MID-AMERICA TRANSPORTATION CENTER

Report # MATC-KU: 152-3

Final Report

WBS: 25-1121-0005-152-3

# Low-Cost 3-D LIDAR Development for Transportation

**Christopher Depcik, PhD**
Professor
Department of Mechanical Engineering
University of Kansas

**Nate Ahlgren**
Graduate Student
Department of Mechanical Engineering
University of Kansas

**Dang M. Tran**
Undergraduate Student
Electrical Engineering and Computer Science
Wichita State University

**Hongsheng He, PhD**
Assistant Professor
Electrical Engineering and Computer Science
Wichita State University

2023

A Cooperative Research Project sponsored by
U.S. Department of Transportation- Office of the Assistant Secretary for Research and Technology

# Low-Cost 3-D LIDAR Development for Transportation

Christopher Depcik, Ph.D.
Professor
Department of Mechanical Engineering
University of Kansas

Nate Ahlgren
Graduate Student
Department of Mechanical Engineering
University of Kansas

Hongsheng He, Ph.D.
Assistant Professor
Electrical Engineering and Computer Science
Wichita State University

Dang M. Tran
Undergraduate Student
Electrical Engineering and Computer Science
Wichita State University

A Report on Research Sponsored by

Mid-America Transportation Center

University of Nebraska–Lincoln

July 2023

# TECHNICAL REPORT DOCUMENTATION PAGE

| | | |
|---|---|---|
| 1. Report No.<br>25-1121-0005-152-3 | 2. Government Accession No. | 3. Recipient's Catalog No. |
| 4. Title and Subtitle<br>Low-Cost 3-D LIDAR Development for Transportation | | 5. Report Date<br>July 2023 |
| | | 6. Performing Organization Code<br>Enter any/all unique numbers assigned to the performing organization, if applicable. |
| 7. Author(s)<br>Christopher Depcik, Ph.D., https://orcid.org/0000-0002-0045-9554<br>Hongsheng He, Ph.D., https://orcid.org/0000-0002-2810-865X | | 8. Performing Organization Report No.<br>25-1121-0005-152-3 |
| 9. Performing Organization Name and Address<br>University of Kansas, Department of Mechanical Engineering, 3144C Learned Hall, 1530 W. 15th Street, Lawrence, Kansas, USA, 66045-4709<br>Wichita State University, Department of Electrical Engineering and Computer Science, Wichita, KS, 67260, USA | | 10. Work Unit No. |
| | | 11. Contract or Grant No.<br>69A3551747107 |
| 12. Sponsoring Agency Name and Address<br>Office of the Assistant Secretary for Research and Technology<br>1200 New Jersey Ave., SE<br>Washington, D.C. 20590<br><br>Mid-America Transportation Center<br>Prem S. Paul Research Center at Whittier School<br>2200 Vine St.<br>Lincoln, NE 68583-0851 | | 13. Type of Report and Period Covered<br>Final Report, August 2020 -June 2023 |
| | | 14. Sponsoring Agency Code<br>MATC TRB RiP No. 91994-67 |
| 15. Supplementary Notes<br>Conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration. | | |


16. Abstract

Mobile light detection and ranging (LIDAR) technology offers a significant opportunity to increase transportation safety and efficiency. However, most commercial systems are prohibitively expensive for usage. Building on past efforts, two relatively inexpensive three-dimensional single point LIDAR systems were generated. The third generation system demonstrated efficiency, ease of use, and options for various resolution levels under the cost of $500. The fourth generation system enhanced the system by removing leading edge bias while creating a standalone system that outpaces the prior generation by 12% in acquisition rate with a resolution four times as dense. Subsequent use of this system with an adaptive active fusion network for scene depth estimation provides upsampling and use of a convolutional spatial propagation network (CSPN) helps refine the full depth map. The performance of proposed network was evaluated and compared with the state-of-the-art methods on the NYUv2 dataset, and the experiment demonstrated its outperformance in reconstruction accuracy, reliance on the number of laser scans, and robustness. Subsequently employing a novel Iterative Statistical Outliers Removal (ISOR) method removes noise from the system and the upsampling methods. Finally, through the combination of upsampling and employing the ISOR method, a lower accuracy point cloud from the fourth generation single-point 3-D LIDAR system can be enhanced to create an accurate 3-D point cloud bordering on real time. This would make the complete system applicable for all transportation-based environments.


| | | | |
|---|---|---|---|
| 17. Key Words<br>Safety, Risk, Laser radar, Adaptive active fusion, Transportation | 18. Distribution Statement<br>No restrictions. | | |
| 19. Security Classif. (of this report)<br>Unclassified | 20. Security Classif. (of this page)<br>Unclassified | 21. No. of Pages<br>58 | 22. Price |

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Convolutional Neural Networks (CNN)
Convolutional Spatial Propagation Layer Network (CSPN)
Department of Transportation (DOT)
Graphics Processing Unit (GPU)
Graphical User Interface (GUI)
High Performance Computing (HPC)
Iterative Statistical Outliers Removal (ISOR)
Light Detection and Ranging (LIDAR)
Mean Absolute Error (MAE)
Mean Square Error (MSE)
Partial Differential Equation (PDE)
Rectified Linear Unit (ReLU)
Recurrent Neural Network (RNN)
Red, Blue, Green (RGB)
Reversed Huber (berHu)
Root Mean Square Error (RMSE)
Serial Clock Line (SCL)
Serial Data Analyzer (SDA)
Static Random-Access Memory (SRAM)
Statistical Outliers Removal (SOR)
Three-Dimensions (3-D)
Two-Dimensions (2-D)
United States (U.S.)
Universal Serial Bus (USB)
Volts Direct Current (VDC)

## Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

Abstract

Mobile light detection and ranging (LIDAR) technology offers a significant opportunity to increase transportation safety and efficiency. However, most commercial systems are prohibitively expensive. Building on past efforts, two relatively inexpensive three-dimensional single point LIDAR systems were generated. The third generation system demonstrated efficiency, ease of use, and options for various resolution levels with a cost of under $500. The fourth generation system enhanced the system by removing leading edge bias while creating a standalone system that outpaces the prior generation by 12% in acquisition rate with a resolution four times as dense. Subsequent use of this system with an adaptive active fusion network for scene depth estimation provides upsampling and use of a convolutional spatial propagation network (CSPN) helps refine the full depth map. Performance of the proposed network is evaluated and compared with the state-of-the-art methods on the NYUv2 dataset, and the experiment demonstrates its outperformance in reconstruction accuracy, reliance on the number of laser scans, and robustness. Subsequently employing a novel Iterative Statistical Outliers Removal (ISOR) method removes noise from the system and the upsampling methods. Finally, through the combination of upsampling and employing the ISOR method, a lower accuracy point cloud from the fourth generation single-point 3-D LIDAR system can be enhanced to create an accurate 3-D point cloud bordering on real time. This would make the complete system applicable for all transportation-based environments.

# Chapter 1 Development of Low-Cost 3-D LIDAR Systems

## 1.1 Background

As evident when reviewing past and current research projects supported by the United States Department of Transportation (USDOT), mobile light detection and ranging (LIDAR) technology offers a significant opportunity to increase transportation safety and efficiency [1]. Unfortunately, these projects typically rely on expensive commercial hardware that prevents widespread usage and potentially mitigates LIDAR's overall benefits. Previous efforts within the PI's laboratory resulted in the construction of an inexpensive LIDAR system ($300) that was able to capture up to 700k data points for a respectively accurate point cloud; hence, setting the stage for extensive implementation [2]. However, data collection took too long for effective usage. As a result, this project built on past research by increasing the sampling and data collection rate while enhancing connectivity and improving the post-processing of the point cloud information. Overall, this ensures that a low-cost and mobile LIDAR system will be suited for transportation-based safety outcomes.

## 1.2 Problem Statement

LIDAR cameras are a remote sensing method that enables effective monitoring of the transportation environment [1]. LIDAR employs the use of near visible light waves to map the surrounding environment in three-dimensions (3-D) [3]. Current applications include forest mapping to track growth, modeling forest fire behavior, classifying land and environmental types, and charting various other environments for a variety of purposes [4-7]. Ground-based LIDAR systems can recognize various road types and identify defects in their respective surfaces while monitoring the environment surrounding roads for potential dangers, such as landslides [8, 9]. While these applications illustrate LIDAR's propensity to provide accurate and detailed

representations, it is often costly to collect these data while respectively difficult to analyze the point cloud files that result from the collocation of this information [4]. Commercial LIDAR systems are highly capable; however, their individual cost ($6k to $100k [10]) might be excessive and outweigh their benefits [11]. As a result, a cost-effective LIDAR system is needed and a previous effort within the PI's laboratory demonstrated success in generating 3-D point clouds [2] with equivalent accuracy to commercial systems used in other USDOT research projects [12]. However, the previous system was too slow (i.e., two hours for a 700,000 data point cloud) and image recognition needed improvement. Therefore, this project moved to a more powerful LIDAR rangefinder to increase the sampling rate while incorporating new hardware and software capabilities via image filtering techniques. These objectives required a collaboration between Mechanical Engineering and Electrical Engineering and Computer Science investigators and students. Two 3-D single point LIDAR systems are detailed in this chapter. The first system was developed by undergraduate students as part of a capstone design project at the University of Kansas. Details regarding this system (third generation) were published in the American Society of Mechanical Engineers International Mechanical Engineering Conference & Exposition and material in this report is taken from the corresponding paper [13]. The second system was created by a graduate student (fourth generation) at the University of Kansas and expanded the capabilities of the third generation system. Enhanced image filtering techniques are described in later chapters.

## 1.3 3-D LIDAR Hardware

Previous work in this area at the authors' institution included the fabrication of two working prototypes. The first was assembled using a Garmin LIDAR-Lite v3 device as the range sensor and an Arduino Mega 2560 v3 as the microprocessor [2]. This system had a final cost of

less than $300 and was able to generate 3-D point clouds with an accuracy similar to many commercial systems. However, it had limited portability since it required a hard connection to a large Direct Current power supply and needed a significantly longer amount of time (e.g., 130 minutes for a 700,000 data point cloud) than commercial systems. The second prototype upgraded the hardware to a Garmin LIDAR-Lite v3HP rangefinder to increase update rates while lowering current consumption and enhancing the microprocessor to a Raspberry Pi 4 Model B. This lowered system runtime by increasing clock frequency and Static Random-Access Memory (SRAM) capacity. This second prototype redesigned housing for the LIDAR rangefinder and two 28BYJ-48 Stepper Motors. While the housing was assembled with primarily 3-D printed components, after testing it was found to have stability deficiencies and was inadequate in supporting the weight of the motors, leading to offset points in the generated point clouds.

For the third and fourth generation prototypes, a Garmin LIDAR-Lite v3HP optical sensor was selected. The Garmin product was chosen as it provided sufficient performance for the LIDAR system's application at a reasonable cost ($150), since low cost is a key project outcome. The other options considered were the TF03 Long-Distance LIDAR Module ($230) and TeraRanger Evo 60m USB ToF Rangefinder ($118). The Garmin LIDAR Lite v3HP provides the fastest update rate while requiring the lowest current, along with being packaged at a reasonable size and weight to ensure smooth translation during data acquisition. Although it is not the least expensive rangefinder available, it is the ideal component for this application as it provides accurate distance readings (+/- 2.5 cm at distances > 2 m) within its range for a relatively low cost.

For both the vertical and horizontal motors, HiLetgo 28BYJ-48 stepper motors were chosen due to their low cost and high precision. Three other motors were considered, a SureStep

Nema 8 Bipolar stepper motor, and both STEPPERONLINE short and SureStep full-bodied Nema 17 stepper motors. The 28BYJ-48 is capable of 64 steps per revolution, which itself does not outpace the other motors (200, 200, and 400 steps per revolution, respectively). However, coupled with a gear reduction ratio of 64, the 28BYJ-48 can produce a total output of 4096 steps per revolution, outperforming the others significantly. In addition to its higher precision, the 28BYJ-48 motors are unipolar whereas the other motors considered are bipolar, meaning that switching direction is more complicated and requires an H-bridge logic board, like a HiLetgo STMicroelectronics L298N. Due to the fact that the system oscillates horizontally while capturing data, the ability to switch direction quickly and easily is important. An H-bridge logic board would also require an additional 6 VDC battery supply, which increases the size and cost of the system. The 28BYJ-48 motors require a HiLetgo ULN2003 driver board; however, this board is capable of powering itself from the microcontroller and it is designed to have a simple plug connection to the motor, eliminating the chance for a wiring mistake. The 28BYJ-48 stepper motors also have the benefit of being less expensive than the others. The drawback of this motor is a limited torque output of 3.5 N×cm, which still outperforms the Nema B at 1.6 N×cm but falls short of the Short Body Nema 17 and Full Body Nema 17 motors, at 13 N×cm and 48 N×cm, respectively.

To run the code that controls the motors, rangefinder, and stores data in an efficient manner, a relatively powerful microcontroller is needed. After researching cost-effective microcontrollers, three options were analyzed: Raspberry Pi 4B, Rock Pi 4 Model C, and Odroid-XU4. While the Rock Pi 4 and Odroid-XU4 offered similar performances to the Raspberry Pi at nearly identical prices, the Raspberry Pi had the highest combination of processing speeds (clock frequency) and available SRAM. A marginally larger size and slightly

higher cost were not significant enough factors to justify selecting another processor with slightly less performance capabilities. Another Raspberry Pi strength is its library of programming languages. For instance, Raspberry Pi's support includes Scratch, Python, C++, and JavaScript.

1.4 System Setup

To best fit the chosen stepper motor model, it was decided to utilize previous custom 3-D printed components produced from a Stratasys Mojo 3-D printer (layer thickness: 0.018 cm) designed specifically for the motors selected. Placed between the horizontal motor and the component containing the rangefinder was a needle-roller thrust bearing (McMaster-Carr model number: 5909K34). This allowed for a U-Bracket to be equally supported on all sides. A multipurpose flanged sleeve bearing (McMaster-Carr model number: 7815K24) was used to support the end opposite from the motor shaft by being placed through the rangefinder mount and the U-Bracket. Both items helped minimize any undesirable interference of vibrations or tilt without prohibiting full rotation range in both the vertical and horizontal directions. A detailed Computer Aided Design drawing of the third generation assembly can be seen in Figure 1.1.

Subsequently, the LIDAR turret (everything above the horizontal stepper motor in fig. 1.1) was redesigned to accommodate bearings that, while also taking weight off the motor, allow for movements that are more precise by reducing the degrees of freedom between the motor and the component that it was moving. A potentiometer was also added that was driven off the motor that drives horizontal motion to address problems in the third generation with the overall fourth generation hardware seen in Figure 1.2.

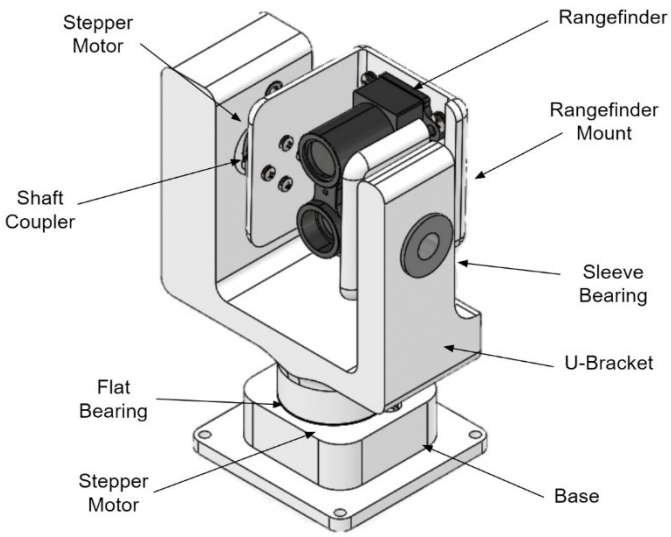


**Figure 1.1** Assembly of the LIDAR housing showing the complete third generation system

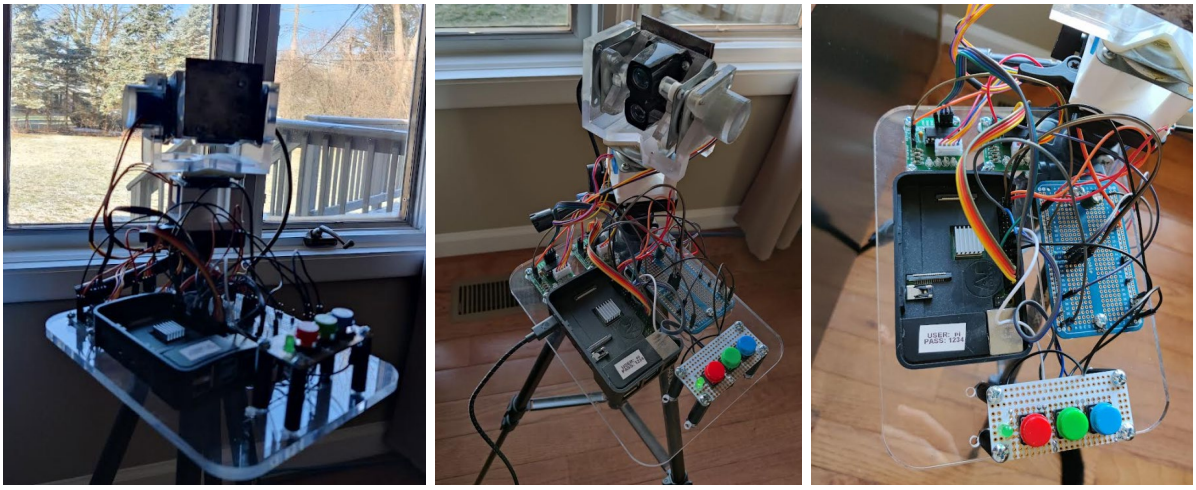

**Figure 1.2** Various images of the fourth generation LIDAR prototype system.

The overall wiring diagram for both the third and fourth generations is shown in Figure 1.3. The rangefinder utilized Serial Clock Line (SCL) and Serial Data Analyzer (SDA) wires. The SCL acted to synchronize all data transfers over an Inter-Integrated Circuit connection, whereas the SDA was responsible for transfer of actual data. Both connections allowed for

communication between the rangefinder and the Raspberry Pi 4B's general purpose input/output pins. The SCL and SDA connections were paired with pull-up 4.7 kΩ resistors to restore the SCL and SDA signals to high when the Raspberry Pi was not transmitting a low signal. The pull-up resistors also ensured that the connections were given a well-defined voltage. The entire system was powered by an external battery source (SlimThin 10000 mAh) connected to the system with a Micro-Universal Serial Bus (USB) to USB cord.

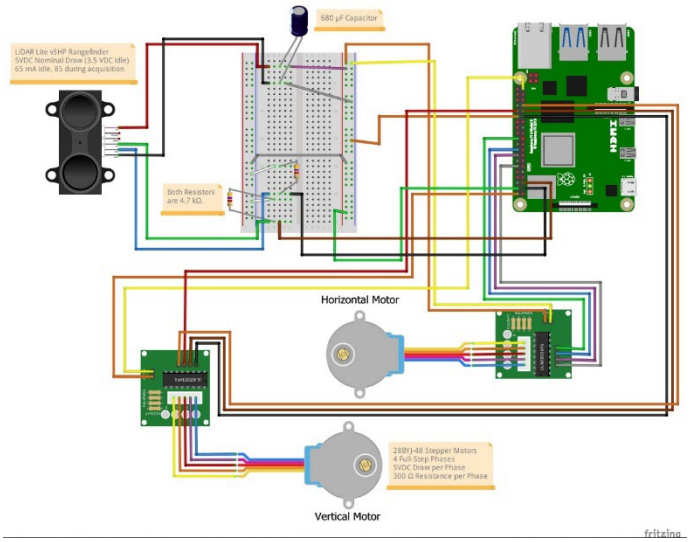


**Figure 1.3** Wiring schematic for third and fourth generation LIDAR systems

The C++ language was first selected to program both LIDAR systems because of the amount of control and precision it can provide. However, with no previous understanding of basic coding in this language, there was a struggle trying to edit and adjust code functionality. Ultimately, Python was chosen for its similarity to MATLAB and ample online resources regarding the use of stepper motors. The code for the third generation prototype is available at: https://github.com/depcik/LIDAR/tree/main/2021. The code for the fourth generation prototype is available at: https://github.com/depcik/LIDAR/tree/main/2023.

## 1.5 Third Generation LIDAR Results and Discussion

A Graphical User Interface (GUI) was created to provide user interaction with the system. The GUI was created in Python 3 on the Raspberry Pi to be used in conjunction with an official Raspberry Pi 7" Touch Screen Display. The GUI displays a visual menu with four interactable buttons: Low Accuracy, Medium Accuracy, High Accuracy, and Exit as seen in Figure 1.4. When a button is selected on the touchscreen, a command is sent to the Raspberry Pi 4 to perform the desired outcome. The three accuracy buttons (low, medium, and high) have embedded text to describe their intended purpose. Pressing each accuracy button starts the LIDAR system and runs the code to allow for motor rotation and scanning to create a 3-D point cloud. The buttons from left to right increase the time it takes for a scan to complete while improving the accuracy of the 3-D point cloud. The accuracy is enhanced by increasing the number of horizontal data points taken per degree that creates a denser 3-D point cloud with less space in-between data points. It should be noted that the three options share the same number of vertical data points taken and only vary in the horizontal direction. The amount of vertical data points remains constant since the perceived slight enhancement in image quality on the vertical axis is deemed initially inconsequential to the significant increase in scan time.

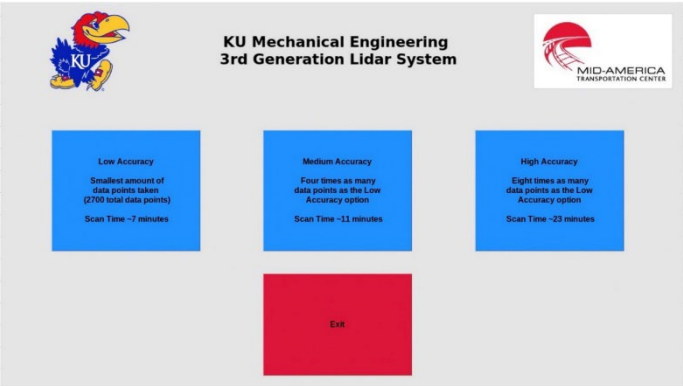


**Figure 1.4** Screenshot of the Graphical User Interface for the third generation LIDAR system

The left button results in the lowest number of data points, while the middle button increases the number of horizontal data points scanned by four times, and the right button increases the number of horizontal data points scanned by eight times. These options produce 2700, 10800, and 21600 total data points, respectively, when the default values for degrees swept in the horizontal (32.16°) and vertical (42.19°) direction are used. The approximate time to complete a scan for the buttons from left to right are seven minutes, eleven minutes, and twenty-three minutes, respectively.

As an initial test of the system, a bag of sugar with noticeable crinkling was placed on a chair in front of a wall in Figure 1.5. Scanning the image took 7.24 minutes and a 3-D point cloud, 2-D point cloud, and 2-D contour plot can be seen in Figure 1.6.


**Figure 1.5** Bag of sugar on a chair in front of a wall

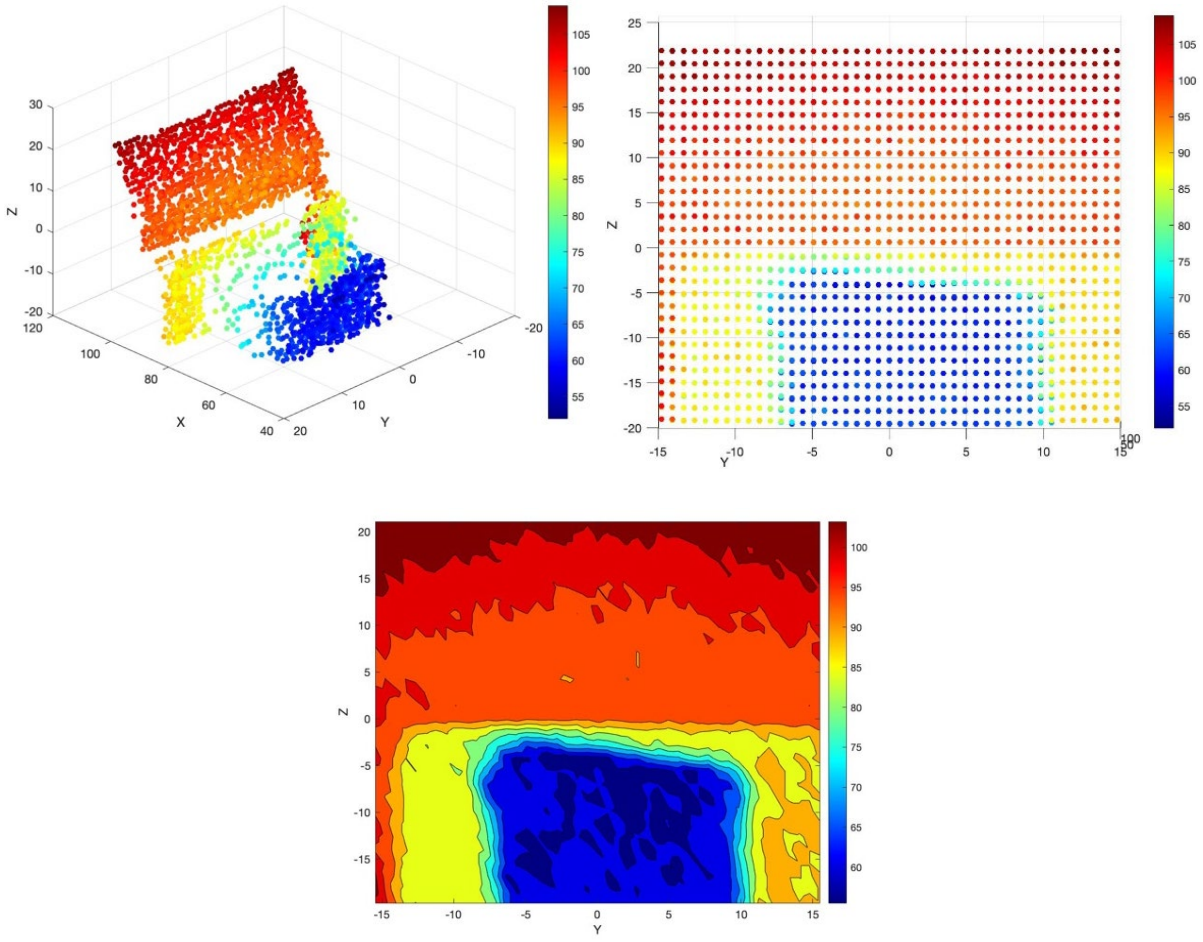


**Figure 1.6** (Top Left) 3-D point cloud, (Top Right) 2-D point cloud, and (Bottom) 2-D contour plot of the bag of sugar in fig. 1.5

While the outline of the sugar bag and chair can be easily seen, there were issues with the system detecting a wall in the background. The different red colors indicate dissimilar depth distances, which should not occur as the wall was a uniform distance away from the system. The 2-D point cloud also had too much space in between each point, limiting the resolution of each plot produced. Although the image was relatively good in a short amount of runtime, sacrificing time to achieve a more accurate point cloud became more ideal.

Since the rangefinder was rotating, it effectively moved away from the wall (as a function of its starting point) during data collection. Thus, to correct the depth issue, the distance

measurement from the LIDAR was multiplied by the cosine of the horizontal angle as well as the cosine of the vertical angle. By taking both angle measurements into account, the correct depth was recorded. An object with more defined edges and unique shapes was used for the final test. Figure 1.7 shows a shoe with uneven parts and edges on top of a box. Figure 1.8 is the 3-D point cloud, 2-D point cloud, and 2-D contour plot created from the MATLAB code with the new data. The total scan time for this test was 7.26 minutes.

**Figure 1.7** Shoe on top of box used for testing third generation prototype

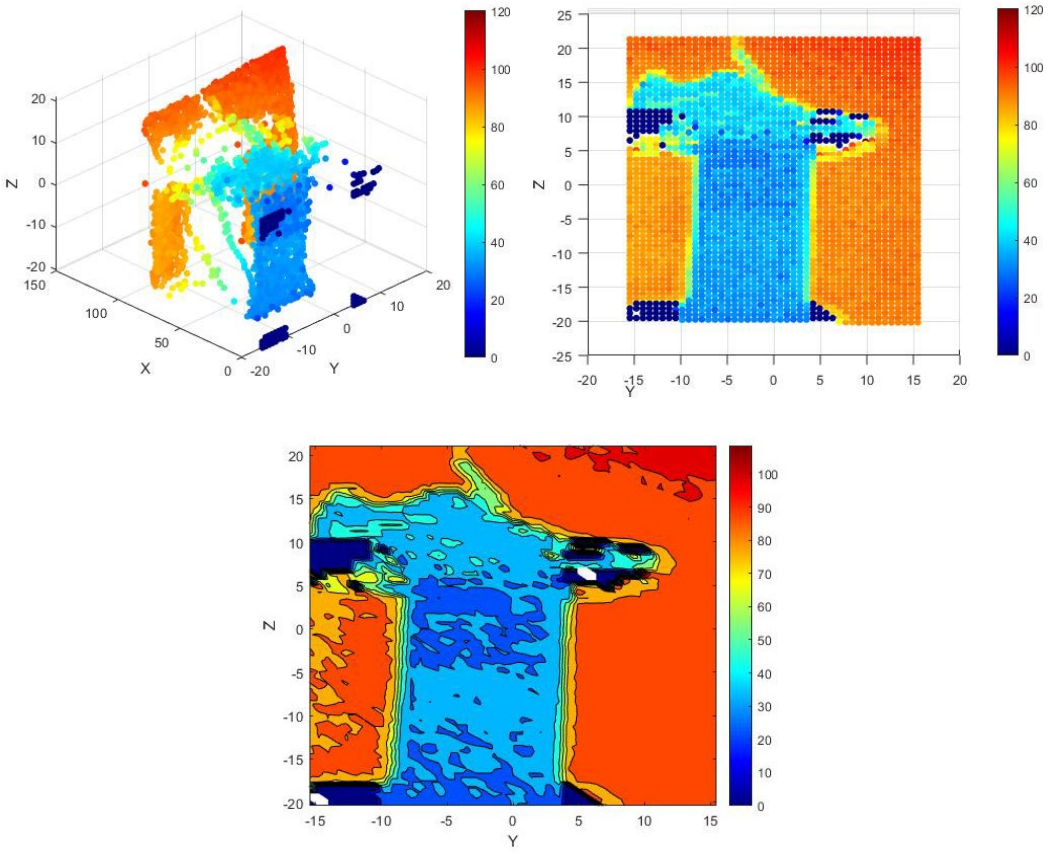


**Figure 1.8** Low accuracy (Top Left) 3-D point cloud, (Top Right) 2-D point cloud, and (Bottom) 2-D contour plot of the shoe in fig. 1.7

Overall, the outline of the shoe was evident along with the correct wall depth when looking at the 2-D images; however, the 3-D point cloud did not provide a recognizable image. Thus, the next step was to increase the number of data points recorded to improve accuracy. This was accomplished two different ways: full stepping and half stepping. In full stepping, data are recorded each full step of the motor whereas before, data was only recorded at the end of the entire motor step. This led to four times as many data points and a more accurate 3-D point cloud, 2-D point cloud, and 2-D contour plot of the same shoe and box setup in Figure 1.9 with an overall run time of 11.27 minutes.

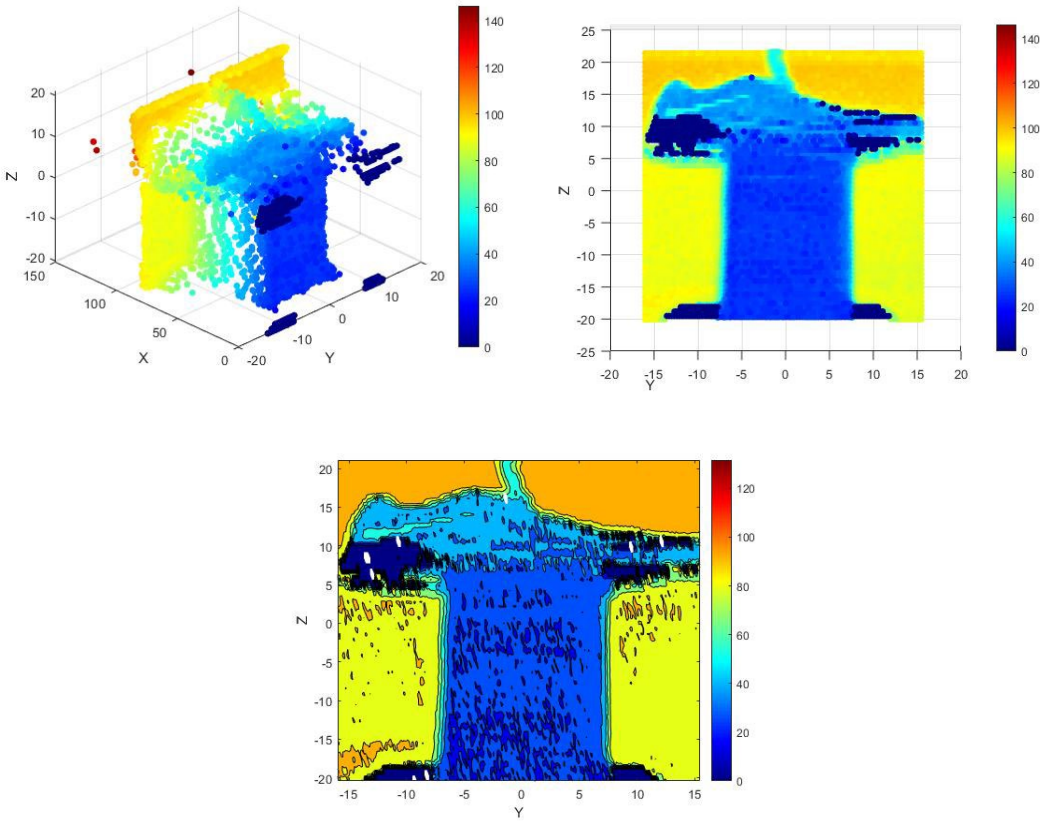


**Figure 1.9** Medium accuracy (Top Left) 3-D point cloud, (Top Right) 2-D point cloud, and (Bottom) 2-D contour plot of the shoe in fig. 1.7

This resulted in a more noticeable shoe shape in the 3-D point cloud (e.g., heel, facing, and toe cap) with the 2-D images illustrating more ridges along the shoe. To further increase the number of data points, half stepping was used where an additional step of data collection occurred in between the full steps. Half stepping produces eight times as many points as the low accuracy code and two times as many as full stepping. The same shoe-on-box setup was used and the 3-D point cloud, 2-D point cloud, and 2-D contour plot results are seen in Figure 1.10, respectively. In all three figures, the outline of the shoe becomes more evident along with a distinct heel, sole, collar, facing, tongue, and toe cap. Interestingly, the logo of the shoe becomes

more noticeable as it starts to reflect the laser. The run time for this experiment was clocked at 22.58 minutes.

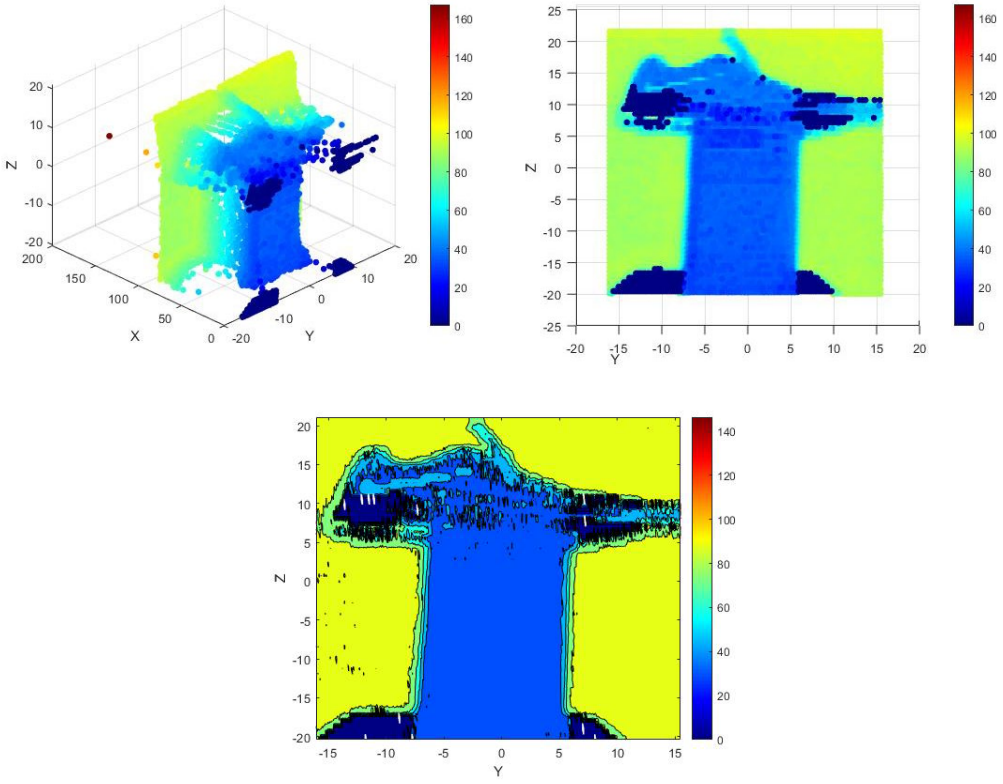


**Figure 1.10** High accuracy (Top Left) 3-D point cloud, (Top Right) 2-D point cloud, and (Bottom) 2-D contour plot of the shoe in fig. 1.7

Overall, three different levels of measuring accuracy were generated, able to produce point clouds of 2700, 10800, and 21600 points, resulting in images of low, medium, and high accuracy, respectively. The scan times are relatively short, with the most accurate at slightly over 23 minutes, the medium accuracy at approximately 11 minutes, and the low accuracy at a little over 7 minutes. In addition to being able to capture images quickly and efficiently, the system is also cheaper than many alternatives with a total cost of less than $500. The system also remains

respectively simple to use as the additions of a GUI, clean packaging, and an external battery allow users to operate the system at any location.

## 1.6 Fourth Generation LIDAR Results and Discussion

Throughout the project, the third generation prototype was benchmarked and analyzed to determine the abilities of the previous iteration and investigate where improvements could be made for the fourth generation prototype. This effort emphasized two key areas where improvements would make the biggest impact. The first is eliminating the "leading edge bias" present in the third generation prototype, as well as in early iterations of the fourth generation Python script. This bias can be seen on the left in Figure 1.11 with significant removal of the bias as demonstrated on the right.

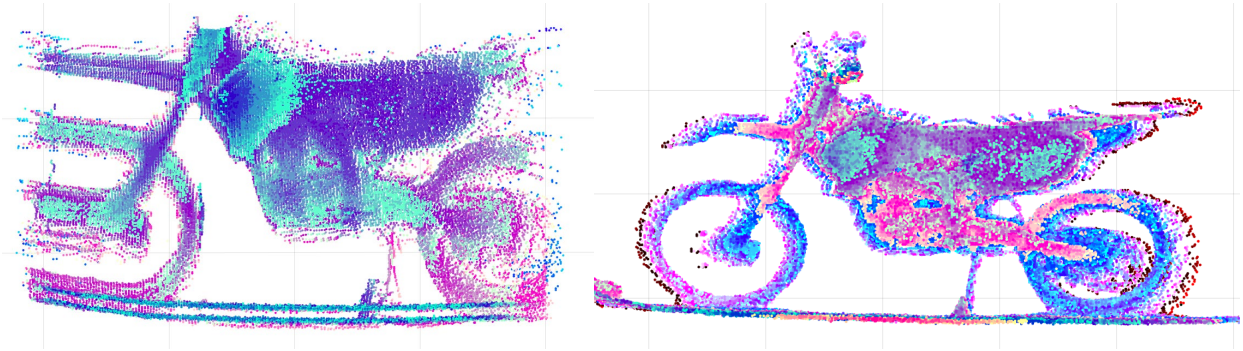

**Figure 1.11** (Left) Displays capabilities and issues with the high frequency vertical (HFV) program; (Right) Shows capabilities of the updated HFV program

By the end of the project, the leading edge bias was largely eliminated, and the maximum resolution of the LIDAR system was quadrupled per unit area. To make the system more versatile, three resolution settings were configured so that one could choose the resolution required for a given scan. This allows the operator to do quick, low resolution scans when

precision is not paramount and slower, high resolution scans when more detailed scenes are being measured. The program was also configured to run from boot using three push-to-start buttons to scan a predefined range at each level of resolution.

To demonstrate the applicability of the fourth generation system to classify land and the environment in a real world scenario, Figure 1.12 provides a 3-D LIDAR point cloud of trees and converging fence lines as shown in Figure 1.13. The tree overhanging the fence line can be seen in the point cloud through respective shading. Additionally, the tree located behind the fence line can be captured with relatively good accuracy. Similarly, Figure 1.14 provides a 3-D LIDAR point cloud of a tree and runoff ditch as shown in Figure 1.15. Both the tree and runoff ditch can be distinguished from the 3-D LIDAR point cloud demonstrating the successful ability of the fourth generation prototype to identify land and the environment.

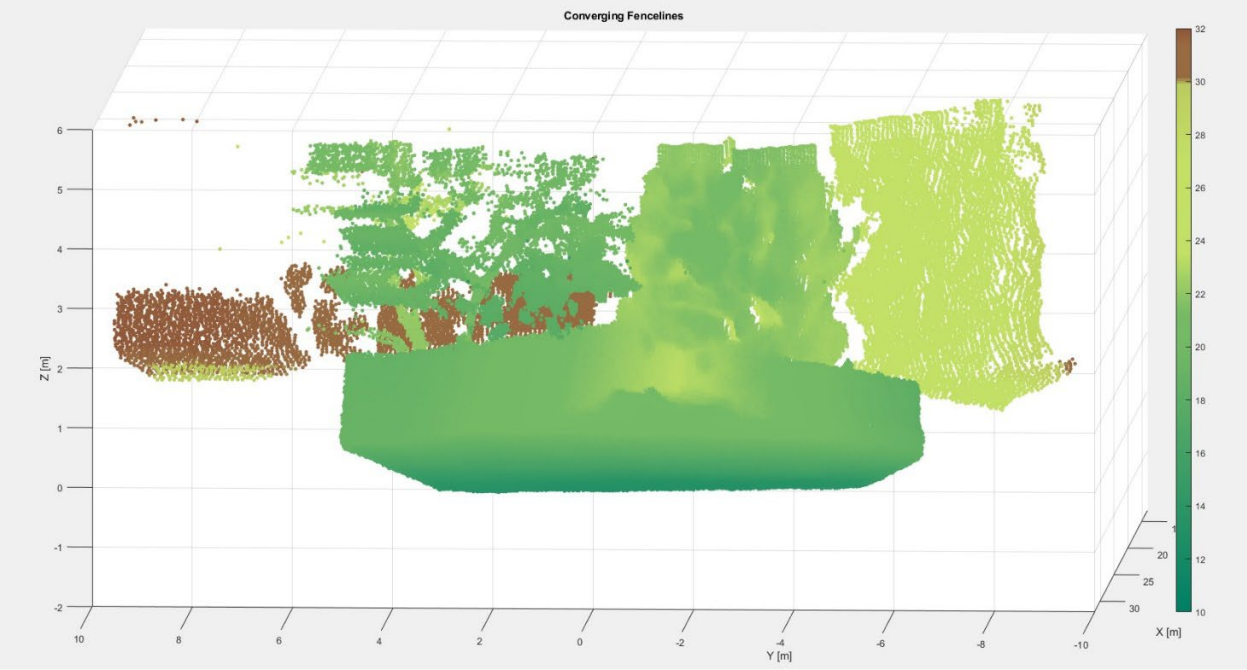


**Figure 1.12** 3-D LIDAR scan of trees and converging fence lines as shown in the picture of fig. 1.13

**Figure 1.13** Picture of converging fence lines and two trees, one overhanging the fence and the other behind the fence

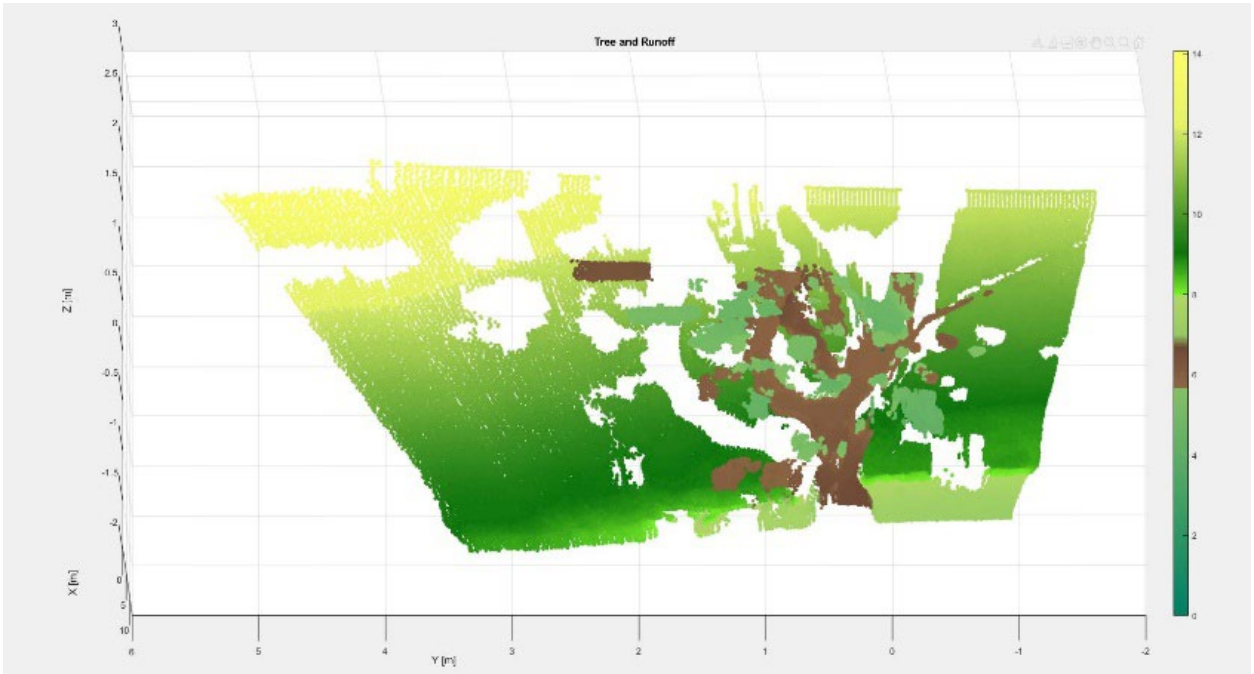


**Figure 1.14** 3-D LIDAR scan of tree and runoff ditch as shown in the picture of fig. 1.15

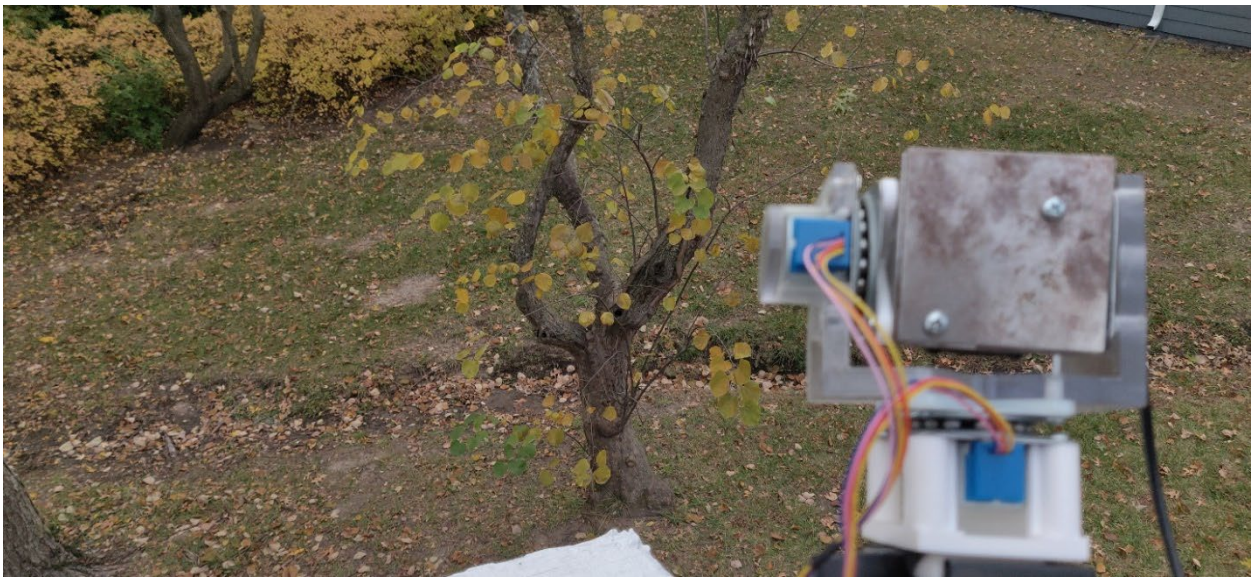

**Figure 1.15** Picture of tree and runoff ditch with the tree in front of the ditch

Subsequently, the fourth generation prototype was tested to determine if it could identify automobiles in a real world environment. Figure 1.16 provides a 3-D LIDAR scan of a static 1974 Volkswagen Beetle as shown in Figure 1.17. The 3-D point cloud demonstrates high accuracy in capturing the vehicle as evident through the wheel wells and the open door. Figure 1.18 provides a 3-D LIDAR scan of a parking lot as shown in Figure 1.19. The closer vehicle (Ford Ranger) is seen with high accuracy, whereas the vehicle farther away (Kia Soul) starts to blend in with the environment.

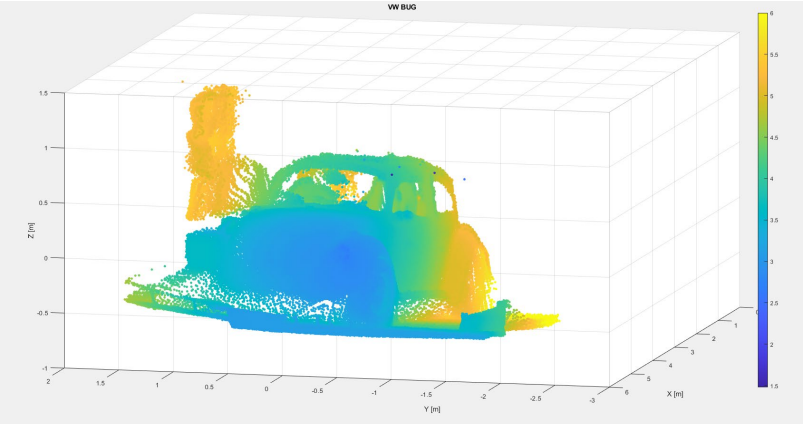


**Figure 1.16** 3-D LIDAR scan of the 1974 Volkswagen Beetle as shown in the picture of fig. 1.17


**Figure 1.17** Picture of a 1974 Volkswagen Beetle

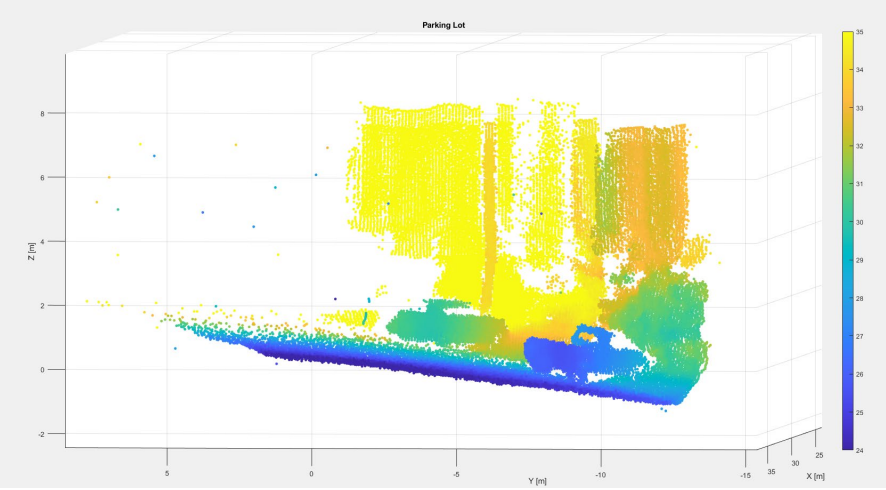


**Figure 1.18** 3-D LIDAR scan of a parking lot with a Ford Ranger (closer) and Kia Soul (farther)

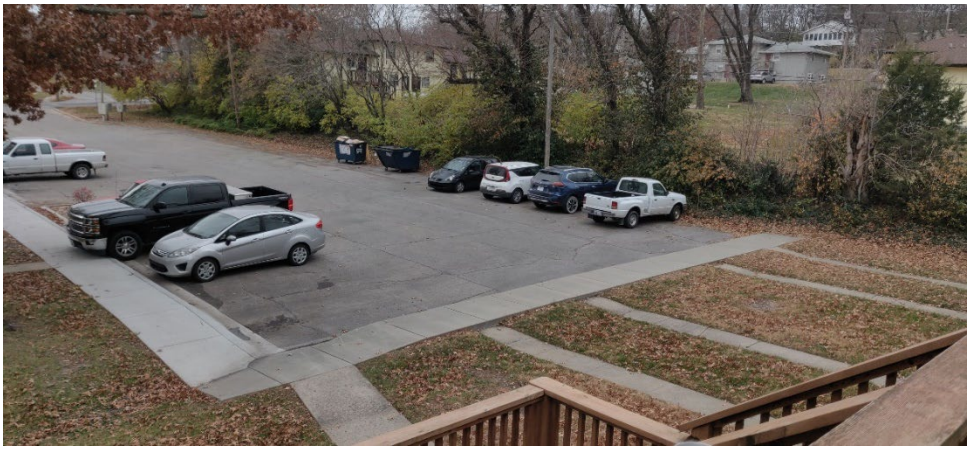

**Figure 1.19** Picture of parking lot with Ford Ranger (closer) and Kia Soul (further). It is worth noting that the scan was taken overnight to minimize vehicle movement, resulting in the addition of vehicles in the image that were not present in the scan.

The fourth generation prototype includes a user interface to allow for standalone, push-to-start functionality with varying resolutions. This means off-the-grid applications would only need a USB-C 5 VDC power source to run the device and conduct measurements. It builds off the third generation prototype in providing three different accuracy options where run time increases as accuracy increases. It outpaces its predecessor with a 12% higher acquisition rate and has a configurable resolution that is four times as dense at peak resolution. Additionally, this device can be run using only a power bank and USB-C cable to maximize its off-the-grid capabilities. The lidar system is run by connecting it to power, waiting for the “ready” light to illuminate, and selecting the scan precision. Final packaging is constructed to ensure the prototype is as durable and versatile as possible. This includes a 0.25” acrylic sheet that holds all the hardware and mounts to a tripod to maximize stability and point cloud accuracy. The resulting fourth generation prototype is suited for static analysis of the environment (e.g., land, vehicles), but is not fast enough to capture moving vehicles. The combination of all these factors

results in an ideal starting place for others to enhance the output and create an inexpensive system that could be widely utilized. Subsequently deployment within the transportation sector will aid in the recognition of low-quality or hazardous roadways and surfaces helping to create a safer environment.

# Chapter 2 Depth Sensing Model

Material in the next two chapters is taken from the published paper of the PI and Co-PI and their students [14]. Depth sensing is an essential problem for LIDAR performance. In this report, we propose a depth completion model that can improve the performance of depth sensing in existing LIDAR systems. The proposed method is a convolutional neural network model, following encoder-decoder architecture with convolutional spatial propagation layer network (CSPN) for refinement and saliency detection plugin for visual attention. The preprocessing layer takes a red, blue, green (RGB) image as input, and produces back three different outputs: saliency map, edge map, and a uniform grid. The three outputs are fused with the depth map and produce information centralized depth regions to become the inputs of the encoder-decoder model. The spatial propagation layer takes the original depth map as input to produce affinity matrices describing global information. The output of the encoder-decoder model uses affinity information to iteratively perform the refining step. Through various evaluation metrics, such as Root Mean Square Error (RMSE), Mean Square Error (MSE), Mean Absolute Error (MAE) and Delta $\delta_k$, the proposed model illustrates the outperformance over CSPN and Sparse-To-Dense models.

## 2.1 Background

Although there has been significant progress in controlling the quality of LIDAR scans, research regarding the LIDAR sampling rate is still an open challenge. The main limitation of the current cost-effective LIDAR systems is their time-consuming sampling rate compared with 3-D LIDAR counterparts. There are two main approaches for increasing the sampling rate: hardware-based and software-based. Although the hardware-based approach has more successful results and research interests [15], this approach requires an additional hardware update cost, which may

be a trade-off with the cost-efficiency characteristics of the product. Meanwhile, the software-based approach is still a fresh field that has not been thoroughly discovered yet.

For the software-based approach, there are two main methods to increase the depth sensing density: 1) point cloud upsampling and 2) depth completing. Although there has been significant research for point cloud upsampling [16-18], this method requires a dense point cloud input, which is unsuitable for the real-time requirement of a depth sensing system. The second approach (building a depth completion model) is a more promising method for a cost-effective LIDAR system, since it can generate the depth stream in real-time.

Depth completion models have illustrated its importance and effectiveness in various applications, such as target detection [19, 20], 3-D scene reconstruction [15, 21], and surface defect detection [22-24]. These depth estimation methods rely on accurate 3-D LIDAR or Line-Scan LIDARs, which capture dense depth maps of a scene in real time. These sensor fusion methods cannot be directly applied on single-point depth sensors [25], which only sample one point in a scan. The primary challenge in low-cost single-point LIDARs is to balance scanning density and real-time performance, which are determined by the scanning strategy. For instance, a high-resolution point cloud requires a sufficient sampling distribution of scans, which is time-consuming to move the single-point sensor and perform scans. A mechanism that optimizes the sampling ratio and distribution will significantly improve scanning efficiency and real-time performance. The second challenge is the blurry outputs an image-based depth estimation model usually obtains, such as the depth completion model using Convolutional Neural Networks (CNNs) [26, 27]. The blurry results are attributed to the predictions solely coming from local extracted features. Spatial propagation and affinities learning have been employed to obtain a

sharper depth estimation by maintaining  global information and iteratively refining the depth predictions [26, 28-30].

The rest of this chapter is organized as follows: First, we introduce the convolutional neural network architecture of the depth completion model and its essential components/layers. Next, we theoretically describe two additional add-on components of the network: 1) Saliency Detection Layer for visual attention mechanism, and 2) Convolutional Spatial Propagation for depth resolution refining. We then introduce evaluation metrics, experiment settings and results. Finally, the conclusion section sums up our opinions of the results and future research directions.

## 2.2 Model Architecture

We proposed a depth completion model that takes RGB and sparse depth map as input, and outputs a refined dense depth image, as shown in Figure 2.1. The model contains a preprocessing layer using a visual attention mechanism, an encoder-decoder neural architecture, and a convolutional spatial propagation layer for better quality results. The preprocessing layer takes an RGB image as input and produces three different outputs: saliency map, edge map and a uniform grid. The three outputs are fused with the depth map to produce information centralized depth regions for input into the encoder-decoder model. The spatial propagation layer takes the original depth map as input and produces an affinity matrix describing global information. The output of the encoder-decoder model uses affinity information to iteratively complete the refining step. The final output of the model after the $n^{th}$ iteration is the refined dense depth map, which is ready for 3-D point cloud conversion.

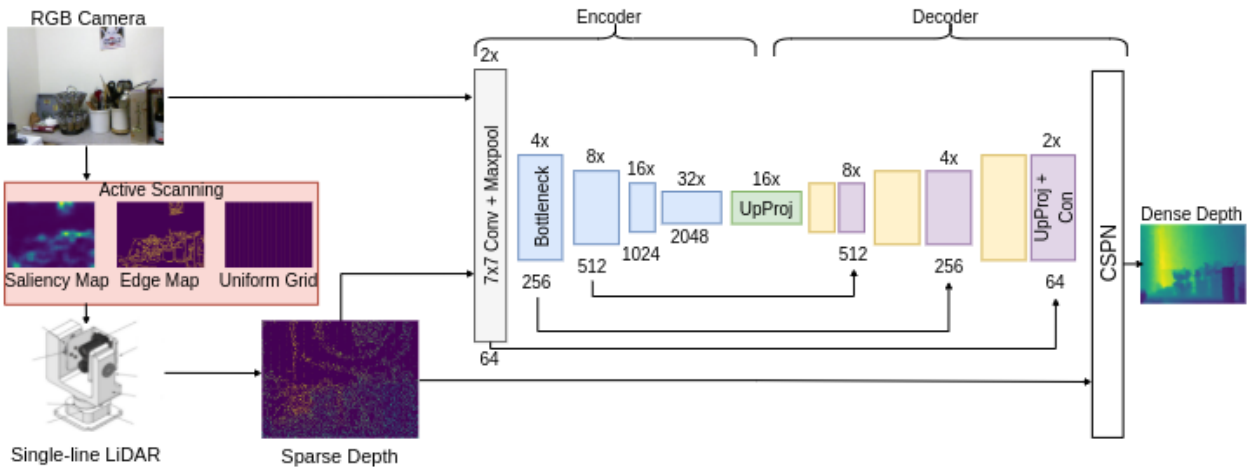


**Figure 2.1** The overall framework of our depth completion network. Given a RGB-D input (304 × 228 × 4), the tensor first goes through 7 × 7 convolutional layer, following with 4 downsampling steps and 4 upsampling steps. Sparse depth map is directly mapped as affinity to guide the depth completion. Notice the integration of the third generation LIDAR system as described in the previous chapter.

The method presented in this research uses RGB and the sparse depth map to predict a dense depth image. The main challenge when developing the depth reconstruction model is the difficulty in maintaining the global spatial information. Although convolution can capture local relations, conventional convolutional-based models lack the ability to capture global information. In this chapter, we design a spatial propagation network that can capture global information [28]. Global features are represented in the form of affinity matrices, which can be used for depth refinement.

To reconstruct depth information, the model takes a sparse depth map $D_0 \in R^{m \times n}$ aligned with the RGB image $X \in R^{m \times n}$ as input. The edge and saliency maps are extracted from an RGB image and merged with the uniform grid map into a binary mask. The mask map is used to guide the depth extraction process, concentrating the most essential information on the depth map for the estimation. The extracted depth map and original RGB are used as inputs for the model. The

depth map is further routed directly to a spatial propagation layer to generate affinity matrices, which are concatenated with the encoder-decoder output.

The model contains an encoder, which starts with a large convolution size of seven by seven kernels that have max pooling, followed by four consecutive bottlenecks. The feature maps are down-sampled by half their input size while the feature channel dimension is doubled. Following He et al. [31], each bottleneck is a block of three consecutive convolution layers with kernel choices $[1,3,1]$ respectively. The Rectified Linear Unit (ReLU) activation function is used at each layer. The bottleneck layers are used to compress information of the input feature map into a latent representation.

The output feature map from the encoder is fed to the decoder, which is a sequence of four up-sampling blocks. Following Laina et al. [32], each up-sampling block contains a padded convolution layer (that does not change the size of the feature map input), an up-projection layer, followed by a ReLU layer. Each up-sampling step uses information from the feature channel to estimate back pixels in the up-scale feature map, which increases the shape of the feature map by two while reducing the feature channel dimension by $2^2$.

The sparse depth map $D_0 \in R^{m \times n}$ is routed directly to the Convolutional Spatial Propagation Network (CSPN) layer to generate affinity matrices $D_n$ through $n$ iteration steps. The CSPN [28] uses a recurring relation between nearby pixels to propagate spatial information in a specific direction. Affinity matrices are generated by combining the propagation results from different directions. The Convolution Spatial Propagation Network uses a convolution kernel to instruct the propagating, which allows for the comopuation of affinity matrices in a single iteration as anisotropic diffusion process. This approach increases the computation efficiency and learning ability [28].

*2.2.1 Encoder*

The encoder extract feature maps from RGB-D tensor $304 \times 228 \times 4$ uses convolution operations. Feature maps are generated by applying a trainable kernel window size $k$ on the specific stride $s$ and padding $p$. The sliding kernel window on the tensor input compresses information locally and returns an embedded representation of features in a hidden dimension. During this phase the channel dimension is also increased, while the width and height dimensions are reduced by half on each sampling step.

The encoder contains four down-sampling layers. Each layer contains three residual blocks with bottleneck architecture, as represented in Figure 2.2. [31]. It contains an identity and residual information flow. In the identity path (top), original information from the input is maintained. Meanwhile, information from residual path (bottom) is learned by the model and identifies the number of useful layers from the learning process. Information from both paths is combined using concatenation ($\oplus$ operator). The ReLU activation function is used following the He weight initialization method [33] for convolutional layers.

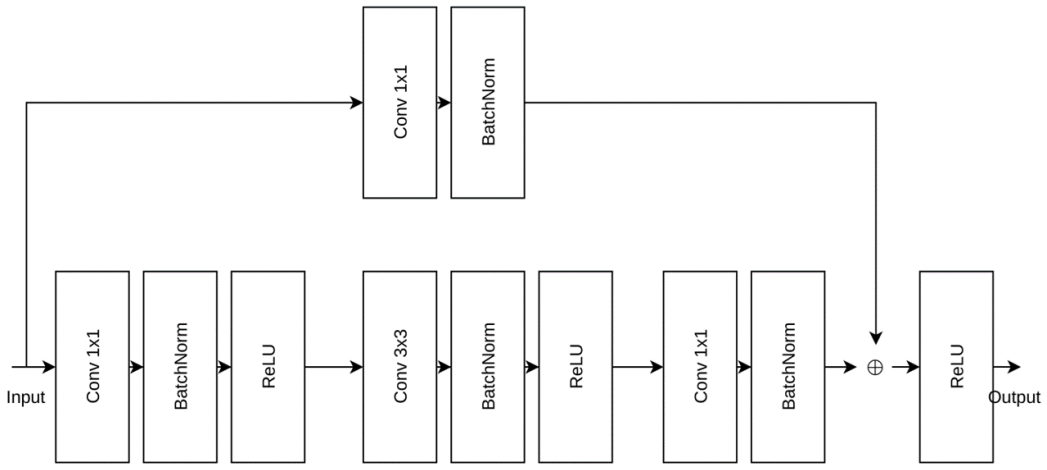


**Figure 2.2** Structure of the bottleneck residual block

*2.2.2 Decoder*

The decoder component contains four up-sampling layers, where each layer doubles the output dimensions (width and length) using compressed information within the channel dimension. Each up-sampling step is an up-projection block defined by Laina et al. [32]. To improve the robustness of unpooling, each layer is concatenated with a $5 \times 5$ convolution and a ReLU activation. This up-sampling architecture is known as an up-convolution block. Additionally, by stacking up four up-convolution blocks, the feature maps can be scaled up 16 times in width and height. Further following an extension proposed by Laina et al. [28], we adapt the residual block architecture for up-projection, which can be described by Figure 2.3.

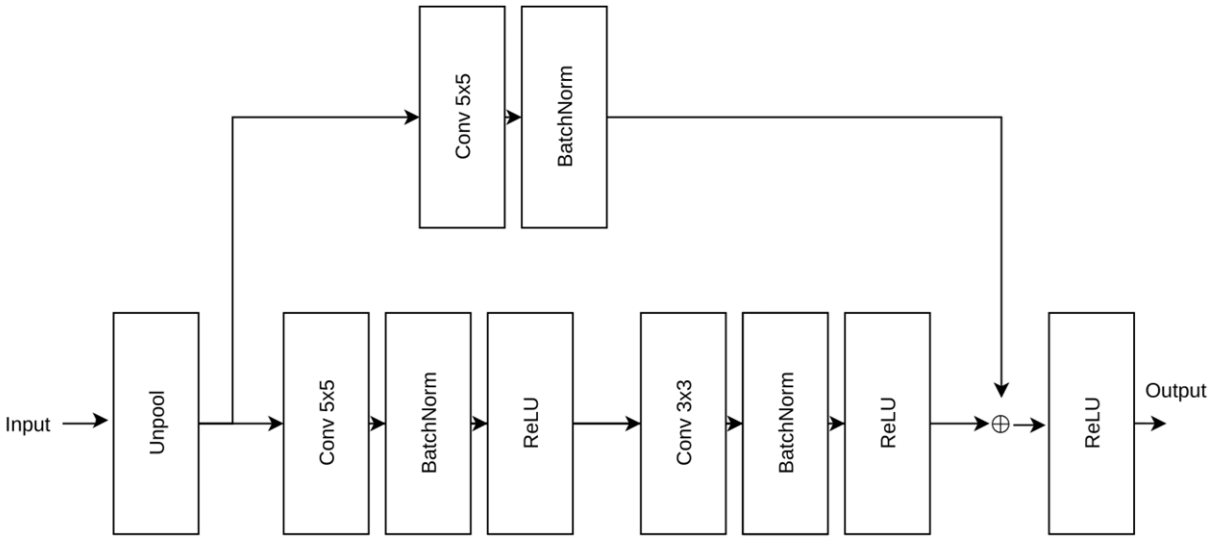


**Figure 2.3** Structure of the bottleneck residual block

*2.2.3 Adaptive Active Fusion*

Visual attention mimics the behavior of human vision by concentrating on important segments, therefore it is applied to condense information from the full depth map. This approach allows the depth completion model to omit redundant information, only highlighting the most

important information of an image. Based on this idea, the visual attention layer can significantly reduce the number of depth inputs required overhead and boost the depth prediction performance. In this chapter, we apply a heuristic-based saliency method named log-spectrum saliency, which is a static method applied on a single image. Log-spectrum saliency uses log form representation of an image to perform salient localization. Log-spectrum of an image is a frequency domain representation and has been commonly used in the fast semantic categorization problem [34]. Images having the same semantic characteristics (such as indoor/outdoor and maximum capturing distance) will have the same log patterns. Log-spectrum representation is first used in saliency localization by Hou and Zhang [34]. The process minimizes the information redundancy from the original image so only important pixels remain. The log-spectrum method provides a convenient way to represent duplicating information in the frequency domain.

Given an image $D \in R^{m \times n \times c}$, its log spectrum $\mathcal{L}(f)$ in the frequency domain is obtainable by using the Fourier Transform method $J$. We define $\mathcal{R}(f)$ as the spectral residual of an image, which can be obtained through statistical analysis of $\mathcal{L}(f)$. More specifically, $\mathcal{R}(f)$ can be defined as follows:

$$\mathcal{R}(f) = \mathcal{L}(f) - \mathcal{A}(f) \tag{2.1}$$

where $\mathcal{A}(f)$ is a general shape of log spectra and defines the filtered region on the frequency domain. $\mathcal{A}(f)$ can be seen as a controllable prior hypothesis. However, Hou and Zhang [34] used a heuristic method to approximate the size of this filter region using convolution operation on log-spectrum input. $\mathcal{A}(f)$ can be approximated as follows:

$$\mathcal{A}(f) = \frac{1}{n^2} J_n * \mathcal{L}(f) \tag{2.2}$$

where $*$ is a convolution operation, $J_n$ is a kernel of ones $n \times$ n, defined as:

$$J_n = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \dots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \tag{2.3}$$

Spectral residual $\mathcal{R}(f)$ is a compressed representation of the saliency map in the frequency domain. It suppresses unimportant information and maintains non-trivial parts. To obtain the saliency map $\mathcal{S}(f)$ in the spatial domain, Inverse Fourier Transformation is applied as follows:

$$\mathcal{S}(f) = \mathcal{J}^{-1}\left[exp\left(\mathcal{R}(f)\right)\right]^2 \tag{2.4}$$

where $\mathcal{J}^{-1}$ is the Inverse Fourier Transformation and $\mathcal{R}(f)$ is a frequency domain signal.

We recognized that saliency extraction alone is not sufficient for the depth prediction problem. Significant information is not obtainable using a visual attention mechanism, such as shapes and local boundaries. Therefore, an edge detection model is integrated along with the saliency method. Here, we apply the standard Canny edge detecting method [35] due to its computational efficiency for real-time application.

The Canny detection algorithm uses intensity characteristics of an edge to perform the recognition. The first order derivative gradient can be obtained in the horizontal direction $G_x$ and vertical direction $G_y$ by convolution with Sobel kernel. The obtained derivatives can be used to estimate gradient and direction as follows where $G$ is the local gradient and $\Theta$ is its direction angle.

Once the gradient $G$ and orientation angle $\Theta$ at each pixel are obtained, the Canny algorithm can be applied to detect valid edges. Canny performs a non-maximum suppression on each pixel before returning all possible edges existing in the image. The hysteresis thresholding

is then used to filter out invalid edges $E(G, \Theta)$. For a more detailed overview about the edge detection algorithm, see Canny [35].

Although detected edges and saliency maps provide more compacted representation of the input in centralized regions, depth prediction performance using these features alone is worse compared to uniform depth sampling [28], especially in non-centralized regions. To increase the generality of the depth mask, we perform two further preprocessing steps: 1) reduce the point density on the saliency region, and 2) integrate uniform grid mask to capture information from non-centralized regions. The grid mask $G$ contains 1000 pixels uniformly distributed through the *xy*-dimension.

The binary mask for depth image is obtained by applying the element-wise product $\oplus$ on the filtered saliency map, edge map, and uniform grid mask as follows:

$$\mathcal{D} = \mathcal{S}^*(f) \odot E(G, \Theta) \odot \mathcal{G} \tag{2.5}$$

where $\mathcal{S}^*$ is the filtered saliency map.

*2.2.4 Spatial Propagation*

Given a sparse depth input $D_0 \in R^{m \times n}$, the spatial propagation layer generates a feature map in hidden space $H \in R^{m \times n \times c}$ where $c$ is the feature dimension. We denote $H_{i,j}$ is a cell of feature map $H$, $H_0$ is the feature map $H$ at iteration 0, $H_{i,j,t}$ is the cell at $i, j$ of H at iteration $t$. Using these notations, Convolutional Spatial Propagation Network (CSPN) defines the recurrence relation of hidden state $H_{i,j,t+1}$ as follows:

$$H_{i,j,t+1} = \kappa_{i,j}(0,0) \odot H_{i,j,0} + \sum_{a,b=\frac{-(k-1)}{2}}^{\frac{(k-1)}{2}} \kappa_{\mathrm{i,j}}(\mathrm{a,b}) \odot \mathrm{H_{i-a,j-b,t}} \tag{2.6}$$

where $\kappa \in R^{k\times k}$ is the normalized kernel size $k$, $H_{i,j,0}$ is a cell of $H$ at iteration 0, $H_{i-a,j-b,t}$ is a cell of $H$ at iteration $t$. More specifically, $\kappa_{i,j}$ can be defined by normalizing the equation:

$$\kappa_{i,j}(a,b) = \begin{cases} \dfrac{\hat{\kappa}_{i,j}(a,b)}{\sum_{a,b\neq 0}\left|\hat{\kappa}_{i,j}(a,b)\right|} & \text{if a,b not 0} \\ 1 - \sum_{a,b\neq 0} \kappa_{i,j}(a,b) & \text{otherwise} \end{cases} \tag{2.7}$$

where $\hat{\kappa} \in \mathcal{R}^{k\times k}$ is a kernel size $k$. After $n$ iterations, CSPN returns a tensor $H_n \in R^{m\times n\times c}$, which is concatenated with a feature map from encoder-decoder components, and generates refined results.

Figure 2.4 illustrates an intuitive explanation of the spatial propagation mechanism. Pixels of the depth image are used as the input for the Recurrent Neural Network (RNN) model. Pixel $x_{k,t-1}$ informs pixel $x_{k,t}$ through a recurrence relation between hidden states $h_{k,t-1}$ and $h_{k,t}$. Liu et al. [30] proposed linear recurrent propagation through a one-dimensional (1-D) sequence as shown in Figure 2.4a. The idea is extended for multiple connectors [30] (fig. 2.4b) and convolutional propagation [28] (fig. 2.4c).

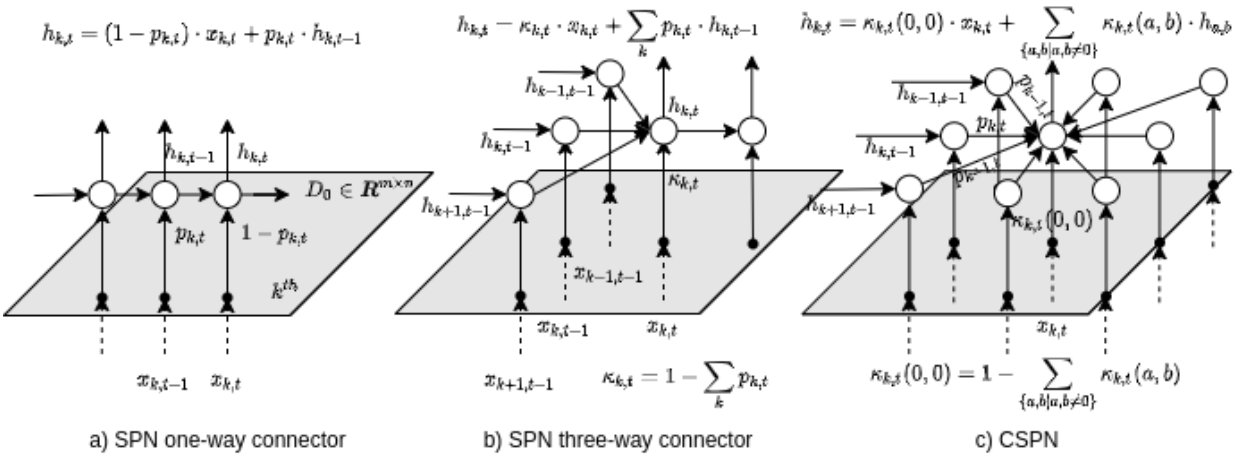


**Figure 2.4** Different spatial propagation strategies. Inputs of the recurrent model is sequence of pixels in depth image. a) Spatial information is propagated linearly in specific direction as Recurrent Neural Network (RNN). b) Spatial information is propagated in specific many-to-one pattern (three-way connector). c) Spatial information is propagated using neighbor pixels, similar to convolutional kernel operator.

## 2.3 Optimizing Function

To train the multiple regression model for depth estimation, we adopt the Reversed Huber loss function [36]. According to Ma and Karaman [26], the Reversed Huber function is designed to be less sensitive to large weight values than $L_2$, and less sensitive to small weight values than $L_1$. More specifically, Reversed Huber (denoted as berHu) is defined as follows:

$$\mathcal{B}(e) = \begin{cases} |e| & |e|<c \\ \dfrac{e^2 + c^2}{2c} & \text{otherwise} \end{cases} \tag{2.8}$$

where $e$ is the absolute difference between predicted depth $D_n \in R^{m\times n}$ and ground truth $D^* \in R^{m\times n}$ and $c$ is the 20% maximum absolute error threshold within the working batch.

## 2.4 Evaluation Metrics

To compare depth estimation performance from different approaches, we use three different metrics that calculate depth residuals:

1. Root Mean Square Error (RMSE)

$$E_{RMSE}(D, D^*) = \sqrt{\frac{1}{|D|}\sum_{d \in D} |d^* - d|^2} \tag{2.9}$$

2. Mean Square Error (MSE)

$$E_{MSE}(D, D^*) = \frac{1}{|D|}\sum_{d \in D} (d^* - d)^2 \tag{2.10}$$

3. Mean Absolute Error (MAE)

$$E_{MAE}(D,\ D^*) = \frac{\sum_{d \in D} |d^* - d|}{|D|} \tag{2.11}$$

where $D \in R^{304 \times 228}$ is the predicted depth image, $D^* \in R^{304 \times 228}$ is the ground truth, and $d$, $d^*$ are corresponding pixels of $D$, $D^*$. Beside residual measurement, we also evaluate model accuracy using pixel relative similarity with predefined thresholds. The method is known as Delta, which is defined as follows:

$$\delta_t = max\left(\frac{d^*}{d}, \frac{d}{d^*}\right) < t \tag{2.12}$$

where $t$ is a threshold value and $\delta_t \in [0,1]$. The common choices for $t$ are $\{1.02, 1.05, 1.10\}$. Delta with lower value of $t$ is much more sensitive when comparing pixel-similarity between two depth images.

## 2.5 Point Cloud Representation

Once the resulting depth map is obtained from the fusion model, we convert it to a 3-D point cloud for further processing. Point cloud representation opens more options for post-

processing algorithms, especially in filtering and refinement. Similarly, point cloud input from the single-point sensor (third generation system) must be converted into an equivalent depth frame before being sent to the fusion model. Therefore, we need a conversion method that can interchangeably transform depth map to point cloud. We apply the depth-to-cloud conversion mechanism using intrinsic camera parameters obtained from the chessboard calibration procedure. Given a depth image $D_n \in R^{304\times228}$, the algorithm maps each depth value pixel at the $u$-th row and $v$-th column to appropriate point coordinate $(x, y, z)$ using focal lengths $f_x$, $f_y$ , center point offsets $G_x$, $G_y$, and skew values following the corresponding equation:

$$D(u,v)\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{pmatrix}f_x & s & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{pmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix} \tag{2.13}$$

where $D(u, v)$ is the depth value of the pixel $(u, v)$ on the depth image. More specifically, the point coordination can be defined as:

$$\begin{bmatrix}x\\y\\z\end{bmatrix} = \begin{pmatrix}f_x & s & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{pmatrix}^{-1} D(u,v)\begin{bmatrix}u\\v\\1\end{bmatrix} \tag{2.14}$$

where $f_x$, $f_y$, $c_x$, $c_y$, and $s$ can be obtained following the sensor calibration procedure.

## 2.6 Experiments

To evaluate the performance of the visual attention-based model, we trained and validated our model on the NYUv2 dataset [37]. The NYUv2 dataset consists of 464 diverse indoor scenes, where each sample is represented by an RGB image and depth frame. We used 249 scenes for training and preserved 215 for testing. To increase the training and testing size, we followed the augmentation procedure described in Sun et al. [38]. In total, there are 47,585 generated samples for training and 655 samples for testing. The dataset consists of various types

of indoor environments such as living room, basement, kitchen, and bedroom. Each input is a pair of RGB frame $X \in R^{640\times480}$ and complete depth frame $D^* \in R^{640\times480}$. To make the frames suitable for input into the model, both RGB and depth image are first down-sampled and center-cropped into a fixed size of 304 x 228. Each sparse depth map $D_0 \in R^{304\times228}$ is generated from the completed depth map $D^*$ that contains approximately 6,000 pixels. The complete depth map $D^*$ is used as ground truths for the training process. The models are developed in Pytorch, visual attention layers are implemented using OpenCV in Python. The models are trained on 100 epochs using adaptive learning rate initialized with $lr = 0.01$. The models are trained through Beoshock High Performance Computing (HPC) with Graphics Processing Unit (GPU) node (Nvidia Tesla V100) and $\geq 64Gb$ memory resource. The training process took on average four days on the server.

Following the proposed procedure in [28], the weights of the Encoder-Decoder are first initialized from the pretrained model on ImageNet dataset. We found that with the learning rate $lr = 0.01$, the training took average two days to complete and stopped early, while $lr = 10^{-3}$ extended the training process for week, but also expressed to be overfitting. To balance performance between early-stopping and overfitting problem, we developed adaptive learning optimizers with $lr = 10^{-3}$ with a learning rate scheduler (will reduce the learning rate 20% after 3 consecutive epochs without performance improving), which is only triggered when the loss value is sufficiently low. Using previous baselines as benchmark [28], we decided that RMSE $<$ $0.12$ is a good start condition to trigger the learning rate scheduler.

2.7 Results and Discussions

Table 2.1 shows the performance of different depth estimation models on various metrics. Residual metrics (↓) computes difference between predicting depth and ground truth. The

smaller residual value indicates the better the model performs. In other hand, Delta metrics (↑) computes similarity between prediction and ground truth. Larger $\delta_t$ value indicates that it is harder to visually distinguish differences between prediction and ground truth depth map. According to the table, the proposed model obtains smallest residual values ($E_{MSE} = 0.003028$, $E_{RMSE} = 0.055027$, $E_{MAE} = 0.019414$) and highest $\delta_t$ values ($\delta_{1.02} = 0.999205$, $\delta_{1.05} = 0.999938$, $\delta_{1.10} = 0.999992$), illustrates the outperformance over CSPN and Sparse-To-Dense. More specifically, our model achieves $1.6\times$ better residual value than CSPN, and $11.3\times$ better than Sparse-to-Dense in MSE. The proposed model is the only method obtain over 99% accuracy in both three scale $t$ values.

**Table 2.1** Quantitative comparison of three depth completion models on NYUV2 validation set

| | This chapter | CSPN | Sparse2Dense |
|---|---|---|---|
| MSE | 0.003028 | 0.004888 | 0.034214 |
| RMSE | 0.055027 | 0.069914 | 0.184969 |
| MAE | 0.019414 | 0.022062 | 0.087443 |
| $\delta_{1.02}$ | 0.999205 | 0.998425 | 0.987042 |
| $\delta_{1.05}$ | 0.999938 | 0.999830 | 0.997955 |
| $\delta_{1.10}$ | 0.999992 | 0.999973 | 0.999472 |

To know whether depth estimation models suitable for real-time application, we additionally evaluated the run time performances of three models on NYUv2 test set as illustrated in fig 2.5. Both depth completion models can produce dense depth map in less than

0.2s, which is less than average duration for human reaction. More specifically, both our proposed model and CSPN can produce full dense $304 \times 228$ depth map within $5 - 19$ ms. Sparse-to-Dense model can produce outputs even faster, within $0.8 - 4$ ms. However, the quality of depth map constructed from CSPN and our proposed model has significantly higher resolution than Sparse-To-Dense, which indicates the trade-off between resolution and frame rate.

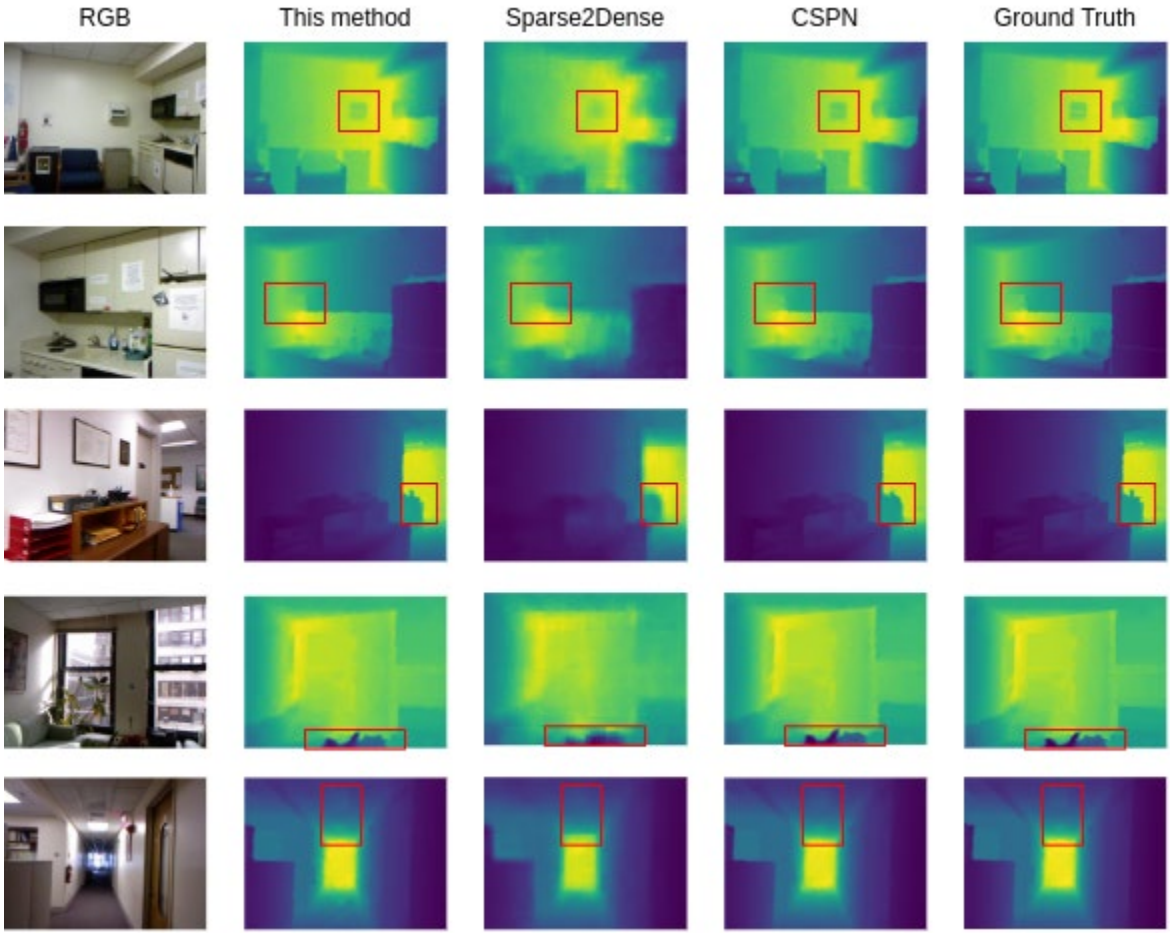


**Figure 2.5** Qualitative result of the proposed active fusion model. Sparse depthmap is collected by composing three different sampling layers: 1) saliency map-ping with density downsampling, 2) edge detecting, 3) grid-sketch distributing. The composing depth map (second column) contains 6, 000 pixels, focusing on "most important" parts of an image.

## 2.8 Conclusions

This chapter proposed a fusion model for depth estimation and sensor integrating, using cost-efficient sensing components to develop a robust and real-time perception system. We have shown that visual attention mechanism helps improving the performance of depth prediction while requires less depth samples. Saliency attention approach can decide most essential and informative pixels from the original image. We proposed different depth sampling strategies, to balance scanning density and real-time performance. The proposed method has shown to be suitable for systems containing single-point depth sensors (see Chapter 1). Through experiments, visual attention mechanism was proved to be an effective technique to improve depth predicting. Despite the achieved system performance, the method is developed on indoor environment so far, where maximum depth distance $\leq 10$ m. The presented work has the potential for development of cost-efficiency 3-D perception system from fusing.

# Chapter 3 Point-Cloud Outliers Removing

The obtained point cloud from an upsampling model inevitably suffers from noise contamination, due to the imperfect nature of depth estimation model performance and the inherent noise of the acquisition device. Therefore, it is necessary to perform filtering operations on output point clouds in order to obtain better point clouds before post processing. In this chapter, we propose the Iterative Statistical Outliers Removal method (ISOR), a variation of the Statistical Outliers Removal (SOR) family, that is faster and can handle an incomplete point cloud.

## 3.1 Background

Point cloud filtering approaches have been well-studied and summarized by Han et al. [35]. The current state-of-the-art point cloud filtering approaches that directly work on point cloud input can be categorized into 7 groups: 1/ Partial Differential Equation based method (PDE), 2/ Neighborhood-based method, 3/ Projection-based method, 4/ Signal processing-based method, 5/ Statistical-based method, 6/ Hybrid filtering technique, and 7/ Other technique.

The statistical-based method has shown to be the fastest and most computational efficient approach; thus, making it suitable for real-time problems. The statistical-based method makes filtering decisions based on analyzed geometric and spatial features, such as color and intensity. An important performance measure of a point cloud filtering approach is the time complexity. Due to the significantly large number of point samples in the input (from hundreds of thousands to millions of points per scene), the computation of the filtering algorithm for a direct point cloud input can be significantly time-consuming. As a result, it is necessary to develop a filtering algorithm that can filter the point cloud effectively for real-time performance.

One of the standard statistical-based point cloud filtering algorithms is the Statistical Outliers Removal (SOR), which has shown to perform well for small and medium scenes. The main limitation of this algorithm is when the captured scene contains more than $400,000$ points. Because of the complexity and specificity of the point cloud data structure, the point cloud processing algorithm can be significantly time intensive. There is no method so far that can handle point cloud filtering process in real time. Secondly, with the current LIDAR systems (both third and fourth generations), it is not possible to capture a dense point cloud in real-time. Therefore, there is a requirement to have an algorithm that can handle partial point clouds.

3.2 Statistical-based filtering – Statistical Outliers Removal (SOR)

SOR-based approaches extract geometric information from local densities. The extracted features are then used to compute statistical estimations and make final filtering decisions as illustrated in fig. 3.1. Using local densities, the SOR-based approach is applicable for any point cloud input, without further required assumptions. The common choice of geometric features for the filtering process are surface normal, curvature differentiation, average distance. Since each estimation is done within a local density, the SOR-based approach is expendable for parallel computing. The main challenge with the basic SOR method is the run-time performance. Although the method runs faster compared to surface interpolation and projection-based approaches [39], the SOR method is only suitable for small- and medium- scale point clouds. The run-time significantly decreases when the number of points increases over 400,000 [40]. The second challenge of the SOR filtering method is the requirement of having a complete point cloud as an input, which makes it suitable for post-processing. When the number of points in the 3-D scene significantly large, the filtering decision is slower due to the waiting time overhead for the complete input, which prevents it from being applicable for real-time problems.

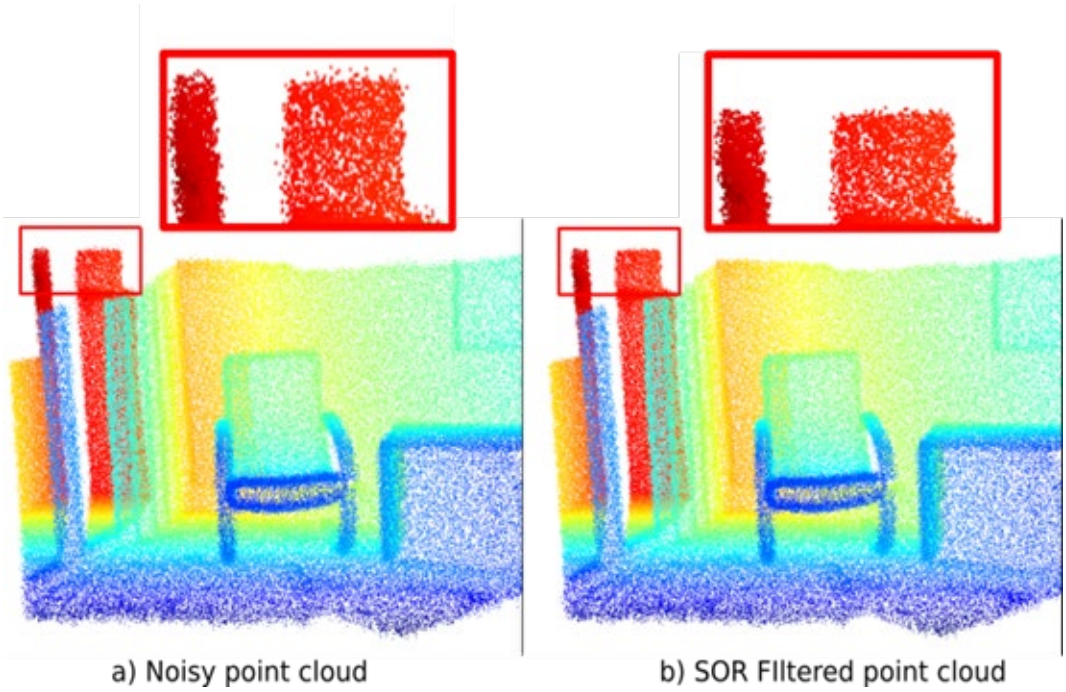


**Figure 3.1** Example of performance of SOR filtering algorithm. Applying SOR filtering on small-scale point cloud (196,133 points), which takes about 13.84 seconds to finish

## 3.3 Iterative Statistical Outliers Removal (ISOR)

The Iterative Statistical Outliers Removal (ISOR) method is an extension of the SOR method and designed to handle an incomplete point cloud and achieve faster performance. By including external memory stacks storing collecting points, ISOR has ability to handle a partial point cloud in the memory, instead of waiting for the complete input. A designed state-machine is used to make filtering decisions, whenever a SOR operator is applied on a partially collected point cloud. The simple decision-making strategy is based on number of points existing in the stacks, the direction movements and velocity of the servos. The SOR operator is a combination of a 4-step function: 1) local densities generating; 2) geometry extracting; 3) statistical estimating; and 4) statistical filtering.

Given the original point cloud input $P = \{p_1, p_2, \dots, p_n\}$, SOR first creates local densities using the Kd-tree method. This Kd-tree will generate $n$ local densities. Each local density $Q_i = \{q_1^i, q_2^i, \dots, q_k^i\}$ with respect to querying point $p_i (i \in n)$, contains $k$ nearest neighbor points surrounding $p_i$. The SOR operator then extracts the average distance of $Q_i$ with respect to $p_i$ as follows:

$$d_{P_i} = \frac{1}{K} \sum_{j \in Q_i} \left|\left| p_i - q^i{}_j \right|\right| \tag{3.1}$$

where $q_j^i \in Q_i$ and $\left|\left| p_i - q_j^i \right|\right|$ is Euclidean distance between $p_i$ and $q_j^i$ . The extracted geometric features $d_{P_i}$ are used to estimate back local distribution following:

$$LD(p_i) = \frac{1}{K} \sum j = 1^k e^{\left( \frac{-\left|\left| p_i - q_j^i \right|\right|}{d_{P_i}} \right)} \tag{3.2}$$

where $LD(p_i) \in [0,1]$ is statistical estimation of $p_i$, and $d_{P_i}$ is the average distance of $Q_i$ with respect to $p_i$. Use of a local distribution $L(p_i)$ indicates the likelihood of point $p_i$ to be an outlier. A point $p_i$ is locally considered to be an outlier if its distance to its neighbors is exceptionally high. More specifically, outliers can be defined as:

$$outliers = \{p_i | LD(p_i) \geq \text{threshold}\} \tag{3.3}$$

## 3.4 Evaluation Metrics

To guarantee the quality of the point cloud output, we evaluated point cloud refining algorithms on the obtained point cloud from the previous experiment. We compared the developed ISOR method with standard filtering SOR on 5 different indoor scenes from the depth completion experiment, measuring run-time and D-mean metric for accuracy performance. The input depth map obtained from depth completion model first is converted to a point cloud.

Additional outliers are added uniformly into the point cloud to challenge the algorithm performances at different scales. The D-mean metric is defined as follows:

$$d(P, P^*) = \frac{1}{2}\mathrm{d}'(\mathrm{P}, \mathrm{P}^*) + \frac{1}{2}\mathrm{d}'(\mathrm{P}^*, \mathrm{P}) \tag{3.4}$$

where $P, P^*$ are origin point cloud and filtered point cloud respectively, $d$ is D-mean point cloud distance, $d'$ is Euclidean point cloud distance which can be further defined as:

$$d'\left(P_{\text{target}}, P_{\text{ref}}\right) = \sum_{i=1}^{N} \left| |p_i - q_i| \right| \tag{3.5}$$

where $p_i$ is a point in target point cloud $P_{\text{target}}$, $q_i$ (locating in $P_{\text{ref}}$ ) is the closest point to $p_i$. The D-mean metric can be used to measure point cloud differences.

## 3.5 Results and Discussion

The qualitative result of both methods is shown in fig. 3.2. The second column represents the point cloud obtained from cloud conversion directly from the previous experiment. As one can see, both ISOR and SOR methods can filter out "global outliers" – outliers that are recognizable to not be a part of the origin point cloud. For less obvious regions, one can see that both ISOR and SOR methods mutually agree on which points are outliers (3rd and 4th columns). While the SOR method performs a more difficult filtering task, ISOR attempts to maintain the origin information. There are white strikes on the ISOR's results, which can be explained by local property of the approach.

We compare run-time and accuracy performance (using D-mean) of the two methods in Table 3.1. The ISOR algorithm runs faster than SOR algorithms in small and medium point cloud scales. With the point cloud size $69{,}312 = 304 \times 228$, ISOR can run within 1 s while SOR requires 14 s. The problem can be an extraneous problem when the size of point cloud increases

to $\geq 100{,}000$, the run-time performance becomes an important issue. D-mean is used to measure the difference between target point cloud and the ground truth. The smaller value of D-mean, the better performance the method can achieve. As shown, the ISOR method also has a smaller D-mean value in all scenarios. This result indicates two items: 1) By observing more complete and global context, SOR can remove more points from the image, which can also remove important information from the image; 2) Since it only makes a decision at the local-scale, ISOR filters less points from the original image, which makes it faster while still retaining more information from the original input.

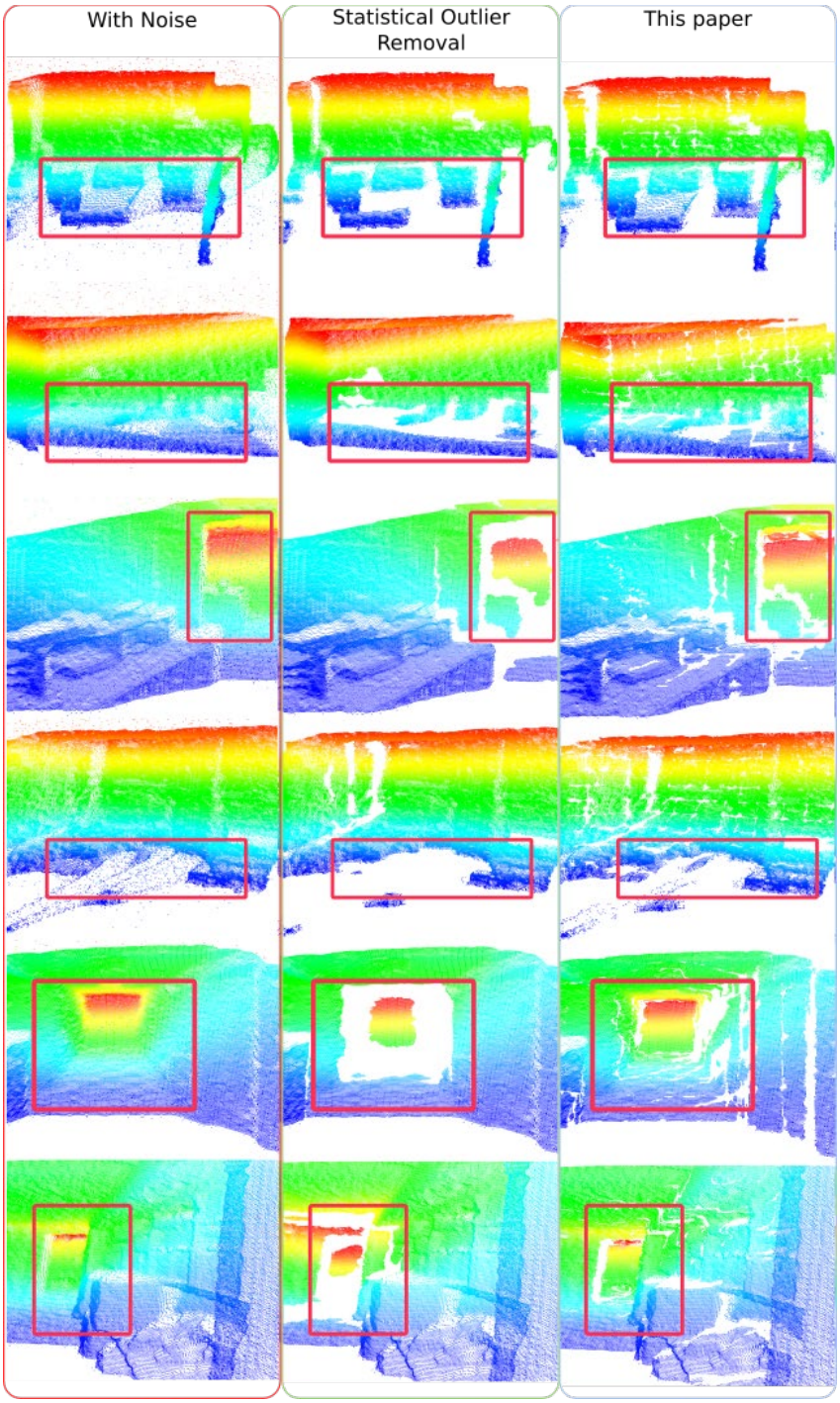


**Figure 3.2** Qualitative results of the point cloud refinement methods

**Table 3.1** Quantitative comparison of two statistical outlier removal methods

| | This Chapter (ISOR) | | SOR | |
|---|---|---|---|---|
| | Runtime (s) ↓ | D-mean ↓ | Runtime (s) ↓ | D-mean ↓ |
| Office 1 | 1.132 | 32.288 | 14.664 | 556.623 |
| Office 2 | 1.081 | 27.598 | 15.277 | 689.443 |
| Office 3 | 1.064 | 3.6011 | 15.386 | 21.1999 |
| Office 4 | 1.008 | 43.447 | 14.108 | 4661.4 |
| Office 5 | 1.072 | 32.288 | 14.204 | 556.623 |
| Table | 7.209 | 8.381 | 95.665 | 139.174 |

## 3.6 Conclusions

The third and fourth generation single point 3-D LIDAR systems are a significant improvement on their predecessors, and through upsampling to increase the depth sensing density, a highly accurate point cloud can be generated. However, the system is still respectively slow to be utilized in a real-time scenario when the system is configured for high accuracy. In addition, upscaling the obtained point cloud from the single point 3-D LIDAR system inevitable results in noise. Thus, a filtering operation is required to obtain better point clouds. Here, the use of a novel Iterative Statistical Outliers Removal (ISOR) method demonstrates high accuracy in removing outliers. Moreover, through the combination of upsampling and employing the ISOR method, a lower accuracy point cloud from a single-point 3-D LIDAR system can be enhanced to generate an accurate 3-D point cloud bordering on real time. This would make the complete system applicable for all transportation-based environments.

References


1. Williams, Keith, Michael Olsen, Gene Roe, and Craig Glennie. 2013. "Synthesis of Transportation Applications of Mobile LIDAR." *Remote Sensing*, 5 (9): 4652.

2. Wiklund, Theodore, Mark Heim, Jaret Halberstadt, Michael Duncan, Deven Mittman, T. DeAgostino, and C. Depcik. 2019. "Design and Development of a Cost-Effective LIDAR System for Transportation." *ASME 2019 International Mechanical Engineering Congress and Exposition*, Salt Lake City, UT.

3. Puente, I., H. Gonzalez-Jorge, J. Martinez-Sanchez, and P. Arias. 2013. "Review of mobile mapping and surveying technologies." *Measurement*, 46 (7): 2127-2145.

4. Kelly, M. and S. Di Tommaso. 2015. "Mapping forests with Lidar provides flexible, accurate data with many uses." *California Agriculture*, 69 (1): 14-20.

5. Garcia-Gutierrez, J., L. Goncalves-Seco, and J. C. Riquelme-Santos. 2011. "Automatic environmental quality assessment for mixed-land zones using lidar and intelligent techniques." *Expert Systems with Applications*, 38 (6): 6805-6813.

6. Yang, B. S., Z. Wei, Q. Q. Li, and J. Li. 2013. "Semiautomated Building Facade Footprint Extraction From Mobile LiDAR Point Clouds." *IEEE Geoscience and Remote Sensing Letters*, 10 (4): 766-770.

7. Chiang, K. W., G. J. Tsai, Y. H. Li, and N. El-Sheimy. 2017. "Development of LiDAR-Based UAV System for Environment Reconstruction." *IEEE Geoscience and Remote Sensing Letters*, 14 (10): 1790-1794.

8. Kromer, R. A., D. J. Hutchinson, M. J. Lato, D. Gauthier, and T. Edwards. 2015. "Identifying rock slope failure precursors using LiDAR for transportation corridor hazard management." *Engineering Geology*, 195 93-103.

9. Neupane, S. R. and N. G. Gharaibeh. 2019. "A heuristics-based method for obtaining road surface type information from mobile lidar for use in network-level infrastructure management." *Measurement*, 131 664-670.

10. Lienert, Paul and Stephen Nellis. 2019. "Cheaper Sensors Could Speed More Self-Driving Cars to Market by 2022." Available from: https://www.reuters.com/article/us-autos-autonomous-lidar/cheaper-sensors-could-speed-more-self-driving-cars-to-market-by-2022-idUSKCN1TD2MY.

11. McIntosh, Daniel. 2019. "Utilization of Lidar Technology - When to Use It and Why." Kentucky Transportation Cabinet: University of Kentucky, Lexington. Available from: https://rip.trb.org/Results?txtKeywords=lidar#/View/1638641.

12. Tarko, Andrew P., Kartik B. Ariyur, Mario A. Romero, Vamsi Krishna Bandaru, and Cheng Liu. 2014. "Stationary LiDAR for Traffic and Safety Applications – Vehicles

Interpretation and Tracking." Purdue University. Available from: https://pdfs.semanticscholar.org/e4a2/6ba366fd06c19830b68371abf3012fa30ba6.pdf.

13. Bennett, Jarod, Mather Saladin, Daniel Sizoo, Spencer Stewart, Graham Wood, Thomas DeAgostino, and Christopher Depcik. 2021. "Design of an Efficient, Low-Cost, Stationary LiDAR System for Roadway Condition Monitoring." *ASME 2021 International Mechanical Engineering Congress and Exposition*.

14. Tran, Dang M., Nathan Ahlgren, Christopher Depcik, and Hongsheng He. 2023. "Adaptive Active Fusion of Camera and Single-Point LiDAR for Depth Estimation." *IEEE Transactions on Instrumentation and Measurement*, 72 1-9.

15. Park, Kihong, Seungryong Kim, and Kwanghoon Sohn. 2018. "High-precision depth estimation with the 3d lidar and stereo fusion." *2018 IEEE International Conference on Robotics and Automation (ICRA)*.

16. Yu, Lequan, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. 2018. "Pu-net: Point cloud upsampling network." *Proceedings of the IEEE conference on computer vision and pattern recognition*.

17. Li, Ruihui, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. 2019. "Pu-gan: a point cloud upsampling adversarial network." *Proceedings of the IEEE/CVF international conference on computer vision*.

18. Fan, Haoqiang, Hao Su, and Leonidas J Guibas. 2017. "A point set generation network for 3d object reconstruction from a single image." *Proceedings of the IEEE conference on computer vision and pattern recognition*.

19. Ma, Jiayi, Linfeng Tang, Meilong Xu, Hao Zhang, and Guobao Xiao. 2021. "STDFusionNet: An infrared and visible image fusion network based on salient target detection." *IEEE Transactions on Instrumentation and Measurement*, 70 1-13.

20. Li, Yue, Devesh K Jha, Asok Ray, and Thomas A Wettergren. 2015. "Feature level sensor fusion for target detection in dynamic environments." *American Control Conference (ACC)*.

21. Sandström, Erik, Martin R Oswald, Suryansh Kumar, Silvan Weder, Fisher Yu, Cristian Sminchisescu, and Luc Van Gool. 2022. "Learning online multi-sensor depth fusion. in European Conference on Computer Vision." Springer.

22. He, Yu, Kechen Song, Qinggang Meng, and Yunhui Yan. 2019. "An end-to-end steel surface defect detection approach via fusing multiple hierarchical features." *IEEE transactions on instrumentation and measurement*, 69 (4): 1493-1504.

23. Cheng, Xun and Jianbo Yu. 2020. "RetinaNet with difference channel attention and adaptively spatial feature fusion for steel surface defect detection." *IEEE Transactions on Instrumentation and Measurement*, 70 1-11.

24. Yeung, Ching-Chi and Kin-Man Lam. 2022. "Efficient fused-attention model for steel surface defect detection." *IEEE Transactions on Instrumentation and Measurement*, 71 1-11.

25. Blankenau, Isaac, Daniel Zolotor, Matthew Choate, Alec Jorns, Quailan Homann, and Christopher Depcik. 2018. "Development of a Low-Cost LIDAR System for Bicycles." SAE International.

26. Ma, Fangchang and Sertac Karaman. 2018. "Sparse-to-dense: Depth prediction from sparse depth samples and a single image." *2018 IEEE international conference on robotics and automation (ICRA)*.

27. Lee, Byeong-Uk, Hae-Gon Jeon, Sunghoon Im, and In So Kweon. 2019. "Depth completion with deep geometry and context guidance." *2019 International Conference on Robotics and Automation (ICRA)*.

28. Cheng, Xinjing, Peng Wang, and Ruigang Yang. 2018. "Depth estimation via affinity learned with convolutional spatial propagation network." *Proceedings of the European conference on computer vision (ECCV)*.

29. Park, Jinsun, Kyungdon Joo, Zhe Hu, Chi-Kuei Liu, and In So Kweon. 2020. "Non-local spatial propagation network for depth completion." *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, Proceedings, Part XIII 16*. Springer.

30. Liu, Sifei, Shalini De Mello, Jinwei Gu, Guangyu Zhong, Ming-Hsuan Yang, and Jan Kautz. 2017. "Learning affinity via spatial propagation networks." *Advances in Neural Information Processing Systems*, 30.

31. He, Hongsheng, Yan Li, and Jindong Tan. 2016. "Rotational Coordinate Transformation for Visual-Inertial Sensor Fusion. in Social Robotics." Cham: Springer International Publishing.

32. Laina, Iro, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. 2016. "Deeper depth prediction with fully convolutional residual networks." *2016 Fourth international conference on 3D vision (3DV)*.

33. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *Proceedings of the IEEE international conference on computer vision*.

34. Hou, Xiaodi and Liqing Zhang. 2007. "Saliency detection: A spectral residual approach." in *2007 IEEE Conference on computer vision and pattern recognition*.

35. Canny, John. 1986. "A computational approach to edge detection." *IEEE Transactions on pattern analysis and machine intelligence*, (6): 679-698.

36. Owen, Art B. 2007. "A robust hybrid of lasso and ridge regression." *Contemporary Mathematics*, 443 (7): 59-72.

37. Silberman, Nathan, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. 2012."Indoor segmentation and support inference from rgbd images." *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part V 12*. Springer.

38. Sun, Zhanghao, David B Lindell, Olav Solgaard, and Gordon Wetzstein. 2020. "SPADnet: deep RGB-SPAD sensor fusion assisted by monocular depth estimation." *Optics express*, 28 (10): 14948-14962.

39. Han, Xian-Feng, Jesse S Jin, Ming-Jie Wang, Wei Jiang, Lei Gao, and Liping Xiao. 2017. "A review of algorithms for filtering the 3D point cloud." *Signal Processing: Image Communication*, 57 103-112.

40. Ning, Xiaojuan, Fan Li, Ge Tian, and Yinghui Wang. 2018. "An efficient outlier removal method for scattered point cloud data." *PloS one*, 13 (8): e0201280.