.894	0.466	0.200	N.A.	0.941
7	19	184,671	20,104	4.1	0.763	0.745	0.000	0.383	0.390	0.879
8	5	291,776	13,756	5.1	0.191	0.957	0.001	0.270	N.A.	0.991
9	13	259,111	42,174	6.0	0.235	0.917	0.686	0.193	0.735	0.956
10	16	278,342	47,597	8.1	0.287	0.897	0.694	0.065	0.806	0.926
Weighted average							0.370	0.165	0.596	0.953
Weighted average of the six deepest parts							0.460	0.879	0.596	0.918

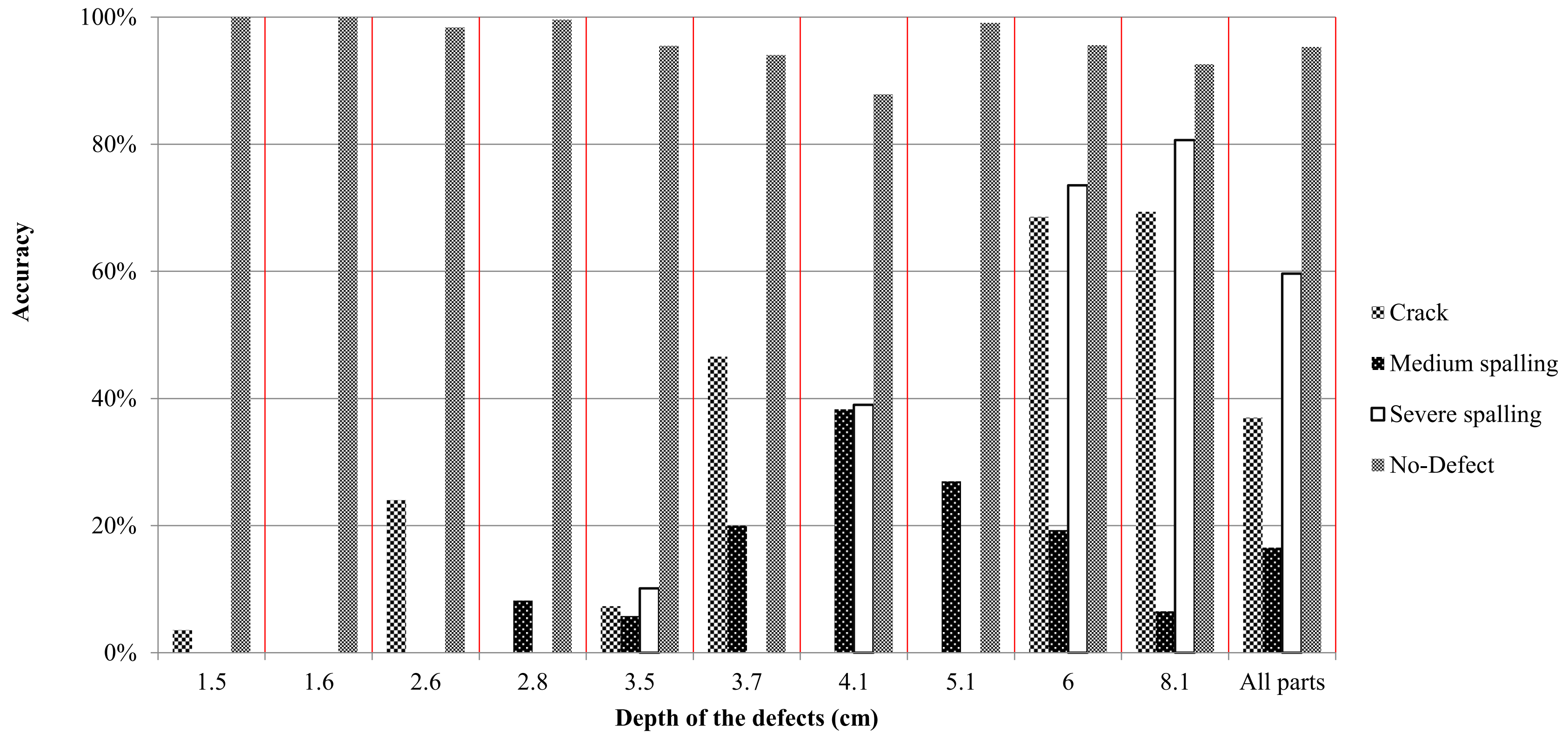


Figure 4-21 Chart of the output results of the testing on detecting types and severity levels of defects (Adapted PointNet++)

4.4 Summary and Conclusions

This chapter proposed a method for detecting surface defects of concrete bridges using point clouds and DNN. The proposed method is based on PointNet and PointNet++, which are adapted to detect defects in LiDAR scanned datasets. Training and testing datasets are collected from four concrete bridges in Montréal and annotated manually. The point cloud dataset prepared in five areas, which contain more than 51 million points and 2,572 annotated defects. Points are annotated into five classes, crack, light spalling, medium spalling, severe spalling, and no-defect. For binary segmentation, all the defects are considered as one class. The dataset split over three parts of training (70%), evaluation (24%), and testing (6%).

The following conclusions can be stated: (1) The trained models performed better in detecting the deeper defects, (2) The adapted PointNet++ performed better than the adapted PointNet on detecting defects in binary classes segmentation. The adapted PointNet++ reached the accuracy of 78.8% and the adapted PointNet could reach the accuracy of 74.9%; (3) Applying the sensitivity analysis on the adapted PointNet showed increasing the block's density more than the density of the actual dataset does not affect the results in PointNet method, and (4) PointNet++ applied on detecting the types (the accuracies of 35.7% for cracks and 83.8% for spalling are achieved), and the severity levels of defects (the accuracies of 37% for cracks, 16.5% for medium spalling, and 59.6 for severe spalling are achieved). There are two limitations in this study: (1) The training datasets are small, and (2) all kinds of defects are categorized in only one class. As future work, preparing more annotated LiDAR scanned point clouds of bridges to expand the training datasets is expected to increase the accuracy of defect detection. In addition, by adding more classes to the training datasets, this method would perform better in detecting the specified defects.

CHAPTER 5 Summary, Conclusions and Future Work

5.1 Summary of Research

This chapter reviewed the concepts, methods, and technologies that are used in the current research. Based on the literature, the LiDAR and UAV technologies can solve the data collection issues for bridge inspection, especially for the inaccessible elements of bridges. The integration of these two technologies aims to define an automated method for the bridge's data collection. Moreover, point cloud analysis using deep learning method can semantically segment LiDAR generated point clouds, which is the core idea of this research's concrete surface defect detection method.

A LiDAR-equipped UAV platform is designed to collect 3D point cloud data using a 2D LiDAR scanner. The design satisfies the main identified requirements and constraints for structural inspection. The platform is realized and tested in an indoor and outdoor environment.

A method for detecting surface defects of concrete bridges using point clouds and DNN is proposed. The method is based on PointNet and PointNet++, which are adapted to detect defects in LiDAR scanned datasets. Five areas of the point cloud datasets (containing three flipped datasets) are used in training, evaluation, and testing.

5.2 Research Conclusion and Contributions

The results of an outdoor test of the designed LiDAR-equipped UAV platform in flying mode show that the point clouds' accuracy is not enough to detect surface defects smaller than 10 cm, which is mainly because of the low accuracy of the available light-weighted LiDAR scanner. A high resolution terrestrial 3D LiDAR (FARO Focus 3D) is used to generate point clouds for the rest of the study and validate the proposed method for detecting surface defects of concrete bridges.

The following conclusions can be stated for the proposed DNN based defect detection method: (1) The trained models performed better in detecting the deeper defects, (2) PointNet++ performed better than PointNet on detecting defects in binary classes segmentation, (3) Increasing the block's density more than the density of the actual dataset does not affect the results in PointNet method, and (4) PointNet++ can segment defects based on types and severity levels.

The research contributions are: (1) Proposing a method for data collection using LiDAR and UAV to increase the accessibility to most parts of bridges for inspection and automate the process of bridge inspection, (2) Proposing a DNN-based method to process the collected bridge point clouds without converting them to other visual representations (e.g. images, voxels). The proposed model is validated on real collected point clouds in detecting defects. Promising results have been obtained despite the small-sized training dataset. The accuracies of 74.9% (adapted PointNet) and 78.8% (adapted PointNet++) in detecting defects are achieved in binary semantic segmentation. In detecting types of defects, the accuracies of 83.8% and 35.7% in detecting spalling defects and crack defects, respectively, are achieved. Also, the accuracies of 87.9% and 46.0% in detecting spalling defects and crack defects deeper than 3cm are achieved. Moreover, in detecting the severity levels of defects, the accuracies of 59.6%, 16.5%, and 37.0% are achieved in detecting severe spalling, medium spalling, and crack defects. Also, the accuracies of 59.6%, 87.9%, and 46.0% are achieved in detecting severe spalling, medium spalling, and crack defects deeper than 3cm.

5.3 Limitations and Future Work

Our limitation for the LiDAR-equipped UAV platform is the quality of the available LiDAR. Also, the weather conditions (e.g. wind) affect the accuracy of the results.

The future work is validating the designed LiDAR-equipped UAV platform using a more accurate light-weighted 3D LiDAR to reach more accurate point clouds. Velodyne Puck LITE (Velodyne LiDAR, 2019) is a light-weighted available 3D LiDAR, which is weighted by almost 590 grams. It may be used on the designed LiDAR-equipped UAV. This LiDAR is 3D and replace the 2D LiDAR and the servo motor. When we use a LiDAR with acceptable accuracy, the designed platform will use to validate the UAV path planning part (Bolourian & Hammad, 2019).

The limitation in point cloud analysis part of the study is the small-size training dataset. Moreover, there is no previous work on concrete surface defect detection using point clouds without converting them to images, so, it is not possible to compare the results to other related methods.

As future work, preparing more annotated LiDAR scanned point clouds of bridges to expand the training datasets is expected to increase the accuracy of defect detection. In addition, by adding more data to the training datasets, this method would perform better in detecting the types and severity levels of the specified defects. Also, in future work, the results of the defect detection method will be used to locate the defects in the bridge information model (BrIM) by using the clustering methods and the industry foundation class (IFC).

REFERENCES

(PCA), P. C. (n.d.). *Concrete Information.* Portland Cement Association (PCA).

About ROS. (n.d.). Retrieved from ROS: http://www.ros.org/about-ros/

Adhikari, R. S., Moselhi, O., & Bagchi, A. (2014). Image-based retrieval of concrete crack properties for bridge inspection. *Automation in construction, 39*, 180-194.

Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., & Savarese, S. (2016). 3d semantic parsing of large-scale indoor spaces. *IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 1534-1543). Las Vegas Valley.

Armesto-González, J., Riveiro-Rodríguez, B., González-Aguilera, D., & Rivas-Brea, M. T. (2010). Terrestrial laser scanning intensity data applied to damage detection for historical buildings. *Journal of Archaeological Science, 37*(12), 3037-3047.

Bachrach, A., Samuel, P., He, R., & Roy, N. (2011). RANGE–Robust autonomous navigation in GPS-denied environments. *Journal of Field Robotics, 28*(5), 644-666.

Bags. (n.d.). (ROS) Retrieved from ROS wiki: http://wiki.ros.org/Bags

Bertussi, S., Szenher, P., & Bai, S. (2017). *spin_hokuyo*. (GitHub Inc.) Retrieved from GitHub: https://github.com/RobustFieldAutonomyLab/spin_hokuyo

Bogosian, B., Gharakhanian, N., Khanoyan, G., Toorian, A., & Ohanian, R. (2016). *3D Scanning Assembly.* Glendale: Glendale Community College.

Bolourian, N., & Hammad, A. (2019). Path Planning of LiDAR-Equipped UAV for Bridge Inspection Considering Potential Locations of Defects. *Advances in Informatics and Computing in Civil and Construction Engineering*, (pp. 545-552).

Brock, A.; Lim, T.; Ritchie, J.M; Weston, N. (2016). Generative and discriminative voxel modeling with convolutional neural networks. arXiv: 1608.04236(2).

Buda, M., Maki, A., & Mazurowski, M. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 249-259.

Canadian Aviation Regulations (CARs) and standards. (2017). Transport Canada.

Chen, Xiaozhi, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. (2017). Multi-view 3d object detection network for autonomous driving. *Computer Vision and Pattern Recognition (CVPR).*

Dissanayake, M. G., Newman, P., Clark, S., Durrant-Whyte, H. F., & Csorba, M. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on robotics and automation*, 229-241.

Engelcke, M., Rao, D., Wang, D. Z., Tong, C. H., & Posner, I. . (2017). Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. *International Conference on Robotics and Automation (ICRA)*, (pp. 1355-1361). Singapore.

FARO. (2012). *FARO Laser Scanner Focus3D X130*. (FARO Technologies Inc.) Retrieved from FARO: https://www.faro.com/products/construction-bim-cim/faro-focus/

FARO. (2017). *Quality Setting Function on the Focus3D*. (FARO) Retrieved from https://knowledge.faro.com/Hardware/3D_Scanners/Focus/Quality_Setting_Function_on_the_Focus3D

Field, T., Leibs, J., & Bowman, J. (2010). *Rosbag Package*. Retrieved from ROS Wiki: http://wiki.ros.org/rosbag/

Freimuth, H., Müller, J., & König, M. (2017). Simulating and executing UAV-assisted inspections on construction sites. *The 34th International Symposium on Automation and Robotics in Construction (ISARC).*

Girardeau-Montaut, D. (2015). *CloudCompare*. Retrieved from http://www.cloudcompare.org/

Girardeau-Montaut, D., Roux, M., Marc, R., & Thibault, G. (2005). Change detection on points cloud data acquired with a ground laser scanner. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 36*(3), W19.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Convolutional Networks. In *Deep learning.* Cambridge: MIT press.

Guldur, B., Yujie, Y., & Hajjar, J. F. (2015). Condition assessment of bridges using terrestrial laser scanners. *Structures Congress*, (pp. 355-366). Portland.

HDF-Group. (2018). *HDF5 User's Guide.* Retrieved from https://portal.hdfgroup.org/display/HDF5/HDF5+User%27s+Guide

Hendrix, A. (2014). *Ubuntu ARM install of ROS Indigo*. Retrieved from http://wiki.ros.org/indigo/Installation/UbuntuARM

Hershberger, D., Gossow, D., & Faust, J. (2018). *rviz Package Summary*. (Robotic Operating System) Retrieved from ROS: http://wiki.ros.org/rviz

HOKUYO AUTOMATIC, C. (2014). *Distance Data Output/UTM-30LX.* Retrieved 2017, from https://www.hokuyo-aut.jp/search/single.php?serial=169

Honkavaara, E., Kaivosoja, J., Mäkynen, J., Pellikka, I., Pesonen, L., Saari, H., . . . Rosnell, T. (2012). Hyperspectral reflectance signatures and point clouds for precision agriculture by light weight UAV imaging system. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci 7*, (pp. 353-358). Melbourne.

Huber, D. (2014, 04 11). *Futuristic infrastructure inspection in Pennsylvania.* Retrieved from http://aria.ri.cmu.edu/archives/category/media

Hugh Durrant-Whyte, Tim Bailey. (2006). Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. *IEEE Robotics & Automation Magazine*, 9.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *International Conference on Machine Learning*, (pp. 448-456).

Jiang, J., Miyagusuku, R., Yamashita, A., & Asama, H. (2017). Glass confidence maps building based on neural networks using laser range-finders for mobile robots. *IEEE/SICE International Symposium on System Integration*, (pp. 405-410).

Kim, M. K., Sohn, H., & Chang, C. C. (2014). Localization and quantification of concrete spalling defects using terrestrial laser scanning. *Journal of Computing in Civil Engineering*, 29(6), 04014086.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koch, C., Georgieva, K., Kasireddy, V., Akinci, B., & Fieguth, P. (2015). A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure. *Advanced Engineering Informatics, 29*(2), 196-210.

Kohlbrecher, S. (2011). *Mapping Using Logged Data*. Retrieved from ROS Wiki: http://wiki.ros.org/hector_slam/Tutorials/MappingUsingLoggedData

Laefer, D. F.; Truong-Hong, L.; Carr, H.; Singh, M. (2014). Crack detection limits in unit based masonry with terrestrial laser scanning. *NDT & E International*, 62, 66-76.

LeCun, Y. (1989). Generalization and network design strategies. *Connectionism in perspective*, 143-155.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*.

Li, Z., Y. Yan, Y. Jing, and S. G. Zhao. (2015). The Design and Testing of a LiDAR Platform for a UAV for Heritage Mapping. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *40.*

Lidar, V. (2019). *Velodyne Puck LITE*. Retrieved from Velodine Lidar.

Liu, W., Chen, S., & Hauser, E. (2011). LiDAR-based bridge structure defect detection. *Experimental Techniques, 35*(6), 27-34.

Maekawa, D. (n.d.). *Daiki Maekawa*. Retrieved from GitHub: https://github.com/DaikiMaekawa

MANIFOLD. (n.d.). (SZ DJI Technology Co., Ltd.) Retrieved from https://www.dji.com/manifold

MATRICE 100. (2016). (SZ DJI Technology Co., Ltd.) Retrieved from https://www.dji.com/matrice100/info

Nasrollahi, M; Bolourian, N.; Zhu, Z.; Hammad, A. (2018). Designing LiDAR-equipped UAV Platform for Structural Inspection. *International Symposium on Automation and Robotics in Construction (ISARC).* Berlin.

Olsen, M. J.; Kuester, F.; Chang, B. J.; Hutchinson, T. C. (2009). Terrestrial Laser Scanning-Based Structural Damage Assessment. *Journal of Computing in Civil Engineering, 24*(3), 264-272.

(2008). *Ontario Structure Inspection Manual (OSIM).* St. Catharines, ON: Ministry of Transportation.

pcl_ros. (n.d.). (ROS) Retrieved from ROS wiki: http://wiki.ros.org/pcl_ros

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 1-17.

Qi, C. R., Su, H., Mo, K., & Guibas, L. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Computer Vision and Pattern Recognition (CVPR)*, (pp. 4-23). Honolulu, Hawaii.

Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*.

Qi, C. R.; Su, H.; Nießner; M., Dai, A.; Yan, M.; Guibas, L. J. (2016). Volumetric and multi-view CNNs for object classification on 3D data. *IEEE conference on computer vision and pattern recognition*, (pp. 5648-5656).

Rehman, S. K., Ibrahim, Z., Memon, S. A., & Jameel, M. (2016). Nondestructive test methods for concrete bridges: A review. *Construction and Building Materials, 107*, 58-86.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. *International Conference on Artificial Neural Networks.* Thessaloniki, Greece.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research, 1*(15), 1929-1958.

Stormbee demo days. (2017). (STORMBEE CO.) Retrieved from http://www.stormbee.eu/

Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3D shape recognition. *IEEE international conference on computer vision*, (pp. 945-953). Santiago, Chile.

Teza, G., Galgaro, A., & Moro, F. (2009). Contactless recognition of concrete surface damage from laser scanning and curvature computation. *NDT & E International, 42*(4), 240-249.

Tsai, Y. J. and Li, F. (2012). Critical assessment of detecting asphalt pavement cracks under different lighting and low intensity contrast conditions using emerging 3D laser technology. *Journal of Transportation Engineering, 138*(5), 649-656.

Venator, E. (2015). *Point Cloud Library (PCL)*. (Robot Operating System) Retrieved 02 20, 2015, from http://wiki.ros.org/pcl

Wallace, L., Arko, L., Christopher, W., & Darren, T. (2012). Development of a UAV-LiDAR system with application to forest inventory. *Remote Sensing, 4*(6), 1519-1543.

Winkvist, S., & Rushforth, E. (2013). Towards an autonomous indoor aerial inspection vehicle. *The Industrial Robot, 40*(3), 196-207.

Wise, M. (2009). *Using The Hokuyo Node*. Retrieved from ROS Wiki: http://wiki.ros.org/hokuyo_node/Tutorials/UsingTheHokuyoNode

Yoder, L., & Sebastian, S. (2016). Autonomous exploration for infrastructure modeling with a micro aerial vehicle. *Field and service robotics, 113*, 427-440.

Zhang, J., & Singh, S. (2014). LOAM: Lidar Odometry and Mapping in Realtime. *Robotics: Science and Systems.* Berkeley.

Zhou, Y., & Tuzel, O. (2018). Voxelnet: End-to-end learning for point cloud based 3d object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 4490-4499).

Zink, J., & Barritt, L. (2015). *Unmanned aerial vehicle bridge inspection demonstration project.* No. MN/RC 2015-40.

Appendices

Appendix A – Robot Operating System (ROS)

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aims to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms (http://www.ros.org/about-ros/). ROS website documented the installation tutorial on its website. Here is the link of ROS Indigo installation:

http://wiki.ros.org/indigo/Installation/Ubuntu

For installing on Manifold use:

http://wiki.ros.org/indigo/Installation/UbuntuARM

Necessary steps and probable errors that may occur in ROS Indigo installation have explained in this section.

1. On the Ubuntu desktop, the taskbar is in the left side. The first icon is "Search", select it and type "terminal" :

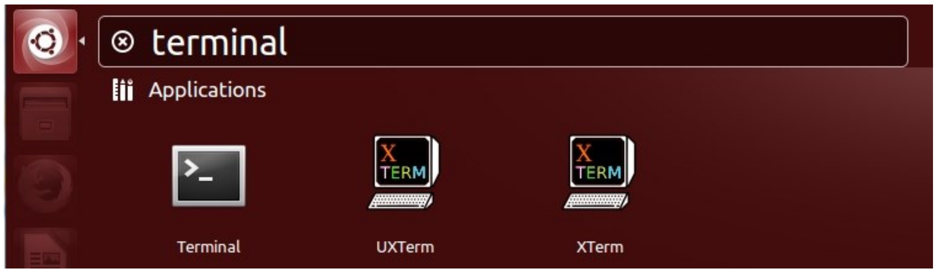
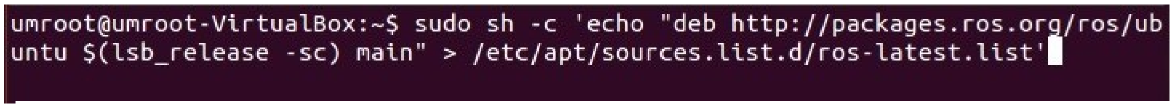


Figure A-1

2. Open "Terminal" and type this command and press Enter to setup the sources list:

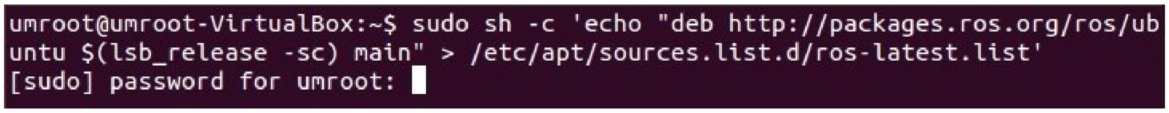
```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb
_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```



```
umroot@umroot-VirtualBox:~$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ub
untu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Figure A-2

Then type Ubuntu password:

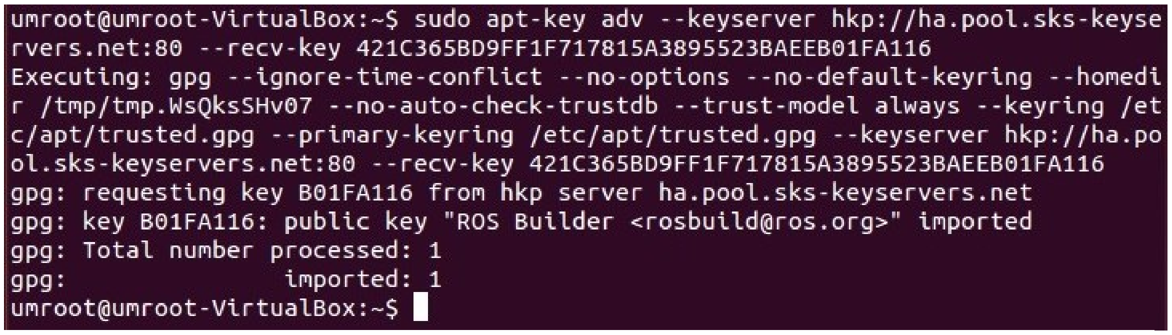


```
umroot@umroot-VirtualBox:~$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ub
untu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
[sudo] password for umroot:
```

Figure A-3

3. Type this command and press Enter to set up your keys:

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:
80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```



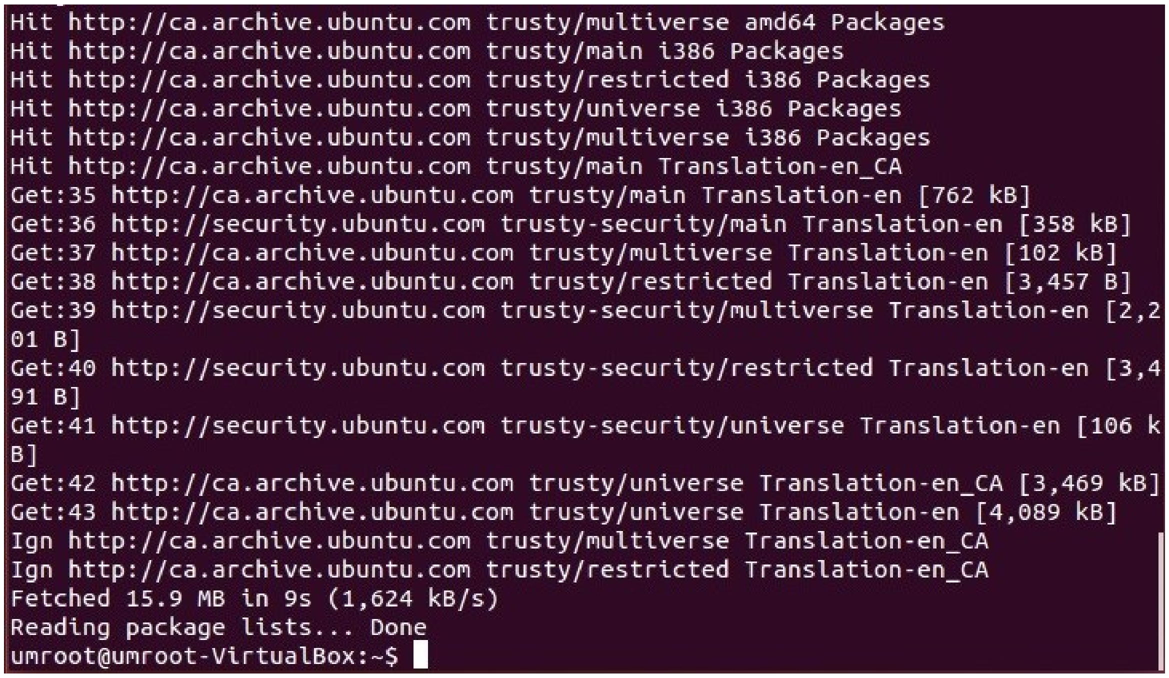
```
umroot@umroot-VirtualBox:~$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyse
rvers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedi
r /tmp/tmp.WsQksSHv07 --no-auto-check-trustdb --trust-model always --keyring /et
c/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg --keyserver hkp://ha.po
ol.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
gpg: requesting key B01FA116 from hkp server ha.pool.sks-keyservers.net
gpg: key B01FA116: public key "ROS Builder <rosbuild@ros.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
umroot@umroot-VirtualBox:~$
```

Figure A-4

4. Type this command and press Enter to make sure the Debian package index is up-to-date :

```
$ sudo apt-get update
```

Some operations similar to the following figure will appear:



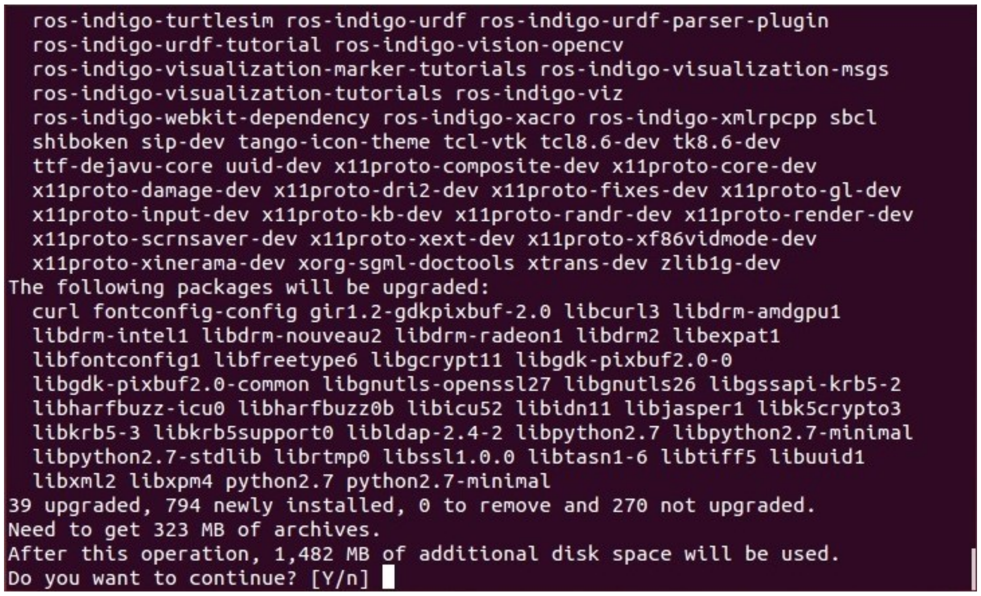
```
Hit http://ca.archive.ubuntu.com trusty/multiverse amd64 Packages
Hit http://ca.archive.ubuntu.com trusty/main i386 Packages
Hit http://ca.archive.ubuntu.com trusty/restricted i386 Packages
Hit http://ca.archive.ubuntu.com trusty/universe i386 Packages
Hit http://ca.archive.ubuntu.com trusty/multiverse i386 Packages
Hit http://ca.archive.ubuntu.com trusty/main Translation-en_CA
Get:35 http://ca.archive.ubuntu.com trusty/main Translation-en [762 kB]
Get:36 http://security.ubuntu.com trusty-security/main Translation-en [358 kB]
Get:37 http://ca.archive.ubuntu.com trusty/multiverse Translation-en [102 kB]
Get:38 http://ca.archive.ubuntu.com trusty/restricted Translation-en [3,457 B]
Get:39 http://security.ubuntu.com trusty-security/multiverse Translation-en [2,2
01 B]
Get:40 http://security.ubuntu.com trusty-security/restricted Translation-en [3,4
91 B]
Get:41 http://security.ubuntu.com trusty-security/universe Translation-en [106 k
B]
Get:42 http://ca.archive.ubuntu.com trusty/universe Translation-en_CA [3,469 kB]
Get:43 http://ca.archive.ubuntu.com trusty/universe Translation-en [4,089 kB]
Ign http://ca.archive.ubuntu.com trusty/multiverse Translation-en_CA
Ign http://ca.archive.ubuntu.com trusty/restricted Translation-en_CA
Fetched 15.9 MB in 9s (1,624 kB/s)
Reading package lists... Done
umroot@umroot-VirtualBox:~$
```

Figure A-5

5. Type the following command, in order to install the full package of ROS Indigo.

```
$ sudo apt-get install ros-indigo-desktop-full
```

Answer "Y" to the following question:



```
  ros-indigo-turtlesim ros-indigo-urdf ros-indigo-urdf-parser-plugin
  ros-indigo-urdf-tutorial ros-indigo-vision-opencv
  ros-indigo-visualization-marker-tutorials ros-indigo-visualization-msgs
  ros-indigo-visualization-tutorials ros-indigo-viz
  ros-indigo-webkit-dependency ros-indigo-xacro ros-indigo-xmlrpcpp sbcl
  shiboken sip-dev tango-icon-theme tcl-vtk tcl8.6-dev tk8.6-dev
  ttf-dejavu-core uuid-dev x11proto-composite-dev x11proto-core-dev
  x11proto-damage-dev x11proto-dri2-dev x11proto-fixes-dev x11proto-gl-dev
  x11proto-input-dev x11proto-kb-dev x11proto-randr-dev x11proto-render-dev
  x11proto-scrnsaver-dev x11proto-xext-dev x11proto-xf86vidmode-dev
  x11proto-xinerama-dev xorg-sgml-doctools xtrans-dev zlib1g-dev
The following packages will be upgraded:
  curl fontconfig-config gir1.2-gdkpixbuf-2.0 libcurl3 libdrm-amdgpu1
  libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libdrm2 libexpat1
  libfontconfig1 libfreetype6 libgcrypt11 libgdk-pixbuf2.0-0
  libgdk-pixbuf2.0-common libgnutls-openssl27 libgnutls26 libgssapi-krb5-2
  libharfbuzz-icu0 libharfbuzz0b libicu52 libidn11 libjasper1 libk5crypto3
  libkrb5-3 libkrb5support0 libldap-2.4-2 libpython2.7 libpython2.7-minimal
  libpython2.7-stdlib librtmp0 libssl1.0.0 libtasn1-6 libtiff5 libuuid1
  libxml2 libxpm4 python2.7 python2.7-minimal
39 upgraded, 794 newly installed, 0 to remove and 270 not upgraded.
Need to get 323 MB of archives.
After this operation, 1,482 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

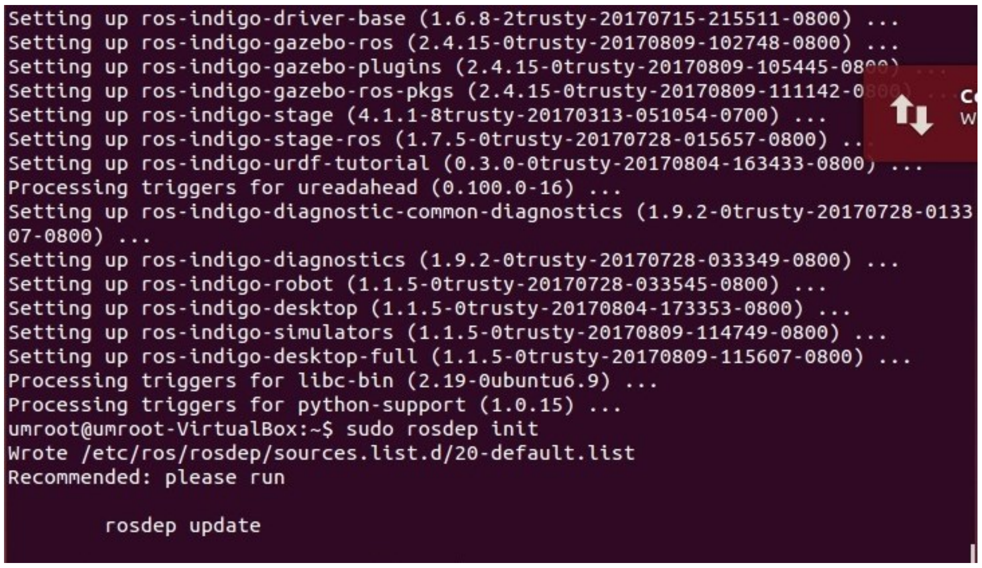
Figure A-6

If in this step you faced an error, try to install desktop version by:

```
$ sudo apt-get install ros-indigo-desktop
```

6. Type this command and press Enter to initialize rosdep :

```
$ sudo rosdep init
```



```
Setting up ros-indigo-driver-base (1.6.8-2trusty-20170715-215511-0800) ...
Setting up ros-indigo-gazebo-ros (2.4.15-0trusty-20170809-102748-0800) ...
Setting up ros-indigo-gazebo-plugins (2.4.15-0trusty-20170809-105445-0800) ...
Setting up ros-indigo-gazebo-ros-pkgs (2.4.15-0trusty-20170809-111142-0800) ...
Setting up ros-indigo-stage (4.1.1-8trusty-20170313-051054-0700) ...
Setting up ros-indigo-stage-ros (1.7.5-0trusty-20170728-015657-0800) ...
Setting up ros-indigo-urdf-tutorial (0.3.0-0trusty-20170804-163433-0800) ...
Processing triggers for ureadahead (0.100.0-16) ...
Setting up ros-indigo-diagnostic-common-diagnostics (1.9.2-0trusty-20170728-0133
07-0800) ...
Setting up ros-indigo-diagnostics (1.9.2-0trusty-20170728-033349-0800) ...
Setting up ros-indigo-robot (1.1.5-0trusty-20170728-033545-0800) ...
Setting up ros-indigo-desktop (1.1.5-0trusty-20170804-173353-0800) ...
Setting up ros-indigo-simulators (1.1.5-0trusty-20170809-114749-0800) ...
Setting up ros-indigo-desktop-full (1.1.5-0trusty-20170809-115607-0800) ...
Processing triggers for libc-bin (2.19-0ubuntu6.9) ...
Processing triggers for python-support (1.0.15) ...
umroot@umroot-VirtualBox:~$ sudo rosdep init
Wrote /etc/ros/rosdep/sources.list.d/20-default.list
Recommended: please run

        rosdep update
```

Figure A-7

If you faced "rosdep: command not found" error, then install new commands by these two commands respectively:

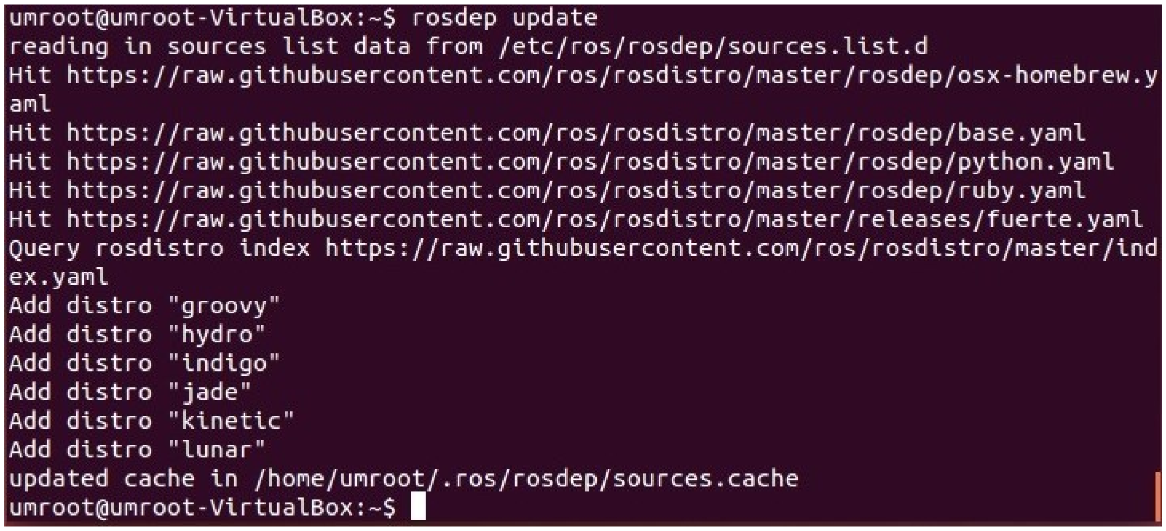
```
$ sudo apt-get install python-pip

$ sudo apt-get install python-rosdep
```

Now try again to initialize rosdep.

7. The following command is necessary during initializing rosdep :

```
$ rosdep update
```



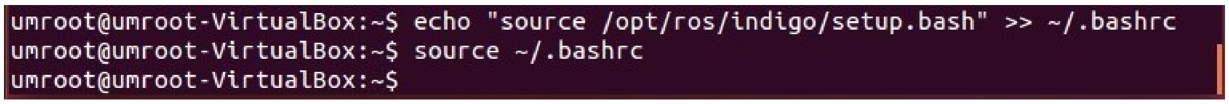
```
umroot@umroot-VirtualBox:~$ rosdep update
reading in sources list data from /etc/ros/rosdep/sources.list.d
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/osx-homebrew.y
aml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/base.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/python.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/ruby.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/releases/fuerte.yaml
Query rosdistro index https://raw.githubusercontent.com/ros/rosdistro/master/ind
ex.yaml
Add distro "groovy"
Add distro "hydro"
Add distro "indigo"
Add distro "jade"
Add distro "kinetic"
Add distro "lunar"
updated cache in /home/umroot/.ros/rosdep/sources.cache
umroot@umroot-VirtualBox:~$
```

Figure A-8

8. Type following commands separately for environment setup :

```
$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

$ source ~/.bashrc
```

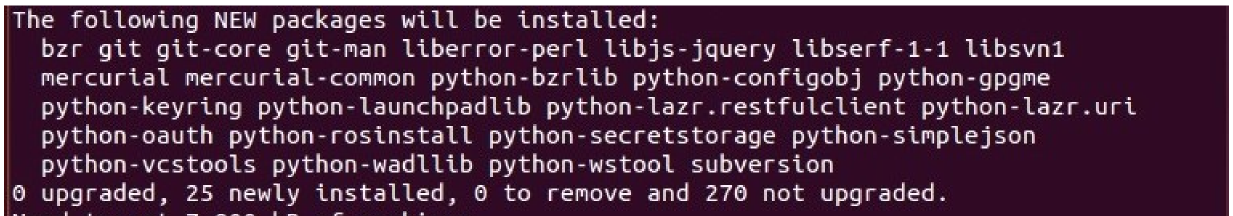


```
umroot@umroot-VirtualBox:~$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
umroot@umroot-VirtualBox:~$ source ~/.bashrc
umroot@umroot-VirtualBox:~$
```

Figure A-9

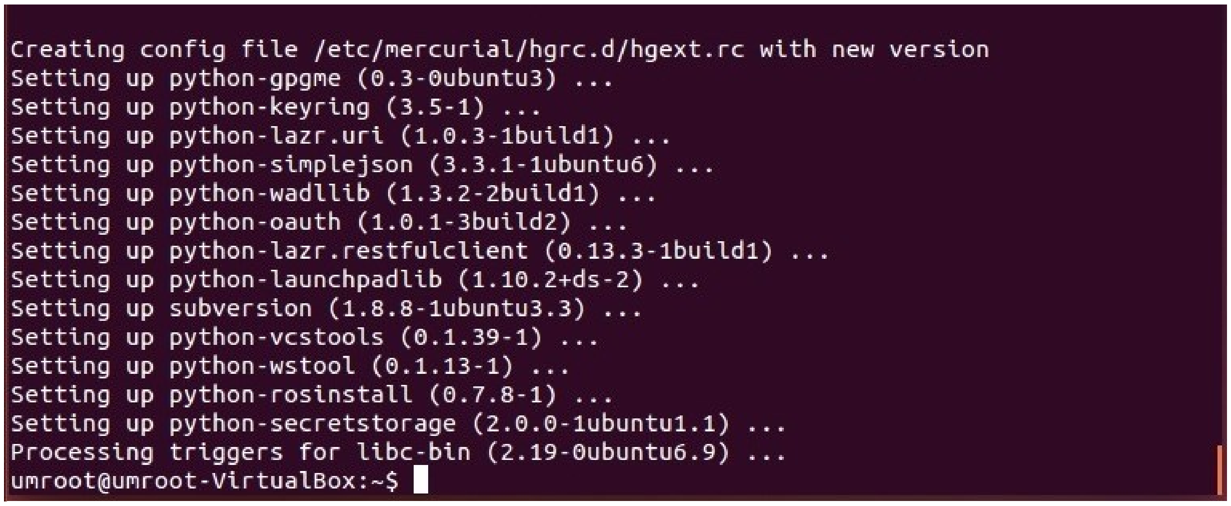
9. Getting rosinstall (frequently used command-line tool in ROS) is the last step of installing ROS. Type this command and press Enter :

```
$ sudo apt-get install python-rosinstall
```



```
The following NEW packages will be installed:
  bzr git git-core git-man liberror-perl libjs-jquery libserf-1-1 libsvn1
  mercurial mercurial-common python-bzrlib python-configobj python-gpgme
  python-keyring python-launchpadlib python-lazr.restfulclient python-lazr.uri
  python-oauth python-rosinstall python-secretstorage python-simplejson
  python-vcstools python-wadllib python-wstool subversion
0 upgraded, 25 newly installed, 0 to remove and 270 not upgraded.
```

By typing “Y”, accept to continue:



```
Creating config file /etc/mercurial/hgrc.d/hgext.rc with new version
Setting up python-gpgme (0.3-0ubuntu3) ...
Setting up python-keyring (3.5-1) ...
Setting up python-lazr.uri (1.0.3-1build1) ...
Setting up python-simplejson (3.3.1-1ubuntu6) ...
Setting up python-wadllib (1.3.2-2build1) ...
Setting up python-oauth (1.0.1-3build2) ...
Setting up python-lazr.restfulclient (0.13.3-1build1) ...
Setting up python-launchpadlib (1.10.2+ds-2) ...
Setting up subversion (1.8.8-1ubuntu3.3) ...
Setting up python-vcstools (0.1.39-1) ...
Setting up python-wstool (0.1.13-1) ...
Setting up python-rosinstall (0.7.8-1) ...
Setting up python-secretstorage (2.0.0-1ubuntu1.1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.9) ...
umroot@umroot-VirtualBox:~$
```

Figure A-11

If you want to install ROS on an ARM-based CPU computer (like MANIFOLD) do not forget to unset GTK_IM_MODULE environment variable set by this code:

```
unset GTK_IM_MODULE
```

Otherwise, you cannot run RViz and you face to “segment fault” error.

A.1 ROS workspace (catkin workspace)

Catkin workspace is a directory where you modify, build, and install catkin packages (http://wiki.ros.org/catkin/workspaces). To create and build a catkin workspace using the “mkdir” command. It makes the folders:

```
$ mkdir -p ~/catkin_ws/src
```

Then go into the folder by:

```
$ cd ~/catkin_ws/
```

And create the workspace by typing:

```
$ catkin_make
```

You can see this folder (catkin_ws) in the “home” directory. It contains three folders, ‘build’, “devel” and “src”. The “catkin_make” command creates a “CMakeLists.txt” link in the “src” folder. Inside the “devel” folder, there are several setups.*sh files. Sourcing any of these files will overlay this workspace on top of your environment. Before continuing, source the new setup.bash file:

```
$ source devel/setup.bash
```

A.2 Hector Slam

In order to use the Hokuyo laser scanner, it is necessary to add some packages to ROS. Hokuyo UTM-30LX is a 2D laser detector that does not have an internal GPS, so we should use SLAM

(Simultaneous Localization and Mapping) technology to create a map by laser scanning. We should install “hector_slam”, “hector_mapping” and “hokuyo_node” packages.

It is necessary to run this code in a Terminal it at first to run ROS:

```
$  roscore
```

While ROS is running, in a new Terminal install “hector SLAM” package:

```
$ sudo apt-get install ros-indigo-hector-slam
```

“Hector SLAM” is a full package that installs “hector mapping” and “hector geotiff” with itself that is necessary for creating a map using a laser scanner. Each package has one or some launch files. For example, the hector SLAM package has nine launch files and ‘tutorial.launch’ is the one for launching created maps.

A.3 Hokuyo node

Install “hokuyo_node” package using this command:

```
$ sudo apt-get install ros-indigo-hokuyo-node
```

Appendix B - Hector Slam Example

Now all the packages that are necessary to use Hokuyo UTM-30LX are installed, just it is necessary to make a launch file to use hector SLAM, hector mapping, and Hokuyo node in order to create a map. One option is to edit “tutorial.launch” file that is in the hector SLAM package and the other option is to use pre-written packages. A related example made by Daiki Maekawa (https://github.com/DaikiMaekawa/hector_slam_example) gets used in this survey.

Extract the zipped folder, rename it to "hector_slam_example" and copy it into the "src" folder in the "catkin_ws" folder.

In a new Terminal type:

```
$ source ~/catkin_ws/devel/setup.bash
```

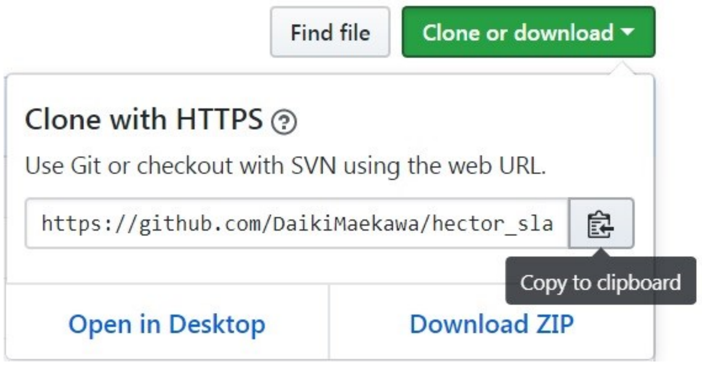
Now install hector SLAM example:

```
$ rosdep install hector_slam_example
```

In the case of facing an error on installing the package, there is another way of installation. In this way, before downloading the "hector_slam_example" package, it is recommended to install the newly released "catkin-tools" package that has a "catkin build" command for making new packages instead of "catkin_make". So, use this command line to install "catkin-tools":

```
$ sudo apt-get install python-catkin-tools
```

Open the package's link (https://github.com/DaikiMaekawa/hector_slam_example), click on "Clone or download", click on "Copy to clipboard".



Use Git or checkout with SVN using the web URL.

```
https://github.com/DaikiMaekawa/hector_sla
```

Figure B-1

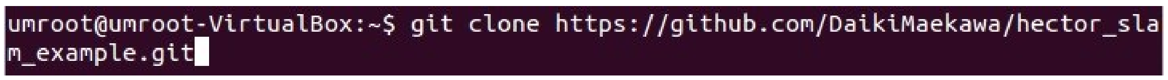
After the clone link copied to the clipboard, go to ROS, create a folder named “git” by this command line:

```
$ mkdir -p ~/git
```

Then go into the folder by:

```
$ cd ~/git/
```

Type “git clone” as a new command and paste the copied link of “hetor_slam_example” after that:



```
umroot@umroot-VirtualBox:~$ git clone https://github.com/DaikiMaekawa/hector_sla
m_example.git
```

Figure B-2

The package is in the “git” folder now and next step in to link it to the catkin workspace source folder. So go into catkin workspace source folder:

```
$ cd ~/catkin_ws/src/
```

Then use this command:

```
$ ln -s ~/git/hector_slam_example/
```

Then type “ls” in the command line, and see the list of folders and files in the source folder. Make sure that “hector_slam_example” is not in red color, and if it is, a mistake occurred in the previous steps. Check again from the cloning step to find the source of error.

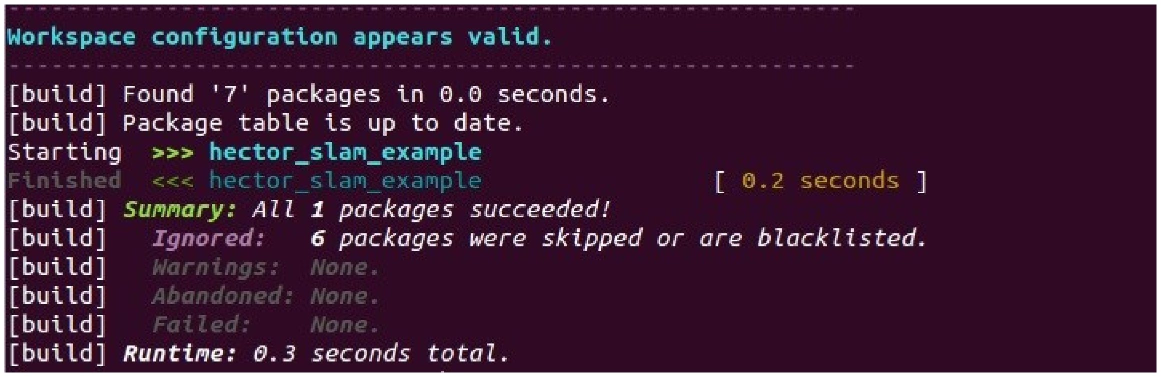
By this command, go back to catkin workspace root:

```
$ cd ~/catkin_ws/
```

Then build a package by:

```
$ catkin build hector_slam_example
```

These notes will appear if everything goes well.



```
------------------------------------------------------------
Workspace configuration appears valid.
------------------------------------------------------------
[build] Found '7' packages in 0.0 seconds.
[build] Package table is up to date.
Starting  >>> hector_slam_example
Finished  <<< hector_slam_example                [ 0.2 seconds ]
[build] Summary: All 1 packages succeeded!
[build]   Ignored:   6 packages were skipped or are blacklisted.
[build]   Warnings:  None.
[build]   Abandoned: None.
[build]   Failed:    None.
[build] Runtime: 0.3 seconds total.
```

Figure B-3

B.1 Creating a Map Using Hokuyo UTM-30LX

Turn Hokuyo on and plug in its USB. Make sure that the power light is on. In a Terminal type:

```
$ ls -l /dev/ttyACM0
```

You will see something similar to:

```
crw-rw-XX- 1 root dialout 166, 0 2009-10-27 14:18 /dev/ttyACM0
```

If XX is rw: the laser is configured properly.

If XX is --: the laser is not configured properly and it is necessary to:

```
$ sudo chmod a+rw /dev/ttyACM0
```

For the hokuyo_node to work properly, a ROS core must be running. In a new terminal:

```
$ roscore
```

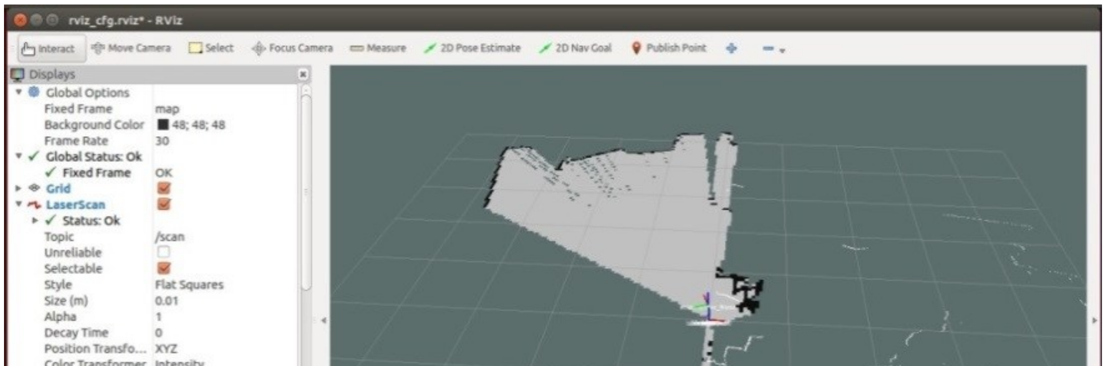
In a new terminal type:

```
$ source ~/catkin_ws/devel/setup.bash
```

Now run ‘Hector SLAM example’ by:

```
$ roslaunch hector_slam_example hector_hokuyo.launch
```

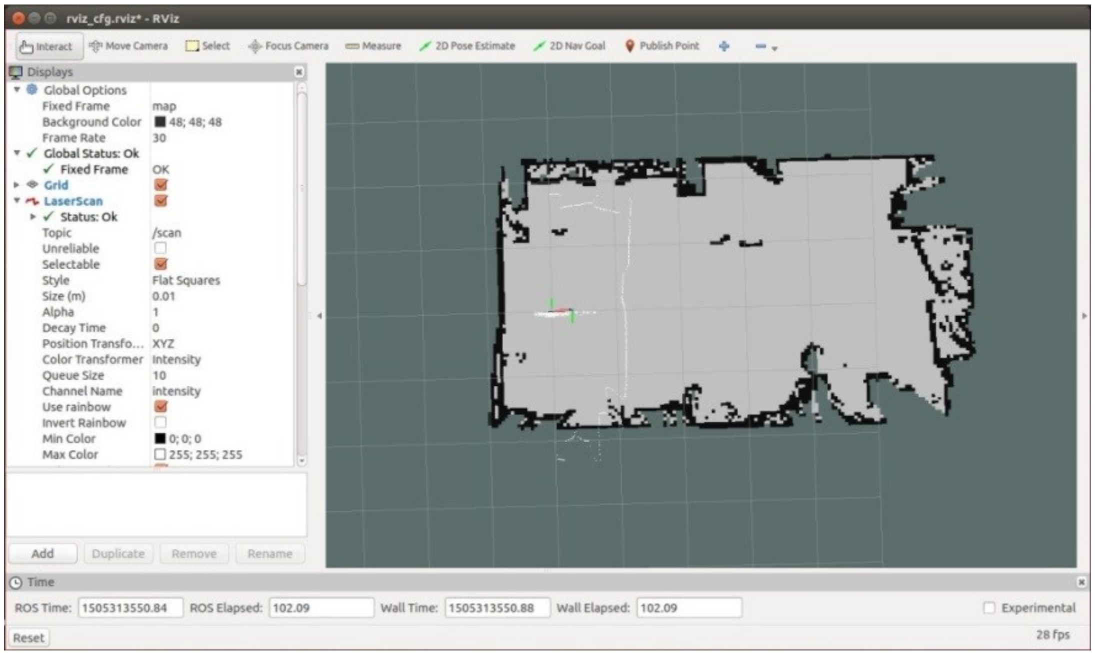
The lunch file will open RViz software that shows the scanned area and create a map.



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
LaserScan	✓
✓ Status: Ok	
Topic	/scan
Unreliable	☐
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transfo...	XYZ
Color Transformer	Intensity

Figure B-4

You can move the laser to map the room. As this software plot the map in 2D, you should move Hokuyo just vertically to scan the surfaces parallel to the ground surface and do not turn it or pitch it during scanning.



```
▾ Global Options
    Fixed Frame           map
    Background Color      ■ 48; 48; 48
    Frame Rate            30
▾ ✓ Global Status: Ok
    ✓ Fixed Frame         OK
▸ Grid                    ✓
▾ LaserScan               ✓
  ▸ ✓ Status: Ok
    Topic                 /scan
    Unreliable            ☐
    Selectable            ✓
    Style                 Flat Squares
    Size (m)              0.01
    Alpha                 1
    Decay Time            0
    Position Transfo...   XYZ
    Color Transformer     Intensity
    Queue Size            10
    Channel Name          intensity
    Use rainbow           ✓
    Invert Rainbow        ☐
    Min Color             ■ 0; 0; 0
    Max Color             ☐ 255; 255; 255

ROS Time: 1505313550.84   ROS Elapsed: 102.09   Wall Time: 1505313550.88   Wall Elapsed: 102.09   ☐ Experimental
28 fps
```

Figure B-5

B.2 Recording the Data

While RViz is running and the scanner is scanning, you can record the data and the map while creating by "rosbag" command. At first, you should create a folder for bag files:

```
$ mkdir ~/bagfiles
```

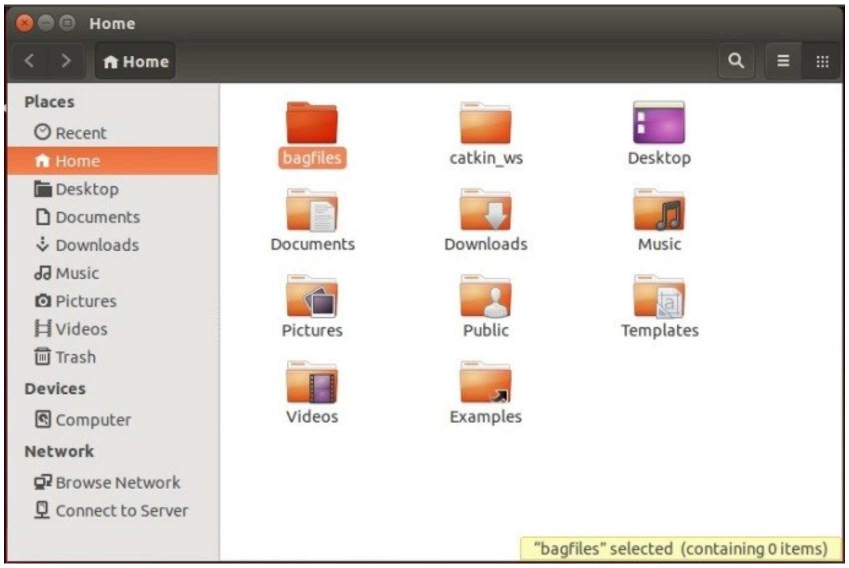
Go into the folder by:

```
$ cd ~/bagfiles
```

While RViz is scanning and you are in bag files folder type this command in a new terminal (Field, Leibs, & Bowman, 2010):

```
$ rosbag record -a
```

After scanning, stop recording by pressing Ctrl+C in the recording terminal. Pressing Ctrl+C in a terminal stops a command that is running. You can find the recorded file (.bag file) in the bag files folder.



"bagfiles" selected (containing 0 items)

Figure B-6

It is possible to play it with rosbag command, but before that, it is necessary to open an RViz window:

```
$ roslaunch hector_slam_launch tutorial.launch
```

When the RViz window opened, in a new terminal type:

```
$ rosbag play <your bag file name>
```

You should write the full name of the bag file, for example, 2017-09-13-11-34-56.bag

Each bag file name contains the exact date and time of the start of recording.

You can find the information about a bag file by this command:

```
rosbag info <your bagfile>
```

Size of a bag file of 548 seconds scanning is around 1,346 MB, a bag file of 496 seconds of scanning is 1,298 MB and a bag file of 93 seconds of scanning is 66 MB.

B.3 Mapping a Hallway Hsing Hokuyo

Hokuto UTM-30LX got tested to create a map of EV-Building 8th floor’s hallway two times. In the first experiment, Hokuyo is connected to a Laptop and in the second time, it connected to the MANIFOLD microcomputer that we plan to use it on the drone.

Hokuyo UTM-30LX Laser Range finder, Lenovo Thinkpad Core i7 processor Laptop, and a 12V ANKER power bank (to support power for Hokuyo) is used on a cart and scanned hallway of EV-Building 8th floor.

The experiment took around 6 minutes and you can see the created map on the next page. I should add that in this case, ROS works on virtual Ubuntu OS that uses 5 processors of an Intel Core i7 processor and 24 GB memory as RAM.

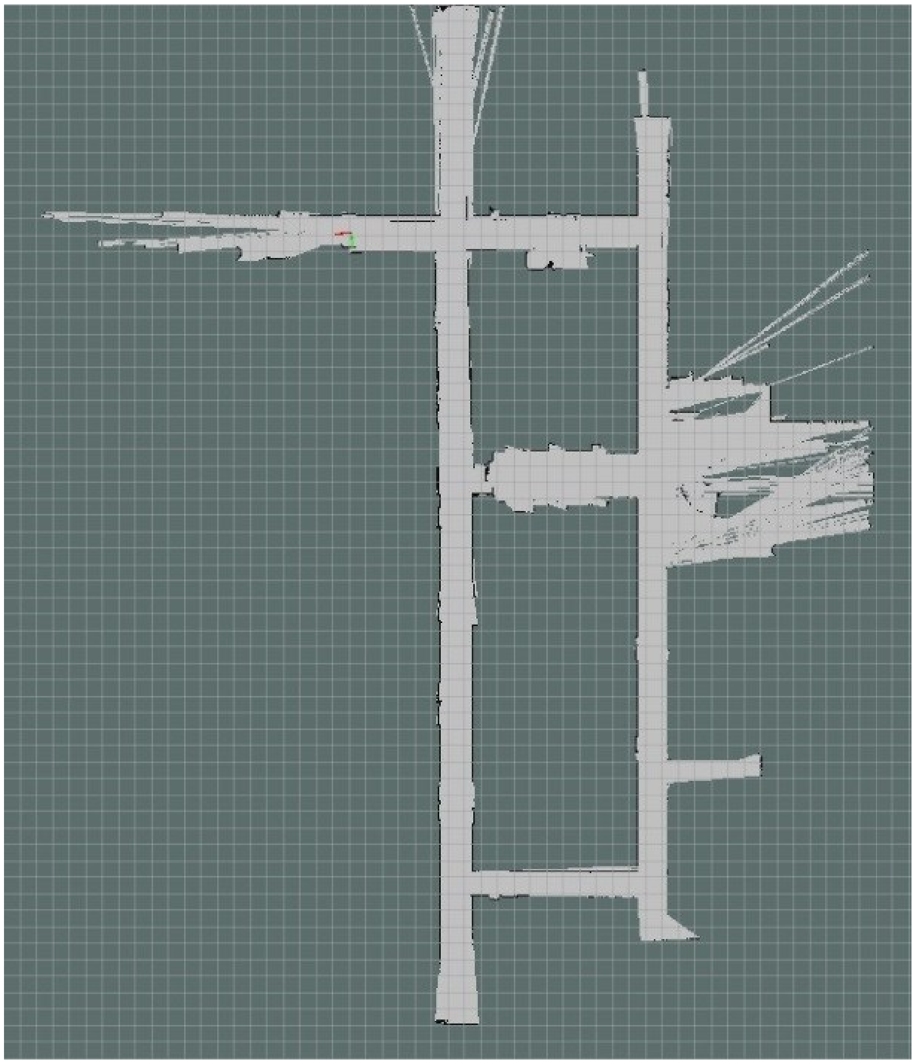


Figure B-7

Hokuyo UTM-30LX Laser Range finder, MANIFOLD ARM MPcore processor microcomputer, DJI MATRICE 100 drone (to supply power for MANIFOLD) and a 12V ANKER power bank (to support power for Hokuyo) is used on a cart and scanned the hallway of EV-Building 8th floor.

The experiment took around 8 minutes and you can see the created map here. It should be added that in this case, ROS worked on Ubuntu OS that uses Quad-core 4-Plus-1 ARM MPcore processors and 2 GB memory as RAM.

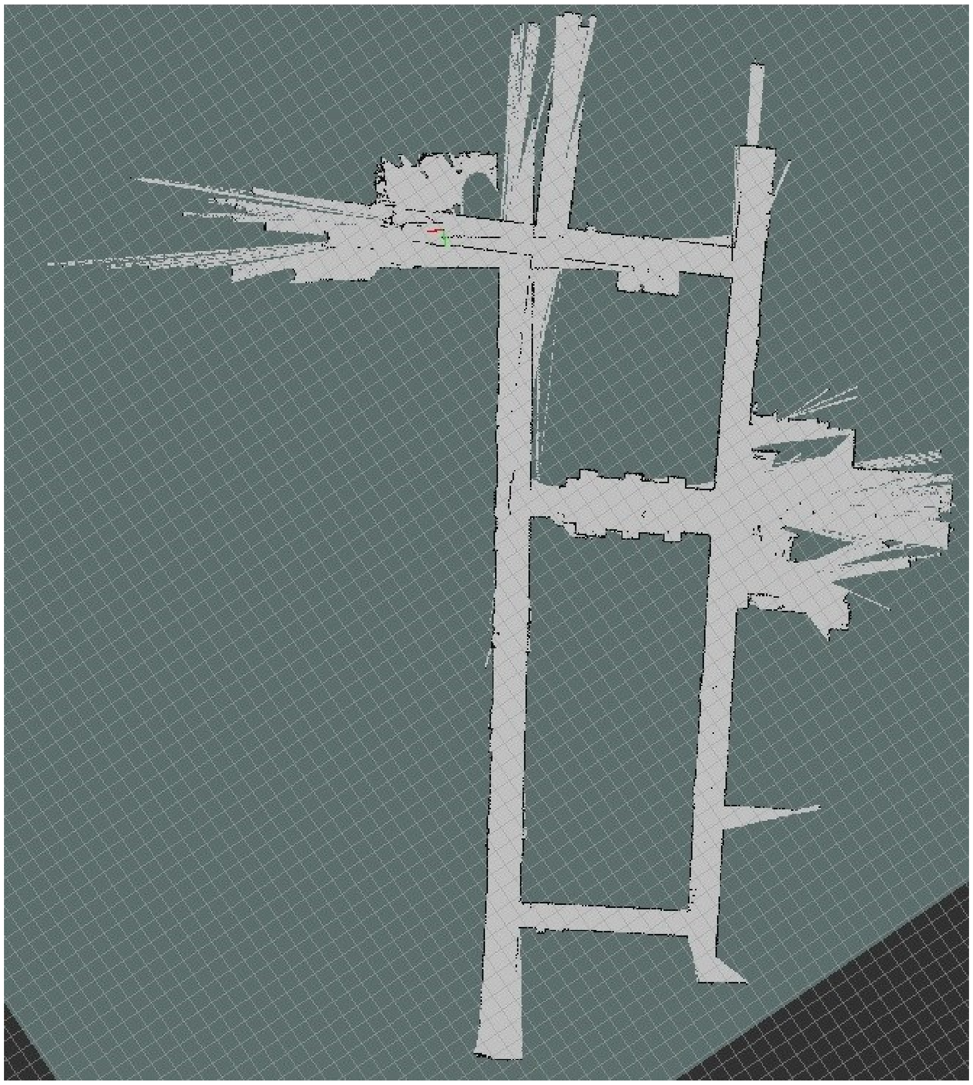
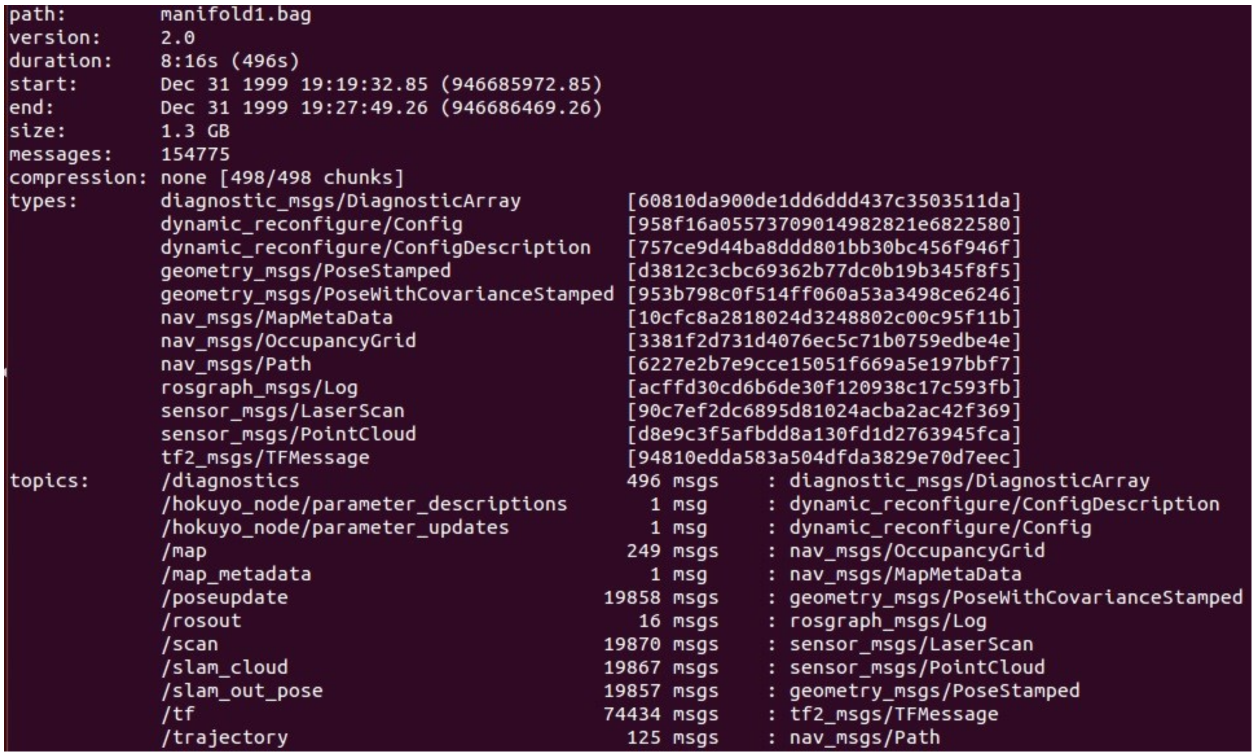


Figure B-8

B.4 Converting bag Files to Point Cloud

“Bag” is a ROS file format and named because of .bag extension. Bag files store ROS messages data. While recording output of Hokuyo laser scanner, bag files record “sensor_msgs/LaserScan” and “sensor_msgs/PointCloud” messages. To see a bag file information use this command line:

```
$ rosbag info session*.bag
```



```
path:        manifold1.bag
version:     2.0
duration:    8:16s (496s)
start:       Dec 31 1999 19:19:32.85 (946685972.85)
end:         Dec 31 1999 19:27:49.26 (946686469.26)
size:        1.3 GB
messages:    154775
compression: none [498/498 chunks]
types:       diagnostic_msgs/DiagnosticArray         [60810da900de1dd6ddd437c3503511da]
             dynamic_reconfigure/Config              [958f16a05573709014982821e6822580]
             dynamic_reconfigure/ConfigDescription   [757ce9d44ba8ddd801bb30bc456f946f]
             geometry_msgs/PoseStamped               [d3812c3cbc69362b77dc0b19b345f8f5]
             geometry_msgs/PoseWithCovarianceStamped [953b798c0f514ff060a53a3498ce6246]
             nav_msgs/MapMetaData                    [10cfc8a2818024d3248802c00c95f11b]
             nav_msgs/OccupancyGrid                  [3381f2d731d4076ec5c71b0759edbe4e]
             nav_msgs/Path                           [6227e2b7e9cce15051f669a5e197bbf7]
             rosgraph_msgs/Log                       [acffd30cd6b6de30f120938c17c593fb]
             sensor_msgs/LaserScan                   [90c7ef2dc6895d81024acba2ac42f369]
             sensor_msgs/PointCloud                  [d8e9c3f5afbdd8a130fd1d2763945fca]
             tf2_msgs/TFMessage                      [94810edda583a504dfda3829e70d7eec]
topics:      /diagnostics                            496 msgs    : diagnostic_msgs/DiagnosticArray
             /hokuyo_node/parameter_descriptions       1 msg     : dynamic_reconfigure/ConfigDescription
             /hokuyo_node/parameter_updates            1 msg     : dynamic_reconfigure/Config
             /map                                    249 msgs    : nav_msgs/OccupancyGrid
             /map_metadata                             1 msg     : nav_msgs/MapMetaData
             /poseupdate                           19858 msgs    : geometry_msgs/PoseWithCovarianceStamped
             /rosout                                  16 msgs    : rosgraph_msgs/Log
             /scan                                 19870 msgs    : sensor_msgs/LaserScan
             /slam_cloud                           19867 msgs    : sensor_msgs/PointCloud
             /slam_out_pose                        19857 msgs    : geometry_msgs/PoseStamped
             /tf                                   74434 msgs    : tf2_msgs/TFMessage
             /trajectory                             125 msgs    : nav_msgs/Path
```

Figure B-9

Bag files can just visualize in ROS Rviz and need to convert to other popular point cloud formats like .pcd and .ply to open in other software. There is a package named "pcl_ros" that has a "bag_to_pcd" node for converting bag files to PCD (Point Cloud Data) files. This is the command line for "bag_to_pcd":

```
$ rosrun pcl_ros bag_to_pcd <input_file.bag> <topic> <output_dir
ectory>
```

As "pcl_ros" tries to convert "pointcloud2" messages as a point cloud file, it seems that it cannot convert manifold bag files to a pcd file.

The old format “sensor_msgs/PointCloud” is not supported in PCL. The “laser_assembler” package changes laser messages to “PointCloud2”. As the “PointCloud” message is old, it is necessary to make ROS publish laser scanner messages as “PointCloud2”. “laser_assembler” and “laser_geometry” are so recommended by ROS users in order to publish messages in a 3D view (by using “PointCloud2” messages). Both of them are under the category of the “laser_pipeline” package.

Appendix C – Compute Canada

Compute Canada enables Canadian researchers to perform world-class research using Advanced Research Computing (ARC) strategies. The organization helps researchers who need ARC in all disciplines and at all scales; from individual researchers to some of the largest international research collaborations in the world.

Compute Canada is in partnership with regional organizations ACENET, Calcul Québec, Compute Ontario and WestGrid.

https://www.computecanada.ca/

Cedar (located in Simon Fraser University) and Graham (located in Waterloo University) are general purposes clusters composed of a variety of nodes including large memory nodes and nodes with accelerators. They went into service in the summer of 2017. You can log in to either one using SSH and the same password you use at ccdb.computecanada.ca. A home directory will be automatically created for you the first time you log in.

Download MobaXterm (that supports SSH) from this link and run it:

https://mobaxterm.mobatek.net/download.html

Start a local terminal and write one of these SSH commands to log in to a cluster: (Use ccdb username in the command lines)

```
$ ssh <username>@graham.computecanada.ca
```

```
$ ssh <username>@cedar.computecanada.ca
```

The password of the login is the same as the ccdb account (for both Cedar and Graham). If the "Welcome message" is shown, you are logged in successfully.

C.1 Submitting a Job

Right-click on the left sidebar and choose "New empty file" to create a "sh" file to define a command line as a job. Write a name that ends with ".sh" (name-of-the-job.sh) and double click on the created sh file. Start to write your job in MobaTextEditor.

A simple sh job:

```
#!/bin/bash
#SBATCH --time=00:01:00
echo 'Hello, world!'
sleep 30
```

Save the written file and remember its path address. In the terminal, go into the folder of the saved job by "cd" command (eg cd tensorflow/pointnet). The terminal is in "/home/username/" by default.

1. Use sbatch to run the job:

```
[<username>@gra-login3 ~]$  sbatch <job name>.sh
```

2. If you faced an error about the account, add this line to the .sh file:

```
#SBATCH --account=def-hammad
```

If submission worked, you will see:

```
Submitted batch job 3130294
```

This number (3130294) is the job ID.

3. To see the running/pending jobs:

```
[<username>@gra-login3 ~]$ squeue -u <username>
```

4. Information about a completed job:

```
[<username>@gra-login3 ~]$ sacct -j <job ID>
```

5. To cancel a running/pending job:

```
[<username>@gra-login3 ~]$ scancel <job ID>
```

By default the output is placed in a file named "slurm-", suffixed with the job ID number and ".out", e.g. “slurm-123456.out”, in the directory from which the job was submitted. You can use the “--output” command in sh file to specify a different name or location.

C.1 Installing TensorFlow

There is a tutorial for installing TensorFlow:

https://docs.computecanada.ca/wiki/Tensorflow

Summary: (some parts are from other documentations)

1. You can see where you are by:

```
[<username>@gra-login3 ~]$ pwd
/home/<username>
```

2. To see all the available modules: (look at the core modules part)

```
[<username>@gra-login3 ~]$ module avail
```

3. Find the version of the python you want to work on and load it:

```
[<username>@gra-login3 ~]$ module load python/3.6.3
```

4. **4.** Create a new Python virtual environment:

```
[<username>@gra-login3 ~]$ virtualenv tensorflow
```

5. Activate your newly created Python virtual environment:

```
[<username>@gra-login3 ~]$ source tensorflow/bin/activate
```

And see:

```
(tensorflow) [<username>@gra-login3 ~]$
```

6. Install TensorFlow into your newly created virtual environment:

```
(tensorflow) [<username>@gra-login3 ~]$ pip install tensorflow-gpu
```

Now, TensorFlow is installed and a directory named “tensorflow” should have appeared.

7. Go into the tensorflow folder:

```
(tensorflow) [<username>@gra-login3 ~]$ cd tensorflow
```

8. In the left panel, right click and create a “New empty file” and name it “tensorflow-test.py”. Open it (by double click) and write these lines and save it:

```python
import tensorflow as tf
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
sess = tf.Session()
print(sess.run([node1, node2]))
```

9. Create another “New empty file” and name it “tensorflow-test.sh” for running the job. Open it (by double click) and write these lines into it and save it:

```
#!/bin/bash
#SBATCH --gres=gpu:1        # request GPU "generic resource"
#SBATCH --cpus-per-task=6   # maximum CPU cores per GPU request: 6 on Cedar, 16 on Graham.
#SBATCH --mem=32000M        # memory per node
#SBATCH --time=0-03:00      # time (DD-HH:MM)
#SBATCH --output=%N-%j.out  # %N for node name, %j for jobID

module load cuda cudnn python/3.6.3
source ~/tensorflow/bin/activate
python ./tensorflow-test.py
```

Don’t forget to wait till it will completely upload. Don’t forget to specify the exact python version that is loaded.

10. Then write:

```
(tensorflow) [<username>@gra-login3 tensorflow]$ sbatch tensorflow-test.sh
```

If it works, you should see:

```
Submitted batch job 3132013
```

This number (3132013) is the job ID.

A file named “gra956-3132013.out” will appear, right click on it and choose “Open with default text editor”. It should be like this:

```
2018-03-01 20:41:50.604981: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1212] Found device 0 with properties:
name: Tesla P100-PCIE-12GB major: 6 minor: 0 memoryClockRate(GHz): 1.3285
pciBusID: 0000:83:00.0
totalMemory: 11.91GiB freeMemory: 11.62GiB
```

```
2018-03-01 20:41:50.605045: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1312] Adding visible gpu devices: 0

2018-03-01 20:41:51.025649: I tensorflow/core/common_runtime/gpu/gpu_device.cc:993] Creating TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 11250 MB memory) -> physical GPU (device: 0, name: Tesla P100-PCIE-12GB, pci bus id: 0000:83:00.0, compute capability: 6.0)

Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)

[3.0, 4.0]
```

C.2 SBATCH Command Options

All the SBATCH command options are explained in https://slurm.schedmd.com/sbatch.html. Some important options are explained here.

#SBATCH --account=<account>

This line is to specify an account for the job. The account name is something like "def-hammad" under the "Group Name" in the "Account details" on Compute Canada.

#SBATCH --begin=<time>

Submit the batch script to the Slurm controller immediately, like normal, but tell the controller to defer the allocation of the job until the specified time.

```
  --begin=16:00
  --begin=now+1hour
  --begin=now+60          (seconds by default)
  --begin=2010-01-20T12:34:00
```

#SBATCH --cpus-per-task=<ncpus>

Number of CPUs you want to allocate to the job. It is recommended to use maximum 6 for Cedar and 16 for Graham per GPU.

#SBATCH --deadline=<OPT>

To remove the job if no ending is possible before this deadline (start > (deadline - time[-min])). Default is no deadline. Valid time formats are:

HH:MM[:SS] [AM|PM]

MMDD[YY] or MM/DD[/YY] or MM.DD[.YY]

MM/DD[/YY]-HH:MM[:SS]

YYYY-MM-DD[THH:MM[:SS]]]

#SBATCH --gres=<list>

--gres=gpu:1

--gres=gpu:2,mic=1

--gres=gpu:kepler:2

#SBATCH --mem=<size[units]>

Specify the real memory required per node. Default units are megabytes. Different units can be specified using the suffix [K|M|G|T].

#SBATCH --output=<filename pattern>

--output=Class-%N-%j.out # %N for node name, %j for jobID

Appendix D – Python Code for Flipping the Dataset

```
import os
import glob
import sys
import numpy as np
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
ROOT_DIR = os.path.dirname(BASE_DIR)
sys.path.append(BASE_DIR)

data_dir = os.path.join(ROOT_DIR, 'data')
flipped_dir = os.path.join(data_dir, 'mirrored')
if not os.path.exists(flipped_dir):
    os.mkdir(flipped_dir)
anno_paths = [line.rstrip() for line in open(os.path.join(BASE_DIR,
'meta/anno_paths.txt'))]


for anno_path in anno_paths:
    print(anno_path)
    elements = anno_path.split('/')
    area = os.path.join(flipped_dir+'/'+elements[-3])
    if not os.path.exists(area):
        os.mkdir(area)
    part = os.path.join(area+'/'+elements[-2])
    if not os.path.exists(part):
        os.mkdir(part)
    output_dir = os.path.join(part+'/'+'Annotations')
    if not os.path.exists(output_dir):
        os.mkdir(output_dir)
    input = np.loadtxt(data_dir+'/'+'bridge'+'/'+elements[-
3]+'/'+elements[-2]+'/'+elements[-2]+'.txt', dtype=np.float, delimiter='
')
    num=len(input)
    out = input
    for i in range(num):
        out[i,0] = -input[i,0]
    np.savetxt(part+'/'+elements[-2]+'.txt', out, fmt='%.3f %.3f %.3f %d
%d %d')
    for f in glob.glob(os.path.join(data_dir, 'bridge', anno_path,
'*.txt')):
        out_filename = os.path.basename(f)
        input = np.loadtxt(f, dtype=np.float, delimiter=' ')
        print (f)
        num=len(input)
        print (num)
        out = input
        for i in range(num):
            out[i,0] = -input[i,0]
        np.savetxt(output_dir+'/'+out_filename, out, fmt='%.3f %.3f
%.3f %d %d %d')
```