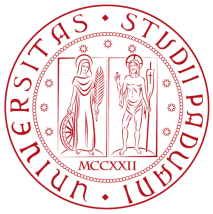


UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria dell'Informazione

Tesi di Laurea

Point Cloud Lossless Compression for LiDAR Sensor Data using 2D-oriented and 3D-oriented Methods

Relatore

prof. Andrea Zanella

Correlatori

dott. Marco Giordani

dott. Paolo Testolina

Laureando

Francesco Nardo

1162799

Anno Accademico 2019/2020

21 Settembre 2020

Ai miei genitori e a mio fratello,
che mi hanno sempre sostenuto
Al mio relatore e correlatori,
per la loro disponibilità e gentilezza
Ai miei più cari amici,
per il loro supporto

Abstract

Modern LiDAR sensors simplify the collection of large 3D point-clouds. This three-dimensional mapping of the surrounding environment can be used both for object recognition or topography.

Storage and transmission of data generated by LiDAR sensors are one of the most challenging aspects of their deployment.

The objective of this thesis was to efficiently compress point-clouds from the LiDAR Velodyne VLP-16 sensor. The data have been collected in different modes: static scenario, dynamic scenario and moving sensor.

Two main different techniques for LiDAR frame compression were compared and implemented: the 2D- and the 3D-oriented one. Their performance was analyzed in terms of compression efficiency and quality of the decompressed frame compared to the original.

Moreover, several archiving and compression methods (based on the representation of the clouds through two-dimensional matrices) have been studied: image-based or video-based formats (both intra- and inter-frame) and finally a solution based on dictionary compression.

Only lossless compression methods have been examined for both the techniques.

This thesis also presents some advanced strategies based on spherical coordinates to improve image-based compression results.

We demonstrated that, thanks to the matrix form in which LiDAR frames are saved, compression methods already known like those for two-dimensional images have given equivalent results, if not better, than those designed for three-dimensional point-clouds.

Sommario

I moderni sensori LiDAR semplificano la raccolta di grandi nuvole di punti 3D. Questa mappatura tridimensionale dell'ambiente circostante é impiegata sia per tecniche di riconoscimento degli oggetti sia nella topografia.

L'archiviazione e la trasmissione della grande mole di dati generata da questi sensori é uno degli aspetti più impegnativi del loro sviluppo.

L'obbiettivo di questa tesi é comprimere efficentemente le nuvole di punti acquisite dal sensore LiDAR Velodyne VLP-16. I dati sono stati raccolti con diverse modalitá: scenario statico, scenario dinamico e sensore in movimento.

Sono state confrontate e implementate due metodologie di compressione di frame LiDAR: una 2D-, l'altra 3D-oriented. Se ne sono analizzate le prestazioni in termini di efficienza di compressione e qualità del frame decompresso rispetto all'originale.

Entrambe le metodologie sono esclusivamente lossless.

Inoltre si sono studiate tecniche (2D-oriented, dove la nuvola é rappresentata attraverso matrici bidimensionali) di archiviazione e compressione: basate su formati standard per immagini o video (sia intra- sia inter-frame) ed infine si é proposta una soluzione basata su compressione a dizionario.

Questa tesi presenta anche delle strategie volte a migliorare i risultati della compressione, utilizzando le coordinate sferiche al posto delle classiche coordinate cartesiane.

Grazie alla forma matriciale in cui vengono salvati nativamente i frame LiDAR, i metodi di compressioni già noti per immagini bidimensionali hanno dato risultati in linea, se non migliori, di quelli pensati per nuvole di punti tridimensionali.

Contents

List of Figures

List of Tables

Glossary

deflate Lossless data compression file format that uses a combination of LZW and Huffman coding. 5, 9, 17, 30, 32, 34–36

dictionary A set of strings contained in a data structure maintained by the encoder. When the encoder finds a match, it substitutes a reference to the string's position in the data structure. ii, 9, 11, 17, 30, 32–34, 36

image-based Compression method that implements image formats. ii, 2, 8, 14, 15, 20, 22, 26, 32–36

inter-frame It is a frame in a video compression stream which is expressed in terms of one or more neighboring frames (through prediction). ii, 16, 17, 28–30, 32, 35

intra-frame Video compression technique that does not involved adjacent frames. 28, 32, 35

LiDAR Light Detection and Ranging *or* Laser Imaging Detection and Ranging. ii, iii, 1, 2, 6, 7, 10, 11, 13–15, 19, 26, 32, 35

lossless It is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data. By contrast, lossy compression permits reconstruction only of an approximation of the original data, though usually with greatly improved compression rates. ii, iii, 3, 5, 8–10, 14–16, 20, 22, 26, 28, 29, 31, 35, 36

octree A tree (data structure) where each internal node has exactly eight children. 6–8, 11, 13, 21, 22, 32, 33, 35

point-cloud A set of individual points in a 3-Dimensional space. ii, 1–3, 5–9, 13, 14, 16, 17, 20, 28, 32, 35, 36

video-based Compression method that implements video formats. ii, 8, 15, 17, 32, 33, 35

voxel Volumetric Picture Element, the equivalent of a 2D-image pixel but on a regular grid in a 3-Dimensional space. 7, 8

Acronyms

5G Fifth Generation (Cellular Networks). 1, 2

AVI Audio Video Interleave. 15

Bpp Bytes per Point. 19, 20, 28, 30, 32, 36

CSV Comma-separated Values (Microsoft © Excel). 11, 13

DCT Discrete Cosine Transform. 35

FOV Field Of View. 15

GPS Global Positioning System. 1

H264 Advanced Video Coding. 16, 28

JPEG-LS Lossless JPEG (Joint Photographic Experts Group). 6, 7, 14, 20, 24

LAS LASer (binary format for archiving point-cloud data). 20

LZW Lempel–Ziv–Welch. vii, 2, 9, 17, 30, 32

MJ2 Motion JPEG 2000. 15, 16, 28, 32, 35

MPEG-4 Moving Picture Experts Group multimedia container part 4 or MP4. 15, 16, 28

MSE Mean Squared Error. 19, 22

NaN Not a Number. 24

PCAP Packet Capture (Velodyne © default file format). 5, 11, 13, 19, 20, 24, 28

PCD Point Cloud Data. 2, 5, 11, 13, 20

PCL Point Cloud Library. 13, 22

PNG Portable Network Graphics. 6, 7, 14, 20, 24, 32, 34–36

PPM Portable Pixmap Format. 14, 24, 32

PSNR Peak Signal to Noise Ratio. 8, 15, 19, 20, 22, 26–28, 31–33

RGB Red Green Blue. 7, 14, 28

SNR Signal to Noise Ratio. 27

TIFF Tagged Image File Format. 6, 7, 14, 24

V2X Vehicle to Everything. 1

VLC Variable Length Code. 6

XYZ Cartesian coordinates. 26, 33

ZIP Compression using Deflate algorithm. 5, 9, 14

Chapter 1

Introduction

LiDAR (Laser Imaging Detection and Ranging) is a remote scanner technique that allows to determine the distance from an object or surface using a laser pulse.

Similar to a radar, which uses radio waves instead of light, the distance is determined by measuring the time elapsed between the emission of the pulse and the reception of the back-scattered signal. It has terrestrial, airborne, and mobile applications.

A LiDAR sensor is equipped with various laser beams on different inclinations. These rays are able to rotate several times per second allowing to make a digital 3D representation of the target.

Therefore, each rotation produces a point-cloud which represents the mapping of the space surrounding the sensor.

A point-cloud consists of a set of individual 3D points. Each point, in addition to having a 3D (x,y,z) position, i.e., spatial attribute, may also contain a number of other attributes such as color, reflectance, surface normal, etc. There are no spatial connections or ordering relations specified among the individual points.

1.1 Applications

Today the combination of GPS (Global Positioning System) and airborne LiDAR systems is widely used to monitor glaciers (this sensor has the ability to reveal the slightest growth or decrease) or to study the tree coverings of a forest, measuring the foliage density.

LiDAR also has many uses in archaeology, geo-morphology or architecture due to its abilities to detect subtle topographic features.

However, in the future, autonomous vehicles may use LiDAR for obstacle detection and avoidance to navigate safely through environments thanks to computer vision and/or object detection techniques.

Combining the use of LiDAR sensors with both radar and color camera lead to a more precise detection of pedestrians, cyclists and other vehicles. In [LKK19] the V2X (vehicle to everything) concept is introduced: each vehicle is able, thanks to a special network, to share with others the collected information.

This hypothetical scenario is based on the speed of 5G (fifth generation standard for cellular networks) technology [DS17]. The key is that each vehicle equipped with this sensor will not process the data collected in a local computer. Rather, it will send them in real-time to a *data*

center from where, once processed, instructions will be sent back (thanks to the low latency of 5G) to the on-board computer for driving the vehicle.

1.1.1 The Importance of Compression

In order to avoid buffering problems and reduce the storage costs for this amount of information, it is important to compress the data generated by the LiDAR sensor as much as possible before sending it through the 5G network [Mäm+19].

A LiDAR can generate tons of thousands to millions of points every second (for example in our tests it generated about $300\,000\,\frac{\text{point}}{\text{s}}$, in a 1200 RPM configuration).

Another challenge is represented by the real-time compression required to send frames to the *data center* and get an immediate response. Therefore, the compression must take as short time as possible and cannot be based on future frames, but only on the present one and, possibly, the past ones.

1.2 State of Art

Currently, there is not a compression standard for point-clouds. However, the most commonly used formats are PCD (Point Cloud Data) [Lib11] and LASzip [Ise13]. The performance of these two solutions will be discussed in Sec. 2.1.

Since point-clouds have a three-dimensional nature, geometric compression methods are the most common in the literature. Some of them, such as octree and triangular meshes are described in Sec. 2.3.

Other methods, described in Subsec. 2.4.1, are based on image compression. They usually give better results than geometric-based methods, as confirmed also by the results (presented in Sec. 4.4) of the trials proposed in this thesis.

1.3 Improvements

Instead of implementing already existing image-based and geometric-based compression methods, in this thesis some improvements are proposed to 2D-oriented compressions. Using spherical coordinates instead of cartesian ones, thus the maintenance of a bi-dimensional matrix data form (the correlation between the coordinate angles and the point position are saved in the 2D projection). This solution achieves better compression results, as we can see in Chapter. 4.

Furthermore, the spherical coordinates can be improved with an angle linear interpolation (Subsec. 3.3.2). Predicting the angles and computing the differences with the original one allows us to save only the prediction error, and so to achieve a better quality, as can be seen in Subsec. 4.4.2.

Finally, an evolution of the image-based method was studied. The best results were indeed achieved through either a video-encoding or a customized image-encoding based on Lempel-Ziv-Welch (LZW) and Huffman compression.

1.4 Thesis Overview

First, the related works and the currently state of art about lossless point-cloud compression will be introduced (Ch. 2).

Secondly, we will describe in detail the tests carried out and specifically the various compression methodologies developed by us (Ch. 3).

Finally, the performance metrics and our results will be shown and compared (Ch. 4).

As said before, some 2D-oriented compression methods give better results than the 3D-oriented ones. With the improvements proposed the results are beyond enhanced (Ch. 5).

Chapter 2

Related Works

In this chapter previous works related to both two-dimensional and three-dimensional compression methods will be introduced.

The following methods represent the starting point on which we developed our algorithm: some of them were directly designed for point-clouds while others such as image or video methods have been adapted to our study case.

2.1 PCD and LASzip

In the introduction (Ch. 1) the currently most used formats for storing and processing point clouds were mentioned. The PCD format [Lib11] and LASzip [Ise13].

Now we describe these formats and express some considerations on their limits.

2.1.1 PCD

First of all, the PCD (Point Cloud Data) is directly supported by Matlab's *Computer Vision ToolBox* and it has three different encoding: '`ascii`', '`binary`' or '`compressed`'.

We tried to compress the data by using the `compressed PCD` format and the results are reported in (Sec. 4.2).

The results were even worse than the standard Velodyne data format (PCAP) because PCD is not a real compression algorithm: it saves all the points coordinates, beam after beam in a dedicated file for each frame[1].

2.1.2 LASzip

LASzip is an evolution of the ZIP, an archive file format that usually uses the Deflate algorithm (lossless) [Deu96], specifically designed to compress point-clouds.

The results of this method are available in [Ise13] and reported in (Sec. 4.2).

They are comparable with those obtained during our tests, instead of classical ZIP where the overall compression is not very efficient.

[1] All the cartesian coordinates from a frame are saved in a table with only three row (x, y, z) with '`ascii`' and Huffman encoding.

2.2 LASComp

A further compression method is LASComp, which is based on predicted coordinates (x, y, z) of a point from the previous one, and the prediction error is saved with a VLC (variable length code) encoding.

The results (shown in [MŽ11]) are intermediate between the Cartesian and the Spherical compression obtained by us in PNG, JPEG-LS and TIFF formats (Sec. 3.3).

As mentioned LASComp main advantage from the previous methods is the coordinates (x, y, z) prediction. Then the error between the prediction and the "true" (original) position of the point is computed and saved.

A similar technique will be described in Subsec. 3.3.2 (spherical coordinates), where a small errors, within a restricted domain of possible values, allow us to use a more efficient saving format.

However, the 3.3.2 strategy offers better results as we have implemented the elevation angle prediction. This prediction is perfect, resulting in the benefit of not having to save any errors, which obviously implies less storage space.

2.3 Geometric Methods

Due to the three-dimensional nature of point-clouds, geometric compression methods are the most common in the literature.

2.3.1 Triangular Meshes

The pioneering work of compressing geometrical data was carried out by Taubin and Rossignac (1998) [TR98], who published a method for compressing triangular meshes. Their algorithm divided the triangle mesh into triangular strips. The vertices were then arranged according to their appearances in the triangular strips and coded with a linear prediction schema. In this way, instead of storing the absolute coordinates, only differences between the predicted and the actual positions of vertices were stored. Needing a topology, this algorithm cannot be directly applied to LiDAR datasets.

2.3.2 Octree

The geometric-based algorithms usually exploit the spatial organization of the points to encode them in a structure like an octree in order to reduce the amount of information.

Octrees are an extension of binary trees, useful for partitioning three-dimensional spaces. Specifically, an octree is a tree in which each internal node has exactly eight children. Each child represents $1/8$ of the parent space (analogously with binary search tree, each level of the space is half splitted along each axis, therefore in $2^3 = 8$ partitions) [Sam88]. Each point collected by the LiDAR is represented by the leaf which contains it, so the encoding precision grows with the growing of the number of levels. [SK06]. However, there is always a quantization error: the only way to avoid it would be to have unlimited levels, which is impossible.

The geometry-based solution presented in this thesis (Sec. 3.2) is based on octree compression.

2.3.3 Other geometrical structures

A structure similar to the octree approach is the Voxel Grid (VG). The VG sub-sampling technique is based on a grid of 3D voxels[2].

This technique has been traditionally used in the area of computer graphics to subdivide the input space and reduce the number of points [KB04]. VG algorithm defines a Voxel Grid in the 3D space and for each voxel a centroid is chosen as the representative of all the points that lie on that voxel (LiDAR compression on voxel in [Kam+12]).

Moreover in [SPS12], considering the voxels as spheres, a fast algorithm was developed.

The use of a structure allows to perform some operations like fast searching of the neighbors to reduce the amount of points visualized or to get a more or less precise representation of the point-cloud. In this case the key point is to find an efficient representation by removing redundancy.

Hence a lossy compression system based on plane extraction which represent the points of each scene plane as a Delaunay triangulation and a set of points/area information is discussed in [Mor+14].

2.4 Image Methods

As mentioned, this thesis is based also on the use of classical image compression techniques and therefore on the conversion from point-clouds to two-dimensional images.

This expedient is already used in scientific papers but is quite rare compared with the geometric one.

In [HN15] the mapping of spherical coordinates to 2D coordinates was deeply analyzed, from cartographic considerations to equirectangular projection maps and then a compression based on panoramas was discussed.

An alternative solution is presented in [GK15] based on height map encoding over a planar (2D) domain as a basis.

Anyway, after the conversion process, these methods propose to compress those images by well-known file formats, like PNG, TIFF or JPEG-LS.

2.4.1 Matrix form

In [Bee19] a different strategy that outperform the previous methods and standards is described. Although LiDAR data is often visualized as point-clouds in a 3D space, it is important to note that the raw measurement is simply a distance value (at specific angles) and it is no longer necessary to choose a wise 3D into 2D mapping.

Since LiDAR frames have a matrix form, as will discuss in Ch. 3.3, it is very easy to apply the existing algorithms for 2D image compression. In fact 2D images are processed as one (grayscale) or three (RGB) matrices, therefore seeing the matrix of tuples (x, y, z) given by the LiDAR like three matrices (one per coordinate) allows us to apply the already existing algorithms for 2D image compression.

Because the matrix form is determinated by the data collecting method[3], there is a strong correlation between a field content and its position, which grants a value continuity and so a

[2] A voxel is a volumetric picture element, the equivalent of a 2D-image pixel but on a regular grid in a 3-Dimensional space.

[3] In fact, as we will see in Sec. 3.1, each row corresponds to a beam and each column to a sequential measure.

proper functioning of the image compression algorithms, as can be proved by the results (visible in Sec. 4.4).

To treat the matrix like an image, it is needed to cast the coordinates from float to unsigned integers (in our trials we opted for 16 bits because 32 bits did not show any improvement). This causes a data quantization (Subsec. 4.4.2) and so a limited but, still very high, PSNR (Peak Signal to Noise Ratio) (not infinite like in the theoretical lossless).

Before the casting, the coordinates must be remapped as in eq. 2.1 in order to reduce the quantization error on the 16 bits encoding.

$$\text{IMG} = \left\lfloor \frac{\text{original} - \min}{\max - \min} \cdot (2^{16} - 1) \right\rceil \tag{2.1}$$

where max e min are respectively the maximum and minimum coordinate value in the frame.

However as can be seen in the results (Subsec. 4.4.2), very high PSNR are obtained (in Matlab) due to an estimator function (eq. 2.2). This function takes the value range and estimates the *scale* and $offset$ factors. These allow to avoid as much as possible a quantization rounding, hence enhancing the PSNR.

The casting from float to uint16 (unsigned integer of 16 bits) is done by means of:

$$\text{IMG} = uint16(round(XYZ * scale) + offset) \tag{2.2}$$

where XYZ is the original tensor with float coordinates.

The image-based solution presented in this thesis (Sec. 3.3) is based on this stratagem.

2.5 Video Methods

As already mentioned, in this thesis we developed a video-based compressor directly derived from that of images.

In 2017, the MPEG[4] group issued a call for proposals on Point Cloud Compressor (PCC), and since then it has been evaluating and improving the performances of the proposed PCC video compressors.

PCC is an open standard for compactly representing 3D point-clouds, which are the 3D equivalent of the very well-known 2D pixels.

In [Sch+19] the main developments and technical aspects of this ongoing standardization effort are described.

Similarly to what was done in this thesis, the research immediately differentiated three scenarios based on the dynamic or static nature of the 3D sensor or of the subject.

Currently the standard is still under development (which means that the performances are not yet known) but two different classes are foreseen.

The first class is totally based on a video of still images and it's well suited for static scenarios. However, as we will establish in Ch. 3.4, the traditional motion estimation based on 2D macro-blocks is not well suited to compensate color patches having forms and locations that vary with high frequencies and it has limitations in exploiting temporal correlations.

The second class is hybrid and is based both on voxels and images. A point-cloud is encoded in an octree structure through levels of detail (LoD)[5]. Then an instantiation from the geometry

[4]Moving Picture Experts Group.

[5]The points are grouped into voxels and the voxels have a size given by the level of detail.

decoder is passed to an image encoder, where the re-coloring module assigns colors to the refined vertices, by taking colors from the original (uncompressed) point-cloud.

2.6 Dictionary Methods

A dictionary coder operates by searching for matches between the text to be compressed and a set of strings contained in a data structure (called 'dictionary') maintained by the encoder.

When the encoder finds a match, it substitutes a reference to the string's position in the data structure.

2.6.1 Lempel–Ziv–Welch

Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 [Wel84] as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978 [ZL78].

The method described by Welch encodes sequences of 8-bit data as fixed-length 12-bit codes. The codes from 0 to 255 represent 1-character sequences consisting of the corresponding 8-bit character, and the codes 256 through 4095 are created in a dictionary for sequences encountered in the data as it is encoded.

At each stage in compression, input bytes are gathered character by character until a sequence is formed with no code yet in the dictionary. The code for the sequence (without the last character) is added to the output, and a new code (for the sequence with the last character) is added to the dictionary.

2.6.2 Deflate

The Huffman code [Huf52] is a particular type of optimal prefix code, meaning that there is no whole code word in the system that is a prefix (initial segment) of any other code word in the system.

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols.

In [Deu96], the LZW compression standard combined with the Huffman coding is presented. This lossless technique is called deflate and is one of the most used for both image ([W3C03]) and ZIP compression.

In fact, it manages to combine the advantages of a dictionary compression such as the elimination of duplicate strings together with the advantages of a Huffman coding such as the reduction of the number of bits required.

In this thesis we implemented this combination of algorithms (Sec. 3.5). The results (Sec. 4.6) are indeed good, as redundancies are removed and entropy is reduced in the data stream to be compressed.

2.7 Further Methods

Other interesting methods that we have not considered because they do not match with the specifications of this research are:

1. [CD19]: A bit mask is applied on the raw data packet in order to set to zero the n least significant bits of each measurement, thus creating repeating zero patterns. A conventional lossless data compression algorithm is then used to compress the raw data. This simple method is designed to be used in embedded applications with low available processing power (compatible with existing processing chains) but it introduces a loss of accuracy.

2. [Tu+19a]: it uses a neural network for prediction, but this does not guarantee that the encoding is lossless.

3. [Tu+19b]: designed for real-time application, it uses a neural network and also inter-frame compression.

4. [Al213]: the distribution of the LiDAR points is approximated in one or more planes and the further away points are discarded, reducing the storage space for the dataset. Compression ratios of 17.8% are reported. Since the methods we tested have better performance, this method was not tested.

Chapter 3

LiDAR Point-Cloud Compression

In this chapter we describe the acquisition of the datasets and the involved compression methods. The LiDAR frames obtained from five different ambient condition (datasets) are compressed using the octree method as 3D-based compression and various strategies (image, video and dictionary coding) as 2D-based compression.

The datasets collection and the used compression methods will be now detailed described.

3.1 Data Acquisition

3.1.1 Datasets

The results presented in this thesis are the outcome of the compression performed on five different datasets created with the frames acquired from a LiDAR Velodyne VLP-16 (16 beam) in dual mode (two points are saved for each beam: the nearest and that with the strongest intensity; if the points coincide only one point is saved).

Three datasets were collected at 600 RPM[1]. At each rotation of the beams one frame is collected, so that 600 RPM correspond to 600 frame/ min or 10 fps[2] and 1200 RPM to 20 fps. For each rotational speed, a dataset was captured both in a static scenario and in a movement scenario (of people, the LiDAR remained stationary) plus a last dataset at 600 RPM with a shaking sensor and a static scenario.

In every dataset, the LiDAR was located on a table in an internal environment (a small conference room in the Department of Information Engineer of the University of Padova, which can be seen in Fig. 3.1) with multiple objects (chairs, table, etc.). In the dynamic datasets there are also moving people.

Each dataset was saved in PCAP format with the *VeloView* application, then opened in Matlab thanks to the *velodyneFileReader* module and converted in CSV or Binary PCD[3] to be used in Python, C++ and Matlab.

Each CSV or PCD file represents a frame that is organized in a 16-line matrix (corresponding to the 16 beams) whose inputs are tuples (x, y, z, i) containing the coordinates and the intensity of the point detected by that beam and whose columns represent the temporal instants of sample collection.

[1] Rotations per Minute.
[2] Frame per Second.
[3] One CSV or PCD per frame.

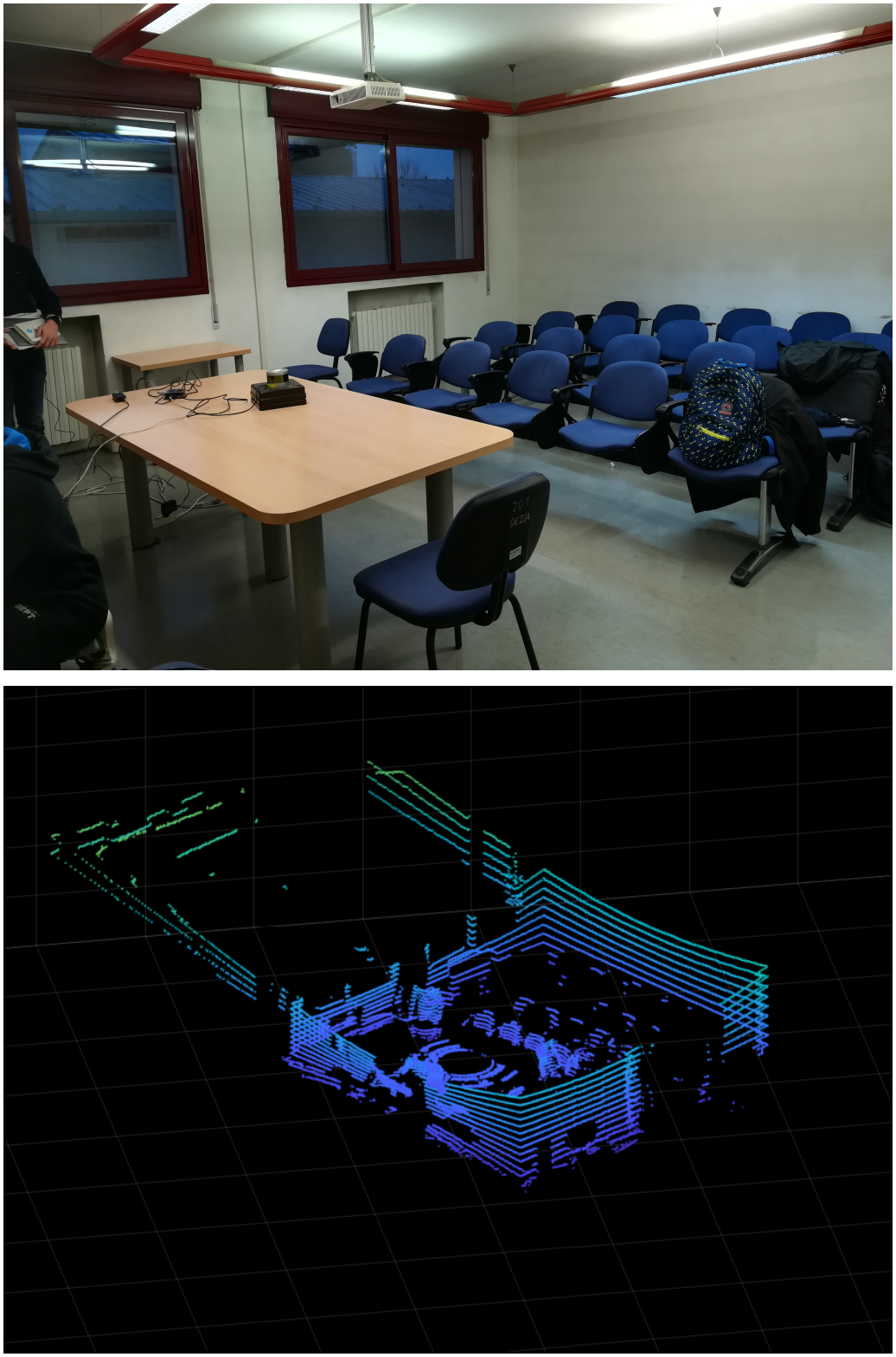


Figure 3.1: Real and LiDAR scanned environment during data collection.

In the CSV files, the intensity was not taken into consideration as the objective was to compress only the information relative to the position of the points while in the PCD ones the intensity was saved and subsequently processed to obtain a feedback from the compression algorithms and to be able to view the point-clouds, as shown in the results.

3.1.2 Points Extraction

The Matlab module imports the PCAP files produced by the LiDAR, and organizes the point-cloud in a tensor whose format is not documented. To the best of our ability we have reconstructed the possible scheme used to store the data into tensor and an extractor has been implemented from it.

We have experimentally verified that this scheme allows us to distinguish the dual points [4] from the normal ones and to obtain a final tensor with only valid points.

However in Matlab this "filtering" has not occurred, therefore all the valid and invalid points have been compressed without distinction.

3.2 Geometry-based

In the trials, the PCL (Point Cloud Library) library[5] [RC11] was used to do a geometry-based compression using octrees (as described in Subsec. 2.3.2).

This library offers 12 different resolution profiles (see Tab. 3.1). Each profile sets the most suitable number of levels for a fixed resolution. All the profiles were tested.

1. `LOW_RES_ONLINE_COMPRESSION_WITHOUT_COLOR` $1\,cm^3$ resolution, without color, online fast encode
2. `LOW_RES_ONLINE_COMPRESSION_WITH_COLOR` $1\,cm^3$ resolution, color, online fast encode
3. `MED_RES_ONLINE_COMPRESSION_WITHOUT_COLOR` $5\,mm^3$ resolution, without color, online fast encode
4. `MED_RES_ONLINE_COMPRESSION_WITH_COLOR` $5\,mm^3$ resolution, color, online fast encode
5. `HIGH_RES_ONLINE_COMPRESSION_WITHOUT_COLOR` $1\,mm^3$ resolution, without color, online fast encode
6. `HIGH_RES_ONLINE_COMPRESSION_WITH_COLOR` $1\,mm^3$ resolution, color, online fast encode
7. `LOW_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR` $1\,cm^3$ resolution, without color, offline efficient encode
8. `LOW_RES_OFFLINE_COMPRESSION_WITH_COLOR` $1\,cm^3$ resolution, color, offline efficient encode
9. `MED_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR` $5\,mm^3$ resolution, without color, offline efficient encode
10. `MED_RES_OFFLINE_COMPRESSION_WITH_COLOR` $5\,mm^3$ resolution, color, offline efficient encode
11. `HIGH_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR` $1\,mm^3$ resolution, without color, offline efficient encode
12. `HIGH_RES_OFFLINE_COMPRESSION_WITH_COLOR` $1\,mm^3$ resolution, color, offline efficient encode

Table 3.1: List of octree compression profiles.

[4] The dual points are those for which a single incident ray sent by LiDAR returns two distinct echoes.
[5] A C++ library for point-clouds management.

3.3 Image-based

The image-based compressor (in the matrix form presented in Subsec. 2.4.1) has been implemented both in Python and in Matlab, using traditional lossless image compression algorithms.

In Python the involved codecs were PNG, TIFF, JPEG-LS, PPM and zipped PPM[6], in Matlab only PNG was used. These formats were selected in order to compare the trial results (Ch. 4) with those in [Bee19].

Different strategies had been used to pack LiDAR points into images. Each of them is presented below.

3.3.1 Cartesian compressions

In the cartesian compression the point-cloud coordinates are simply packed into one or three images. From the matrix of tuples (x, y, z) three matrices are extracted: one per coordinate (X, Y and Z). Then each matrix is remapped and processed as explained in Subsec. 2.4.1.

Cartesian single-channel compression

The points have been saved in three single-channel images, assigning each coordinate (X, Y and Z) to the unique channel of the three images.

Cartesian tri-channel compression

The points have been saved in a tri-channel (RGB) image, assigning the X coordinate to channel R, Y to G and Z to B.

3.3.2 Spherical compressions

As said in Subsec. 1.3, some improvements in image-based compression have been made. They are now presented in this subsection.

To achieve the results reported in Sec. 4.4, the points were converted into spherical coordinates: the *radius* (ρ) and the two angles (*elevation* θ and *azimuth* ϕ), using the following formulas:

$$\begin{aligned} \rho &= \sqrt{x^2 + y^2 + z^2} \\ \theta &= \arctan \frac{\sqrt{x^2 + y^2}}{z} \\ \phi &= \arctan \frac{y}{x} \end{aligned} \tag{3.1}$$

Real angles

The previously defined *radius* and *azimuth* were saved in different grayscale images. During decompression, the *elevation* angle is linearly interpolated from the row index. This trial was implemented only in Python.

[6] An PPM file (non compressed standard) saved with a ZIP compression.

Angle prediction

This methods only saves the radius in a grayscale image.

In fact during the decompression, the *elevation* and *azimuth* angles are linearly interpolated from the row and column indexes. To be more precise, given the LiDAR FOV (Field Of View) (the angle between two outer beams) and the image shape $N \times M$, it is possible to estimate *azimuth* and *elevation* angles (in radians) of the point in the i-th row and j-th column of the image:

$$\hat{\theta} = \frac{\text{FOV}}{2} - i\frac{\text{FOV}}{N} \tag{3.2}$$

$$\hat{\phi} = \frac{\pi}{2} - j\frac{2\pi}{M} \tag{3.3}$$

The Eq. 3.2 and 3.3 are linear interpolations. The extreme values are the same of that given by Matlab library; other environments (or other LiDARs) could organize data in a different way, however this does not affect the goodness of the linear approximation just described.

Angles differences

Furthermore, an advantage from the prediction of the *elevation* and *azimuth* channels could be achieve.

Unlike the previous trial, the absolute angle was not saved, instead its difference with the value obtained by a linear interpolation, it was. This allows us to save some space and to increase the PSNR (about 5 dB) due to a value cast in a much restricted domain.

To minimize the dispersion and achieve values close to zero, the angles differences were remapped in $(-\pi, \pi]$.

3.4 Video-based

The video-based compressor was implemented in Matlab and represents an evolution of the previous image-based compressor (presented in Ch. 3.3).

As already mentioned in Subsec. 2.4.1, Each frame from the LiDAR source is represented in a matrix of tuples (x, y, z) from which we can extract two-dimensional matrices, i.e.[7] images.

Now instead of directly saving single-color images or a three-color image for each single frame, all the images relating to a dataset can be sequenced and compressed as if they were a single colored video or three grayscale videos.

However instead of casting the coordinates from float to 16 bits unsigned integers we have to use only 8 bits per integer due to the specific limit of the chosen video format. So eq. 2.1 holds with the 2 powered by 8 instead of 16. The results of these extreme quantization is shown in the next chapter.

Matlab's *VideoWriter* library was used to write the video object. This class uses different lossless video compression method and codec as AVI, MJ2 or MPEG-4.

[7] Id est.

3.4.1 Inter-frame

An inter-frame is a frame in a video compression stream which is expressed in terms of one or more neighboring frames (through prediction). This kind of prediction tries to take advantage of the temporal redundancy between neighboring frames to enable higher compression rates.

The MPEG-4 (also known as MP4) is a digital multimedia container format most commonly used to store video and audio. In this thesis we chose to use the recent H264 codec (compatible with the MPEG-4 container) to compress still images.

This video compression standard requires particular specifications such as images with 8-bit color depth and precise dimensions (height and width) for the frames.

Therefore, in order to comply with these conditions it was necessary to standardize all the frames within a dataset and carry out a re-shape operation. So a procedure (using null points) capable of reconstructing the original matrix structure during the lossless decompression was developed.

Despite all the efforts, in the results, it will be clear the extreme importance, both in term of quantization error and inter-frame prediction, of a lossless video compression.

3.4.2 Intra-frame

Motion JPEG 2000 (MJ2) was always intended to coexist with MPEG. Unlike MPEG, MJ2 does not implement inter-frame coding; each frame is coded independently using JPEG 2000 (similar with what happened in Sec. 3.3).

This makes MJ2 more resilient to propagation of errors over time, more scalable, and better suited to networked and point-to-point environments, with additional advantages over MPEG with respect to random frame access, but at the expense of increased storage and bandwidth requirements, as we will see in Chapter 4.

3.4.3 Cartesian compression

In the cartesian compression the point-cloud coordinates are simply packed into one or three sequences of frames. From the matrix of tuples (x, y, z) three matrices are extracted: one per coordinate (X, Y and Z). Then each matrix is remapped and processed as previously explained.

Then, in the same way as described in Subsec. 3.3.1, two different approaches were developed: either a video with 24 bits color depth (8 bits per channel) or three grayscale (8 bits color depth) videos are saved.

3.4.4 Spherical compression

In Subsec. 3.3.2 we presented different ways to represent or interpolate angles. However after an image-based compressor result analysis we ended considering only the method which uses angles differences from the prediction.

As mentioned, the error between the prediction and the original position of the point is really small, so a restricted domain of possible values, allow us to cast and save the difference in a more efficient way.

Therefore, depending on the strategy (grayscale o colored video), frame streams containing *radius* and *azimuth* angle prediction error have been saved.

3.5 Dictionary-based

In this thesis the Deflate algorithm introduced in Sec. 2.6 was implemented in Matlab using the algorithm from [Bar20] and the *Huffman Coding* library.

3.5.1 Video-based method analogies

Within this method, the compression is similar to the video-based one (Sec. 3.4), where the coordinates need to be casted from float to 8 bits unsigned integers, in order to fulfill the [Wel84] specifications.

Another similarity with the video encoder it's represented by the inter-frame compression. In fact from each datasets we were able to extract a sequence of frames and from them a sequences of matrices representing the bi-dimensional matrix X, Y or Z which are the absolute position (x, y, z) in a 3D space.

As in the previous method (video), this one uses the matrices described in Subsec. 2.4.1, and the expedients described in Ch. 3.3. So we tested:

- **Cartesian Compression**: the point-cloud coordinates (matrices X, Y and Z) are simply packed into three different sequences of frames.
- **Spherical Compression**: frame streams containing *radius* and *azimuth* angle prediction error have been saved.

However, instead of being able to create tri-channel video objects, here the compression can only be done one channel at a time.

3.5.2 Lempel-Ziv-Welch coding

A given sequences of matrices (within all the dataset) was linearized (row-major[8] order) and became the *text* described by Welch [Wel84]. In this *text* the characters are the unsigned numbers represented exactly by 8 bits. These means that each matrix sequences of properties in a specific dataset needs only one dictionary for all the frames. This common dictionary represents the inter-frame aspect of this compression.

3.5.3 Huffman coding

To the dictionary defined above is then applied the Huffman coding to remove any remaining redundancy. Cleverly the symbols of Huffman's coding ([Huf52]) are none other than the words contained in the LZW dictionary. This allows us to exploit and reduce the entropy.

Finally from this coding we were able to save a binary file for each property of a dataset. Note that a property can be the succession of all X matrices (x coordinate of the points within a point cloud) as well as the succession of a reflection characteristic matrix due to the versatility and power of this lossless method.

[8] It's a method for storing multidimensional arrays in a linear ones. In row-major order, the consecutive elements of a row reside next to each other.

Chapter 4

Performance Results

In this chapter the results from the previous compression methods (Ch. 3) will be presented.

First, in order to compare the trials, we will introduce some performance metrics as compression efficiency and quality of compression.

Then this chapter will be ended by a comparison between all the methods.

4.1 Performance Metrics

The following metrics have been used to measure the performance of our algorithms:

1. **Savings rate**: let *compr* be the size (in bytes) of the compressed dataset and *raw* be the size of the raw dataset, which is the PCAP file from the LiDAR. The *savings rate* is given by the following formula:

$$savings\ rate = 1 - \frac{compr}{raw} \tag{4.1}$$

2. **Bpp (Bytes per Point)**: let *compr* be the size (in bytes) of the compressed dataset and p be the total number of points contained in it. The Bpp is defined by the following formula:

$$Bpp = \frac{compr}{p} \tag{4.2}$$

3. **PSNR**: the *MSE* is the mean squared error between two matrices, I and K of equal size, defined as follows:

$$MSE\ (Mean\ Squared\ Error)(I,K) = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

The PSNR, used to quantitatively evaluate the introduced error, is defined by the following formula:

$$PSNR(I,K) = 20 \cdot \log_{10}\left(\frac{MAX_{I,K}}{\sqrt{MSE(I,K)}}\right) \tag{4.3}$$

where $MAX_{I,K} =$ the maximum value in matrices I and K

For each method we will see the following graphs:

1. **Compression efficiency**
   The plots of the *Saving rates* and *Bpp*: for each format/profile it is shown the mean on all the datasets of these metrics.

2. **Compression quality**
   For each strategy/profile it is shown the mean of the PSNR on all the frames from all the datasets.

4.2 PCD and LASzip Compression

PCD

We tried to compress the data by using the `compressed PCD` format and the results are reported in (Tab. 4.1):

Savings Rate [%]	Bytes per Point [byte]	PSNR [dB]
−104.8	7.10	∞

Table 4.1: PCD rates, Bpp and PSNR using the `compressed` format

PCD is a format that has the advantage of a quick saving time compared to other standards. It is lossless, as evidenced by the PSNR, but it has a negative Savings Rate (defined in Sec. 4.1). This implies that it needs more than double the storage space compared to the original file saved in PCAP. Obviously this is not a good result for a compression algorithm.

LASzip

The results of this method are available in [Ise13] and reported in (Tab. 4.2).

The savings rate is referred to the original uncompressed file of [Ise13] which is LAS (LASer): a binary format for archiving point-cloud data. In term of comparison the bit-rate (Bpp) seems to be quite more interesting since as mentioned is comparable with the image-based cartesian methods.

Savings Rate [%]	Bytes per Point [byte]	PSNR [dB]
85	2.15	∞

Table 4.2: LASzip compression rates, Bpp and PSNR

However, note that from a comparison between LASZip and image-based compression (Sec. 4.4), the 2D-oriented methods based on PNG or JPEG-LS will result more effective.

4.3 Geometry-based Compression

4.3.1 Compression efficiency

As shown in Fig. 4.1 the octree savings rate decreases with the increasing of the resolution[1].

In the *color* (even numbered) and *without color* (odd numbered) profiles the rate is the same, since there is not any stored information about the point color. Between the *online* (the first six profiles, which favor encoding speed) and the *offline* (the last six, which favor encoding efficiency) modes there is only little difference.

The profiles with the best resolution (5, 6, 11 and 12) do not give good results as they save little more than half space.

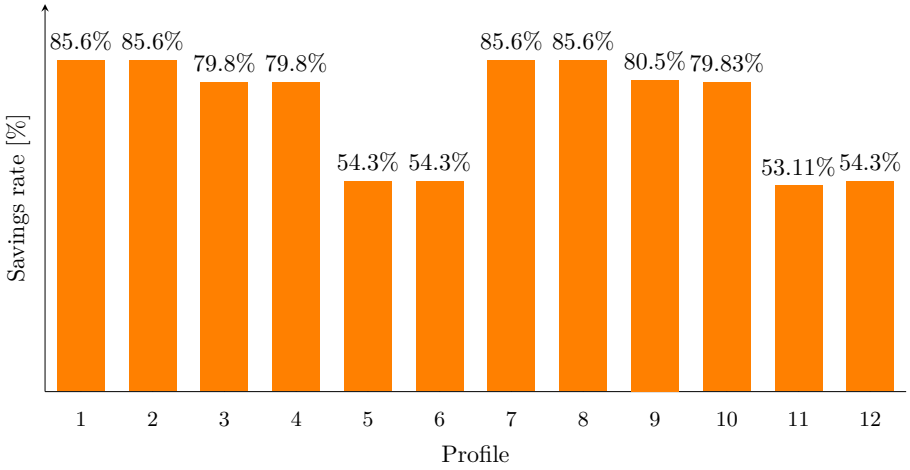


Figure 4.1: Octree savings rates.

According to the *saving rates*, the bytes required to store a point (Fig. 4.2) increase with the resolution increasing. They are however limited from 1 to 4 bytes per point.

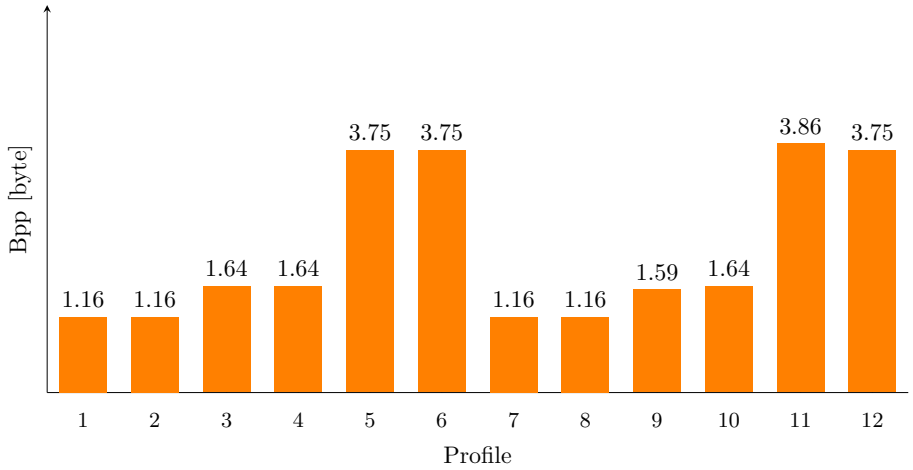


Figure 4.2: Octree Bytes per Point.

[1]As can be seen in Tab. 3.1, profiles 5, 6, 11 and 12 have the higher resolution, while 1, 2, 7 and 8 the lowest.

4.3.2 Compression quality

In order to compare the geometry-based compressor with the image-based one, it was chosen to use PSNR as quality metric. However, with octree, it is not feasible to use the usual PSNR definition (eq. 4.3), due to the possible merging of two, or more, points. As a matter of fact, if some points are very close to each other (the distance between them is in the same order of the quantization one), they are merged in only one point of the octree (and consequently in the final cloud, obtained after the decompression); this makes not clear how to calculate the MSE (which is required to evaluate the PSNR).

The solution is to pair each point in the original cloud with the corresponding one in the final cloud, even if more than one point of the original is matched with the same point of the final cloud.

Unfortunately, the used library (PCL), does not offer any tool to allow this pairing.

For the implementation, it was necessary to resort some approximate methods for matching the points between the two clouds. As mentioned above, these methods do not provide the real PSNR, but they can yield a good estimation of it. The trials have been made with the following two methods:

1. **Clouds with unique point:** for each point of the original cloud a new cloud with only that point is generated. The new cloud is then compressed in an octree and then decompressed; eventually the MSE is evaluated between the point in question and the one in the final cloud.

2. **Minimum Euclidean distance:** Each point from the original cloud is paired with the point from the final cloud at the minimum Euclidean distance.

The achieved results are very similar. In Fig. 4.3 are reported the PSNRs obtained with the second method. The *color* and *without color* profiles gave the same results, so we grouped them into one column.

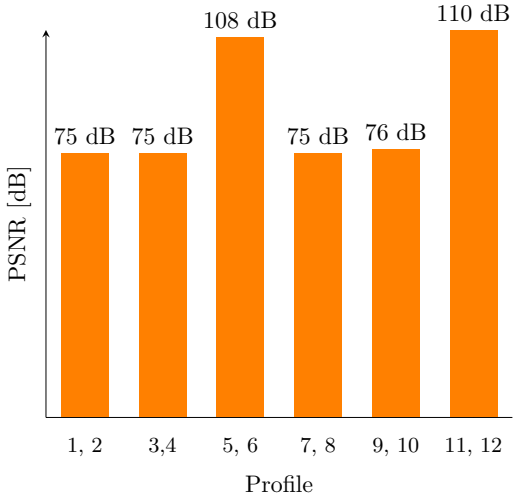


Figure 4.3: Octree compression PSNRs.

As we can see the best results are obtained with profiles 5, 6, 11, 12, which gave also the worst result in savings rate. Anyhow, all profiles are over 75 dB and can thus be considered lossless.

4.4 Image-based Compression

4.4.1 Compression efficiency

Firstly we can observe that saving three grayscale images gives slightly better results than saving one tri-channel image, both for cartesian (Fig. 4.4, 4.5) and spherical (Fig. 4.7, 4.8) methods.

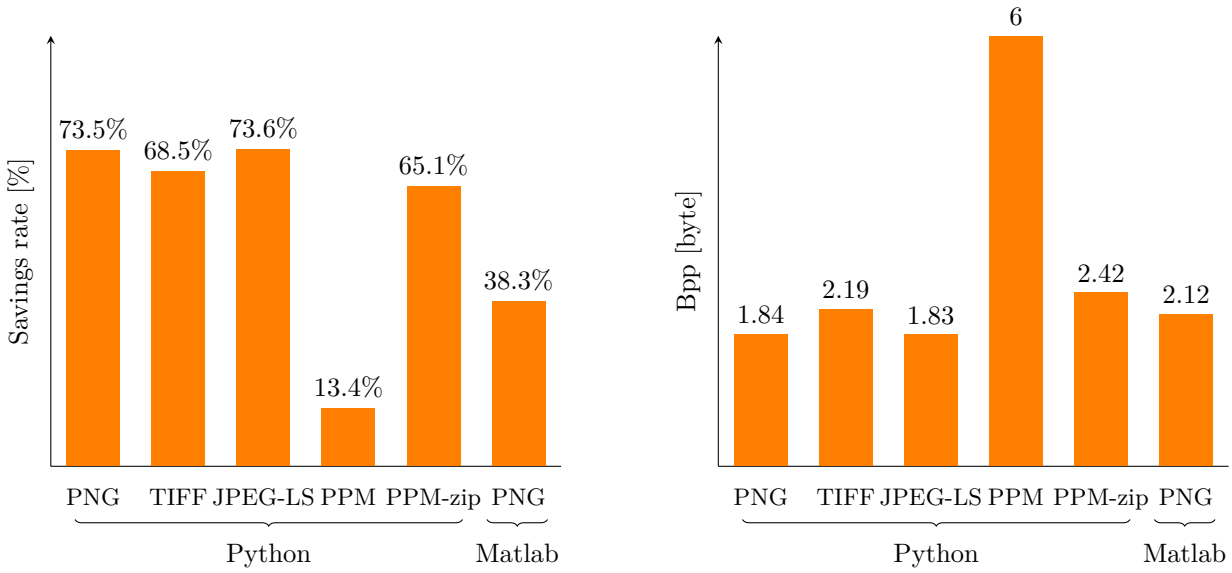


Figure 4.4: Frame packed in three single-channel images, the first for the X coordinate, the second for Y and the last for Z.

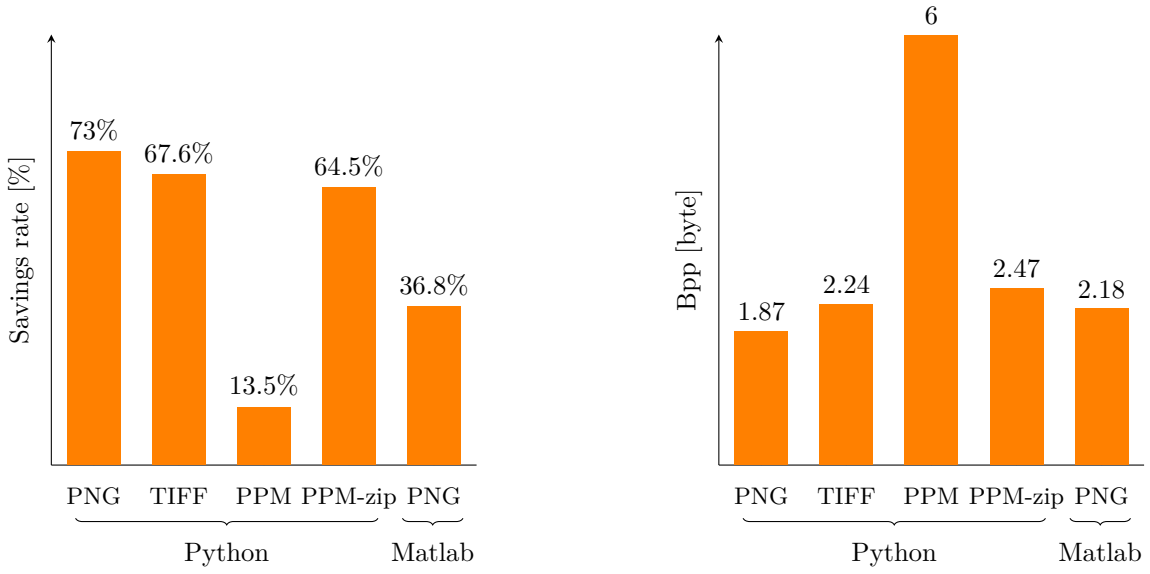


Figure 4.5: Frame packed in a tri-channel image, where the coordinates XYZ have been saved in the three color channels.

With savings rates over 70 % for cartesian methods (Fig. 4.5) and over 85 % for spherical ones (Fig. 4.8) PNG and TIFF give an excellent compression level, and nowadays they are both, two of the most common codecs. On the other hand, the JPEG-LS is not much used, but obtains results comparable with those of PNG and TIFF. Finally PPM is not a compression codec[2], so it gives much worse results compared to the other investigated methods; however its zipped version obtains results only slightly lower than the other formats.

Looking at all the previous results (Fig. 4.4, 4.5), we can exploit an interesting comparison in terms of savings rate between algorithms implemented in Python and Matlab. In Subsec. 3.1.2 we explained that rough data are saved in PCAP files, which contains tensors with valid (numerically) or invalid (NaN) points. In Matlab no distinction has been made about point validity and so all the points have been compressed. This preserves the original data structure, but, as can be seen in the results, it penalizes the savings rate. This happened because the compressed frames, storing more points, have a greater dimension (in bits) than the Python counterpart and so a lower percentage, in relation to the original PCAP. Instead the bytes per point results (Fig. 4.4, 4.5) are only slightly different, because the greater file size is balanced by the higher number of compressed points.

Saving the points using absolute spherical coordinates, instead of cartesian ones, gives better results in terms of compression (Fig. 4.6). The higher savings rate is due to the elevation angle, which is constant for each beam (every row in the matrix has the same elevation angle).

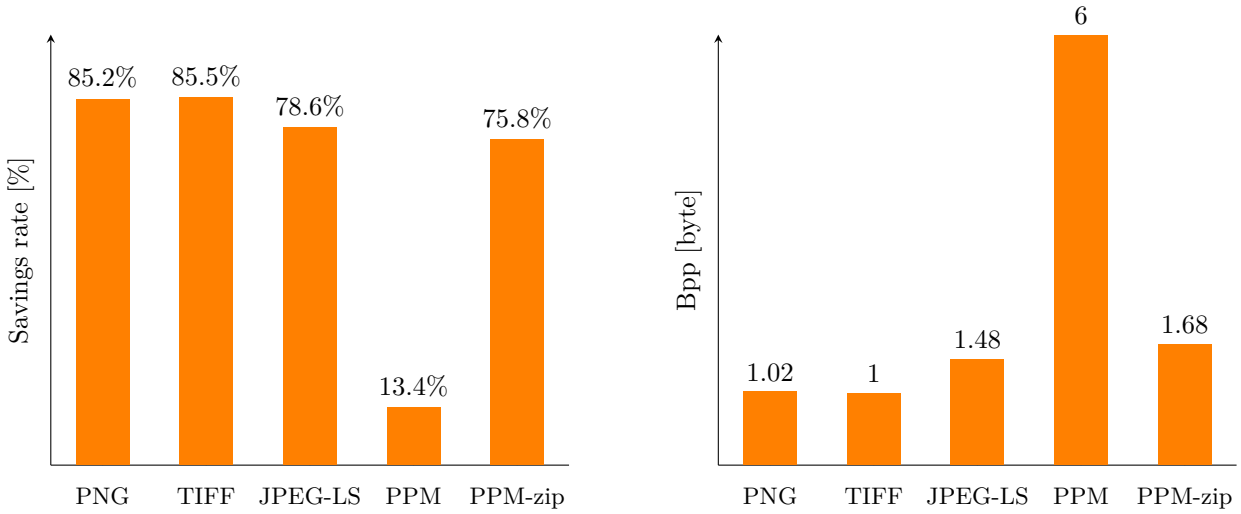


Figure 4.6: Frame packed into three single-channel images, in the first has been saved the radius, in the second the absolute elevation and in the third the absolute azimuth.

Furthermore saving the difference between angles and their prediction allows to further raise the savings rate, in particular for JPEG-LS and zipped PPM, because the difference is often null and these codecs work well with zeros numbers (Fig. 4.7 and 4.8).

[2]PPM uses always 6 Byte to store a point: 3 coordinates as 16 bit integers.

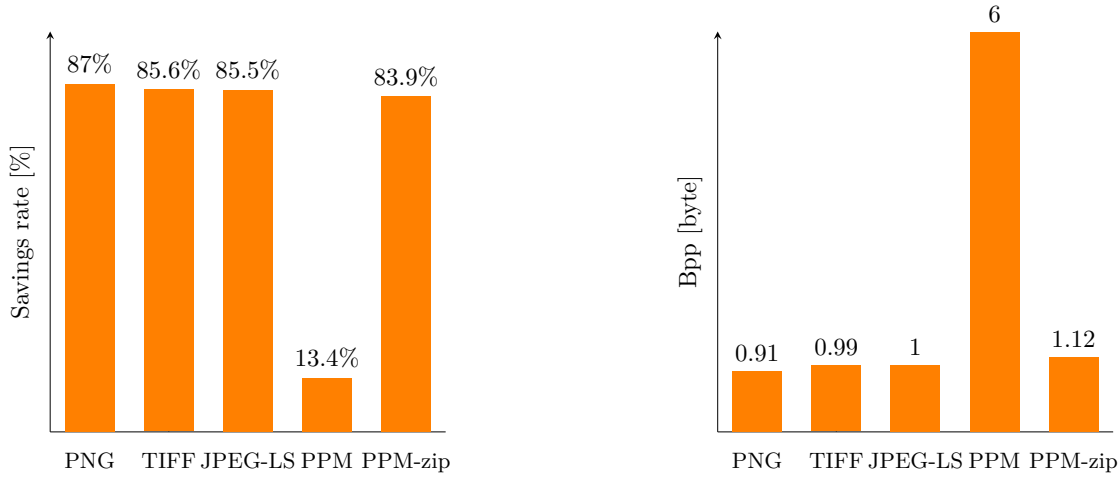


Figure 4.7: Frame packed in three single-channel images, the first containing the radius, the second the difference between real elevation and its interpolation and the third the difference between real azimuth and its interpolation.

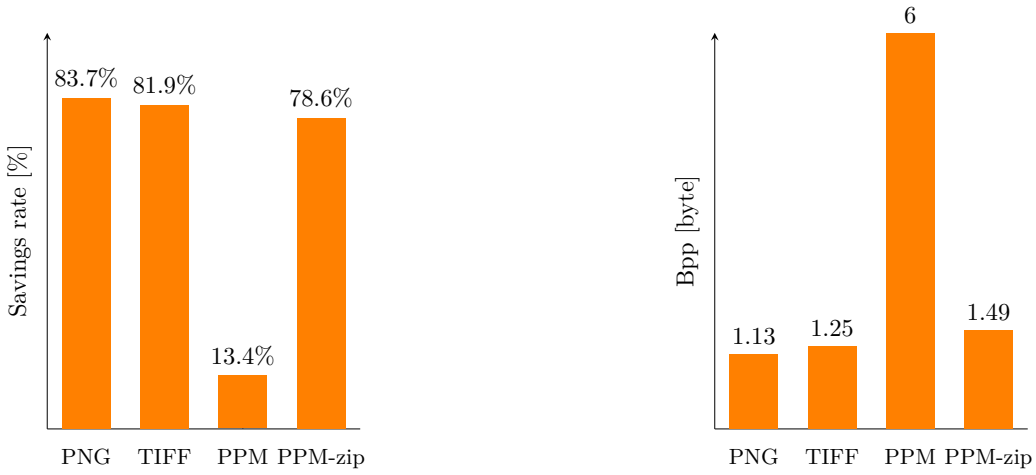


Figure 4.8: Frame packed in a tri-channel image, the first channel containing the radius, the second the difference between real elevation and its interpolation and the third the difference between real azimuth and its interpolation.

Finally, a solution based exclusively on angles predictions is presented in Tab. 4.3. The points have been converted into spherical coordinates and only the radius has been saved in a single-channel image. This trial has been implemented only in Matlab (PNG encoding).

Savings rate [%]	Bytes per point [Bytes]
79.93	0.69

Table 4.3: Frame packed in a single-channel image, containing only the radius, *azimuth* and *elevation* angles are predicted during decompression.

4.4.2 Compression quality

We conclude the image-based section by examining and evaluating the losses due to compression. Two sources of errors can be identified, due to the *packing* methods used:

1. **Quantization errors**: the type of LiDAR generated points is a single precision *floating point*; so it is necessary to remap them in the range of 16 bit unsigned integers and round the result to the nearer integer (Eq. 2.1 in Subsec. 2.4.1).

2. **Prediction errors**: in the strategy with spherical coordinates without storing *azimuth*, an error in its approximation is done.

However, the codecs used to compress the images, are all lossless, therefore they don't introduce any loss.

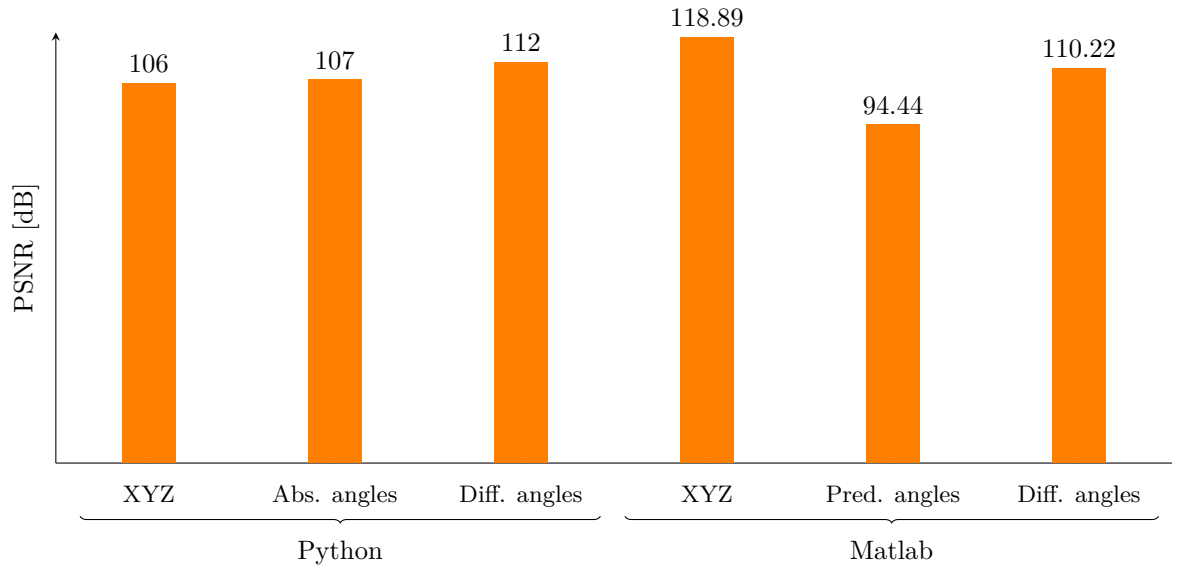


Figure 4.9: Image-based compression PSNRs.

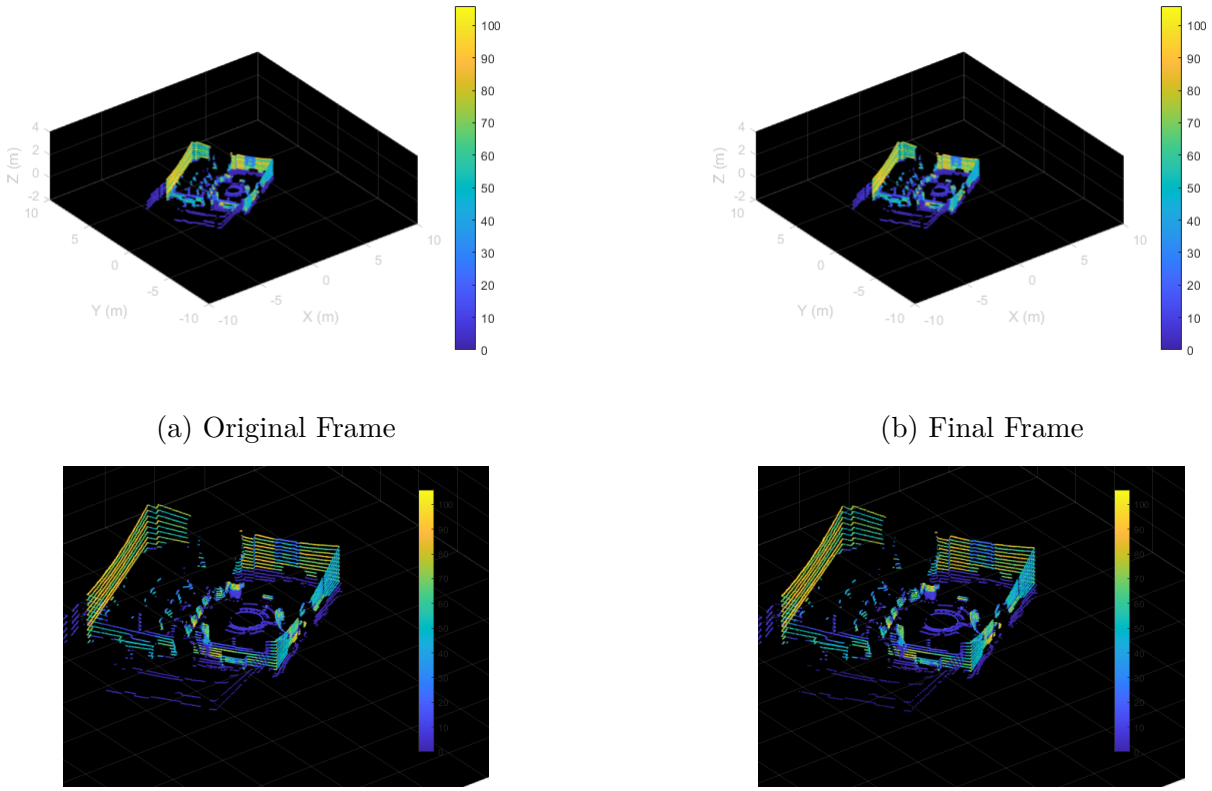
As can be seen in Fig. 4.9, using cartesian coordinates (*XYZ*) or spherical ones (*angles*) with real angles leads to similar PSNRs. Instead, saving the differences between real angles and their prediction gives a higher PSNR because the difference is very slight[3] and so we are able to achieve a smaller quantization error on the angles.

PSNR values exceed 100 dB for all packing strategies, so this compression method can be considered lossless.

Furthermore the very high PSNR (118dB) in Matlab cartesian compression (XYZ in fig. 4.9) is due to an estimator function. This function takes the value range and estimates the $scale$ and $offset$ factors. This allows, if possible, to avoid a quantization rounding, hence enhancing the PSNR. The casting from float to uint16 (Unsigned 16bit) is done by means of: $\text{IMG} = uint16(round(XYZ * scale) + offset)$ where XYZ is the original tensor with float coordinates.

[3] the difference is null for elevation and in the order of 1 % of the real value for azimuth.

Finally snapshots made in Matlab are shown in Fig. 4.10, to subjectively evaluate the quality of the frame after compression. All images are referred to the last frame of the 600 RPM dynamic dataset. The compression method was the spherical one with the azimuth difference illustrated in Subsec. 3.3.2.



(a) Original Frame

(b) Final Frame

(c) Original Detail

(d) Final Detail

Figure 4.10: Comparison between original and decompressed frame with intensity map.

Quantization error

As said before, most of the error is due to quantization. There is a way to calculate the theoretical uniform quantization error via the *quantization SNR (Signal to Noise Ratio)* Λ_q [Lau11]:

$$\Lambda_q = 3\frac{M_a}{V_{sat}^2}2^{2b} \tag{4.4}$$

where $M_a = \sigma_a^2 + m_a^2$ is the statistical power of the coordinates, $V_{sat} = \frac{\max - \min}{2}$ half of the range of coordinate values and b the number of bits used in the coding.
The quantization $PSNR$[4] can be estimated from the SNR in this way:

$$PSNR_q \cong \Lambda_q \frac{\max^2}{M_a} \tag{4.5}$$

Where max is the maximum coordinate value. In the proposed trials the estimated $PSNR_q$ is 107 dB[5], which is coherent with the obtained result (Fig. 4.9).

[4] In linear scale.

[5] Evaluated on the original tensors, so valid for cartesian methods.

4.5 Video-based Compression

4.5.1 Compression efficiency

In figure 4.11 the savings rate and the Bpp are shown for both cartesian and spherical with difference methods. Firstly we can observe that saving three grayscale images gives slightly better results than saving one tri-channel image (RGB) for the spherical method and the opposite for the cartesian one.

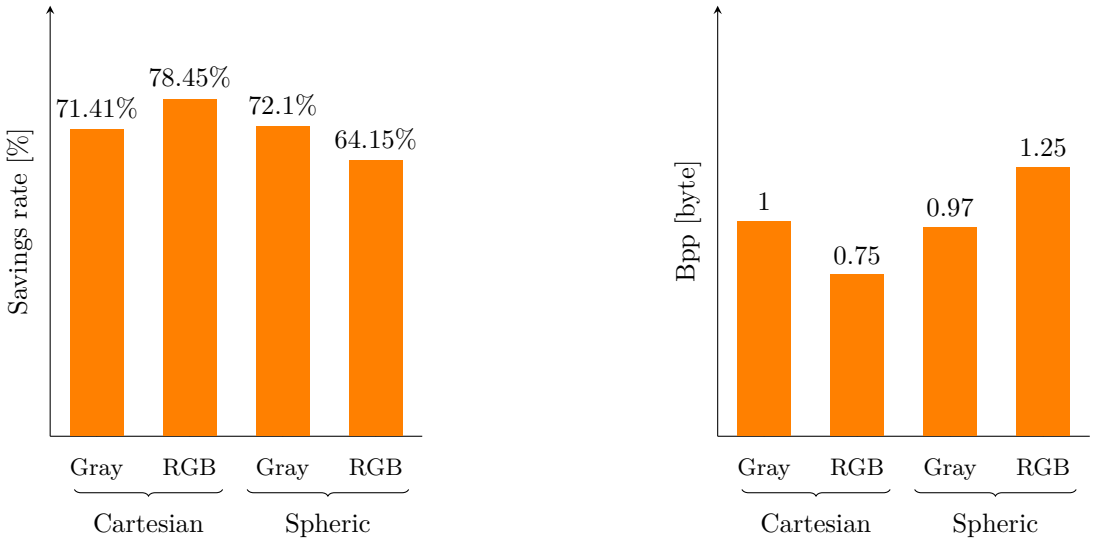


Figure 4.11: Frames sequence packed in three grayscale video or in a single colored (RGB) video.

4.5.2 Compression quality

We already discussed the sources of errors (Subsec. 4.4.2): since videos are a succession of images the errors continue to be the same. However video's frames need a 24 bit color depth re-parted as 8 bit per channel. This means that the original matrices have to be casted (Eq. 2.1) into 8 bit unsigned integers data format, hence worsening the performance (Fig. 4.12).

The MJ2 codec involved to compress the videos, is intra-frame and lossless, therefore it not introduce any loss.

Although the PSNR is still high (78 dB) and can be defined as lossless, such a large quantization error necessarily introduces artifacts for both the cartesian (Fig. 4.13c) and the spherical (Fig. 4.13b) compressor.

Another important result has been achieved within this thesis: in figure 4.13d it is possible to notice the results of a not completely lossless compression. In fact, this screen-shot was collected in Matlab using the MPEG-4 container with H264 inter-frame codec. This standard can guarantee compressions for point-clouds with saving rates up to 97% compared to the original PCAP file.

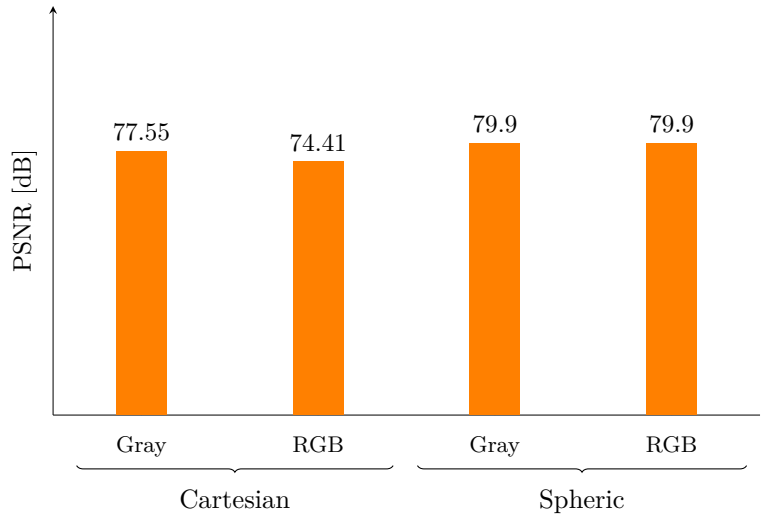
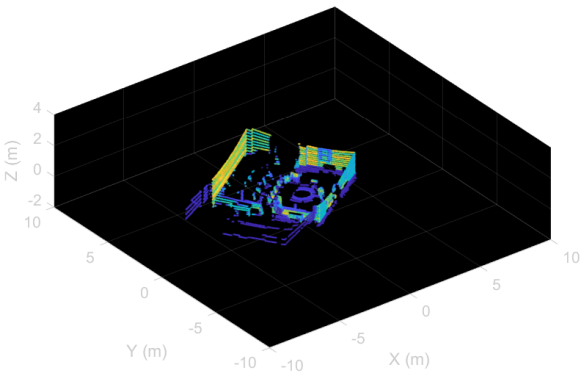
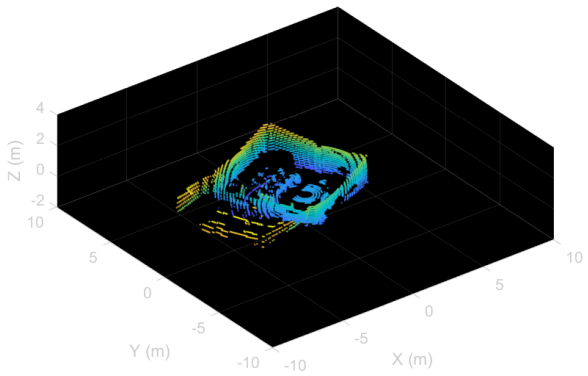


Figure 4.12: Video-based compression PSNRs.

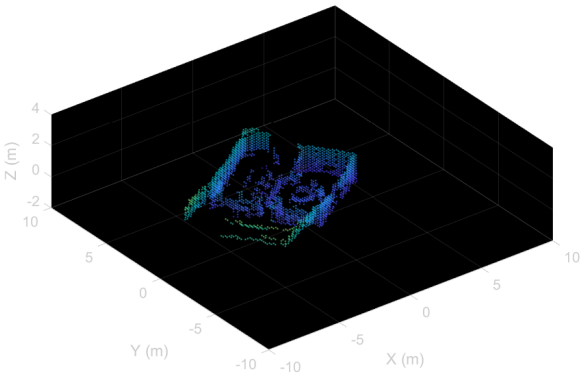
However, even when the lossless mode is set, it is evident from the figure (Fig. 4.13d) that this compression is not lossless, but the inter-frame compression almost completely ruins the data structure (during playback, the dataset visually appears as a rotating vortex of dots).



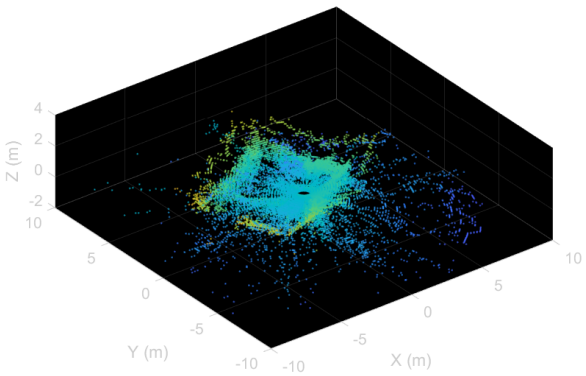
(a) Original Frame



(b) Spherical Compression



(c) Cartesian Compression



(d) Inter-frame Compression

Figure 4.13: Comparison between original and decompressed frame to underline the artifacts.

4.6 Dictionary-based Compression

4.6.1 Compression efficiency

As mentioned in Sec.3.5 this strategy of compression could be considered inter-frame since it shares the same dictionary for all the frames. Hence the mean results are shown both in term of static and dynamic datasets (differences defined in Sec. 3.1).

In figure 4.14 are shown the savings rate and the Bpp for both the cartesian and the spherical difference methods.

Firstly, we can observe that there are no substantial differences between static and dynamic frames. The inter-frame technique is effective in video compression because the differential information is less than the original. In dictionary compression this is not the case, so no tangible differences are exploited.

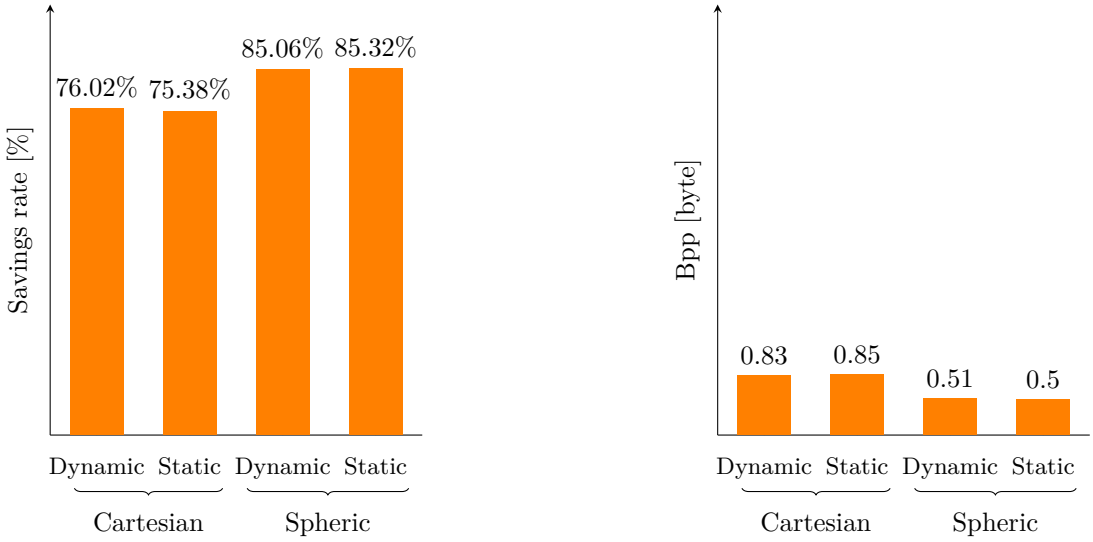


Figure 4.14: Frames sequence compressed with Lempel-Ziv-Welch algorithm and Huffman encoding

With savings rates over 76 % for cartesian methods and over 85 % for spherical ones (Fig. 4.14) Deflate gives an excellent compression level. However what should be noticed is the bitrate: 0.5 byte per point represents the best results from this thesis. The savings rate are not the absolute best, because as already specified (Subcap. 3.1.2), the implementation in Matlab maintains the original structure of the matrix without eliminating invalid points.

4.6.2 Compression quality

There is a strong quantization error due to the LZW specification ([Wel84]) which requires only 8-bit words. This leads to the same conclusions already covered for the video compressor (Subsec. 4.5.2) with the artifacts seen above (Fig. 4.13b and 4.13c).

Lempel-Ziv-Welch and Huffman coding are both lossless, therefore they do not introduce any deterioration.

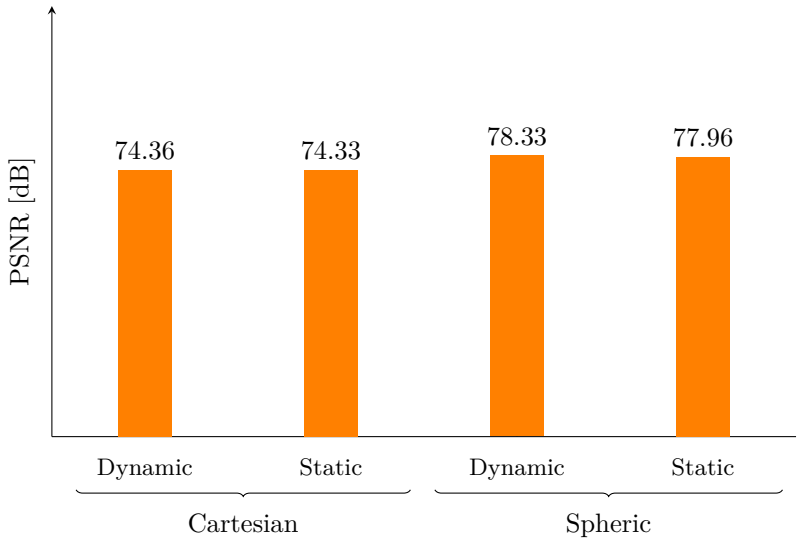


Figure 4.15: Dictionary-based compression PSNRs.

As can be seen in Fig. 4.15, using cartesian coordinates or spherical ones with differential angles leads to similar PSNRs.

PSNR values exceed 74 dB for both strategies and datasets, so this compression method can be considered lossless.

4.7 Comparison

The results between the various datasets were comparable. This fact could be expected for intra-frame compression (as image-based and video-based compressor), however it's interesting for inter-frame compression (as dictionary-based) which managed to remove all the redundancy.

In fact, intra-frame compression does not involved adjacent frames and for the inter-frame one there are no differences between static or dynamic scenarios, also the number of frames per minute (collected) does not affect the efficiency of compression. For this reason only the average results will be show in the summary graph.

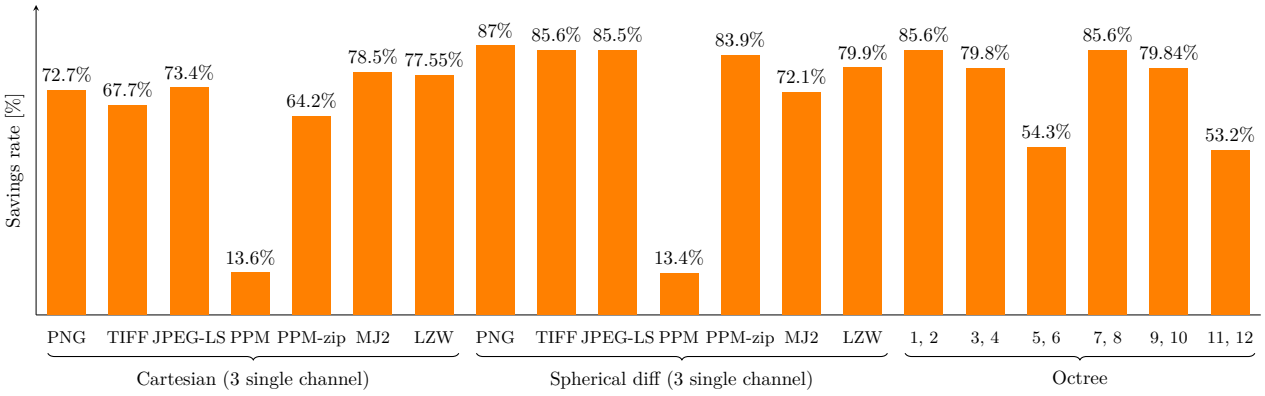


Figure 4.16: Overall Rates.

Let's now compare the results of the 3D-oriented compressor (octree), which is specific for 3D point-clouds like LiDAR observations, with the most efficients 2D-oriented compressors (image-based, video-based, dictionary-based), specifically designed for 2D frames.

In terms of savings rate (Fig. 4.16) the best solution are spherical coordinates with angles differences coded via PNG, which obtains a savings rate of 87 %.

Considering the structure integrity, and the Bpp from Fig. 4.17: Deflate (LZW in the graph) is the most efficient one. However the dictionary-based results are perfectly aligned with those of the image-based one if we consider that a matrices of 8 bit unsigned integers requires half of the space than a 16 bit unsigned integers one.

In general all the results of image-based compression using differential spherical coordinates, except PPM, are the best of all, since video-based inter-framee solution isn't able to preserve the data structure and intra-frame is coded on MJ2, an outdated profile compared to PNG.

There are several results over 80 % both in geometry-based and in image-based, excluding the cartesian methods. Though, leaving out PPM, all the image-based methods deliver better saving rates than the profiles which offer the highest PSNR among octree.

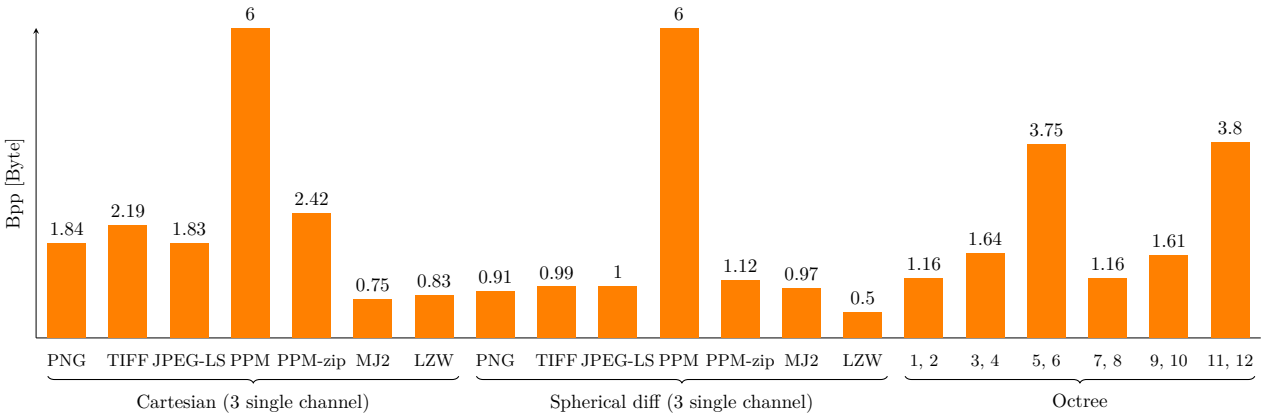


Figure 4.17: Overall Bpp.

In Fig. 4.18, XYZ represents the cartesian coordinates; *Abs.* and *Diff.* represent absolute and prediction difference spherical compression, respectively.

Clearly, only four profiles achieve a PSNR in the order of that of image-based algorithms. Video-based and dictionary-based suffers from a strong quantization error.

While in image-based methods the PSNR grows with the growing of the savings rate, using octrees the PSNR is lower for higher savings rate.

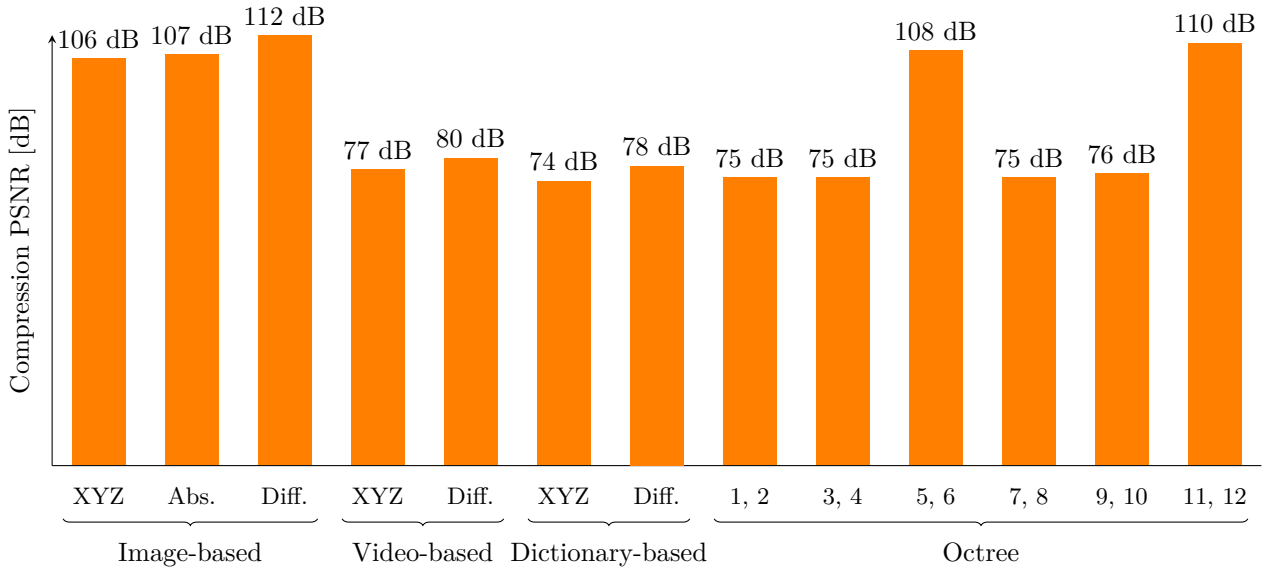
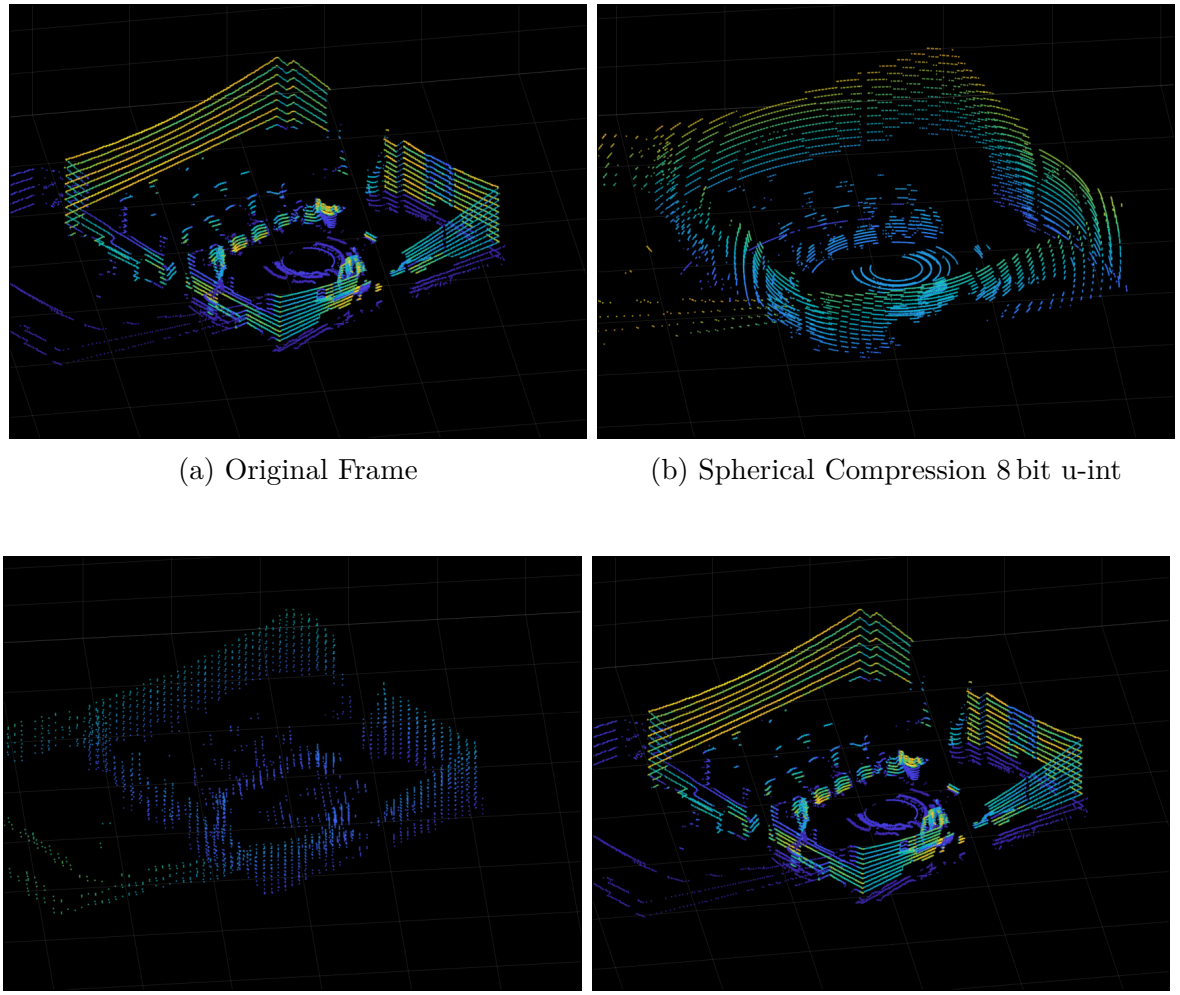


Figure 4.18: Overall PSNRs.

Finally, after all this analysis, a subjective point of view (Fig. 4.19d) confirms that the 2D-oriented method that involved PNG as image-based compressor is the best possible choice.

Both Fig. 4.19b and 4.19d refer to compression based on spherical coordinates. The difference is that the first uses a dictionary-based Deflate encoding that requires matrices with unsigned 8-bit integers, while the latter uses a PNG compression with 16-bit color depth images.

Unfortunately, the strong quantization has led to the following artifact: concentric spherical planes are seen due to too low resolution.



(a) Original Frame

(b) Spherical Compression 8 bit u-int

(c) Cartesian Compression 8 bit u-int

(d) Spherical Compression 16 bit u-int

Figure 4.19: Comparison between original and decompressed frame to underline the compression quality.

Chapter 5

Conclusion

In this thesis the Velodyne VLP-16 sensor was used to collect point-clouds in different modes (static scenario, dynamic scenario and moving sensor) which had to be compressed efficiently.

Two different techniques for frame compression were compared and implemented, 2D- and 3D-oriented.

The first one is a transformation of the points within a frame into two-dimensional pictures, then compressed via traditional image or video (both intra-frame and inter-frame) compression techniques or using the Deflate algorithm. The second one consists of point-cloud codification into a tree structure called *octree*, which allows us to save disk space compared to a direct coding.

This thesis also presented some advanced strategies based on spherical coordinates to improve image-based compression results. Anyway, only lossless compression methods have been examined.

The performance of these these techniques were analyzed in terms of compression efficiency and quality of the decompressed frame compared to the original.

Compression by images (2D-oriented) and by octree (3D-oriented) lead to similar results. This fact is explained by the nature of the LiDAR frames which, despite being clouds of three-dimensional points, are collected by 16 rotating lasers. Hence they are like two-dimensional images with the projection of the surrounding space; instead of colors, distances are saved as information.

In general the results of image-based compression using differential spherical coordinates and PNG encoding are the best of all since video-based intra-frame compression is coded on MJ2, an outdated profile compared to PNG.

Moreover we demonstrated the importance of lossless compression, as inter-frame lossy prediction wasn't able to preserve the data structure.

In particular, the video-based compression may be unsuitable. The noise from the quantization of the coefficients during the DCT (discrete cosine transform), even in the lossless mode, is not negligible (or rather, it is for images, but not for data of other nature).

Good performance were obtained with our algorithms: in particular the image compression with PNG codec reaches 87% of Savings Rate, while the octree compression with profiles 1 and 2 reaches 85.6% of Savings Rate.

Considering the structure integrity, and the Bpp, the Deflate strategy is the most efficient one. However the dictionary-based results are perfectly aligned with those of the image-based one if we consider that a matrix of 8 bit unsigned integers requires half of the space than a 16 bit unsigned integers one. In fact the PNG standard uses Deflate flat as a compression method.

Finally, as PSNR values exceed 100dB for the image compressor and 70dB for the octree, video and dictionary compressor, they can all be considered *lossless*.

A possible future development should be based on dictionary compression. In fact, the performance of an algorithm of this type strictly depends on the *text* to be compressed.

Therefore, in order to exploit repetitive patterns in a succession of point-clouds, it is possible to privilege some data "reading" directions over others.

The best way to sequence the coordinates values (*characters*) in a *text* is an interesting subject for further study.

References

[Al213] T. Al2. "Compression of LiDAR Data Using Spatial Clustering and Optimal Plane-Fitting". In: vol. 2 No. 2. 2013, pp. 58–62. DOI: 10.4236/ars.2013.22008.

[Bar20] Duncan Barclay. "LZW Compression/Decompression". In: *MATLAB Central File Exchange.* (2020).

[Bee19] Peter van Beek. "Image-based compression of LiDAR sensor data". In: *Electronic Imaging* (2019). ISSN: 2470-1173. DOI: 10.2352/ISSN.2470-1173.2019.15.AVM-043.

[CD19] Paul Caillet and Yohan Dupuis. "Efficient LiDAR data compression for embedded V2I or V2V data handling". In: 2019. arXiv: 1904.05649 [cs.RO].

[Deu96] L. Peter Deutsch. *DEFLATE Compressed Data Format Specification version 1.3.* RFC 1951. May 1996. DOI: 10.17487/RFC1951.

[DS17] Ž. Lukač D. Bećirbašić M. Molnar and D. Samardžija. "Video-processing platform for semi-autonomous driving over 5G networks". In: *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)* (2017), pp. 42–46. DOI: 10.1109/ICCE-Berlin.2017.8210584.

[GK15] T. Golla and R. Klein. "Real-time point cloud compression". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 5087–5092. DOI: 10.1109/IROS.2015.7354093.

[HN15] H. Houshiar and A. Nüchter. "3D point cloud compression using conventional image compression for efficient data transmission". In: *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT).* 2015, pp. 1–8. DOI: 10.1109/ICAT.2015.7340499.

[Huf52] D. A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE* 40.9 (Sept. 1952), pp. 1098–1101. ISSN: 2162-6634. DOI: 10.1109/JRPROC.1952.273898.

[Ise13] Martin Isenburg. "LASzip: lossless compression of LiDAR data". In: *Photogrammetric Engineering & Remote Sensing* 79 (Feb. 2013). DOI: 10.14358/PERS.79.2.209.

[Kam+12] J. Kammerl et al. "Real-time compression of point cloud streams". In: *2012 IEEE International Conference on Robotics and Automation.* 2012, pp. 778–785. DOI: 10.1109/ICRA.2012.6224647.

[KB04] Leif Kobbelt and Mario Botsch. "A survey of point-based techniques in computer graphics". In: *Computers Graphics* 28.6 (2004), pp. 801–814. ISSN: 0097-8493. DOI: 10.1016/j.cag.2004.08.009.

[Lau11] N. Laurenti. "Sources of Digital Information". In: *Principles of Communications Networks and Systems*. Ed. by N. Benvenuto and M. Zorzi. 2011. Chap. 3.1.6, pp. 137–196. DOI: 10.1002/9781119978589.ch3.

[Lib11] The Point Cloud Library. "The PCD (Point Cloud Data) file format". In: http://www.pointclouds.org/documentation/tutorials/pcd_file_format.php (2011).

[LKK19] G. H. Lee, K. H. Kwon, and M. Y. Kim. "Ambient Environment Recognition Algorithm Fusing Vision and LiDAR Sensors for Robust Multi-channel V2X System". In: *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*. July 2019, pp. 98–101. DOI: 10.1109/ICUFN.2019.8806087.

[Mäm+19] O. Mämmelä et al. "Evaluation of LiDAR Data Processing at the Mobile Network Edge for Connected Vehicles". In: *2019 European Conference on Networks and Communications (EuCNC)*. June 2019, pp. 83–88. DOI: 10.1109/EuCNC.2019.8802049.

[Mor+14] Vicente Morell et al. "Geometric 3D point cloud compression". In: *Pattern Recognition Letters* 50 (2014). Depth Image Analysis, pp. 55–62. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2014.05.016.

[MŽ11] Domen Mongus and Borut Žalik. "Efficient method for lossless LIDAR data compression". In: *International Journal of Remote Sensing* (2011). DOI: 10.1080/01431161003698385.

[RC11] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011. DOI: 10.1109/ICRA.2011.5980567.

[Sam88] Hanan Samet. "An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures". In: *Theoretical Foundations of Computer Graphics and CAD*. Ed. by Rae A. Earnshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 51–68. ISBN: 978-3-642-83539-1. DOI: 10.1007/978-3-642-83539-1_2.

[Sch+19] S. Schwarz et al. "Emerging MPEG Standards for Point Cloud Compression". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.1 (Mar. 2019), pp. 133–148. ISSN: 2156-3365. DOI: 10.1109/JETCAS.2018.2885981.

[SK06] Ruwen Schnabel and Reinhard Klein. "Octree-based Point-Cloud Compression". In: *Eurographics Symposium on Point-Based Graphics* (2006). DOI: 10.2312/SPBG/SPBG06/111-120.

[SPS12] J. Smith, G. Petrova, and S. Schaefer. "Encoding normal vectors using optimized spherical coordinates". In: *Computers Graphics* 36.5 (2012). Shape Modeling International (SMI) Conference 2012, pp. 360–365. ISSN: 0097-8493. DOI: 10.1016/j.cag.2012.03.017.

[TR98] Gabriel Taubin and Jarek Rossignac. "Geometric Compression through Topological Surgery". In: *ACM Trans. Graph.* 17.2 (Apr. 1998), pp. 84–115. ISSN: 0730-0301. DOI: 10.1145/274363.274365.

[Tu+19a] Chenxi Tu et al. "Point Cloud Compression for 3D LiDAR Sensor using Recurrent Neural Network with Residual Blocks". In: May 2019. DOI: 10.1109/ICRA.2019.8794264.

[Tu+19b] Chenxi Tu et al. “Real-time Streaming Point Cloud Compression for 3D LiDAR Sensor Using U-net”. In: *IEEE Access* PP (Aug. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2935253.

[W3C03] W3C. “Portable Network Graphics (PNG) Specification (Second Edition)”. In: http://www.w3.org/TR/2003/REC-PNG-20031110 (Nov. 2003).

[Wel84] Welch. “A Technique for High-Performance Data Compression”. In: *Computer* 17.6 (June 1984), pp. 8–19. ISSN: 1558-0814. DOI: 10.1109/MC.1984.1659158.

[ZL78] J. Ziv and A. Lempel. “Compression of individual sequences via variable-rate coding”. In: *IEEE Transactions on Information Theory* 24.5 (Sept. 1978), pp. 530–536. ISSN: 1557-9654. DOI: 10.1109/TIT.1978.1055934.