Efficient Storage and Decoding of SURF Feature Points

Kevin McGuinness, Kealan McCusker, Neil O'Hare, and Noel E. O'Connor

CLARITY: Center for Sensor Web Technologies, Dublin City University

Abstract. Practical use of SURF feature points in large-scale indexing and retrieval engines requires an efficient means for storing and decoding these features. This paper investigates several methods for compression and storage of SURF feature points, considering both storage consumption and disk-read efficiency. We compare each scheme with a baseline plain-text encoding scheme as used by many existing SURF implementations. Our final proposed scheme significantly reduces both the time required to load and decode feature points, and the space required to store them on disk.

Keywords: Image features, interest points, quantization, coding

1 Introduction

Speeded up robust features (SURF) based descriptors [1] have become very popular in modern computer vision and multimedia information retrieval engines, primarily due to their exceptional performance and a fast method for detection and extraction. For large datasets, SURF features can be computed offline and stored alongside the images they represent, allowing matching to be performed without the need to extract features each time an image is visited.

A linear search for an object in a database of feature points requires, for each image, three operations: 1) loading all feature points for the image from the disk; 2) decoding these feature points; and 3) computing the distance between all feature points in the query object to all feature points in the target image. Euclidean distance is generally used in practice, and can be implemented using a fixed number of primitive (add, subtract, square) machine operations. As such, the time required to search a large dataset can be dominated by the time spent loading and decoding feature points if the amount of memory required to represent these feature points exceeds the memory capacity of the hardware.

Storage space for feature points also requires consideration. Typical image files, efficiently compressed, require only a few tens or hundreds of kilobytes of disk storage. Depending on the image and parameters specified for the feature extraction algorithm, each image can produce hundreds or even thousands of feature points. A naïve scheme for storing these feature points (e.g. XML) could require significantly storage space than the image file itself. Indeed, one XML

based storage scheme we investigated required over 13GB to store the feature points for 10,000 images; the images required only a single gigabyte of storage.

Visual feature vector compression has previously been considered by Takacs et al. [15], who used entropy coding of quantized feature vectors to reduce the size of feature vectors to approximately 5 bits per coefficient, in their mobile augmented reality system. They subsequently proposed transform coding of SURF and SIFT [11] components using the Karhunen-Loeve transform, followed by quantization and entropy coding, achieving very high compression rates of approximately 2 bits per coefficient [3]. They later achieved even higher compression rates using a compressed histogram of gradients descriptor [2].

While other authors have proposed very effective schemes for compressing SURF feature vectors, these works were focused on low-bitrate descriptors; the objective of this paper is to devise a storage scheme for SURF feature points that is efficient in terms of both storage space and time required to read and decode the features from a disk. This compression scheme, like the scheme used by Takacs et al. [15], is based on quantization and entropy coding of the coefficients. While Takacs et al. simply noted the use of entropy coded SURF features as part of a larger system, we present a more in-depth discussion on how to implement such an encoding scheme, and an analysis of the effect of quantization on the distance between pairs of descriptors. We also present an empirical analysis of decode time, storage requirements and accuracy using the BelgaLogos dataset [9].

It is worth mentioning that exhaustive search is usually impractical for large-scale online image search systems where the query images are not known a-priori. Visual bag of words (BoW) and bag-of-features (BoF) based techniques (e.g. [13, 12, 6]) are generally superior for these systems, in terms of both query time and storage requirements, since only vectors of codewords need to be retained for each image, rather than the individual descriptors. Such systems can, in addition, benefit from BoF based compression techniques such that proposed by Jègou et al. [8] to reduce the dimensionality of the codeword vectors, allowing for very compact image descriptors. Nevertheless, storing the individual descriptors is still important for a variety of applications. Search and indexing systems may benefit from storing the descriptors during an intermediate step in the indexing chain, to be processed at a later stage by other modules like visual word extraction or offline indexing. Mobile applications may be required to transmit descriptors over low-bandwidth connections for cloud based matching and detection. Retaining individual descriptors can also be useful for establishing the precise location of matched objects in images.

The remainder of the paper is organized as follows. Section 2 gives an overview the composition of a SURF feature point and introduces notation for the components. Section 3 describes the dataset we used for analysis and experimentation. Section 4 examines the effect of quantizing SURF coefficients on the square distance between pairs of coefficients. Section 5 discusses quantization strategies, and Section 6 discusses entropy coding of the quantized coefficients. Section 7 evaluates the proposed encoding schemes, and Section 8 concludes the paper.

2 SURF Feature Points

A SURF feature point f is composed of the (x, y) location of the point, a scale value s, an orientation value θ, a Laplacian value $l \in \{-1, +1\}$, and 64 coefficients that make up the descriptor. The coefficients are summations over the horizontal and vertical Haar wavelet filter responses in the $4 \times 4 = 16$ blocks surrounding the feature point. There are four coefficients extracted for each block: $f_1 = \sum d_x$, $f_2 = \sum d_y$, $f_3 = \sum |d_x|$, and $f_4 = \sum |d_y|$, where d_x and d_y are the horizontal and vertical filter responses at each pixel inside a block, and the summations apply over all pixels in the block.

3 BelgaLogos Dataset

The BelgaLogos dataset [9] is composed of 10,000 images manually annotated for 26 logos and trademarks, with 55 images from the dataset provided as query images for these logos. This makes it ideal for evaluating image and feature point matching techniques. We extracted a total of 5,966,580 feature points from these images[1], a mean of 596.7 per image. The range, interquartile range, mean, median, and standard deviation of the number of feature points per image are:

min.	Q.1	median	mean	Q.3	max.	sd.
0	333	517	596.7	756	3938	377.8

The minimum of zero in the above is an outlier – only one image has zero feature points associated with it.

To perform a statistical analysis of the feature points, we randomly sampled 100,000 feature points from the 5,966,580 in the dataset. Figure 1 shows the distribution of feature point components $f_1 \ldots f_4$ for all feature points in the sample set. The f_1 and f_2 components for the sample are in the range $[-0.5, 0.5]$; the f_3 and f_4 are in the range $[0, 1]$. The observed ranges and mean values of the coefficients are:

	f_1	f_2	f_3	f_4
mean	-0.0000235	0.0213	0.109	0.154
min	-0.3709430	-0.408639	0.0	0.0
max	0.4143080	0.460317	0.64189	0.74343

The orientation value are in the interval $[0, 2\pi]$, with modes $\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi\}$. The scale values in the sample set are in the interval $[1.601, 22.680]$.

[1] The number of features extracted is governed by the Hessian threshold, which we set to 0.001.

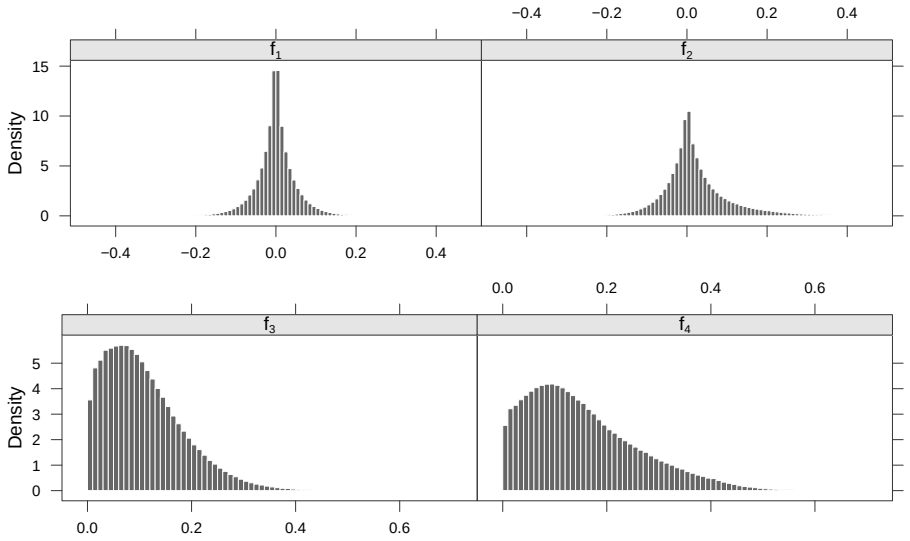

Fig. 1. Histograms showing the distribution of feature point descriptor coefficients (f_1, f_2, f_3, f_4)

4 Coefficient Precision

Matching comparable SURF features (i.e. those with similar scales and Laplacian values) requires computing the Euclidean distance between the coefficient vectors for pairs of descriptors. Common criteria for deciding if two feature vectors match are [14]: 1) comparing against a simple threshold, 2) finding the k nearest neighbors feature vectors and comparing their distance against a threshold, and 3) comparing the the ratio between the nearest and second nearest neighbors (the *nearest neighbor distance ratio*) with a threshold. Reducing the precision of the SURF coefficients allows them to be stored using less bits. Ideally, we would like to reduce the precision so that it does not have a significant affect on the distance between any two vectors. Here we look at the effect of reducing the precision of the coefficients on the distance computation.

Let $\mathbf{u}$ and $\mathbf{v}$ be the coefficient vectors for two SURF feature points, with $\mathbf{v} = (v_1, v_2, \ldots, v_{64})$, and $\mathbf{u} = (u_1, u_2, \ldots, u_{64})$. The square distance between these vectors is:

$$\delta^2(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{64} (u_i - v_i)^2. \tag{1}$$

Any loss in precision can introduce a small error component at each value in $\mathbf{u}$ and $\mathbf{v}$. Let the maximum error introduced by encoding the coefficients u_i and v_i be equal to ϵ. In the worst case, this error value can affect each term in the

distance formula by 2ϵ, giving a square distance of:

$$\delta^2(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{64} (u_i - v_i + 2\epsilon)^2. \tag{2}$$

Substituting $\Delta_i = u_i - v_i$ in the above and multiplying out gives:

$$\delta^2(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{64} \Delta_i^2 + \left[4\epsilon \sum_{i=1}^{64} \Delta_i + 256\epsilon^2 \right]. \tag{3}$$

The first part is just the square distance, so the worst-case error in the square distance computation that results from an error ϵ in each component is:

$$E_{\text{worst-case}}(\epsilon, \Delta) = 4\epsilon \sum_{i=1}^{64} \Delta_i + 256\epsilon^2. \tag{4}$$

Recall that the SURF coefficients f_1 and f_2 are in the interval $[-0.5, 0.5]$ and the coefficients f_3 and f_4 are in the range $[0, 1]$. The maximum distance between any two SURF coefficients is therefore 1, and so the largest value of the summation in the above is 64 (since, $\max \sum_i^{64} \Delta_i = 64$). Substituting this into Eq. (4) gives a general worst-case error:

$$E_{\text{worst-case}}(\epsilon) = 256 \times \epsilon(\epsilon + 1). \tag{5}$$

To keep the square distance error δ_ϵ^2 below some value, we simply choose sufficient digits of precision so that ϵ satisfies:

$$256\epsilon^2 + 256\epsilon < \delta_\epsilon^2. \tag{6}$$

The maximum allowable value of ϵ for a given δ_ϵ^2 can be found by solving the above quadratic inequality and inferring the required precision. The worst case square error in terms of the number of bits of binary precision of each component can be derived by substituting the machine unit roundoff error $\epsilon = 2^{-p}/2$ for p bits of precision in Eq. (5).

$$E_{\text{worst-case}}(p) = 256 \times \frac{2^{-p}}{2} \left(\frac{2^{-p}}{2} + 1 \right) \tag{7}$$

$$= 2^{6-2p} + 2^{7-p}. \tag{8}$$

Usually the value of $k = \sum_{i=1}^{64} \Delta_i$ from Eq. (4) will be significantly less than its worst-case value of 64, which only occurs when the distance between each of the individual components is maximal. It is clear from Figure 1 that the coefficient values are usually close to zero, and so the expected value of k is small. Moreover, the threshold on the Euclidean distance $\delta(\mathbf{u}, \mathbf{v})$ used to match SURF features is usually low. Therefore, although in the worst-case $k = 64$, in the average case – and for cases that will affect the outcome of a whether two are judged to match – $k \ll 64$.

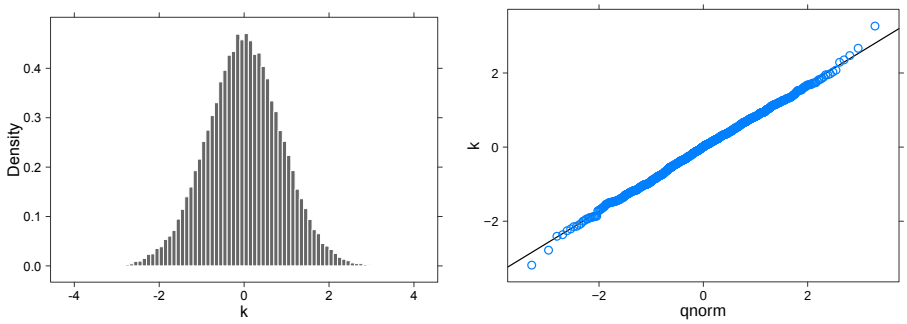

Fig. 2. Histogram (left) showing the distribution of $k = \sum_{i=1}^{64} \Delta_i$ for a sample of 50,000 pairs of feature points. Q-Q plot (right) showing that it is reasonable to assume that k is normally distributed.

Figure 2 (left) shows the distribution of k for a random sample of 50,000 pairs of feature points, and Fig. 2 (right) shows the normal Q-Q plot of 5,000 of these values. It is clear from these figures that the distribution of the sample is very close to normal, and it is reasonable to assume that the value of k in general is also normally distributed. The mean and standard deviation were found to be zero (± 0.0077 with 95% confidence) and 0.88 (c.i. $(0.875, 0.887)$ with 95% confidence). Assuming that the sample is representative, this implies that 75% of all values of k are in the $(-1, +1)$ interval, 97.6% are in the $(-2, +2)$ interval, and 99.9% are in the $(-3, +3)$ interval. The upper bound for error given by $k = 64$ in Eq. 8 is therefore unrealistically high in practice; setting $k = 4$ gives a practical upper bound on the error, while setting $k = 1$ gives the square distance error in the majority of cases. The general formula for square distance error in terms of k given p bits of precision is:

$$E(p, k) = 2^{6-2p} + 2^{1-p}k. \tag{9}$$

Table 1 shows the error values for varying amounts of machine precision and values of k. The first column is the number of bits p of machine precision; the second is the roundoff error for p bits of precision; the remaining columns show the effect of this error on the square distance in the worst and typical cases. The last row, $p = 23$, corresponds to IEEE-754 single precision floating point numbers. The table shows that the error incurred by encoding each coefficient using 16 bits of precision is likely to be acceptable. The error incurred by using 8 bits of precision may also be acceptable, and is worth investigating.

5 Quantizing Coefficients

The discussion in the previous section on the effect of rounding error on the outcome of a distance calculation indicates that encoding the coefficients with 16

Table 1. Worst-case and typical error values for varying amounts of machine precision

p	$\epsilon = 2^{-p}/2$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$E_{\text{worst-case}}$
7	0.00390625	0.01953125	0.03515625	0.05078125	0.06640625	1.00390625
8	0.00195312	0.00878906	0.01660156	0.02441406	0.03222656	0.50097656
9	0.00097656	0.00415039	0.00805664	0.01196289	0.01586914	0.25024414
10	0.00048828	0.00201416	0.00396729	0.00592041	0.00787354	0.12506104
12	0.00012207	0.00049210	0.00098038	0.00146866	0.00195694	0.03125381
14	0.00003052	0.00012231	0.00024438	0.00036645	0.00048852	0.00781274
16	0.00000763	0.00003053	0.00006105	0.00009157	0.00012209	0.00195314
18	0.00000191	0.00000763	0.00001526	0.00002289	0.00003052	0.00048828
20	0.00000048	0.00000191	0.00000381	0.00000572	0.00000763	0.00012207
22	0.00000012	0.00000048	0.00000095	0.00000143	0.00000191	0.00003052
23	0.00000006	0.00000024	0.00000048	0.00000072	0.00000095	0.00001526

or possibly even 8 bits of precision should not significantly affect the outcome. Since all the coefficients f have a range $r = |\max f - \min f| \leq 1$, they can be remapped so that they lie in the interval $[0, 1]$. We therefore only need to encode the significand (mantissa) of the floating point number – all the bits can be used for precision.

A rigorous description of the quantization scheme requires us to define rounding and quantization functions. The rounding function $r\colon \mathbf{R} \to \mathbf{Z}$ returns the nearest integer to x, rounding ties away from zero:

$$r(x) = \begin{cases} \lfloor x + 0.5 \rfloor & x \geq 0 \\ \lceil x - 0.5 \rceil & x < 0. \end{cases} \tag{10}$$

The clamp function $c\colon \mathbf{R} \to \mathbf{R}$ limits the value of x to a given interval $[x_1, x_2]$:

$$c(x; x_1, x_2) = \max\{\min\{x, x_2\}, x_1\}. \tag{11}$$

The linear remapping function $m\colon \mathbf{R} \to \mathbf{R}$ maps values that lie in the interval $[x_1, x_2]$ to values in the interval $[y_1, y_2]$ as follows:

$$m(x; x_1, x_2, y_1, y_2) = y_1 + \frac{(y_2 - y_1)(x - x_1)}{x_2 - x_1}. \tag{12}$$

Using the above definitions, we can define the p bit quantization function $q_p\colon \mathbf{R} \to \mathbf{Z}$, which maps a real value $x \in [x_1, x_2]$ to a p bit integer representation:

$$q_p(x; x_1, x_2) = r(m(c(x; x_1, x_2); x_1, x_2, 0, 2^p - 1)). \tag{13}$$

The corresponding p bit de-quantization function $d_p\colon \mathbf{Z} \to \mathbf{R}$ returns a real number that approximates $x \in [x_1, x_2]$:

$$d_p(y; x_1, x_2) = c(m(y; 0, 2^p - 1, x_1, x_2); x_1, x_2). \tag{14}$$

5.1 8 and 16-bit Encoding

We can define a 16-bit encoded point using the following C structure (a similar structure can be defined for 8-bit coding, using instead 8 bits per coefficient):

```
struct encoded_point {
  uint16_t x, y, scale;
  uint8_t orientation, laplacian;
  uint16_t coefficients[64];
};
```

Assuming images are never larger than 65535×65535, the values x and y in the above can be assigned from the unencoded values by simple rounding:

$$x' = r(x) \tag{15}$$
$$y' = r(y). \tag{16}$$

Assuming that scale values greater than 30 are sufficiently rare that they can be safely approximated by a value of exactly 30, then we can assign the scale value in the above structure as:

$$s' = q_{16}(s; 0, 30). \tag{17}$$

The orientation parameter can be encoded to 8 bits, assuming that an error of 7/10 of a degree can be tolerated in practice.

$$\theta' = q_8(\theta; 0, 2\pi) \tag{18}$$

The Laplacian parameter could be encoded in a single bit, but we use 8 bits so that the structure is aligned to 16-bit boundaries in memory. The 16 blocks of coefficients $f_1 \ldots f_4$ are encoded as:

$$f_1' = q_{16}(f_1; -0.5, +0.5) \tag{19}$$
$$f_2' = q_{16}(f_2; -0.5, +0.5) \tag{20}$$
$$f_3' = q_{16}(f_3; 0, 1) \tag{21}$$
$$f_4' = q_{16}(f_4; 0, 1). \tag{22}$$

The final 16-bit encoding requires 136 bytes per SURF point, before applying any compression algorithms. For the 8-bit quantization, the q_8 quantizer is used above. The 8-bit encoding requires 72 bytes per SURF point.

6 Compression

The symbol entropy values $H(f)$ for each of the four Haar coefficients $f_1 \ldots f_4$ for the BelgaLogos dataset sample were found to be:

	f_1	f_2	f_3	f_4	mean
8-bit	5.64	6.32	6.09	6.60	6.16
16-bit	13.62	14.30	14.07	14.59	14.14

Shannon's source coding theorem for symbol codes bounds the compression rate that can be achieved using an optimal symbol coding algorithm between $H(f)$ and $H(f)+1$, so based on the mean entropies above we can expect compressing the 8-bit encoded coefficients using an optimal symbol coding algorithm to reduce the number of bits to required to encode a coefficient by between 0.84 and 1.84 bits per symbol. This translates reducing amount of space required to store the coefficients by between 10 and 23%.

To encode 8-bit coefficients, we first computed four empirical probability distributions for the quantized coefficients $f_1 \ldots f_4$ by sampling N = 100,000 points from the BelgaLogos dataset. These four sample distributions were then smoothed by convolution with a width 5 Hamming window to give the empirical distributions $P_e(x|n), n \in \{1 \ldots 4\}, x \in \{0 \ldots 255\}$. The empirical distributions were then used to generate prefix codes $C_n(x)$ for each the Haar coefficients using the Huffman algorithm [5]. To minimize any wasted bits due to a block of Huffman codes not being a multiple of 8 bits, we encoded all Haar coefficients of the same type (e.g. f_1) together in large blocks for all SURF points in a file.

7 Evaluation

To evaluate the effect of quantization on the outcome of a matching experiment, we ran a linear search using the internal query set (55 query images) in the BelgaLogos set for the different encoding strategies. Table 2 summarizes the results of the evaluation. In the table, the column marked 'text/xml' refers to an XML-based encoding scheme; 'text/plain' refers to the text encoding scheme used by the OpenSURF[2] library; 'float/32' refers to storing the coefficients as 32-bit IEEE-754 floating point numbers; 'int/16' and 'int/8' refer to simple quantization of the coefficients to 16 and 8 bits as described above; 'huffman/8' refers to entropy coded 8-bit coefficients. The total query times shown are for all 55 images in the internal query set[3].

There is no change in mean average precision (MAP) when using 16-bit precision to encode the coefficients. There is a slight increase in MAP when using 8-bit coefficients. This increase can be attributed to the clustering effect of quantizing the features, which makes them comparable with visual words.

The raw binary encoding schemes gave the fastest decode times, all producing a roughly equal query time of ∼34 seconds per query on our test machine (34 seconds to match a set of query feature points against all feature points in the BelgaLogos set). Huffman encoding of the coefficients more than doubles the decode time to 75 seconds. Both raw and encoded coefficients are significantly faster than using XML encoded features, giving speedups of 97.6% and 94.7%. Compared with the plain text format generated by OpenSURF, the speedups are 88% for raw features and 74.2% for encoded features.

[2] `http://www.chrisevansdev.com/computer-vision-opensurf.html`

[3] The test machine used was an Intel® quad core Xeon® 1.86GHz CPU (E5320) with 4GB RAM running Linux kernel 2.6.31-19 and using an ext4 filesystem.

Table 2. Summary of encoding method performance

	text/xml	text/plain	float/32	int/16	int/8	huffman/8
Dataset size (MB)	13172	3683	1590	795	430	346
Mean file size (KB)	1349	377	163	81	44	35
Total query time (min)	1305	267	32	32	32	69
Mean query time (sec)	1422	291	34	34	34	75
Mean average precision (%)	8.53	8.53	8.53	8.53	8.74	8.74

Entropy coding of the 8-bit coefficients gives the greatest space efficiency, reducing the dataset size by 97.4% compared to XML encoded features, by 90.6% compared to plain text encoded features, and by 78.2% compared to binary coding with 32-bit floating point numbers.

7.1 Comparison with PCA Compression

Correlation among the Haar coefficients within descriptors implies a more compact representation may be obtained using principal component analysis (PCA) to reduce the dimensionality of the feature vectors without loss of important information. This approach is similar to the PCA-SIFT technique described by Ke and Sukthankar [10], except that here we focus on matching using the reconstructed descriptors rather than in the transform domain. The following compares several PCA-based compression schemes with the simple quantization and entropy coding schemes outlined above in terms of storage size, decoding speed, and matching accuracy on the BelgaLogos dataset.

We used 100,000 randomly sampled feature points to derive the PCA eigenvector transform matrix Φ. We used the same sample to derive the empirical means $\mu = (\mu_1, \ldots, \mu_{64})$ of the 64 coefficients, and the PCA transform of the sample to derive the empirical range $[\alpha_i, \beta_i]$ of the transformed coefficients. Dimensionality reduction of a coefficient vector $\mathbf{v}$ is performed by centering around the mean and multiplying by the transform matrix: $\Phi^{\mathrm{T}}(\mathbf{v} - \mu)$, then truncating to the desired number of coefficients. Each retained coefficient v_i is then quantized to 8 bits using Eq.(13) with the empirical minimum α_i and maximum β_i. Decoding a quantized coefficient vector $\mathbf{u}$ is performed similarly: by de-quantizing the stored coefficients and performing the inverse transform $\Phi\mathbf{u} + \mu$.

Table 3 compares the performance of PCA compressed descriptors with 8-bit quantization and entropy coding based descriptors on the BelgaLogos dataset. The column heading 'PCA/50' denotes a 50 element PCA compressed descriptor quantized to use 8-bits per element. It is clear from the table that the 50 element PCA descriptor performs on par with 8-bit entropy coded coefficients in terms of descriptor size and decode time, but suffers a small loss in matching precision. The 40 and 30 element PCA descriptors are both smaller and faster than entropy coded coefficients, but the sacrificed precision reduces matching accuracy. The 20 element descriptor is almost half the size of the entropy coded descriptor, but

Table 3. Comparison of 8-bit PCA compression with quantization and entropy coding

	PCA/50	PCA/40	PCA/30	PCA/20	int/8	huffman/8
Dataset size (MB)	351	294	237	180	430	346
Mean file size (KB)	36	30	24	18	44	35
Total query time (min)	61	53	47	41	32	69
Mean query time (sec)	66	58	51	44	34	75
Mean average precision (%)	8.18	6.94	5.47	2.35	8.74	8.74

suffers an acute reduction in accuracy. Despite their smaller size, none of the PCA-based descriptors match the decode time of the simple 8-bit scheme.

8 Conclusion

We have presented a simple, easy to implement, scheme for quantization and entropy coding of SURF features that significantly reduces both the space required to store these features and the time required to load and decode them. The experiments show that careful quantization of the feature points can be performed without significantly effecting matching performance. Quantization of coefficients to 8 bits also opens the door to an integer only implementation of feature point distances, which may improve performance, particularly for embedded applications. Entropy coding reduces the storage requirements by 20% over raw 8-bit coefficients, but doubles the decoding time. This modest space improvement rarely justifies the additional complexity and decode time; we anticipate that simple 8-bit quantization may give the best balance between decode time and storage requirements for many applications. If space is at a premium, transform coding can be applied to de-correlate the coefficients prior to quantization at the cost of reduced matching accuracy and increased decode times.

More powerful compression techniques like product quantization [7] are capable of achieving an even more compact representation. For example, [4] used this technique to compress SURF feature vectors to around half the size of the Huffman encoded vectors. This approach requires precomputing a codebook using a VQ algorithm (e.g. Lloyd's algorithm), which may take significant time and can potentially produce suboptimal results. The simple quantization approach presented here can be combined with product quantization to reduce the memory footprint of the codebook by a factor of four without significant effect on reconstruction. Codebooks can therefore be designed with four times more cluster centroids and still fit in the same amount of memory, which may be important for performance when the codebook is required to fit in cache memory [7].

An implementation of our SURF encoding schemes is available online and can be downloaded from: `http://kspace.cdvp.dcu.ie/public/surfenc`.

Acknowledgements. This work is supported by Science Foundation Ireland under Grant 07/CE/I1147 (CLARITY: Center for Sensor Web Technologies). Part of this work is supported by the EU Project FP7 AXES ICT-269980.

References

1. Speeded up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.
2. V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod. Chog: Compressed histogram of gradients a low bit-rate feature descriptor. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2504–2511, 2009.
3. V. Chandrasekhar, G. Takacs, D. Chen, S. S. Tsai, J. Singh, and B. Girod. Transform coding of feature descriptors. In *Visual Communication and Image Processing*, 2009.
4. S. Gammeter, A. Gassmann, L. Bossard, T. Quack, and L. Van Gool. Server-side object recognition and client-side object tracking for mobile augmented reality. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2010*, pages 1–8, 2010.
5. D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
6. H. Jègou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87:316–336, 2010.
7. H. Jègou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:117–128, 2011.
8. H. Jègou, M. Douze, C. Schmid, and P. Perez. Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, 2010.
9. A. Joly and O. Buisson. Logo retrieval with a contrario visual query expansion. In *ACM International Conference on Multimedia*, pages 581–584, 2009.
10. Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 506–513, 2004.
11. D. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, pages 1150–1157, 1999.
12. J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.
13. J. Sivic, B. Russell, A. Efros, A. Zisserman, and W. Freeman. Discovering objects and their location in images. In *IEEE International Conference on Computer Vision*, pages 370–377, 2005.
14. R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 1 edition, 2010.
15. G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W. C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *ACM International Conference on Multimedia Information Retrieval*, pages 427–434, 2008.