Compressing ROS Sensor and Geometry Messages with Draco

Thomas Wiemann[1,*], Felix Igelbrink[1], Sebastian Pütz[1,*], Malte Kleine Piening[1], Steffen Schupp[1], Steffen Hinderink[1], Juri Vana[1] and Joachim Hertzberg[1,2]

***Abstract*— Current development in sensor technology allows to capture highly detailed geometric 3D environment models fast and online with laser scanners or RGB-D cameras. However, in most rescue and service scenarios, the available bandwidth between operator and controlled robots is limited. In this paper, we integrate Google's geometry compression library Draco into ROS to reduce the required bandwidth when sending geometry data from a remote-controlled robot to a RViz instance running on the operator's computer. Besides a prototypical implementation, we present an in-depth analysis of the achievable compression rates, accuracy and resulting latency when using Draco for this purpose.**

I. INTRODUCTION

Generating realistic representations of unknown environments and sending the captured information back to the operator is key problem when operating a mobile robot remotely or semi-autonomously. When such operation takes place in unknown environments with no accessible communication infrastructure, available band width becomes an issue. Today, mobile robots equipped with mobile laser scanners or RGB-D cameras are capable of generating realistic environment models in real time on board. A well known example for this is Kinect Fusion and its extensions [1], [2] that are able to generate highly realistic 3D polygon models with textures, even of large scale environments [3]. Besides these RGB-D based approaches, robots equipped with high resolution laser scanners can produce even more accurate environment models at the cost of gathering gigabytes of data. Our own previous work concentrated on reconstructing textured polygonal models from such massive data sets to reduce the amount of data while keeping geometric precision [4], [5]. However, these methods concentrate on reducing the number of triangles in the generated models without taking the encoding of the captured data into account.

In this paper, we present an approach to integrate the open source geometry compression library Draco [6] developed by Google into a set of ROS-based tools to compress 3D environment models captured on a mobile robot. The sensor data is compressed on board the mobile robot and then encoded into special ROS messages. These compressed messages are sent from the robot to the ROS master controlling the robot. A conversion node then decodes the received geometry data into uncompressed ROS messages that can be visualized in RViz in real time. Since our software stack completely relies on either standard ROS geometry messages or our own 3D extensions and RViz plug-ins, our compression method can be used directly with all SLAM and reconstruction methods supporting these message types.

In the following, we describe the integration of the Draco compression library into the existing ROS infrastructure. We present how this library can be used to compress and de-compress large 3D models. To demonstrate the benefits of this approach, we evaluate our approach on different reference data sets that were chosen to cover a number of typical robotic application scenarios. With these reference data sets, we provide a detailed evaluation of encoding and decoding time as well as achievable compression rates within ROS' infrastructure.

II. RELATED WORK

Most previous work on creating compact, realistic environment models concentrates on the algorithmic aspects of computing polygonal surface reconstructions from 3D point cloud data. However, these lines of work do not take the efficient encoding of such models into account. Recent work on geometry compression has provided several methods to compress point cloud and mesh based models with high efficiency. For point cloud data, mostly octree based methods with sophisticated encoding are used. To reduce the amount of transmission data, the octree depth is limited to a determined value depending on the available bandwidth. For 3D geometries, existing approaches take connectivity coding and other metrics into account. Draco, for example, strongly relies on technique called Edgebreaker [7] to compress mesh data. A detailed survey of basic mesh compression technologies is given in [8]. As standards for point cloud and geometry compression are still under development [9], several freely available libraries exist. In [10], Doumanoglou et al. presented an extensive evaluation Google Draco [6], O3dgc [11], Corto [12], and OpenCTM [13] in the context of live streaming of 3D geometry data. They came up two interesting findings that inspired the work presented in this paper: First, Draco outperforms other libraries in scenarios with higher latency, and second, that Draco can also effectively include additional vertex attributes like normals or colors. In this work we exploited this extensibility to also include face attributes like materials into the encoded data stream, which to our knowledge has not been evaluated before.

* Work by Thomas Wiemann is supported by the German Federal Ministry of Education and Research in the project SoilAssist2 (Grant No.031B0684D). Sebastian Pütz is financed by a doctoral grant by Friedrich-Ebert-Stiftung. The DFKI Niedersachsen Lab (DFKI NI) is sponsored by the Ministry of Science and Culture of Lower Saxony and the VolkswagenStiftung. All this sponsoring is gratefully acknowledged

[1] University of Osnabrück, Institute of Computer Science, Knowledge Based Systems Group, 49090 Osnabrück, Germany firstname.lastname@uni-osnabrueck.de

[2]DFKI Niedersachsen Lab, 49090 Osnabrück, Germany joachim.hertzberg@dfki.de

The remainder of this paper is organized as follows: First, we present the integration of our draco-based geometry compression into existing ROS tools and the conversion of the compressed data streams to uncompressed ROS geometry messages. The experiments presented in Section IV concentrate on achievable compression rates, geometric precision of the decompressed messages and resulting latencies due to de- and encoding. The final section discusses the presented results and concludes.

III. INTEGRATING DRACO IN ROS

A. Mesh Tools

Utilizing 3D data in the ROS ecosystem has gained a lot of popularity in recent years. 3D point clouds are generally well supported within ROS through the *Pointcloud2*-Message, which is based on the data format of the Point Cloud Library (PCL) [14]. PCL implements a variety of point cloud based algorithms for robotic applications and is well connected with ROS' infrastructure making processing of 3D point cloud data possible in a number of different ROS packages.

In constrast to point cloud data, the support of 3D polygon data is very limited. An analogous infrastructure to the PCL-based 3D point cloud support is missing. RViz supports the rendering of polygonal robot models in URDF. These models can also be loaded in Gazebo as well as polygonal environment maps, but these representations are linked to static ressources (either to URDF models or to static world files in the user's file system). Sending complex polygonal environment models is currently not efficiently supported by ROS. Standardized messages to exchange 3D maps with annotations are currently missing.

To integrate such maps into ROS, we have developed the *Mesh Tools* software[1]. This package provides a set of tools allowing for the efficienct transmission of meshes within a ROS system as well as visualization in *RViz*. Additionally, processing on meshes is also possible through our *Las Vegas Reconstruction Toolkit* (LVR) [5][2] enabling the development of 3D mesh processing algorithms for robotic applications. Our *Mesh Tools* package provides ROS messages to define 3D geometries, transformation of triangle meshes in ROS' tf-framework, adding vertex and face annotations and tools to efficiently serialize such messages.[3].

The *Mesh Tools* package strictly separates between geometry definition and additional attributes. This separation was chosen deliberately to allow for dynamic on-demand computation of attributes and other data related to an exising mesh without having to transmit the geometry between different nodes every time an annotation changes. This allows to add certain attributes linked to the mesh's vertices or faces on demand. For example, roughness or trafficability information can be computed based on the geometry and added to the mesh on demand. Within this framework, geometry and attributes in different messages are then linked to each other using UUIDs. This format also allows to use meshes as a map representation for navigation with multiple information layers (e.g. costs) added on top.

[1] `https://github.com/uos/mesh_tools`

[2] `https://www.las-vegas.uni-osnabrueck.de`

[3] Details on this package can be found in our RosCon talk: `https://vimeo.com/293617680`

TABLE I: Mesh Tools Message types.

`MeshGeometry`	Basic Geometric Structure of the mesh with vertices, normals and faces.
`MeshVertexColors`	Vertex colors of an existing mesh.
`MeshVertexCosts`	Cost-Layer for a mesh as array of floats.
`MeshMaterial`	A Material, defined by a color and an optional texture ID.
`MeshTexture`	A texture (image) associated to a `MeshMaterial` by it's ID.
`MeshVertexTexCoords`	Texture coordinate referring to a pixel of a correspondinng `MeshTexture`
`MeshMaterials`	Combined materials and texture coordinates.

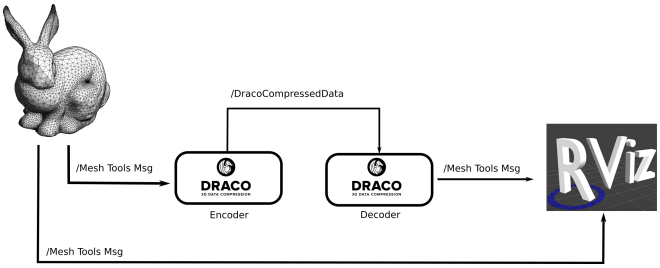


Fig. 1: Transparent compression of mesh data with our tools. When using Draco compressen, the Mesh Tools Messages are encoded and converted to a *DracoCompressedData* message. A receiver node decodes this message and converts it back into a Mesh Tools Message that can be parsed by subscribed nodes like RViz. This process is equivalent to sending the original message directly.

For every part contributing to the full mesh representation, a separate message type has been defined. These basic messages are listed in Table I. For this work, the *MeshGeometry*-Message is the most relevant. This message defines the basic geometric structure of a triangle mesh using and array of vectors representing vertices in $\mathbb{R}^3$, the corresponding normal vectors and an array of vertex indices defining the triangular structure. For evaluation, we use this set of messages for visualization in RViz and the newly implemented compression infrastructure to transparently compress and decompress geometries with Draco.

B. Encoding Geometry with DRACO

When attempting to transmit meshes and point clouds between several remote agents several problems become apparent. Most severely, such data can grow large and quickly overwhelm the bandwidth capabilities of a remote connection. In ROS, this results in high latencies or even dropped messages, due to internal limitations and overflowing buffers. To reduce the bandwidth demands, we integrated the open source compression library DRACO into the ROS message system, to enable compression of point clouds represented in *PointCloud2* messages as well as our own Mesh Tools messages while keeping enough accuracy for robotic applications after decompression.

Draco is a compression algorithm developed by Google [6], enabling the compression of meshes and point cloud data. As both data representations are different in nature, it does not rely on a single compression algorithm but uses multiple different techniques to optimally compress both representations regarding compression ratio, decoding speed and discretization losses. It therefore delivers superior performance when compared to general purpose algorithms like, e.g., gzip due to this specialization.

For point cloud data, Draco mainly relies on an order-optimized encoding by rearranging the points using a kd-tree. Positional data is discretized by a configurable number of quantization bits. While this will naturally result in the loss of spatial resolution, it can be fine tuned regarding requirements for accuracy or visual quality. Draco also supports the compression of arbitrary point attributes, making it well suited for heterogenous data. To compress mesh topology, Draco relies on the Edgebreaker algorithm [7]. Edgebreaker tries to encode a mesh in the form of a spiral, encoding the connectivty of each triangular face in a string while keeping track of the already visited vertices and faces. These strings are then compressed separately by the library.

Although Draco allows to fine-tune many internal parameters, the the compression is mainly influenced by the number of quantization bits used for compression (minimum value is 1 and maximum is 31). They can be set individually for each vertex attribute and allow to handle each attribute differently depending on the required precision. Additionally, a speed setting may be defined, which enables the user to adjust the ratio of compression and decompression time and the expected compression ratio. On compression time, the library then chooses the optimal composition of compression features based on these requirements. This parameterization allows a very fine grained tuning of the compression between optimal precision as well as fast encoding and decoding speed on demand. For our puposes, we stuck with the automatic compression settings and evaluated the quantization settings, since this parameter had the most significant influence on compression rates and geometric quality.

C. ROS Integration

The integration of Draco within the ROS message system should be as transparent as possible without requiring any significant conversion effort to enable a drop-in replacement of the existing uncompressed messages as shown in Fig. 1. Here, two possible paths of publishing a mesh geometry are sketched. In the upper pipeline, the mesh geometry is first sent to our Draco encoder node. This nodes converts the incoming message from the Mesh Tools message into Draco's internal data structures, taking the specified compression parameters into account. This encoder node then generates a byte stream that is capsuled in a *DracoCompressedData* message. Currently this `DracoCompressedData`-message consists only of a byte-array storing the encoded draco data. Additional metadata about the encoded data, compression settings, etc. may be added in future work. Such a message can in principle received by all nodes that can decode such streams. To make these Draco streams compatible with existing nodes that support the Mesh Tools and PointCloud2 messages, we also implemented a decoder node that takes the Draco stream and converts it back to the original uncompressed messages. Both nodes can be parametrized with the desired compression ratio and accuracy. ROS handles the conversion between these wrapper classes and the `DracoCompressedData`-messages automatically using specicalized serialization traits. Therefore no additional work beside adding these wrappers is required to benefit from the compression. On decoding time, the wrappers generate the previously specified messages again for further processing.

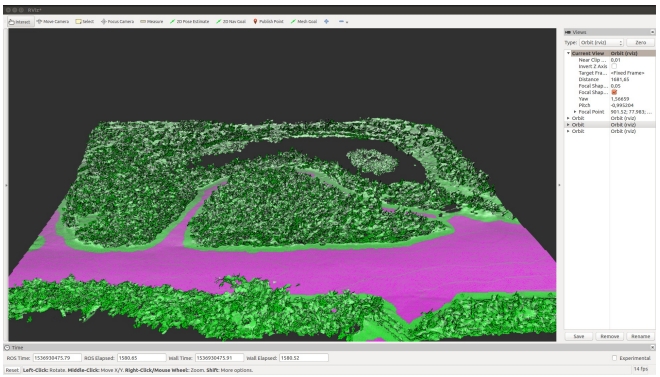


Fig. 2: Vizualization of an annotated triangle mesh in RViz

D. Adding Face Attributes for Meshes

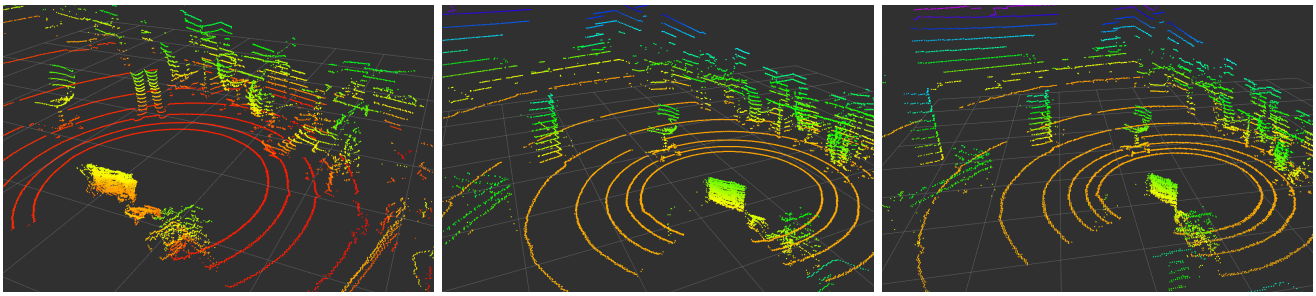
One major drawback when using Draco with triangle meshes is that the library does not support the encoding of per-face attributes directly. However for many applications including face-based information like materials and textures is benefitical. To support such attributes, we implemented a work-around in our tools, that is often used in the Draco community. For all faces, we create a set of additional "virtual vertices" that are included in the encoded vertex buffer. Knowing the number of attributes per face, an attribute array encoded as vertex coordinates can be added to the encoded stream. Knowing the format of these additional attributes, the encoded information can be reconstructed and stored in the published mesh attribute messages.

E. Mesh Visualization for RViz

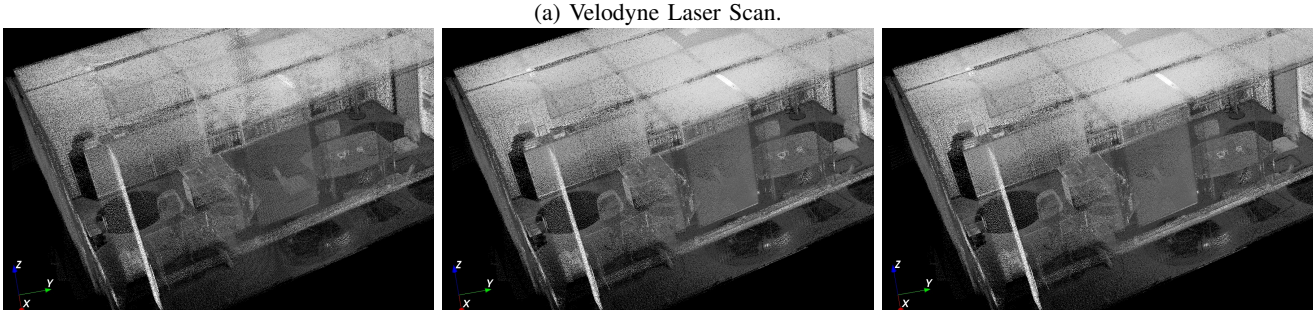
RViz does not natively support mesh messages for rendering and user interaction. For this purpose, we implemented a set of RViz plugins. An example of such an rendering is presented in Fig. 2. Here, a point cloud received from a PointCloud2 message is rendered together with an annotated triangle mesh rendered in wireframe mode. The colors within the mesh indicate a trafficability index, that estimates the roughness of the representated surfaces.

IV. EVALUATION

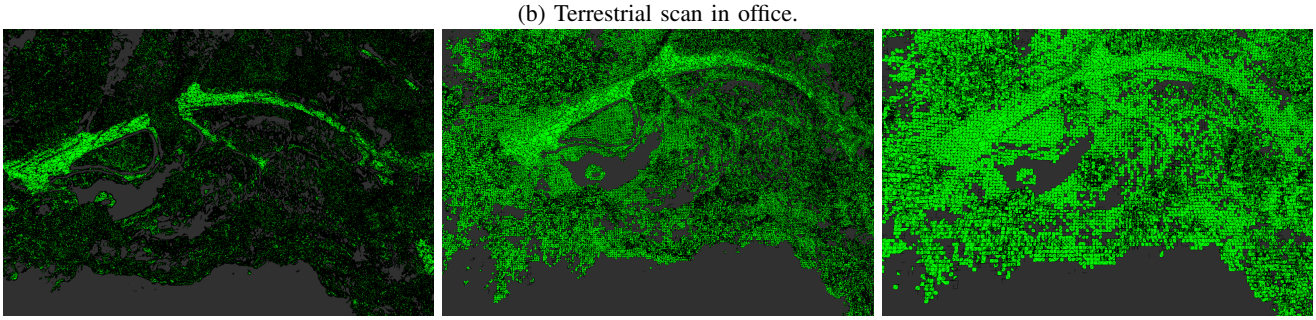
To evaluate the benefits of integrating Draco-based compression into ROS's infrastructure, we evaluated our implementation on triangle meshes and point clouds taken with



(a) Velodyne Laser Scan.



(b) Terrestrial scan in office.



(c) Triangle mesh reconstruction of the Botanical Garden scan.

Fig. 3: Lossy compression of point clouds and meshes using Draco for Velodyne laser scans (a) and a reconstruction based on a terrestrial laser scan taken in the Botanical Garden at the University of Osnabrck (c) with quantization levels of 25, 15 and 10 (left to right).

a terrestrial laser scanner (Riegl VZ 400i) and a Velodyne Puck VLP-16 line scanner. To address the inherently varying structure of different environments, we selected a set of reference data sets that were collected in structured indoor environments as well as cluttered outdoor scenes. All experiments were performed on a Intel Core i5-4690 CPU @ 3.50GHz with 8 GB RAM using Draco version 1.3.5.

The first mesh data set is a polygonal reconstruction from a terrestrial laser scan taken in the Botanical Garden at the University of Osnabrück. The mesh consists of 7.5 million vertices and 11 million triangles. The orginal binary PLY file has a size of 230 MB. This data set was selected to represent a polygonal model of a cluttered scene without many planar parts. In contrast to that, the second polygonal data set contains the facade of a building. It is intended to represent more structured environments. This mesh consists of 2.5 million vertices, 3.8 million triangles and uses 113 MB in PLY representation.

Besides mesh data, we are also interested in Draco's performance in point cloud compression. To cover different typical types of point cloud data, we also evaluated point clouds from the terrestrial scanner and the Velodyne. The Velodyne scan consists of 57.600 points and displays noise in the order of magnitude of 3 cm. The reference scan was taken in our robot lab to represent cluttered indoor environments. The terrestrial scans were taken in an empty office, which mostly features planar surfaces. It consists of 5 million points and uses 127 MB in PLY format, including point normals and intensity values. The second terrestrial scan was taken at the market place at Bremen, Germany. It also features mostly planar features with a medium level of clutter resulting from driving cars, passengers and vegetation. It consists of 14 million points and is 327 MB in size, also including point normals.

In our evaluation, we analyzed the resulting compression rates, encoding and decoding times as well as geometric precision using different compression levels. The general compression level can varied between 0 und 10. In the following experiments we always used the highest compression preset to achieve maximum data reduction. Besides this high level setting, the main parameter that influences data size and geometric precision is the number of quantization bits.

TABLE II: Mean distance errors between original and compressed data and resulting Draco stream sizes for different quantization levels as reported by Cloud Compare.

Dataset	Building		Botanical		Office		Bremen		Velodyne	
Type	Mesh		Mesh		Point cloud		Point cloud		Point cloud	
Quantization	Dst. [m]	Size [kb]	Dst. [m]	Size [kb]	Dst. [m]	Size [kb]	Dst. [m]	Size [kb]	Dst. [m]	Size [kb]
Off	0.000000	70 278	0.000000	98 685	0.000000	59 000	0.000000	171 296	0.000000	301
Q5	1.300922	38 848	3.699721	3 343	0.138391	8	0.021085	4	0.194819	1
Q10	0.007919	40 698	0.030072	5 914	0.004224	1 677	0.006404	974	0.005257	22
Q15	0.000004	45 253	0.001503	17 113	0.000132	9 969	0.000200	20 932	0.000177	69
Q20	0.000273	50 626	0.000012	32 412	0.000000	18 949	0.000006	43 377	0.000006	116
Q25	0.000199	55 635	0.000056	47 436	0.000000	27 976	0.000001	65 820	0.000001	163

TABLE III: Encoding and decoding times for the selected reference data sets.

Dataset	Building		Botanical		Office		Bremen		Velodyne	
Quantization	enc. [ms]	dec. [ms]	enc. [ms]	dec. [ms]	enc. [ms]	dec. [ms]	enc. [ms]	dec. [ms]	enc. [ms]	dec. [ms]
Q5	9972.00	2367.00	24091.00	7076.00	719.84	706.78	1828.70	2043.34	3.41	3.97
Q10	9980.00	2419.00	23931.00	7082.00	1239.27	912.44	2470.75	2176.43	6.88	5.37
Q15	9964.00	2643.00	24147.00	7854.00	1439.83	1060.80	3813.87	2953.39	7.96	5.95
Q20	10210.00	2567.00	24649.00	7445.00	1463.05	1126.32	3964.70	3153.21	7.92	6.33
Q25	10427.00	3107.00	25106.00	7762.00	1521.66	1188.27	4171.64	3335.37	8.12	6.62

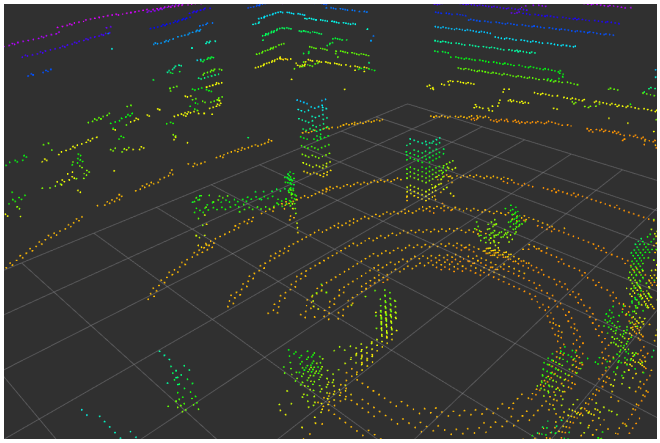


Fig. 4: Discretization errors in a velodyne scan (8 quantization bits).

Here, we varied the setting between a minimum of 5 bits and maximum 25 bits.

Qualitative results for different quantization levels are shown in Fig. 3a. The top row shows the compression of the Velodne scan using 25 , 15 and 10 quantization bits (left to right). In this data set, there are only few visible artifacts arinsing from the chosen compression. In the right picture, a minor discretization effect is visible. This effect becomes more apparent, when the number of quantization bits is further reduced, cf. Fig 4. For polgonal representations, quantization errors effect the not only the positions of the encoded vertices, but also the topology of the meshes, as depicted in Fig. 3c. For high quantization (25 Bits) the structure of the presented mesh is not visibly affected, but with reduced number of available bits, the quality of the representation drastically decreases.

To quantify the resulting geometric errors, we analyzed the metrical distances between the original, uncompressed data and the decoded messages using CloudCompare [15]. The results of this experiments together with the resulting *DracoCompressedData* message sizes are shown in Tab. II. For a setting of 25 bits, the geometric error is negligible for meshes and point clouds, while the algorithm already achieves a high compression of about 21% and 52% for the meshes and 52% respectively 61% for the point clouds. With higher compression levels, the errors increase as expected, especially for the meshes. The relatively small errors for the point clouds can be explained by the used measurement algorithm. Here, the distance of each point in the compressed data set to the original data set is computed. Since the point coordinates are discretized due to quantization, the probability to match a data point from the higher resolution input data is quite high, resulting in relatively small distance errors.

In the office environment, the reported overall errors were lower than in the more cluttered Bremen data set (cf. Tab. II). Similarly, the error on the mesh data sets increased compared to the structured building data sets. For cluttered environments, the compression should therefore be reduced to obtain acceptable results.

To conclude, for quantizations up to 15, the distance error is still acceptable although the number of possible point positions decreases, resulting in a higher possibility of points being represented at the same coordinates. These are not filtered out by the Draco library. For meshes, the distance errors increase more significantly with less than 15

quantization bits and the measured compression is not as high as in the point clouds.

Another important metric for practical applications is the resulting latency. When intregrating Draco in ROS' message-based infrastructure, there are times needed for encoding and decoding a Draco stream. To asses this aspect, we measured the respective times for the chosen reference data sets. The results are presented in Tab. III for different quantization levels. Generally, the recorded encoding times tend to increase a little towards higher quantization levels, but these effects are negligble. Also, decoding seems to be significantly faster for meshes. For point clouds, encoding is still slower, especially for high quantization. Rather than looking at these tendencies, putting the absolute numbers into context, shows that – although data compressen with Draco takes some time – the reported latencies are far below the recording times of the sensors. The Velodyne, for example, operates with a scanning frequency of up to 20 Hz. The reported encoding times are between 3 and 8 ms seconds, which allows to encode each scan in real time. A similar argumentation can be given for the terrestrial scans, where the scanning time is in the order of magnitude of several minutes.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented an infrastructure to compress triangle meshes and point clouds using Draco in ROS. We provided the message definitions necessary to send and render mesh-based 3D geometries as well as adaptor nodes for encoding and decoding these messages with Draco to reduce the size of the sent messages. We provided an extensive evaluation of this approach with respect to achievable compression rates, time consumption for encoding and decoding the data streams and geometric precision. The presented examples prove that our toolchain is able to reduce the amount of sent data in real time with in terms of recording speed of the used sensors on a standard laptop. Even when using high quantization settings, the size of the sent models can be reduced significantly without losing geometric precision. With higher compression, the geometric precision decreases fast, especially for meshes.

In our evaluation we also covered different types of environments. The presented results show, that the structure of the environment may have influence on the achievable precision, especially in cluttered environments.

However, depending on the application context, using our Draco integration might be a good approach to decrease network traffic when bandwith is limited, especially when dealing with point cloud data. With 25 quantization bits, the compression is quasi lossless while reducing the amount of transmitted data significantly. For point clouds, a setting close to 15 might even by acceptable, since the quantization error is still below 1 mm. Here, the amount of sent data can be reduced to 12% of the initial size for the Bremen data set and 17% in the office data set.

Within this evaluation, we always assumed good network conditions when sending the data. In future work, it would be interesting to evaluate how robust the generated streams are in regard to unstable conditions with packet losses and high latencies. Furthermore, our implementation of the Draco compression only supports annotated meshes. The transmission of textures within the compressed stream is currently not supported. The presented messages and RViz plugins however support sending and rendering of textures, but currently, the corresponding images have to be sent in a dedicated data stream.

REFERENCES

[1] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 IEEE International Symposium on Mixed and Augmented Reality*. IEEE, 2011, pp. 127–136.

[2] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Real-time Large Scale Dense RGB-D SLAM with Volumetric Fusion," in *International Journal of Robotics Research Special Issue on Robot Vision*, 2014.

[3] A. Mock, T. Wiemann, D. Borrmann, T. Igelbrink, and J. Hertzberg, "Real time texture generation in optimized large-scale polygon meshes with kinectfusion," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, June 2016, pp. 1–7.

[4] T. Wiemann, H. Annuth, K. Lingemann, and J. Hertzberg, "An extended evaluation of open source surface reconstruction software for robotic applications," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 1, pp. 149–170, 2015.

[5] T. Wiemann, I. Mitschke, A. Mock, and J. Hertzberg, "Surface reconstruction from arbitrarily large point clouds," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, Jan 2018, pp. 278–281.

[6] "Google Draco," accessed March 26, 2019. [Online]. Available: https://github.com/google/draco

[7] J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47–61, Jan. 1999.

[8] J. Peng, C.-S. Kim, and C.-C. J. Kuo, "Technologies for 3d mesh compression: A survey," *Journal of Visual Communication and Image Representation*, vol. 16, no. 6, pp. 688–733, 2005.

[9] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokua, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko, "Emerging mpeg standards for point cloud compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, March 2019.

[10] A. Doumanoglou, P. Drakoulis, N. Zioulis, D. Zarpalas, and P. Daras, "Benchmarking open-source static 3d mesh codecs for immersive media interactive live streaming," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 190–203, March 2019.

[11] "Open 3d Graphics Compression (o3dgc)," accessed March 26, 2019. [Online]. Available: https://github.com/amd/rest3d/tree/master/server/o3dgc

[12] "Corto," accessed March 26, 2019. [Online]. Available: https://github.com/cnr-isti-vclab/corto

[13] "Openctm," accessed March 26, 2019. [Online]. Available: http://openctm.sourceforge.net/

[14] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4.

[15] D. Girardeau-Montaut, M. Roux, R. Marc, and G. Thibault, "Change detection on points cloud data acquired with a ground laser scanner," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, no. part 3, p. W19, 2005.