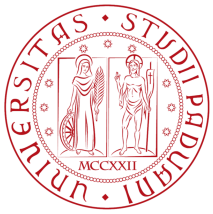


UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria dell'Informazione

Tesi di Laurea

Lossless compression of LiDAR frames

Relatore

prof. Andrea Zanella

Correlatori

dott. Marco Giordani

dott. Paolo Testolina

Laureando

Davide Peressoni

1167497

Anno Accademico 2019/2020

14 Luglio 2020

Abstract

LiDARs are sensors which obtain a three-dimensional mapping of the surrounding environment. In addition to provide accurate localization and mapping information, LiDARs can be used in object detection and in autonomous driving. These sensors produce a large amount of data and so it is necessary to implement compression techniques, especially if the data has to be processed in real time, e.g., for safety-critical services.

In this thesis two different techniques for LiDAR frame compression were compared and implemented, 2D- and 3D-oriented. The performance had been analyzed in terms of compression efficiency, quality of the decompressed frame compared to the original one and time spent to do the compression. Only lossless compression methods have been examined, limited to intraframe operations. We demonstrated that, thanks to the 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.

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

Sommario

I LiDAR sono sensori in grado di ottenere una mappatura tridimensionale dell'ambiente circostante. In aggiunta, trovano impiego per il riconoscimento di oggetti e nella guida autonoma. Questi sensori generano una gran mole di dati e pertanto è necessario adottare delle tecniche di compressione, in particolare nel caso i dati debbano essere elaborati in tempo reale, ad esempio per servizi safety-critical.

In questa tesi 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, qualità del frame decompresso rispetto all'originale e velocità di compressione. Si sono presi in esame metodi di compressione lossless limitandosi a quelli che operano intraframe. Grazie alla forma in cui vengono salvati 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.

Si sono inoltre cercate delle strategie volte a migliorare i risultati della compressione image-based, utilizzando le coordinate sferiche al posto delle classiche coordinate cartesiane.

Contents

List of Figures

Acronyms

Bpp Bytes per Point. 13, 19

FOW Field Of View. 11

GPS Global Positioning System. 1

JPEG-LS Lossless JPEG (Joint Photographic Experts Group). 3–5, 9, 18, 20, 21

LiDAR Light Detection and Ranging *or* Laser Imaging Detection and Ranging. i, ii, 1, 3–7, 9–11, 13, 16, 20, 21, 25

MSE Mean Squared Error. 13

PCD Point Cloud Data. 2, 3

PCL Point Cloud Library. 9, 14–16

PNG Portable Network Graphics. 3–5, 9, 18, 21, 25

PPM Portable Pixmap Format. 9, 18–21, 23

PSNR Peak Signal to Noise Ratio. 3, 6, 10, 13–16, 20, 21, 23, 25

RGB Red Green Blue. 5, 10

SNR Signal to Noise Ratio. 20, 21

TIFF Tagged Image File Format. 4, 5, 9, 18, 21

ZIP Compression using Deflate algorithm. 3, 9

Chapter 1

Introduction

LiDARs (Light Detection and Ranging *or* Laser Imaging Detection and Ranging) are sensors for three-dimensional environment mapping via distances measuring (ranging), done by illuminating the target with laser light and measuring the reflection with a sensor.

Differences in laser return times and wavelengths can then be used to make digital 3D representations of the target. It has terrestrial, airborne, and mobile applications.

LiDARs can map in addition to point positions also the surface reflectance, but, in contrast to traditional photography, cannot map any color. The peculiarity of the LiDAR is that provides three-dimensional information, so it is often used along with other bi-dimensional cameras like the thermal imager.

Today the combination of GPS 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.

However, in the future, a widespread use of the sensor is expected to achieve autonomous driving in the automotive industry thanks to computer vision and/or object detection techniques [LKK19]. As a matter of fact the information from LiDARs arranged over the car body is very useful to detect pedestrians, cyclists and other vehicles. In a *V2X* system the vehicles share this information in order to extend the visibility field.

This hypothetical scenario is based on the speed of 5G 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 Compression importance

Therefore, 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 in fact generates many points every second (for example in our tests it generated about $300\,000\ \frac{\text{point}}{\text{s}}$, in 1200 RPM configuration).

Another challenge is represented by the real-time compression (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.

1.1.1 State of the art

Currently there is not a compression standard for this type of data. 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.

Some developed methods are based on geometric compression (such as octree and triangular meshes). These methods 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. 4.3) of the trials proposed in this thesis.

1.1.2 Improvements

Other than implementing already existing imaged-based and geometric-based compression methods, for this thesis there have been made some improvements in imaged-based compression. Using spherical coordinates instead of cartesian ones there is a correlation between the coordinate angles and the point position in the 2D projection. This correlation brings to better compression results, as we can see in Subsec. 4.3.2.

Furthermore this correlation can be used to make a prediction of the angles. Predicting the angles allows us to save only the prediction error, and so achieve a better quality, as can be seen in Subsec. 4.3.2.

1.2 Thesis overview

We will now see in a more detailed way some exiting compression methods (Ch. 2). This description is followed by and overview of the trials made (Ch. 3) and the obtained results (Ch. 4).

As said before, some image-based compression methods give better results than the geometric-based tested. With the improvements proposed the results are beyond enhanced.

Chapter 2

Related Works

In this chapter we will analyze the main existing compression methods for LiDAR point clouds. The following methods are specifically designed for point clouds, except the image-based ones which are designed for 2D images, but can be used along with a bi-dimensional projection of the LiDAR point cloud.

2.1 PCD and LASzip

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

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

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

Using binary PCD the results are as follows:

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

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 (Peak Signal to Noise Ratio), 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.

In fact PCD is not a compression algorithm: it saves all the points coordinates, one after another[1].

LASzip is an evolution of the ZIP format[2] (lossless), specifically designed to compress point clouds. It saves the difference between two adjacent points using an entropy encoding, so LASzip performance depend on the point order.

The results of this method are reported in [Ise13]. They are comparable with those obtained during our tests, instead of classical ZIP where the overall compression is not very efficient.

Note however that comparing LASZip with Image-based Compression (Ch. 3.3), 2D-oriented methods based on PNG or JPEG-LS result more effective.

[1] The coordinates are saved in a table space/row separated with the '`ascii`' encoding.

[2] The ZIP format usually uses the DEFLATE algorithm, based on Huffman coding [Deu96].

2.2 LASComp

Another 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 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. Because LASComp predicts the coordinates (x, y, z), and then calculates the error between them and the "true" position of the point.

A similar method will be described in Subsec. 3.3.2, where the error is very small, so it implies a restricted domain of possible values and therefore the use of an efficient saving format.

However, the 3.3.2 strategy results better 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

These methods map LiDAR points into geometrical structures and use the structure properties to store them efficiently. Since point clouds have a three-dimensional nature, 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 use 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

Another structure similar to the Octree approach is the Voxel Grid (VG). The VG sub-sampling technique is based on a grid of 3D Voxels. 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 Voxel as spheres, a fast algorithm was developed.

The use of a structure allows to perform some operations like fast searching of the neighbors, reduce the amount of points visualized or 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] mapping a spherical coordinate on 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 can be the one 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 and JPEG-LS. These three methods before using a Huffman-like entropy coding apply a filtering for differential predicting [W3C03]. The value of each pixel is predicted from the values of previous neighboring pixels (above and left in the matrix): in such way the compression takes advantage of image continuity and achieves better results than using a generic compression.

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 (we will discuss that in Sec. 3.1), 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 due to 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 proper functioning of the image compression algorithms, as can be proved by the results (visible in Sec. 4.3).

To treat the matrix like an image, it is needed to cast the coordinates from float to unsigned integers (in our trials it was chosen 16 bits because with 32 bits we have not got any improvement), this causes a data quantization (Subsec. 4.3.2) and so a limited PSNR (not infinite like in the theoretical lossless), but it is still very high.

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

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

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

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

2.5 Further Methods

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

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]: it was designed for real-time application, it uses a neural network and also interframe 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 examined.

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

Chapter 3

Proposed trials

In this thesis are presented the results (visible in Ch. 4) of LiDAR frames compression from five datasets, using the octree method as geometry-based compression and various strategies of image-based compression.

In this chapter the datasets creation and the used compression methods will be 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[1]. Three datasets were collected at 600 RPM[2], while the other two at 1200 RPM. 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 dataset at 600 RPM. 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 dynamic datasets there are also 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[3] to be used in Python and C++.

Each CSV 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, the columns represent the temporal instants of sample collection.

In the trials the intensity was not taken into consideration as the objective was to compress only the information relative to the position of the points.

[1] 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.

[2] Rotations Per Minute. At each rotation of the beams one frame is collected, so 600 RPM means 600 frame/ min or 10 fps.

[3] One CSV per frame.

Figure 3.1: The room during data collection.

3.1.2 Points Extraction

The Matlab module imports PCAP files produced by 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.

3.2 Geometry-based compression

In the trials, the PCL (Point Cloud Library) library[5] [RC11] has been 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` $1\,\mathrm{cm}^3$ resolution, without color, online fast encode
2. `LOW_RES_ONLINE_COMPRESSION_WITH_COLOR` $1\,\mathrm{cm}^3$ resolution, color, online fast encode
3. `MED_RES_ONLINE_COMPRESSION_WITHOUT_COLOR` $5\,\mathrm{mm}^3$ resolution, without color, online fast encode
4. `MED_RES_ONLINE_COMPRESSION_WITH_COLOR` $5\,\mathrm{mm}^3$ resolution, color, online fast encode
5. `HIGH_RES_ONLINE_COMPRESSION_WITHOUT_COLOR` $1\,\mathrm{mm}^3$ resolution, without color, online fast encode
6. `HIGH_RES_ONLINE_COMPRESSION_WITH_COLOR` $1\,\mathrm{mm}^3$ resolution, color, online fast encode
7. `LOW_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR` $1\,\mathrm{cm}^3$ resolution, without color, offline efficient encode
8. `LOW_RES_OFFLINE_COMPRESSION_WITH_COLOR` $1\,\mathrm{cm}^3$ resolution, color, offline efficient encode
9. `MED_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR` $5\,\mathrm{mm}^3$ resolution, without color, offline efficient encode
10. `MED_RES_OFFLINE_COMPRESSION_WITH_COLOR` $5\,\mathrm{mm}^3$ resolution, color, offline efficient encode
11. `HIGH_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR` $1\,\mathrm{mm}^3$ resolution, without color, offline efficient encode
12. `HIGH_RES_OFFLINE_COMPRESSION_WITH_COLOR` $1\,\mathrm{mm}^3$ resolution, color, offline efficient encode

Table 3.1: List of octree compression profiles.

3.3 Image-based compression

The image-based compressor (in the matrix form presented in Subsec. 2.4.1) has been implemented in Python, using traditional *lossless* image compression algorithms. The involved codecs were PNG, TIFF, JPEG-LS, PPM and zipped PPM[6]. These formats were selected in order to compare the trial results with those in [Bee19].

[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.
[6] It is the PPM file saved with a ZIP compression.

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 data is 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 (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. 1.1.2, some improvements in imaged-based compression have been made. They are now presented in this subsection.

To achieve better results (visible in Sec. 4.3), the points have been converted into spherical coordinates[7]: a *radius* (ρ) and 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 three spherical coordinated have been saved in three grayscale images.

Angles difference

Trying to take a better advantage from the elevation and azimuth channels, compared to the previous trial, it has not been saved the absolute angle, but its difference with the value obtained by a linear interpolation. This allows us to save some space and to raise the PSNR (about 5 dB).

[7] We will see in Ch. 5 that such conversion is not needed if the compression in implemented on-board, since LiDARs natively work in spherical coordinates.

The *elevation* and *azimuth* angles are linearly interpolated from the row and column indexes. To be more precise, given the LiDAR *FOW (Field Of View)* (the angle between two outer beams) and the image shape is $N \times M$, 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:

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

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

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.

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

Chapter 4

Performance Results

Now the trial results will be analyzed: for both methods we will describe the compression efficiency, the quality of the compression and the time spent to compress. Then this chapter will be ended by a comparison between geometry-based and image-based compression results.

Before the analysis we will describe the performance metrics used to compare these results.

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:

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

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

4. **Computation time**
   The compression times had been measured on a machine with *Intel Core i5-4210U* @ 1.70GHz running *Linux* 5.4.40-1, *Python* 3.8.2, *g++* 9.3.0 and using the library *PCL* 1.9. All the trails had been run single-threaded.

For each method we will see the following plots:

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 datasets.

3. **Compression time**
   The plot of the computation time for compression in function of the number of points. In this times it is not included the time to load a frame because it could be very different on each scenario (internal or external drive, HDD or SDD, on-board, ...).

4.2 Geometry-based compression

4.2.1 Compression efficiency

As shown in Fig. 4.1 the octree savings rate decreases with the increasing of resolution[1]. In *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 *online* (the first six profiles, which favor encoding speed) and *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.

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.

4.2.2 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 the octree it is not feasible to use the usual PSNR, 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.

[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.

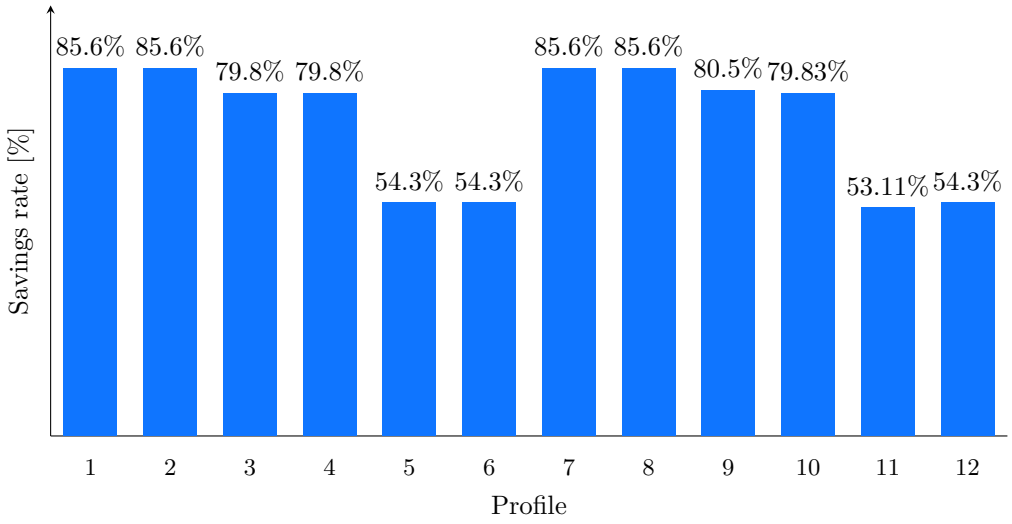


Figure 4.1: Octree savings rates.

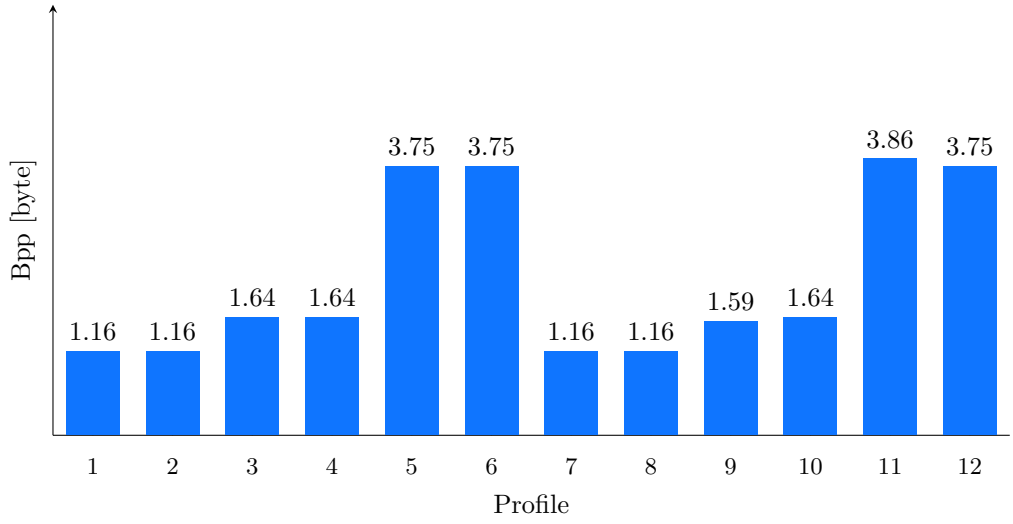


Figure 4.2: Octree Bytes per Point.

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

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

1. **Clouds with unique point:** It is not really a pairing method: for each point of the original cloud a new cloud with only that point is generated, that cloud is compressed in octree and then decompressed; eventually is evaluated the MSE between the point in question and the lonely 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. *Color* and *without color* profiles gave the same results, so we grouped them into one column.

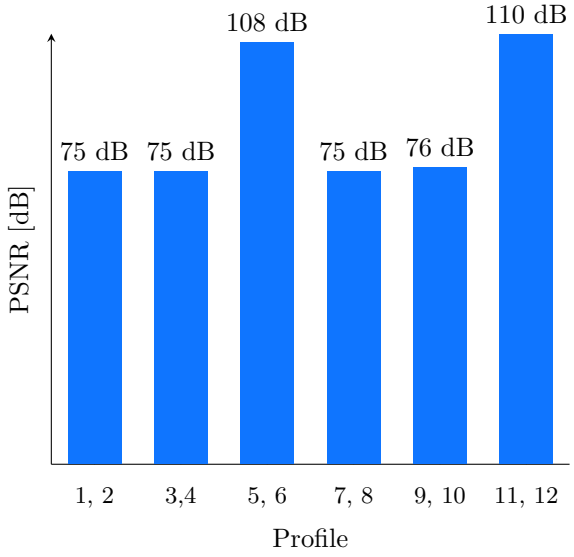


Figure 4.3: Octree compression PSNRs.

As we can see the best results are taken by the profiles 5, 6, 11, 12 but they gave also the worst result in savings rate. Anyhow all profiles are over 75 dB and so can be considered lossless.

4.2.3 Compression time

The compression time can be seen in Fig. 4.4. Leaving out the profile 11[2] (which gives also the worst savings rate), the *color* profiles are equally or slightly slower than the corresponding *without color*.

In the *offline* profiles *without color* the octree is rebuild storing only positions. This should lead to a higher savings rate since only the hierarchical octree data structure is encoded[3]. This rebuilding is very heavy, in particular for profile 11 which has a lot of nodes.

In spite of the name the *online fast encode* profiles are slower than the corresponding *offline efficient encode* ones, because they are intended to be faster on interframe compression[4]. Profiles with a higher resolution takes more time to compress data, as expected.

The mean compression time is around $1.19\,\frac{\mu s}{point}$ which allows to compress $840\,336\,\frac{point}{s}$. Using profile 11 we can handle $434\,622\,\frac{point}{s}$, so also with the slowest profile we can compress in real-time[5].

[2] High resolution, offline, without color.

[3] In the results here presented this is not true, perhaps because the default profiles provided by the PCL library are optimized for structured-light 3D scanner point clouds and not for LiDAR point clouds.

[4] The PCL library support differential encoding between subsequent frames, but in this thesis only intraframe compression methods have been considered.

[5] As said in Sec. 1.1, in our trials the LiDAR had generated about $300\,000\,\frac{point}{s}$ @ 1200 RPM.

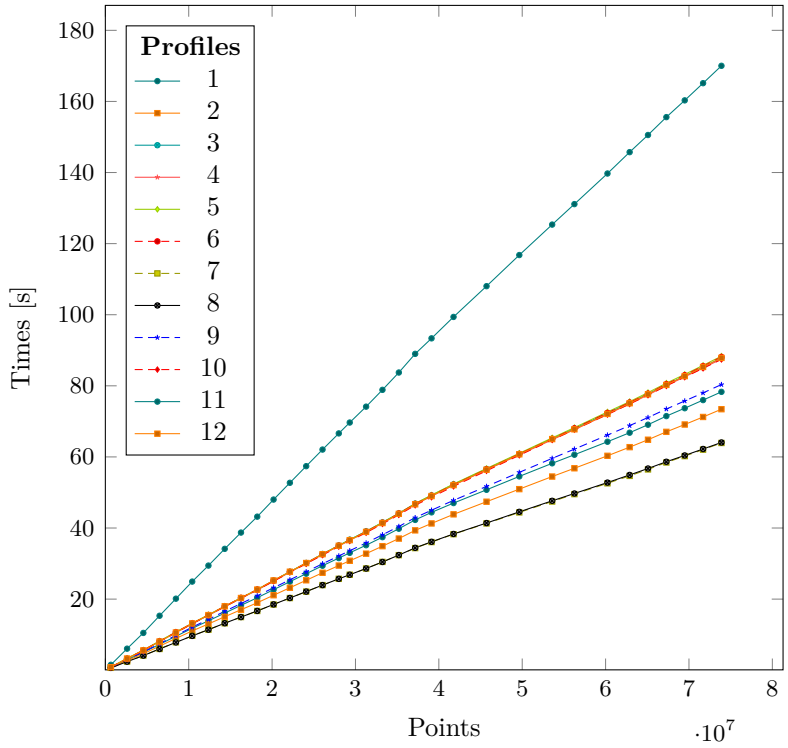


Figure 4.4: Octree compression times.

4.3 Image-based compression

4.3.1 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. 4.5, 4.6) and spherical (Fig. 4.8, 4.9) methods.

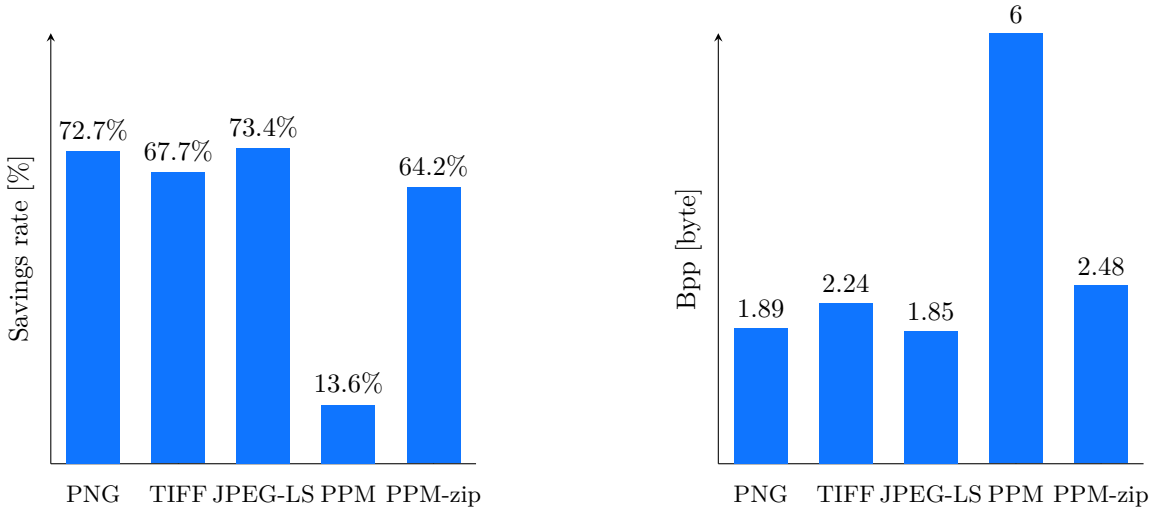


Figure 4.5: Frame packed in three single-channel images, the first for the X coordinate, the second for Y and the last for Z.

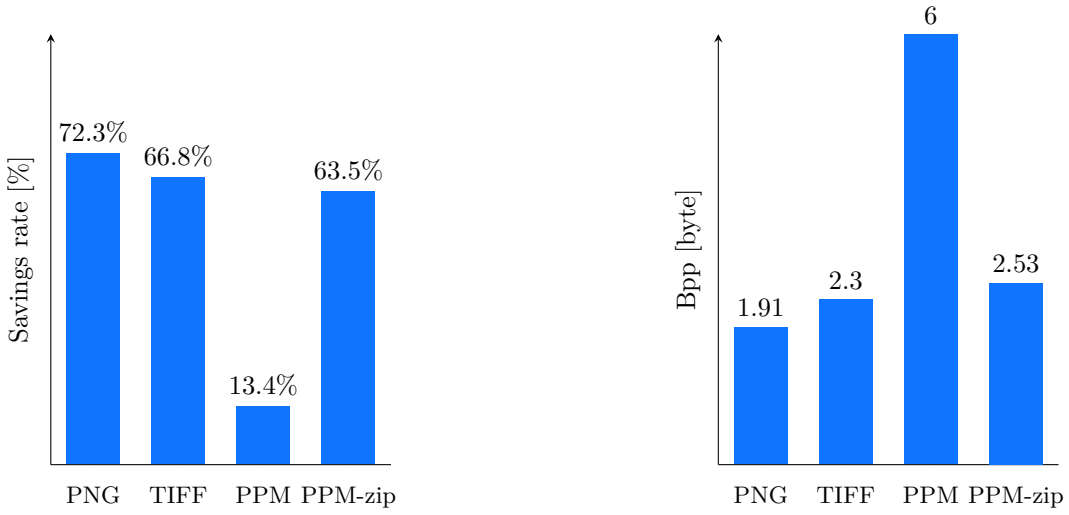


Figure 4.6: Frame packed in a tri-channel image, where the coordinates XYZ have been saved in the three color channels.

With savings rates over 65 % for cartesian methods (Fig. 4.6) and over 80 % for spherical ones (Fig. 4.9) PNG an TIFF give an excellent compression level, and nowadays they are 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[6], 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.

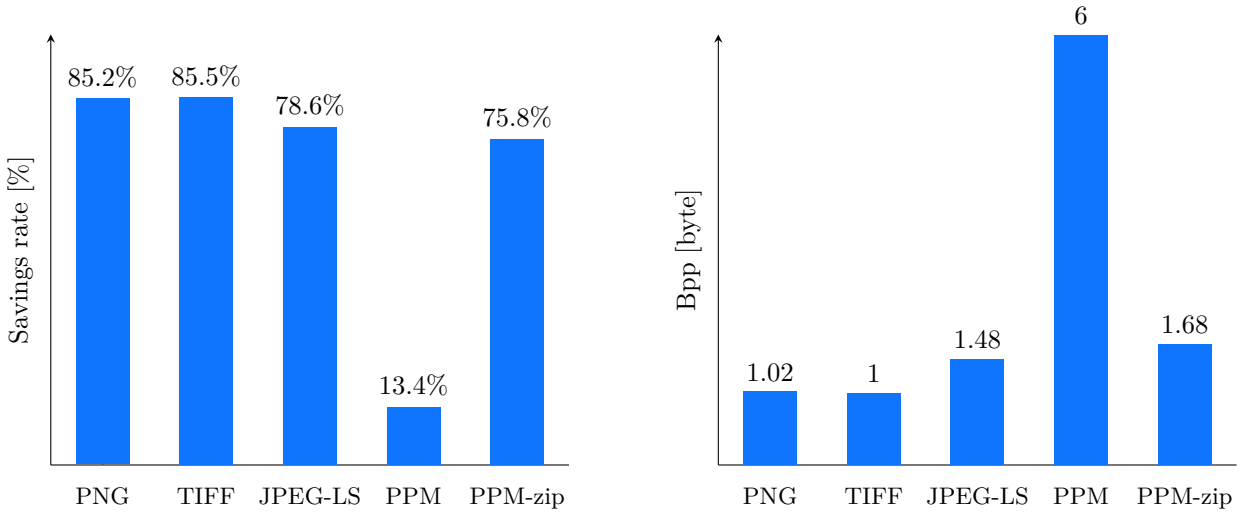


Figure 4.7: Frame packed into three single-channel images, in the first has been saved the radius, in the second the elevation and in the third the azimuth.

Saving the points using spherical coordinates, instead of cartesian ones, gives better results in terms of compression (Fig. 4.5 and 4.7).

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

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). In fact the Bpp (left out PPM) is around ⅔ times that of cartesian method, since only radius and azimuth are stored for each point while the elevation is stored only for the first column[7].

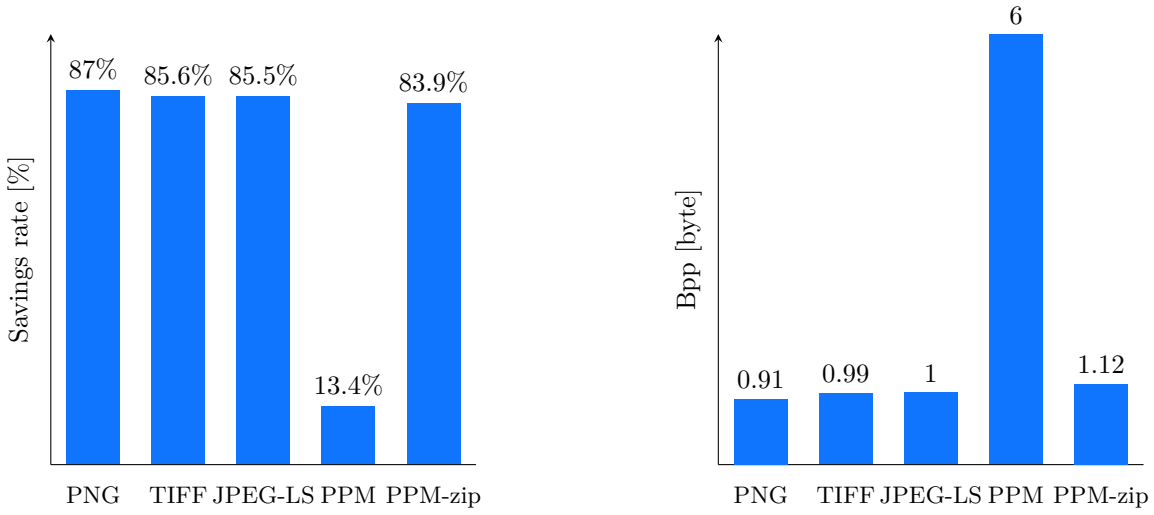


Figure 4.8: Frame packed in three single-channel images, the first containing the radius, the second the difference between real elevation and its interpolation and the third the difference between real azimuth and its interpolation.

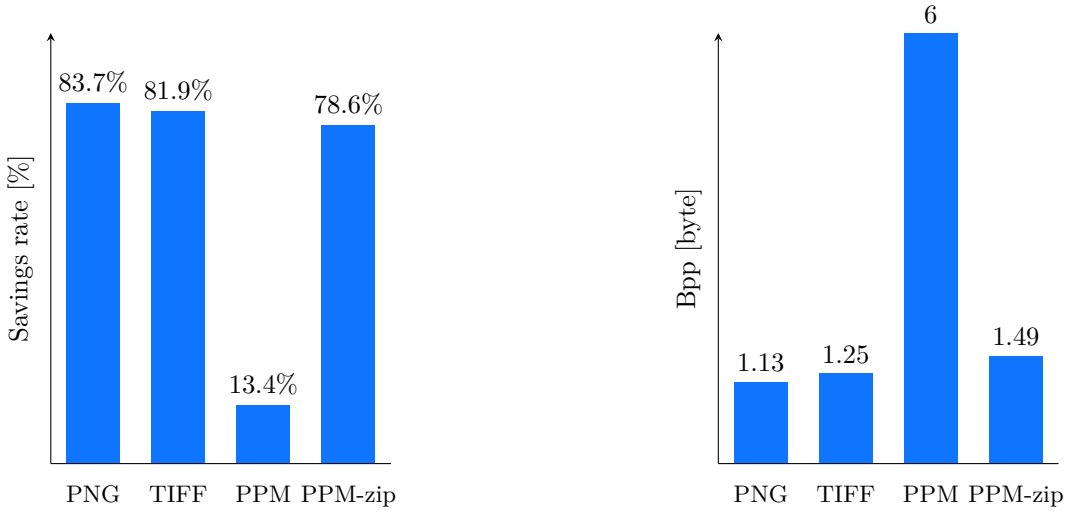


Figure 4.9: Frame packed in a tri-channel image, the first channel containing the radius, the second the difference between real elevation and its interpolation and the third the difference between real azimuth and its interpolation.

[7]This thanks to the differential predicting filter of image compression algorithms (see Sec. 2.4).

Furthermore saving the difference between angles and their prediction allows to raise a little the savings rate, in particular for JPEG-LS and zipped PPM, this because the difference is often null and these codecs work well with zeros.

4.3.2 Compression quality

We can identify the source of errors in quantization: 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 nearest integer (Eq. 2.1 in Subsec. 2.4.1).

On the other hand, the codecs used to compress the images, are all lossless, therefore they do not introduce any loss.

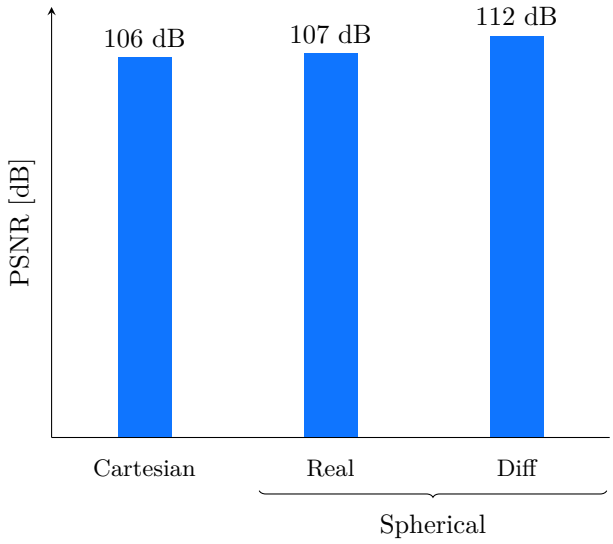


Figure 4.10: Image-based compression PSNRs.

As can be seen in Fig. 4.10, using cartesian coordinates or spherical ones 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 little[8] and so we 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.

Quantization error

As said before all 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]:

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

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

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 $PSNR$[9] 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[10], which is coherent with the obtained result (Fig. 4.10).

4.3.3 Compression time

We saw before that in terms of compression saving three grayscale images is better than saving one tri-channel image; now we can see that this is true also in terms of compression time (Fig. 4.11).

In terms of compression time PPM is the winner (as it does not compress) despite its low performance in terms of savings rate, as illustrated in Fig. 4.6; but its zipped version takes the worst compression time (since it needs the PPM encoding and makes a generic compression). PNG and TIFF are slightly faster than JPEG-LS.

Saving the points using spherical coordinates, instead of cartesian ones, gives better results in terms of compression paying only a little increase in compression time (for the coordinate conversion). Moreover the computation of the difference between angles and predictions does not lead to significant time increasing, but rather this method takes fewer time than that with real angles. This could be attributed to the fact that the elevation channel is totally null and so easy to compress.

Using PNG and TIFF with spherical coordinates, which were the winner methods in compression efficiency, it takes about 177 ns to compress a point, so we can compress around 5 650 000 $\frac{\text{point}}{\text{s}}$.

Since the slowest method (zipped PPM with spherical coordinates difference using one tri-channel image) can compress 1 968 687 $\frac{\text{point}}{\text{s}}$, each of the image-based methods here presented can widely handle a real-time compression.

4.4 Comparison

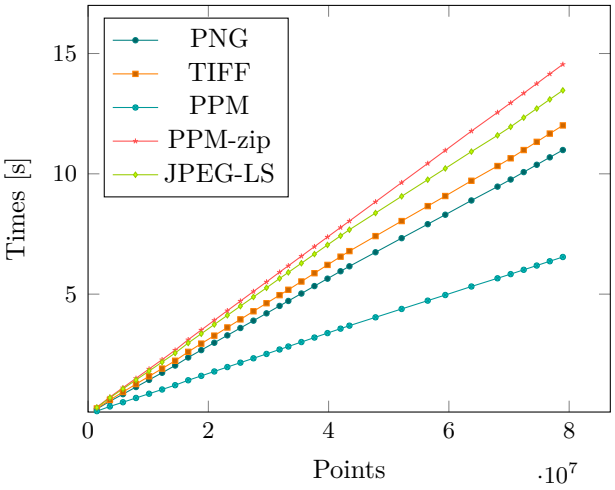
The results between the various datasets were comparable, as could be expected since intraframe compression does not involve the adjacent frames. Therefore, there are no differences between static or dynamic scenarios, also the number of frames per minute does not affect the efficiency of compression. For this reason only the average results have been shown.

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

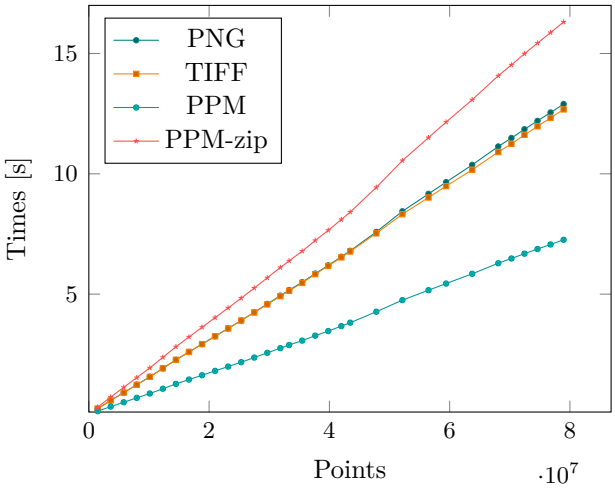
In terms of savings rate (Fig. 4.12) the winner is PNG using spherical coordinates with angles differences, which obtains a savings rate of 87 %. In general all the results of image-based compression using differential spherical coordinates, except PPM, are the best of all.

[9] In linear scale.

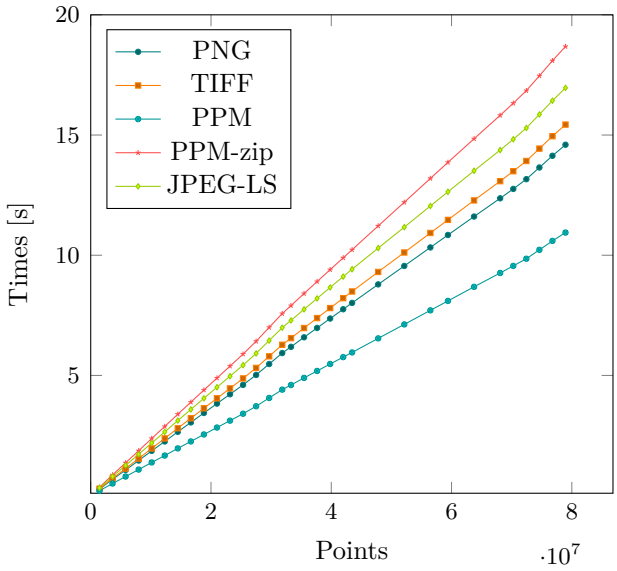
[10] Evaluated on the original tensors, so valid for cartesian methods.



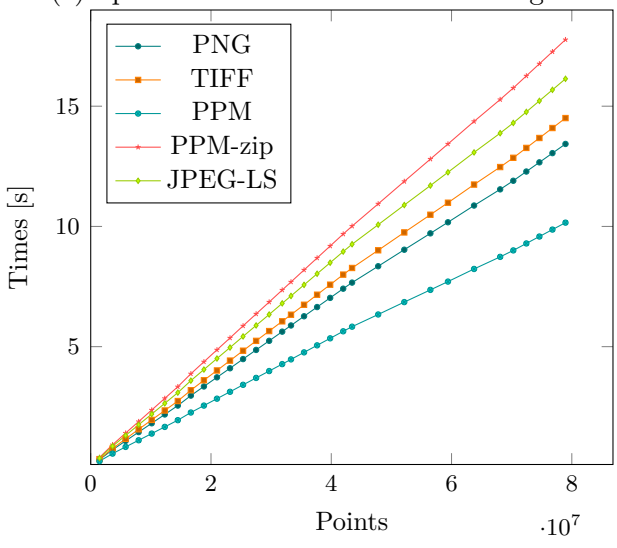
(a) Cartesian. Three single-channel images.



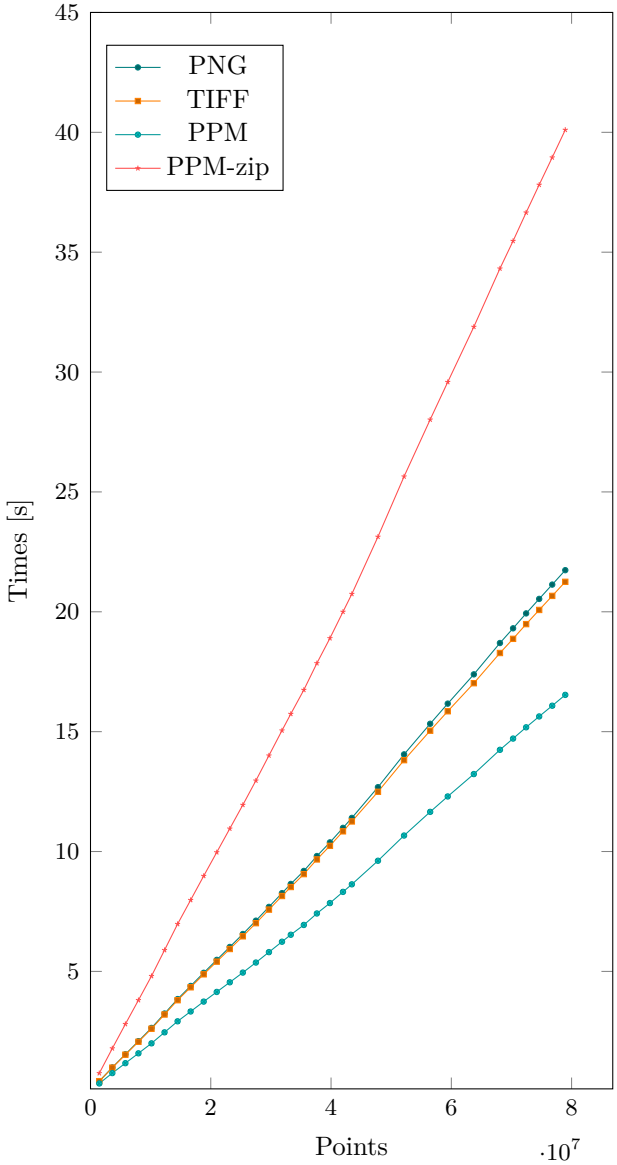
(b) Cartesian. Tri-channel image.



(c) Spherical coordinates with real angles.



(d) Spherical with angle differences. Three single-channel images.



(e) Spherical with angle differences. Tri-channel image.

Figure 4.11: Image-based compression times.

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.

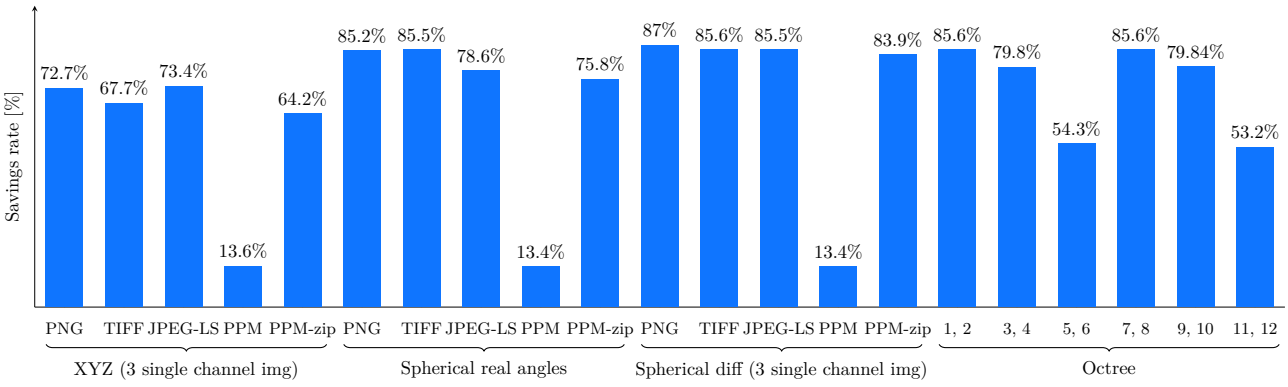


Figure 4.12: Overall Rates.

In fact, as we can see in Fig. 4.13, only four profiles achieve a PSNR in the order of that of image-based algorithms. The winner method is the image-based with differential spherical coordinates (112 dB), as for savings rate.

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.

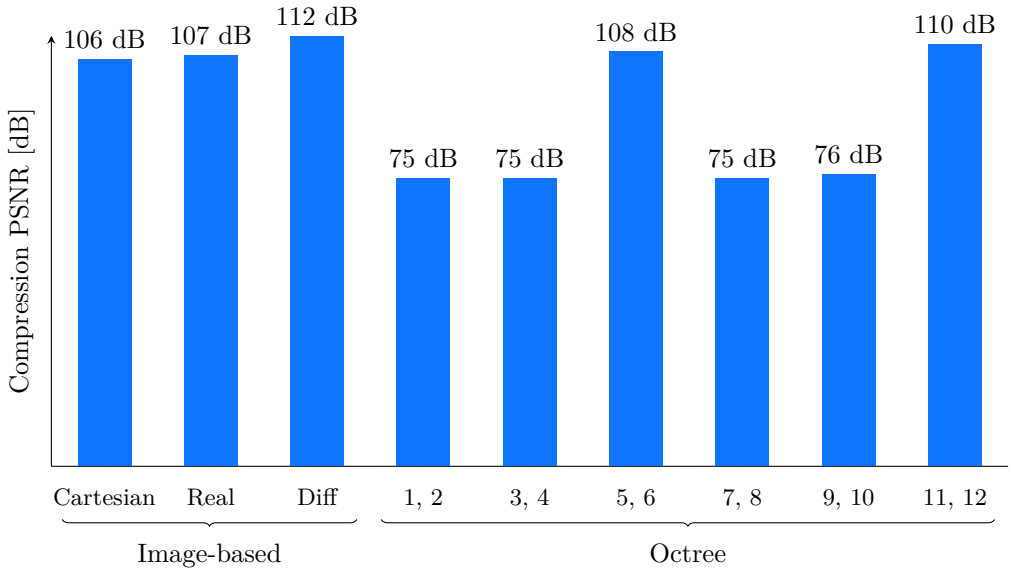
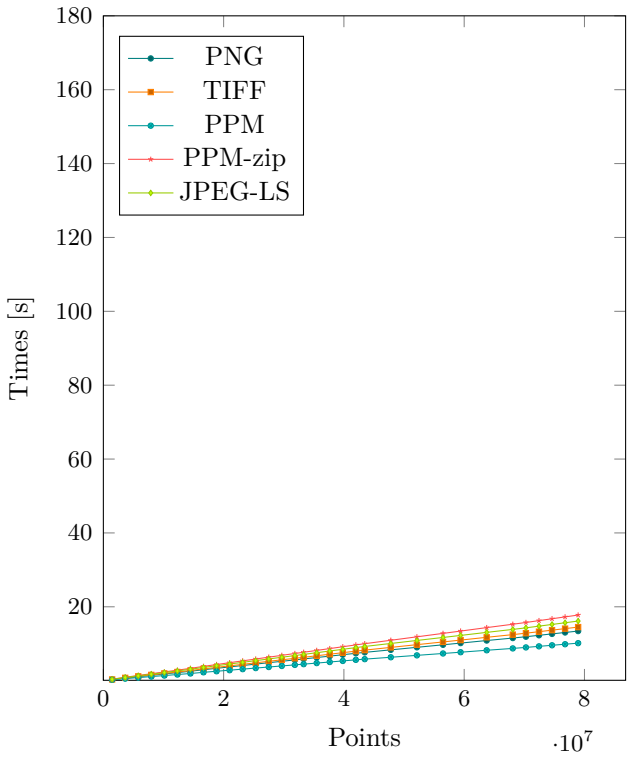


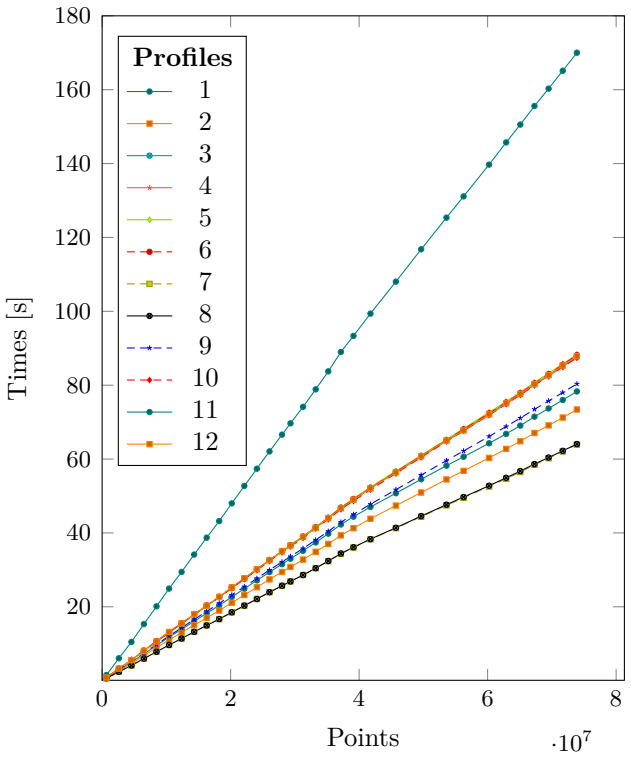
Figure 4.13: Overall PSNRs.

Comparing the compression time we can see that image-based compression is on average four time faster than geometry-based (Fig. 4.14). So using octree is not a good choice even for compression time, besides the savings rate / PSNR trade-off.

As said before, using spherical coordinates is slower than using cartesian ones, but the difference is very small (around 50 $\frac{\text{ns}}{\text{point}}$), so it is worth to use spherical coordinates which achieve better results in terms of savings rate and PSNR.



(a) Compression time for spherical coordinates with angle differences saved as three single-channel images.



(b) Octree compression time.

Figure 4.14: Compression time comparison.

Chapter 5

Conclusion

In this thesis two different techniques have been implemented to compress point-clouds generated by a LiDAR sensor. The first one is a transformation of the points within a frame into two-dimensional pictures and then compressing them via traditional imaging algorithms. Instead, the second one consists in point cloud codification into a tree structure called *octree*, which allows us to save disk space compared to a direct coding.

Compression by images (2D-oriented) leads to better results than compression by octree (3D-oriented), 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.

Very high savings rate were obtained in both 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.

In terms of speed the image-based is about 4 times faster than the octree compressor. Image compression algorithms can be also easily implemented on-board [YVS09; Li+12; Lam+02], further reducing the compression time. Since LiDAR natively works with spherical coordinates[1] implementing the image-based compression with spherical coordinates on-board removes the task to convert the coordinates.

Furthermore, as PSNR values exceed 100 dB for the image compressor and 75 dB for the octree compressor, them both can be considered *lossless*.

A possible future work for improving the obtained results could be using existing video compression techniques, along with the differential spherical coordinates packing presented in this thesis. This method should achieve a higher savings rate (since the video compression operates inter frame) keeping the high PSNR offered by this packing solution.

Another future work could be study if there are some predicting techniques which can be applied to the *radius* for the differential spherical coordinates packing strategy.

[1] The LiDAR measures the radius and knows the elevation from the beam number and the azimuth from its rotation position.

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.

[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.

[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.

[Lam+02] Catherine Lambert-Nebout et al. "On-board Optical Image Compression for Future High Resolution Remote Sensing Systems". In: *Proceedings of SPIE - The International Society for Optical Engineering* 4115 (Sept. 2002). DOI: 10.1117/12.411557.

[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.

[Li+12] Yunsong Li et al. "FPGA Design of Listless SPIHT for Onboard Image Compression". In: Jan. 2012, pp. 67–85. DOI: 10.1007/978-1-4614-1183-3_4.

[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.

[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).

[YVS09] Guoxia Yu, Tanya Vladimirova, and Martin N. Sweeting. “Image compression systems on board satellites”. In: *Acta Astronautica* 64.9 (2009), pp. 988–1005. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2008.12.006.