Low Bit-rate 3D Feature Descriptors for Depth Data from Kinect-style Sensors

Sai Manoj Prakhya[†], Weisi Lin[†], Vijay Chandrasekhar[†,*], Bingbing Liu[*] and Jie Lin[*]

[†]*School of Computer Science and Engineering, Nanyang Technological University. Emails:* `saimanoj001@e.ntu.edu.sg` *and* `wslin@ntu.edu.sg`
[*]*Institute for Infocomm Research, A*STAR, Singapore. Emails:* {`vijay, bliu, lin-j`}*@i2r.a-star.edu.sg*

Abstract

In applications that require an input point cloud to be matched with a set of database point clouds present on a remote server, it is preferable to compress and transfer 3D feature descriptors online, rather than compressing and transferring the whole input point cloud. This is because the former would require much lesser bandwidth and does not require feature extraction on the server. Existing real valued 3D feature descriptors that offer good keypoint matching performance require higher bandwidth for their transfer over the network. On the other hand, the existing binary 3D feature descriptor requires relatively less bandwidth but offers reduced keypoint matching performance. In this paper, we propose to employ lattice quantization to efficiently compress 3D feature descriptors. These compressed 3D feature descriptors can be directly matched in compressed domain without any need for decompression, hence drastically reducing the memory footprint and computational requirements. We also propose double stage lattice quantization to achieve even more compression in the case of SHOT 3D feature descriptor. We provide a spectrum of possible bit rates and achievable keypoint matching performance for three state-of-the-art 3D feature descriptors. Experimental evaluation on publicly available benchmark dataset highlights that the compressed 3D feature descriptors require much lesser bandwidth and yet offer good keypoint matching performance. The source code is made publicly available for the benefit of the community.

Keywords:
3D Feature Descriptors, Compression, Lattice Quantization, 3D Keypoint Matching

1. Introduction

With the availability of sensors such as Microsoft Kinect, Intel RealSense Camera and Structure sensor [1], 3D depth data acquisition has become affordable compared to the expensive laser scanners. These low cost depth sensors are being heavily employed in 3D perception applications spanning the domains of robotics and computer vision. Furthermore, the advent of hand-held mobile devices such as Google Tango, which are equipped with depth sensors, necessitate the development of applications that run with low memory, computational and bandwidth requirements.

Let us consider an application scenario, wherein a 3D point cloud captured with a hand-held mobile device has to be matched with a database of 3D point clouds present on a remote server. This is relevant in applications, in which a user, or an autonomous robot deployed in an unknown environment, needs to perform tasks such as 3D object recognition [2], retrieval [3], reconstruction [4], 3D point cloud search and registration [5, 6, 7], Simultaneous Localization and Mapping (SLAM) [8, 9] and loop closure, while interacting with a remote server. One of the most important steps in all the above mentioned applications, is to establish reliable 3D keypoint correspondences between a source and a target point cloud. 3D feature descriptors that encode the point distributions around the keypoints into a multidimensional vector, are the preferred choice to find reliable 3D keypoint correspondences between two arbitrarily oriented source and target point clouds.

It was observed in 2D image domain that compressing and transmitting 2D feature descriptors requires less bandwidth, has lower power consumption and offers better user experience than transmitting a downsampled JPEG image [10, 11]. Based on this observation, for the considered scenario of matching a source point cloud with a set of database point clouds, it is preferable to compress and transmit 3D feature descriptors, instead of compressing and transmitting the whole point cloud [12, 13]. There are three desirable characteristics while compressing and transmitting 3D feature descriptors:

- First, the compressed feature descriptors should be represented in less number of bits for efficient online transfer with low network latency.
- Second, the compressed 3D feature descriptors should offer high keypoint matching accuracy to retrieve accurate results from the server's database.
- Third, the compressed descriptors should be matched directly without decompression, hence lowering the memory and computational power requirements on the end-devices.

Contributions: In this work, we explore how low we can go in compressing real valued 3D feature descriptors via lattice quantization. We apply lattice quantization technique to compress three state-of-the-art 3D feature descriptors, namely, SHOT [14], RoPS [15] and FPFH [16] and present their achievable keypoint matching performance. We perform compressed domain matching, where the compressed feature descriptors are matched by querying a pre-computed look up table, which is significantly faster than the widely used kd-tree [17] based matching technique. We also propose a double stage lattice quantization to compress the SHOT feature descriptor even further. Based on the extensive evaluation on Bologna Kinect dataset, we provide a spectrum of possible bitrates and achievable keypoint matching performances of these three feature descriptors with some interesting findings. This can help in selecting the appropriate compressed 3D feature descriptor for an application, as per its bandwidth and performance requirements. The source code will be made publicly available at **https://sites.google.com/site/dslqshot/**

2. Related Work

2.1. Compression Techniques in 2D Domain

The process of standardizing compact 2D feature descriptors for visual search applications by MPEG [18, 11] has resulted in the development of various techniques to compress 2D feature descriptors, especially SIFT [19]. The final adopted technique in the MPEG standard is based on scalar quantization [20, 21, 18], which employs small linear transforms and ternary scalar quantization to compress SIFT feature descriptor. The linear transforms and the selection of a subset of transformed descriptor values to be transmitted, were tailored/optimized specifically for SIFT feature descriptor. It is not intuitive to apply those linear transforms to 3D feature descriptors and choosing only a subset of the transformed values, because of the fundamental difference in the construction principles of SIFT and considered 3D feature descriptors, RoPS [15], FPFH [16] and SHOT [14]. The standard also adopted a multi-stage vector quantization [22] from various submitted proposals based on vector quantization. These vector quantization methods require a codebook that is learned from visual vocabularies, which is difficult in 3D domain for two main reasons. First, there is no standard or pre-built vocabularies that are publicly available. Second, it is not clear on how to accommodate the differences that arise from point cloud density variations and types of depth data, such as 2.5D and 3D.

In contrast to the previous methods, i.e., handcrafted scalar quantization, or vector quantization that requires pre-built vocabularies, there is lattice quantization technique [23, 24], which directly compresses normalized histograms or probability distributions based on a fixed codebook without any need for clustering or learning. Lattice quantization has been applied on 2D feature descriptors [24] for low bit rate image retrieval. Firstly, lattice qunatization based compression has offered better performance when compared to Huffman and other tree based coding techniques [25, 23]. Moreover, Huffman tree based coding technique has a major drawback, which is that the number of possible Huffman trees increase drastically with the number of bins in the histogram, necessitating the need for large memory to store those codebooks. Secondly, in lattice quantization, the codebook or the quantization levels are fixed based on the initially set parameters, and this enables compressed domain matching. Compressed domain matching directly matches the compressed descriptors with few look up table queries and hence there is no need to decompress the transmitted descriptors. After considering these advantages, we employed lattice quantization to compress 3D feature descriptors in this work, to achieve greater compression while offering good keypoint matching performance.

2.2. Compressed 3D Feature Descriptors

There was only a single attempt [26], as per our knowledge, to compress 3D feature descriptors for efficient online transfer. The authors tried to employ lattice quantization (also known as *type coding*) to compress 3D feature descriptors [26], however, there were many shortcomings in their proposal. First, they need to decompress the transmitted (compressed) feature descriptors to established correspondences based on Euclidean distance metric. This means that the authors' proposal does not offer any reduction in memory footprint on both the source and the target devices and needs additional computational power for both compression and decompression. Second, they need to employ a specific software library [26] to handle extremely large integers of size 256 bits, which is not desirable to have, on mobile devices. Third, the highest compression that they have reported, required 528 bits to represent a 352 dimensional SHOT feature descriptor. Moreover, they complicate the lattice quantization process by appending the normalization factors at the end of the descriptors and it is not clear if their proposal can be applied to RoPS and FPFH feature descriptors as well. Finally, the computational requirements were not reported and it was only mentioned that the system requires significantly higher amount of time, which makes it is difficult to comprehend its practical use in real world applications.

We propose to employ lattice quantization to compress 3D feature descriptors in a much simpler way and overcome the above mentioned shortcomings of [26]. First, we perform compressed domain matching using a look up table and hence there is no need to decompress the compressed feature descriptors to find keypoint correspondences, thus offering significant reduction in memory footprint on end-devices. Second, our implementation does not require any additional software library, and we can compress a 352 dimensional SHOT feature descriptor to 176 bits, thus requiring much lesser bandwidth, memory and power requirements. Moreover, we improvise the conventional lattice quantizer by a double stage lattice quantization technique and show that it can offer even higher compression with little loss in performance. Also, we compress three state-of-the-art 3D feature descriptors, SHOT [14], RoPS [15] and FPFH [16] and perform rigorous evaluation on achievable bitrate and performance, and highlight the findings based on which, the user can choose the apt compressed 3D feature descriptor for their application. Finally, the compressed feature descriptors in our proposal require much lesser computational time than uncompressed original 3D feature descriptors, which makes them preferable even for on-device applications.

2.3. *Binary 3D Feature Descriptors*

B-SHOT [27], the very first binary 3D feature descriptor, is the most relevant publicly available low bitrate 3D feature descriptor for the considered applications scenario. In B-SHOT (Binary SHOT), the authors proposed an adaptive binarization technique to convert an N-dimensional real valued vector to an N-bit binary vector. Hence, B-SHOT requires 352 bits as SHOT is a 352 dimensional vector. In this work, we apply that adaptive binarization technique on RoPS and FPFH to create 135 bit B-RoPS and 33 bit B-FPFH binary 3D feature descriptors, and use them in experiments to show the performance improvement offered by lattice quantization.

2.4. *Real Valued 3D Feature Descriptors*

Guo et. al [7] have performed a comprehensive evaluation on various 3D feature descriptors and concluded that

- FPFH [16] feature descriptor is the best choice in memory constrained applications as it is comprised of 33 dimensions. However, its computational requirements increase drastically with the density of the point cloud [14].
- SHOT [14] is preferable in the case of dense point clouds as it is computationally efficient and offers high descriptiveness in feature descriptor matching.
- RoPS [15] offers stable performance across different datasets and hence is a good choice if the point cloud characteristics are unknown.

Hence, we create compressed variants of these three state-of-the-art 3D feature descriptors, SHOT, RoPS and FPFH using the proposed technique and evaluate their performance.

3. Lattice Quantization

Scalar quantization can be seen as a function that maps the set of all real values to a finite set of output values, which forms the codebook. The cardinality of the codebook and the distribution of the output values determine the resolution and the quality of the performed quantization. Vector quantization is the generalized extension of scalar quantization to multiple dimensions. While scalar quantization is extensively used in analog to digital conversion, vector quantization, at times is used for data compression.

Lattice quantization is a special case of vector quantization where the codebook is regular, i.e., the real valued input vectors are quantized to a set of output vectors that are equally spaced in the representation space. The cardinality of the codebook determines the number of bits required to represent each possible output vector. All the possible output vectors in codebook are encoded with a unique index, and this index alone is transmitted, hence drastically reducing the number of bits to represent a high dimensional vector.

3.1. *Lattice Quantization to Compress 3D Feature Descriptors*

Let us represent an N-dimensional 3D feature descriptor by $\mathbf{A} = \{a_1, a_2, ..., a_N\}$, which is formed by an aggregation of N/m number of m-dimensional sub-vectors. These m-dimensional sub-vectors represent various properties that are repeatedly calculated around a keypoint. For example, a 352-dimensional SHOT feature descriptor is an aggregation of 32 number of 11-dimensional histograms. Similarly, a 135-dimensional RoPS feature descriptor is made up of 27 number of 5-dimensional sub-vectors and a 33-dimensional FPFH feature descriptor is made of 3 number of 11-dimensional histograms. Let us represent the repeatable m dimensional sub-vector by $\mathbf{B} = \{b_1, b_2, ..., b_m\}$ and N/m such repeatable sub-vectors combine to form an N dimensional 3D feature descriptor $\mathbf{A}$, where $\mathbf{A} = \{\mathbf{B}_1, \mathbf{B}_2, ..., \mathbf{B}_{N/m}\}$.

In lattice quantization, a single m-dimensional real valued sub-vector $\mathbf{B}$, is quantized and mapped to a specific vector in the codebook. Then a unique index representing the mapped vector in the codebook is transmitted over the network. Existing works [23, 24] on compressing 2D feature descriptors claim that, it is ideal to quantize and transmit a single sub-vector $\mathbf{B}$ at a time, which would result in transmission of $\frac{N}{m}$ indices for every N-dimensional feature descriptor. It was reported that there was no improvement in the performance when joint-histogram compression was employed to compress the 2D HOG descriptor [24]. In contrast to that, we have found that, in the case of 3D SHOT feature descriptor, a set of consecutive histograms can be compressed simultaneously, and this offers both higher compression and performance gain, when compared to single histogram compression. For example, we had quantized two consecutive sub-vectors at a time, which would require transmission of only $\frac{N}{2m}$ indices for every N-dimensional descriptor, and this has offered better performance while requiring less number of bits compared to single histogram compression, as shown in Experiment 1 later on.

Therefore, the parameter m may not be limited to the length of a single repeatable sub-vector, and can be extended to accommodate consecutive sub-vectors, which results in much higher compression. However, m can only take a limited set of values for which $\frac{N}{m}$ is always a natural number. Another important parameter in lattice quantization is n, which determines the number of quantization levels in each dimension of the lattice. With an increase in n, the resolution of the quantized vector increases and makes them more distinctive. A codebook with high resolution quantized vectors requires more number of indices to encode them, resulting in greater number of bits to be transmitted per index.

Formally, a lattice quantization process initiated with m and n parameters, involves the construction of a lattice, where each point in the lattice is a m-dimensional probability distribution $\mathbf{Q} = \{q_1, q_2, .., q_m\}$, and each quantity q_i has n quantization levels i.e.,

$$q_i = \frac{c_i}{n}, \; where \; \{c_i, n\} \in Z_+ \; and \; \sum_i c_i = n \qquad (1)$$

Example: A lattice constructed with parameters $m = 5$ and $n = 3$ consists of lattice points that span the space of all possible five dimensional probability distributions, where each dimension can have three possible states $\{0.33, 0.66, 1\}$, other than zero (from Eqn. 1). Some points from this lattice are $\{0.66, 0.33, 0, 0, 0\}$, $\{0, 0.33, 0.33, 0, 0.33\}$, and $\{0, 0, 0, 0, 1\}$.

The constructed lattice is referred to as A_{m-1} lattice and this technique is also referred to as lattice coding or type coding or A_{m-1} lattice quantization in previous works [23, 24, 28]. The number of lattice points in the constructed lattice are given by the multiset coefficient:

$$\left(\!\binom{m}{n}\!\right) = \binom{n+m-1}{m-1} = \frac{(n+m-1)!}{(m-1)!n!} \tag{2}$$

Every input m-dimensional sub-vector $\mathbf{B}$, where $\mathbf{B} = \{b_1, b_2, ..., b_m\}$ is normalized to form a probability distribution $\bar{\mathbf{B}} = \{\bar{b_1}, \bar{b_2}, .., \bar{b_m}\}$, where $\bar{b_i} = b_i / \sum b_i$. If the input sub-vector $\mathbf{B}$ has any negative values, then the largest of those negative values can be added to each element of $\mathbf{B}$, to make it non-negative and then normalization should be performed. After normalization, the resultant probability distribution $\bar{\mathbf{B}}$ is quantized to the closest lattice point $\mathbf{Q} = \{q_1, q_2, .., q_m\}$ in the constructed lattice, based on the steps mentioned below in Algorithm 1. Then index of the mapped lattice point $\mathbf{Q} = \{q_1, q_2, .., q_m\}$ is calculated based on lexicographic enumeration. Finally, $\log_2 \left(\!\binom{m}{n}\!\right)$ bits (from Eqn. 2) are required to represent each index, with fixed length coding.

Algorithm 1: To find the nearest lattice point

We now present the algorithm [28, 24] to quantize a sub-vector $\mathbf{B} = \{b_1, b_2, .., b_m\}$ to the nearest lattice point $\mathbf{Q} = \{q_1, q_2, .., q_m\}$ in the lattice constructed with parameters m and n, which represent the dimension of the input vector and the number of quantization levels in each dimension respectively.

1. Normalize the input sub-vector $\mathbf{B}$ to form a probability distribution $\bar{\mathbf{B}} = \{\bar{b_1}, \bar{b_2}, .., \bar{b_m}\}$, where $\bar{b_i} = b_i / \sum b_i$.
2. Compute $\bar{c_i}$ for $(i = 1, 2, ..., m)$, where $\bar{c_i} = \left\lfloor n\bar{b_i} + \frac{1}{2} \right\rfloor$ and $n' = \sum_i \bar{c_i}$.
3. If $n' = n$, then the computed $\bar{c_i}$ values form the corresponding q_i components of the quantized distribution $\mathbf{Q}$, which is the nearest lattice point.
4. If $n' \neq n$, residual errors e_i are computed, where $e_i = \bar{c_i} - n\bar{b_i}$, and e_i are sorted in ascending order, i.e., $-\frac{1}{2} \leq e_{j_1} \leq e_{j_2} \leq ... \leq e_{j_m} \leq \frac{1}{2}$.
5. If $\delta > 0$, where $\delta = n' - n$, then δ values of $\bar{c_i}$ with largest errors are decremented, and the corresponding q_i components of the quantized distribution $\mathbf{Q}$ are estimated as
$$q_{j_i} = \begin{cases} \bar{c}_{j_i} & j = 1, 2, ..., m - \delta - 1, \\ \bar{c}_{j_i} - 1 & i = m - d, ..., m. \end{cases}$$
6. Else if $\delta < 0$, then $|\delta|$ values of $\bar{k_i}$ with smallest errors are incremented, and the corresponding q_i components of the quantized distribution $\mathbf{Q}$ are estimated as
$$q_{j_i} = \begin{cases} \bar{c}_{j_i} + 1 & i = 1, 2, ..., |\delta|, \\ \bar{c}_{j_i} & i = |\delta| + 1, ..., m. \end{cases}$$

Enumeration: Calculating the unique index

The next step is to find a unique index α that corresponds to the lattice point $\mathbf{Q} = \{q_1, q_2, .., q_m\}$ in the constructed lattice with m and n. A lexicographic enumeration [29, 24, 28] technique is employed to find the unique index α as:

$$\alpha(\mathbf{Q}) = \sum_{j=1}^{n-2} \sum_{i=0}^{q_j - 1} \left(\!\binom{m-j}{n - i - \sum_{d=1}^{j-1} q_d}\!\right) + q_{n-1}$$

This unique index α is estimated for every sub-vector $\mathbf{B}$ of the input 3D feature descriptor $\mathbf{A}$. Please note that α is a whole number and it is bounded by the number of lattice points in the constructed lattice, as shown in Eqn. 2. The estimated indices α_k that correspond to $\mathbf{B}_k$, are concatenated to form the compressed feature descriptor corresponding to the original feature descriptor $\mathbf{A}$, where $k = 1, 2, ..., N/m$, N is the dimension of the 3D feature descriptor $\mathbf{A}$ and m represents the dimension of the sub-vector $\mathbf{B}_k$.

3.2. Compressed Domain Matching

The advantage of lexicographic enumeration [29] process is that, for a fixed lattice structure defined by m and n, there is a one to one correspondence between the enumerated indices α_k and all the lattice points Q_k. Based on this observation, one can pre-compute the Euclidean distances between all pairs of lattice points and store them in a look up table. The look up table (LUT) is a square matrix with each element representing the distance between the lattice points or the probability distributions that correspond to the indices of the LUT. The size of the LUT depends on the number of lattice points (Eqn. 2) in the constructed lattice, as defined by parameters m and n.

Finally, the distance between two feature descriptors $\mathbf{A}_\beta$ and $\mathbf{A}_\gamma$, whose compressed variants are represented by a set of indices $\{\beta_1, \beta_2, ..., \beta_{N/m}\}$ and $\{\gamma_1, \gamma_2, ..., \gamma_{N/m}\}$, can be easily calculated by accessing LUT as

$$Dist(\mathbf{A}_\beta, \mathbf{A}_\gamma) = \sum_{i=1}^{N/m} LUT(\beta_i, \gamma_i), \tag{3}$$

where N is the dimension of $\mathbf{A}_\beta$ and m is the dimension of sub-vector that is compressed via lattice quantization. Hence, the compressed feature descriptors, which are made up of the indices, can be directly matched in the compressed domain, without any need for decompression. Compressed domain matching is significantly faster than kd-tree based feature descriptor matching, because the former requires few look up table queries and their summation to calculate their relative distance.

3.3. Double Stage Lattice Quantization

Gaining inspiration from the success of multistage vector quantization [22] in effectively quantizing the input vector, we propose double stage lattice quantization in this work. Please note that, in general, vector quantization involves learning and storage of codebooks while lattice quantization does not require those steps. Based on our knowledge, this is the first work to

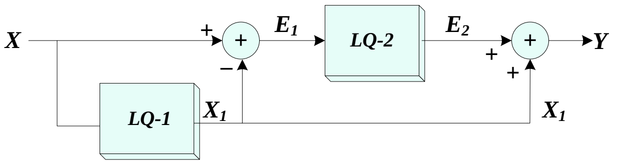

Figure 1: Flow chart of Double Stage Lattice Quantization (DSLQ)

propose double stage lattice quantization (DSLQ) and evaluate its feasibility.

The basic idea behind DSLQ is to divide the quantization process into two stages. In the first stage, lattice quantization is performed on the original input vector, while in the second stage, the error vector between the input vector and the first stage output is quantized. The flowchart of DSLQ is shown in Fig. 1. Lets represent the input vector by $\mathbf{X}$, which is quantized by the first stage lattice quantizer *LQ-1* and produces the quantized distribution $\mathbf{X}_1$. The second stage lattice quantizer *LQ-2* operates on the vector difference $\mathbf{E}_1$, where $\mathbf{E}_1 = \mathbf{X} - \mathbf{X}_1$, and generates the second stage quantized output $\mathbf{E}_2$. At last, the final quantized approximation of the input vector $\mathbf{X}$ is given by $\mathbf{Y}$, where $\mathbf{Y} = \mathbf{E}_2 + \mathbf{X}_1$. Hence, DSLQ uses two indices, one from the first stage and the other from the second stage to represent the input vector.

Multistage vector quantization, in general, requires more number of bits to represent a vector, however, the quantization error is lower when compared to single stage vector quantization. In particular, we show in this work that, with appropriate choice of lattice quantization parameters (m and n), DSLQ can further compress the SHOT feature descriptor when compared to single stage lattice quantization as shown in Experiment 2 later on.

Please note that DSLQ based compressed feature descriptors are not matched in the compressed domain as in single stage lattice quantizers. This is because of the fact that, a better approximation to input vector is achievable with DSLQ, only when the two quantized distributions from both stages are reconstructed by summation on the decoder side. Hence, as in multistage vector quantization, queries to two vector tables and their summation is enough to reconstruct the final output vector from the transmitted indices. Then the reconstructed descriptors can be matched efficiently with kd-tree, resulting in the same computational complexity as traditional 3D feature descriptor matching. Another way to match DSLQ descriptors is to capitalize on the lexicographic enumeration process to generate the quantized distributions, which would save the memory as there wont be any need to store the vector tables; however, it would require additional computations to generate quantized distributions from transmitted indices.

4. Experimental Setup

4.1. Dataset

Hand-held mobile devices such as Google Tango come with depth sensors that provide noisy depth data. In general, the applications that need to interact with a server, run on hand-held mobile devices or on autonomous robots equipped with sensors such as Microsoft Kinect and Asus Xtion Pro Live. Moreover, innumerable applications for indoor mobile robots and 3D object recognition [2, 30] are developed using low-cost Kinect-style depth sensors. Considering these, the Bologna Kinect dataset[1] [31] is the most relevant dataset as it inherently has the noise arising from these low cost portable depth sensors. The relevance and the need to develop 3D feature descriptors that work well on the Bologna Kinect dataset was also highlighted in the recent work on comprehensive evaluation of 3D feature descriptors [7]. This dataset has 17 scenes and 27 models. A model is the 3D point cloud of an object while a scene contains various object models in different orientations, occlusion and clutter. In total, there are 49 scene-model pairs in this dataset and they come with the ground truth 3D transformation between the considered model and the scene.

4.2. Evaluation Methodology

We evaluate and compare the keypoint matching performance of the compressed and uncompressed 3D feature descriptors based on Precision Recall curves [7, 14, 15]. To generate precision recall curves, uniform keypoints are detected on scene, and with the available groundtruth, these scene keypoints are transformed onto the model. By employing uniform keypoints rather than random keypoints [14], we account for keypoints arising from all kinds of neighbourhoods. Next, by detecting the keypoints on the scene first and then finding their corresponding keypoints that only lie on the model, ensures that we account for all possible false positives that arise from the background of the scene and other object models. Once the keypoints on the model and scene are detected, the considered 3D feature descriptors for evaluation are extracted around these keypoints. A uniform keypoint detector with a radius R_{kp} of $0.01m$ was used while all the considered feature descriptors are extracted with a support radius R_{fd} of $0.06m$.

For every model keypoint's feature descriptor, we find the first and the second nearest scene keypoint's feature descriptor. If the ratio of the distance to the first and the second nearest neighbour is less than or equal to a threshold δ then the first nearest neighbour is considered as a correspondence C_r. The threshold δ is varied from 0 to 1 and accordingly the number of correspondences $C_{r\delta}$ on the considered scene-model pair are estimated. Specifically, we set the values of $\delta = \{0.2, 0.4, 0.6, 0.75, 0.85, 0.925, 0.95, 0.975, 1.0\}$, calculated the correspondences $C_{r\delta}$ and then estimated the precision and recall.

The groundtruth matches G_m are the number of model keypoints because, for every model keypoint there lies a corresponding scene keypoint, which can be found via the available groundtruth. The true correspondences $True_{corr}$ are the correspondences from the set of $C_{r\delta}$ that align with the groundtruth. Specifically, a correspondence in $C_{r\delta}$ is considered as a true correspondence T_{corr}, if the matched scene keypoint lies within a threshold ϵ to the transformed model keypoint based on the

[1]The dataset is available at http://vision.deis.unibo.it/research/80-shot

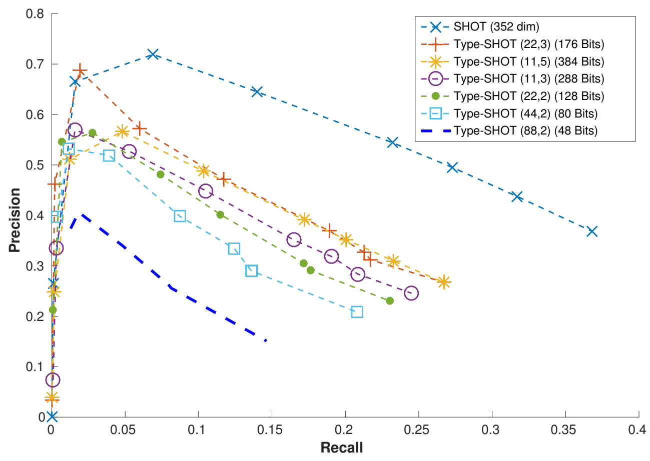

Figure 2: Performance evaluation of Type-SHOT feature descriptors with variation in m and n parameters as described in Experiment 1.

groundtruth. In all our experiments, ϵ is set to $0.01m$ because we uniformly detect keypoints with the same radius[2]. Finally, precision and recall are estimated as

$$Precision = \frac{True_{corr}}{C_{r\delta}} \qquad Recall = \frac{True_{corr}}{G_m}$$

where G_m represents the groundtruth matches G_m, $True_{corr}$ represents true correspondences as explained above, while $C_{r\delta}$ represents the established correspondences that pass the ratio test at a specific δ. By this way, the average precision and recall at a specific δ for 49 scene-model pairs in the Bologna Kinect dataset is calculated to evaluate the considered feature descriptors.

5. Compressing Various 3D Feature Descriptors

In this section, we compress three state-of-the-art 3D feature descriptors [7], SHOT [14], RoPS [15] and FPFH [16], and compare them based on the above mentioned evaluation methodology. As the amount of compression depends on the parameters m and n, here, we experimentally determine the parameters that offer best keypoint matching performance and report the achievable compression.

5.1. Compressing SHOT Feature Descriptor

Experiment 1: *Single Stage Lattice Quantization of SHOT* In this experiment, we compress SHOT feature descriptor by applying lattice quantization with varying m and n parameters. The m and n parameters determine the number of lattice points based on Eqn. 2, and hence, the number of bits required to represent the feature descriptor, i.e., achievable compression. As SHOT is an aggregation of 32 sub-vectors, where each subvector is a 11-bin histogram, we experimented by compressing each 11-bin histogram, i.e., m is set to 11 and varied the parameter n. Then we explored if the consecutive histograms can be combined and compressed together in a single instance, hence we experimented with $m = \{22, 44, 88\}$. After fixing the parameter m, the value of the parameter n is chosen in such a way that the number of lattice points in the constructed lattice is less than 8000 so that time and memory and computational requirements for lattice construction are tractable.

The results are shown in Fig. 2, where the compressed SHOT feature descriptors are shown in this representation, Type-SHOT (m,n) (ABC Bits), where (m,n) are the set m and n parameters while (ABC Bits) tells the number of bits required to represent the compressed descriptor via fixed-length coding.

It can be seen from Fig. 2 that Type-SHOT (22,3) (176 Bits) is the most preferable compressed variant as it achieves consistently high recall and precision, specifically when compared to the higher bitrate variants such as Type-SHOT (11,3) (288 Bits) and Type-SHOT (11,5) (384 Bits). Existing works in 2D feature descriptor compression [23, 24] have reported that there was no improvement in the performance with joint histogram compression. In contrary to that observation, it was found that joint histogram compression with two consecutive 11-bin histograms, i.e., $m = 22$ offers considerably better keypoint matching performance than Type-SHOT with $m = 11$ on this dataset. Another observation is that if we try to compress SHOT even further than 176 bits with single stage lattice quantization, the performance of Type-SHOT drops as can be seen with other low bit variants such as Type-SHOT (22,2) (128 Bits) and Type-SHOT (44,2) (80 Bits). Hence, with the use of single stage lattice quantization, SHOT feature descriptor with 176 bits offers comparatively the best keypoint matching performance.

Experiment 2: *Double Stage Lattice Quantization of SHOT* In this experiment, we apply double stage lattice quantization to compress SHOT feature descriptors even further, and compare the performance with single stage and B-SHOT binary 3D feature descriptor. We employed B-SHOT as it is the only publicly available low bitrate 3D feature descriptor. Please note that the original uncompressed SHOT descriptor requires 11,264 bits (based on IEEE floating point standard) while B-SHOT requires 352 bits.

The results are shown in Fig. 3. Double stage lattice quantization has two sets of m and n parameters. In Fig. 3, the double stage lattice quantized SHOT feature descriptors are represented as DSLQ-SHOT (m_1, n_1) (m_2, n_2) (ABC Bits), where (m_1, n_1) and (m_2, n_2) are the m and n parameters of the first and second stages respectively, and (ABC Bits) tells the number of required bits.

From Fig. 3, DSLQ-SHOT (44,2) (44,2) (160 Bits) offers the second best performance with Type-SHOT (22,3) (176 Bits) being the first. Moreover, DSLQ-SHOT (88,2) (44,2) (128 Bits) offered better performance than Type-SHOT (22,2) (128 Bits). Hence, at a budget of 128 bits per SHOT descriptor, DSLQ-SHOT is preferable over Type-SHOT, highlighting the effectiveness of double stage lattice quantization. It can be seen from Fig. 3 that B-SHOT with 352 bits offers similar performance as that of DSLQ-SHOT at 128 bits. Again, DSLQ-SHOT (88,2) (88,2) (96 Bits) offering similar performance as that of Type-SHOT (22,2) (128 Bits) highlights the effectiveness of double

[2]We provide the source code for the evaluation methodology in our project website (**https://sites.google.com/site/dslqshot/**) to reproduce the results.

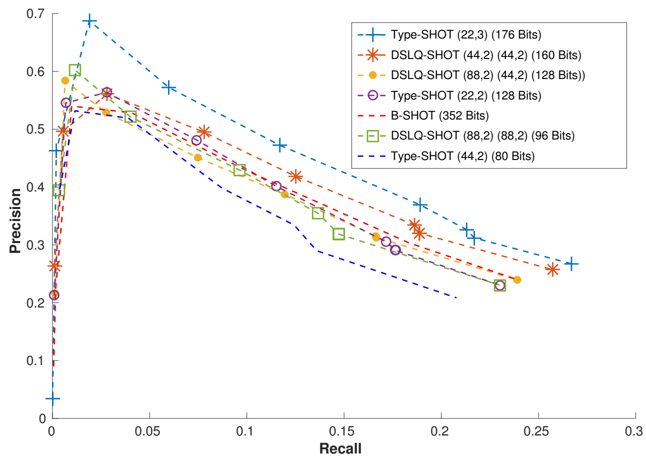

Figure 3: Performance evaluation of Type-SHOT and DSLQ-SHOT compressed descriptors as explained in Experiment 2.

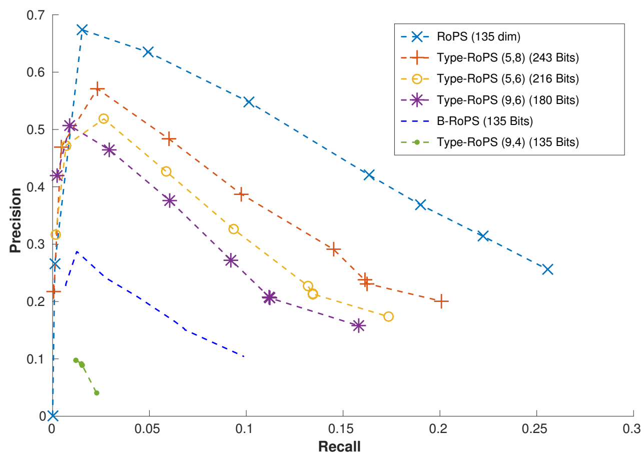

Figure 4: Performance evaluation of Type-RoPS feature descriptors with varying m and n parameters as explained in Experiment 3.

stage lattice quantization in encoding the extra information in less number of bits. It can be noticed that further compression of SHOT with single stage lattice quantization, i.e., Type-SHOT (44,2) (80 Bits) offers much lower performance because of the fact that performance drops at lower bitrate.

Double stage lattice quantization offers two reliable compressed SHOT variants that require 128 and 96 bits which offer better performance than Type-SHOT at 128 bits. While choosing the parameters m and n for DSLQ-SHOT, the parameter n was set to 2 for $m = 44$ *and* 88, because for $n > 2$, there is a drastic increase in both the lattice points and the size of the look up table, which needs to be stored for compressed domain matching of Type-SHOT descriptors. DSLQ-SHOT descriptors either require vector look up tables or decompression based on lexicographic enumeration to reconstruct the quantized vectors, which correspondingly require additional memory or computational time when compared to single stage lattice quantization. However, as this matching and decompression happens in the server, memory and computational requirements should not be a bottleneck. Hence, double stage lattice quantization can be employed to achieve further compression of SHOT while having a boost in the performance when compared to Type-SHOT at the same bitrate.

5.2. Compressing RoPS Feature Descriptor

Experiment 3: *Lattice Quantization of RoPS*
In this experiment, we compress RoPS feature descriptor with single stage lattice quantization. As RoPS descriptor is made up of five properties of a distribution matrix calculated 27 times, we first experimented by setting $m = 5$ and then varied n. Second, RoPS, unlike SHOT and FPFH, is not an aggregation of histograms, hence, we also tried setting m to factors of 135, i.e., $m = \{3, 5, 9, 15, 27, 45\}$ and then varied n. Hence, we report the results of the compressed RoPS variants with $m = \{5, 9\}$, however, setting $m \geq 15$, needed much higher quantization levels (n) to establish feature descriptor correspondences, and the lattice points in the constructed lattice were so high that they required larger memory to store the look up table for compressed domain matching, making them infeasible.

The results are shown in Fig. 4, where RoPS represents the original uncompressed feature descriptor that requires 4320 bits, B-RoPS that requires 135 bits is created by applying the adaptive quantization method proposed in B-SHOT [27] to RoPS feature descriptor, and Type-RoPS (m,n) (ABC Bits) represents a compressed variant of RoPS feature descriptor.

As can be seen from Fig. 4, Type-RoPS (5,8) (243 Bits) offered relatively better keypoint matching performance compared to other compressed RoPS variants. As the bitrate decreases, the performance also drops. However, it can be noted from the figure that at 135 bits, B-RoPS offered better performance than Type-RoPS (9,4) (135 Bits). This is because the number of quantization levels, i.e., $n = 4$ is less and do not capture the variations in the RoPS descriptor. For example, Type-RoPS (5,8) (243 Bits) has eight quantization levels with $n = 8$ and hence can capture the variations in the descriptor's values, resulting in high performing compressed RoPS variant. The reason behind this is that the RoPS descriptor is an aggregation of five statistical properties calculated over a matrix, which do not exhibit any resemblance with probability distributions. Hence, each value in RoPS descriptor can be very similar or close to each other, requiring higher precision/quantization levels to capture their variations and be discriminative from other compressed descriptors after normalization. From this experiment, it can be concluded that a single stage lattice quantized RoPS with 243 bits is the preferred compressed variant for RoPS descriptor.

The double stage lattice quantization (DSLQ) on RoPS descriptor did not offer more compression nor higher discriminative power. This is because RoPS descriptor is not made of histograms but statistical properties, which are similar to each other with less difference. Consequently, the normalization performed in the second stage in DSLQ renders the reconstructed RoPS descriptor to be less distinctive than a single stage lattice quantizer. The aim behind proposing DSLQ was to employ smaller lattice structures, so that less bits are required, and yet achieve good performance, which was not possible with the

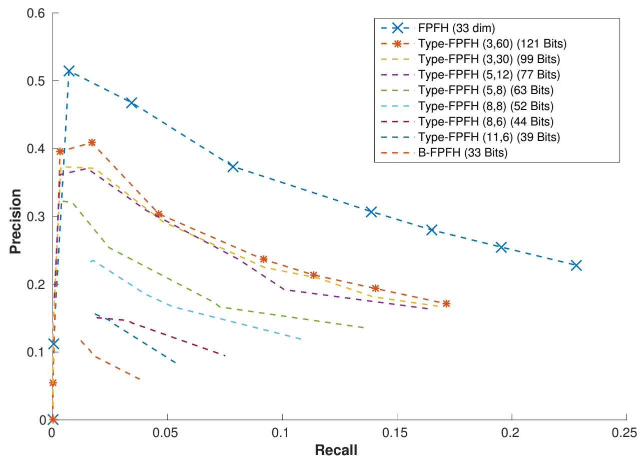

Figure 5: Performance evaluation of Type-FPFH feature descriptors with variation in m and n parameters as described in Experiment 5.

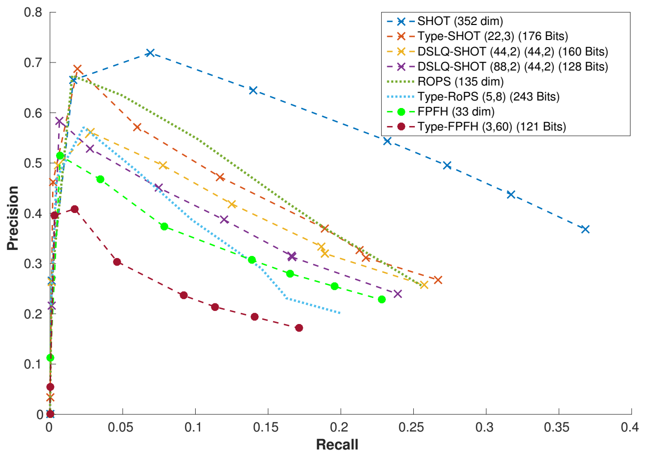

Figure 6: Comparison of the uncompressed and compressed variants of SHOT, RoPS and FPFH feature descriptors.

RoPS feature descriptor.

5.3. *Compressing FPFH Feature Descriptor*

Experiment 4: *Lattice Quantization of FPFH*
We compress FPFH descriptor with the single stage lattice quantization technique. FPFH, being 33 dimensional, a combination of 3 histograms with 11 bins per each histogram, gives only two options for m, i.e., $m = \{3, 11\}$. In order to have more compressed variants, we appended two zeros at the end of the 33 dimensional FPFH, which gave another possibility of $m = 5$. Similarly, we only considered 32 dimensions generating another possibility of $m = 8$. Therefore, we carried out experiments to evaluate the keypoint matching performance of the compressed variants of FPFH generated with $m = \{3, 5, 8, 11\}$ and a correspondingly set n value for which memory and computational requirements are tractable. B-FPFH, created by applying the adaptive quantization from B-SHOT [27] on to FPFH descriptor, is also considered for evaluation.

The results, as shown in Fig. 5, highlight that the performance of the compressed variants decrease accordingly with their bitrate. Type-FPFH (3,60) (121 Bits) offered high keypoint matching performance followed by Type-FPFH with 99, 77, 63 bits and so on. Type-FPFH (11,6) with 39 bits offered better performance that B-FPFH with 33 bits, as can be seen from Fig. 5. Please note that the uncompressed FPFH descriptor requires 1056 bits for its representation, which shows that Type-FPFH (121 Bits) offers a compression ratio of ≈ 8.72.

As seen in the compression of SHOT feature descriptor, double stage lattice quantizer was able to compress more than a single stage lattice quantizer, because it capitalized on compressing consecutive histograms with smaller lattice structures in both of its stages. This, consecutive histogram compression is not possible with FPFH descriptor as it is of 33 dimensional, with only three 11-bin histograms and a relatively smaller descriptor length, to see the advantage of double stage lattice quantization.

5.4. *Comparing the Compressed Variants of SHOT, RoPS and FPFH*

Here, in Fig. 6, we compare the best performing compressed variants of SHOT, RoPS and FPFH descriptors that were found in our previous experiments and the original uncompressed 3D feature descriptors. It can be seen from Fig. 6 that SHOT stands out as the best performing 3D feature descriptor. Next, for *recall* < 0.22, RoPS performs better than Type-SHOT (176 Bits), however, for *recall* > 0.22, Type-SHOT (176 Bits) outperforms RoPS. Also note that SHOT and RoPS require significantly higher memory requirements and feature descriptor matching time.

- Type-SHOT (176 Bits) is the best performing compressed descriptor and is also better than FPFH descriptor.

It can be said that DSLQ-SHOT (160 Bits) and DSLQ-SHOT (128 Bits) are the next best choices to have higher compression than 176 bits and yet achieve good keypoint matching performance. The uncompressed FPFP descriptor's performance comes next to DSLQ-SHOT (128 Bits). Type-RoPS (243 Bits) offered better performance than FPFH for *recall* < 0.13, however, for *recall* > 0.13 the uncompressed FPFH descriptor is better than Type-RoPS (243 Bits).

6. Computational Requirements

We present the computational time required to compute and match SHOT, B-SHOT and Type-SHOT (176 Bits) feature descriptors, as it offered highest keypoint matching performance. We considered a real world scenario where, detected keypoints are matched with feature descriptors and false positive matches are removed via RANSAC algorithm. Uniform keypoints were detected at $0.02m$, considered feature descriptors were extracted with a support size of $0.05m$ and the ransac inlier threshold was set to $0.04m$. The reference implementation for SHOT and RANSAC are from the Point Cloud Library [17].

We employed a CPU with an *Intel Xeon(R) CPU E5-1650 0 @ 3.20GHz* $\times$ *12* and 16 GB RAM with *UBUNTU 14.04* operating system for these experiments.

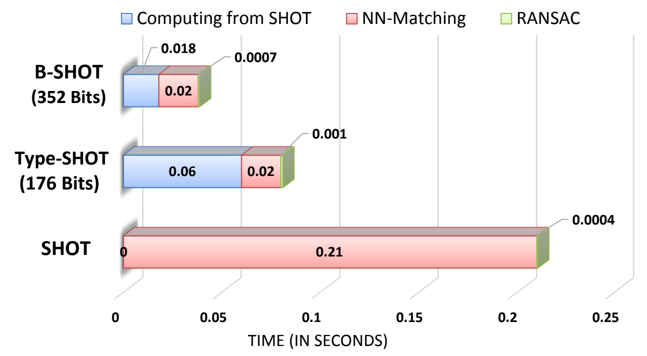

Figure 7: Comparing the computational time for B-SHOT, Type-SHOT (176 Bits) and SHOT feature descriptors.

In this experiment, first, SHOT feature descriptors are extracted around the detected keypoints. Second, B-SHOT and Type-SHOT feature descriptors are calculated from SHOT and the time taken is shown in Fig. 7 as '*Computing from SHOT*'. Third, reciprocal correspondences of SHOT are obtained using kdtree[3], while B-SHOT binary descriptors are matched via Hamming distance and Type-SHOT descriptors are matched in compressed domain. The corresponding time taken for matching is shown in Fig. 7 as '*NN-Matching*'. Finally, the taken by the RANSAC algorithm to find the true correspondences is shown in the figure as '*RANSAC*'. Please note that all the reported values are averaged across all 49 scene-model pairs in the Bologna Kinect dataset.

It is evident from Fig. 7 that B-SHOT is the fastest, followed by Type-SHOT and SHOT feature descriptors. A closer look at the figure indicates that the NN-matching time of both B-SHOT and Type-SHOT are equal and the difference arises only in computing the compressed variants. The fast *NN-matching* of Type-SHOT is justified by the fact that compressed domain matching involves only a few look up table queries and their summation to match two feature descriptors. Next, Type-SHOT takes more time for its creation from SHOT, when compared to B-SHOT, as it encodes much more relevant information through a relatively complex technique to achieve significantly higher keypoint matching performance. This experiment, highlights that Type-SHOT can also be employed in on-device applications to achieve faster matching with only a minimal loss in keypoint matching performance, when compared to original, uncompressed SHOT feature descriptor.

7. Conclusion

In this paper, we compressed three state-of-the-art 3D feature descriptors using lattice quantization and evaluated them on depth data acquired from Kinect-style sensors. More importantly, the compressed 3D feature descriptors can be directly matched in the compressed domain without any need for decompression, hence significantly reducing the memory footprint on the end-devices. We also proposed double stage lattice quantization to achieve even higher compression of SHOT 3D feature descriptor. It was found that the compressed SHOT feature descriptor at 176 bits offers best performance when compared to all other compressed variants and the uncompressed FPFH descriptor on the considered Bologna Kinect dataset. These compressed 3D feature descriptors have huge potential in applications such as 3D point cloud and object search, recognition and retrieval, as they, require significantly lower bandwidth for online transfer, are much faster in feature descriptor matching and have less memory footprint on end-devices.

[3] We employ pcl::registration::CorrespondenceEstimation class from Point Cloud Library (www.pointclouds.org) to estimate reciprocal correspondences, which inherently uses a kdtree for faster matching and retrieval.

[1] Structure Sensor.
URL `http://structure.io/`

[2] T. Faulhammer, A. Aldoma, M. Zillich, M. Vincze, Temporal Integration of Feature Correspondences for Enhanced Recognition in Cluttered and Dynamic Environments, in: Robotics and Automation (ICRA), 2015 IEEE International Conference on, 2015, pp. 3003–3009. doi:10.1109/ICRA.2015.7139611.

[3] M. A. Savelonas, I. Pratikakis, K. Sfikas, Partial 3D Object Retrieval combining Local Shape Descriptors with Global Fisher Vectors.

[4] J. Prankl, A. Aldoma, A. Svejda, M. Vincze, Rgb-d object modelling for object recognition and tracking, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE, 2015, pp. 96–103.

[5] Y. Guo, F. Sohel, M. Bennamoun, J. Wan, M. Lu, An accurate and robust range image registration algorithm for 3d object modeling, Multimedia, IEEE Transactions on 16 (5) (2014) 1377–1390.

[6] S. Prakhya, L. Bingbing, Y. Rui, W. Lin, A Closed-form Estimate of 3D ICP Covariance, in: Machine Vision Applications (MVA), 2015 14th IAPR International Conference on, 2015, pp. 526–529. doi:10.1109/MVA.2015.7153246.

[7] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, N. Kwok, A Comprehensive Performance Evaluation of 3D Local Feature Descriptors, International Journal of Computer Vision (2015) 1–24.

[8] F. Endres, J. Hess, J. Sturm, D. Cremers, W. Burgard, 3-D Mapping with an RGB-D Camera, Robotics, IEEE Transactions on 30 (1) (2014) 177–187. doi:10.1109/TRO.2013.2279412.

[9] S. M. Prakhya, B. Liu, W. Lin, U. Qayyum, Sparse Depth Odometry : 3D Keypoint based Pose Estimation from Dense Depth Data , in: Robotics and Automation (ICRA), 2015 IEEE International Conference on, 2015.

[10] L.-Y. Duan, J. Lin, J. Chen, T. Huang, W. Gao, Compact Descriptors for Visual Search, MultiMedia, IEEE 21 (3) (2014) 30–40. doi:10.1109/MMUL.2013.66.

[11] L.-Y. Duan, T. Huang, W. Gao, Overview of the MPEG CDVS Standard, in: Data Compression Conference (DCC), 2015, 2015, pp. 323–332. doi:10.1109/DCC.2015.72.

[12] V. Morell, S. Orts, M. Cazorla, J. Garcia-Rodriguez, Geometric 3D Point Cloud Compression, Pattern Recognition Letters 50 (2014) 55 – 62, depth Image Analysis. doi:http://dx.doi.org/10.1016/j.patrec.2014.05.016.

[13] J. Navarrete, V. Morell, M. Cazorla, D. Viejo, J. Garca-Rodrguez, S. Orts-Escolano, 3DCOMET: 3D Compression Methods Test Dataset, Robotics and Autonomous Systems (2015) – doi:http://dx.doi.org/10.1016/j.robot.2015.09.028.

[14] S. Salti, F. Tombari, L. Di Stefano, SHOT: Unique Signatures of Histograms for Surface and Texture Description, Computer Vision and Image Understanding 125 (2014) 251–264.

[15] Y. Guo, F. Sohel, M. Bennamoun, M. Lu, J. Wan, Rotational Projection Statistics for 3D Local Surface Description and Object Recognition, International Journal of Computer Vision 105 (1) (2013) 63–86. doi:10.1007/s11263-013-0627-y.
URL `http://dx.doi.org/10.1007/s11263-013-0627-y`

[16] R. B. Rusu, N. Blodow, M. Beetz, Fast Point Feature Histograms (FPFH) for 3D Registration, in: Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, IEEE, 2009.

[17] R. Rusu, S. Cousins, 3D is here: Point Cloud Library (PCL), in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, 2011, pp. 1–4. doi:10.1109/ICRA.2011.5980567.

[18] L. Duan, V. Chandrasekhar, J. Chen, J. Lin, Z. Wang, T. Huang, B. Girod, W. Gao, Overview of the MPEG-CDVS standard,

Image Processing, IEEE Transactions on PP (99) (2015) 1–1. doi:10.1109/TIP.2015.2500034.

[19] D. G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International journal of computer vision 60 (2) (2004) 91–110.

[20] S. Paschalakis, et al., Information Technology - Multimedia content descriptor interface - Part 13: Compact Descriptors for Visual Search, in ISO/IEC DIS 15938-13.

[21] S. Paschalakis, et al., CDVS CE2: Local Descriptor Compression Proposal, in: ISO/IEC JTC1/SC29/WG11/M25929.

[22] J. Chen, L.-Y. Duan, R. Ji, Z. Wang, Multi-stage Vector Quantization towards Low bit rate Visual Search, in: Image Processing (ICIP), 2012 19th IEEE International Conference on, 2012. doi:10.1109/ICIP.2012.6467392.

[23] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, Y. Reznik, R. Grzeszczuk, B. Girod, Compressed Histogram of Gradients: A Low-Bitrate Descriptor, International Journal of Computer Vision 96 (3) (2012) 384–399. doi:10.1007/s11263-011-0453-z.

[24] V. R. Chandrasekhar, Low Bitrate Image Retrieval with Compressed Histogram of Gradients Descriptors, Ph.D. thesis, Stanford University (2013).

[25] T. Gagie, Compressing Probability Distributions, Information Processing Letters 97 (4) (2006) 133 – 137. doi:http://dx.doi.org/10.1016/j.ipl.2005.10.006.

[26] F. Malaguti, F. Tombari, S. Salti, D. Pau, L. Di Stefano, Toward Compressed 3D Descriptors, in: 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on, 2012, pp. 176–183. doi:10.1109/3DIMPVT.2012.9.

[27] S. M. Prakhya, B. Liu, W. Lin, B-SHOT: A Binary Feature Descriptor for Fast and Efficient Keypoint Matching on 3D Point Clouds, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, 2015,.

[28] Y. Reznik, et al., An Algorithm for Quantization of Discrete Probability Distributions, in: Data Compression Conference (DCC), 2011, IEEE, 2011, pp. 333–342.

[29] T. Cover, Enumerative source encoding, Information Theory, IEEE Transactions on 19 (1) (1973) 73–77. doi:10.1109/TIT.1973.1054929.

[30] A. Aldoma, F. Tombari, L. Di Stefano, M. Vincze, A Global Hypothesis Verification Framework for 3D Object Recognition in Clutter, Pattern Analysis and Machine Intelligence, IEEE Transactions on PP (99) (2015) 1–1. doi:10.1109/TPAMI.2015.2491940.

[31] F. Tombari, S. Salti, L. D. Stefano, Performance Evaluation of 3D Keypoint Detectors, International Journal of Computer Vision 102 (1-3) (2013) 198–220. doi:10.1007/s11263-012-0545-4.
URL `http://dx.doi.org/10.1007/s11263-012-0545-4`