EXTRACTING MULTI-SIZE LOCAL DESCRIPTORS BY GPU COMPUTING

Naoyuki Ichimura

National Institute of Advanced Industrial Science and Technology (AIST)
1-1-1, Umezono, Tsukuba, Ibaraki 305-8568, Japan
nic@ni.aist.go.jp

ABSTRACT

This paper presents fast computational techniques for extracting local descriptors from multiple local regions associated with an image feature such as a feature point. Multiple local regions with different sizes are detected by multiplying multiple *scale factors* to the characteristic scale of the image feature. The descriptors obtained from multiple local regions are called *multi-size local descriptors*. Multi-size local descriptors enable us to use various types of feature representation and matching schemes based on many different spatial sizes, which is a promising way to control the balance among the robustness against for occlusions, the invariance, and the distinctiveness of the descriptors to the contents of scenes. Because multi-size local descriptors increases the computational costs of feature extraction, we introduce parallel computational techniques for extracting the multi-size local descriptors consisting of the histograms of gradient orientations through the use of a graphics processing unit (GPU). In particular, we demonstrate that orientation maps are useful for efficient extraction of the multi-size local descriptors. Using orientation maps, we can calculate the descriptors by a table look-up manner. We show implementation details and then conclude with the experimental results that demonstrate the usefulness of GPU computing with orientation maps.

Index Terms— local invariant features, local regions, orientation maps, GPU computing

1. INTRODUCTION

The use of local invariant features is regarded as a promising method for representing the contents of an image. Local invariant features can be extracted by the following two steps[1, 2, 3, 4]: (i) detecting local regions, (ii) calculating descriptors. Figure 1 shows an example of detecting local regions. Descriptors are calculated in the local regions shown by the squares. Local invariant features have two main advantages in describing scenes. The first is their robustness against occlusions; we can use the features of visible portions of the scene even if parts of the scene are occluded. The second advantage is, of course, their invariance. By introducing components such as scale space pyramids, local coordinate systems and norm normalization into feature extraction, we can make local features invariant to deformations and to changes in illumination. Because of these advantages, local invariant features have been widely used as fundamental elements for image matching and object recognition[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].

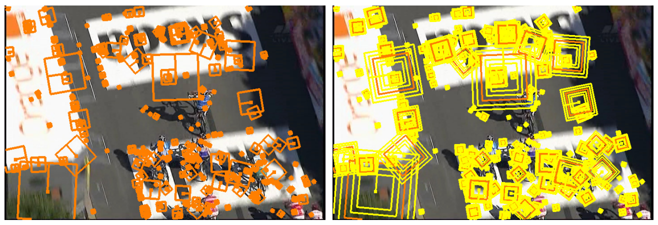

(a) (b)

Fig. 1. Examples of detecting local regions for the image "Tour de France"[13]. (a) A result using a single scale factor. (b) A result using multiple scale factors. In this examples, only 5% of all local regions are shown for the sake of clarity.

Local region detection is performed by determining the positions and sizes of local regions in a scale space pyramid computed from an image. Image features such as feature points and edges and several sampling strategies have been adopted to find the positions of pixels which are used as the centers of local regions[1, 2, 4, 8, 9, 14]. The sizes of local regions are determined by the scales of images in which the center pixels are found, i.e., the characteristic scales[15] of the center pixels. In practice, we determine the sizes of local regions by multiplying the characteristic scales by *a scale factor*. The differences of the characteristic scales of the center pixels lead to the differences of the sizes of the local regions as demonstrated in Fig.1(a). Using the characteristic scales to determine the sizes of local regions enable us to make descriptors invariant to scale changes.

The scale factor has large effect on the robustness against for occlusions, the invariance, and the distinctiveness of descriptors, because it controls the spatial extent used for image feature representation. For example, if the contents of an image include many changes in shape and brightness, descriptors obtained from small local regions that are useful to increase the robustness against for occlusions can have enough invariance and distinctiveness. On the other hand, if there are little changes in shape and brightness, larger local regions are required for the invariance and distinctiveness of descriptors. The scale factor, therefore, should be changed based on the degree of occlusions and the complexity of the contents of scenes. However, the scale factor is normally fixed regardless of the contents of scenes, because it is difficult to predict the degree and complexity to adjust the scale factor.

In this paper, we develop a computational techniques for extracting descriptors from multiple local regions with different sizes which are determined by multiplying multiple scale factors to the characteristic scale. The descriptors obtained from multiple local regions are called *multi-size local descriptors*. Figure 1(b) shows examples of local regions for multi-size local descriptors. For each center pixel, five local regions are prepared in this example. Using multi-size local descriptors obtained from different spatial extents enables us to control the the balance among the robustness against for occlusions, the invariance, and the distinctiveness of the descriptors without using a priori knowledge about the contents of scenes. For example, if the robustness against for occlusions has the priority over the others in image matching and object recognition, the descriptors obtained from the smallest local regions would be most useful. If the distinctiveness is important, utilizing co-occurrence, similarity and/or correlation computed from multi-size local descriptors could be effective. An exhaustive matching among all the sizes of local regions might be a way to increase the invariance. The multi-size local descriptors, therefore, have advantages over the descriptors extracted by using a single scale factor.

A problem in extracting multi-size local descriptors is the increase of computational costs. In order to address the problem, we introduce parallel computational techniques for extracting the multi-size local descriptors consisting of the histograms of gradient orientations through the use of a graphics processing unit (GPU). In particular, we demonstrate that the data structure called orientation maps[12] are useful for efficient extraction of the multi-size local descriptors on a GPU.

A related work on the multi-size local descriptor is Cheng et al.[16]. The purpose of their research is to develop a descriptor which is robust to image deformations due to nonrigid objects and optical systems such as fisheye lens. Although [16] shows that descriptors obtained from multiple local regions with different sizes are useful for the purpose, coping with the increase of the computational costs to calculate descriptors remains as an issue. In particular, since several investigators[6, 7, 8, 9, 10, 12, 14] have demonstrated the advantages of dense sampling of local features which increases the number of local regions used in image matching and object recognition, introducing fast computational techniques is very important to extract the multi-size local descriptors from local regions detected densely. We present a method for extracting the multi-size local descriptors based on the use of a GPU and orientation maps. Although there are GPU-based implementations of algorithms for local invariant features, such as SIFT[17], SURF[18] and HOG[19], algorithms for the multi-size local descriptors using orientation maps have not been considered.

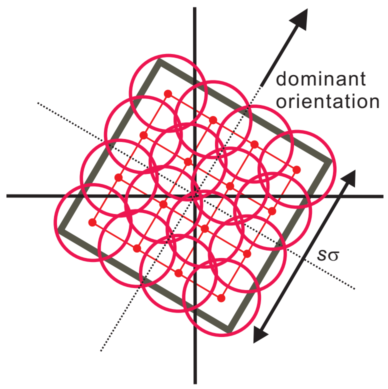


Fig. 2. The arrangement of cells in a local region. The 4×4 cells represented by the circles are arranged in the local region. The size of a local region is determined by multiplying the characteristic scale of the center pixel σ by *the scale factor* s. The local regions with different sizes obtained by changing the scale factor are used to extract the multi-size local descriptors as shown in the example of Fig.1(b).

Below, we explain the descriptor, the orientation maps, and the implementation on a GPU. Then we conclude with a description of our experimental results.

2. REVIEW OF CALCULATING DESCRIPTORS

In this section, we review algorithms commonly used for calculating descriptors, such as SIFT[2], GLOH[3], HOG[6] and DAISY[12], that involve the histograms of gradient orientations.

Multiresolutional analysis by a scale space pyramid is performed for scale invariance. Laplacian of Gaussian (LoG) filters with various scales, together with down sampling of an image, are used to detect local regions by extracting feature points in a scale space pyramid. Scale space extrema are identified to determine the locations and characteristic scales of feature points[15]. The locations of the local regions are the same as those of the feature points. The size of each local region is determined by multiplying the characteristic scale σ by *a scale factor* s as shown in Fig.2. Another scale space pyramid of multiresolutional edge images is generated for the intensity gradients required to calculate the descriptors. Gaussian filters, gradient filters and down sampling are combined to obtain multiresolutional edges. Multiresolutional edges can also be used for local region detection[8, 20]. For example, local regions in Fig.1 are detected using both feature points and edges. The descriptors are calculated in local regions using the intensity gradients obtained from the edge images corresponding to the characteristic scales.

We now explain how the descriptors are calculated. First, cells are arranged in a local region as depicted in Fig.2. Then, the histogram of the gradient orientations of the pixels in each

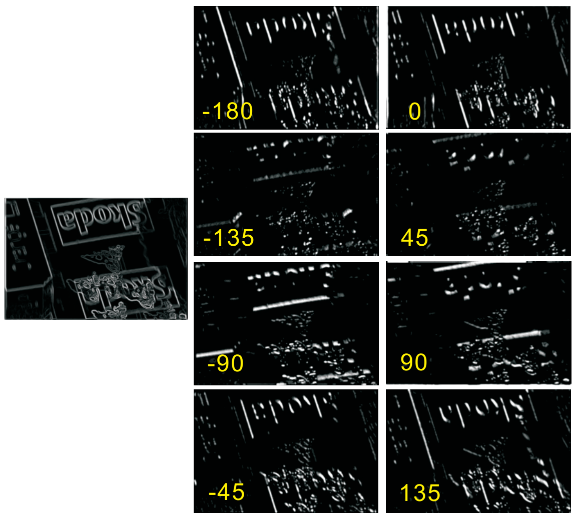


Fig. 3. Examples of orientation maps. Left: an edge detection result for the image shown in Fig.1. Right: the orientation maps corresponding to the eight gradient orientations obtained from the edge image on the left. Each of the maps contains gradient magnitudes for a quantized orientation.

cell is computed. The voting values for the histogram are the gradient magnitudes of the pixels weighted by a Gaussian function located at the center of the local region or cell. The histogram of the cell is shifted for rotation invariance based on the direction of a local coordinate system attached to the local region. The local coordinate system can be set using the dominant orientation proposed in SIFT[2]. A vector is then constructed by concatenating the histograms of the cells. A descriptor is obtained by normalizing the vector for illumination invariance. Since gradient orientations are related to the directions of edges, the descriptor represents the shape information in the cells. The usefulness of this representation in image matching and object recognition has been confirmed[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].

3. CALCULATING MULTI-SIZE DESCRIPTORS USING ORIENTATION MAPS

The gradient orientations and gradient magnitudes of all the pixels in a scale space pyramid are obtained by gradient filters. Using two dimensional arrays corresponding to the bins of the histogram of gradient orientations, we can separate the gradient magnitudes based on the gradient orientations. The two dimensional arrays containing the gradient magnitudes are called *orientation maps*[12]. Examples of orientation maps are shown in Fig.3.

Applying a Gaussian filter to orientation maps, we can aggregate weighted gradient magnitudes locally. The extent of aggregation is determined by the scale of the Gaussian filter. If we choose the scale so that the extent of aggregation fits the size of a cell for the descriptors, the convolution operations performed by the Gaussian filter are equivalent to the computation of the voting values for the histograms of gradient orientations. This allows us to calculate the descriptors merely by looking up the locations of convolved orientation maps corresponding to the centers of cells. No exhaustive access to the pixels in cells is required thanks to orientation maps.

In the conventional methods such as [2, 3, 6], all the pixels in cells should be accessed to calculate the descriptors. Thus if we just port the conventional method to extract multi-size local descriptors, the computational cost is proportional to the number of pixels in multiple local regions. On the other hand, if the method using orientation maps is exploited, the computational cost for extracting multi-size local descriptors is proportional to the number of cells which is much smaller than the number of pixels. This difference is the key to reduce the computational cost.

An issue in using orientation maps is that many convolution operations need to be performed on large numbers of "multiresolutional" orientation maps generated from a scale space pyramid. Thus it is important to examine whether the computational cost of Gaussian filters for multiresolutional orientation maps can actually be exchanged for that of the descriptors calculated without the use of orientation maps.

Since Gaussian filters can be efficiently implemented by utilizing the shared memory in a GPU, the introduction of orientation maps and a GPU could be useful for fast computation of the descriptors. We implement a feature extraction algorithm on the GPU, NVIDIA GTX 480, to confirm the usefulness of this approach; this is discussed in the next section.

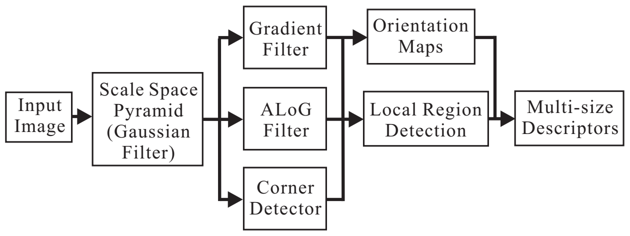


Fig. 4. The flowchart of the feature extraction algorithm that we implement. Local regions are detected based on both feature points and edges. The multi-size local descriptors consisting of the histograms of gradient orientations are calculated using orientation maps.

4. IMPLEMENTATION ON GPU

We apply CUDA(Compute Unified Device Architecture), which was developed by NVIDIA[21], to feature extraction. The data to be processed by the GPU are divided into blocks associated with threads in CUDA. The numbers of blocks and threads are called the *execution configuration* (EC). The number of blocks can be specified by the width and/or the height of a block.

Figure 4 is the flowchart of the feature extraction algorithm that we implement. First, a scale space pyramid is generated from an input image by Gaussian filters with various scales. A gradient filter, approximated LoG filters and corner detectors are then applied to the scale space pyramid for local regions and orientation maps to calculate the descriptors consisting of the histograms of gradient orientations. Blow, we present details of feature extraction including additional processes that are not shown in Fig.4. Because the appropriate choices of the memory in the GPU and ECs are important factors in achieving fast computation, we show their specific type and values.

Image transfer: An input image is transferred from the host (CPU) to the GPU.

Y Component: An intensity image is computed if the input image has colors. The global memory is used. The size of a block is 16×32. Threads are assigned to all the pixels in a block.

Down sampling: Down sampling for a scale space pyramid is performed in the global memory. The EC is the same as that for the **Y Component**.

Gaussian filter: We generate a scale space pyramid using Gaussian filters. The initial scale of the pyramid is 1.6 and the interval of scales is $2^{1/n}$, where n is the number of scales in an octave. The sizes of the Gaussian filters are determined by truncating the domain using the threshold of 10^{-3} for the range . We place the coefficients of the Gaussian filters into the constant memory that has short latency and is read-only in execution of the program. The image data in a block are copied to the shared memory for reuse. The separability of a Gaussian function is exploited. The width of a block in filtering for the rows of the image is 128, and threads are assigned to all the pixels in a block. The size of a block in filtering for the columns is 48×16. Since the maximum number of threads in a block is limited, we divide a block into sub-blocks with the height of eight. We then assign threads to all the pixels in a sub-block. Filtering for the pixels in the remaining sub-blocks is carried out sequentially by the assigned threads.

Gradient filter: The intensity gradients of the scale space pyramid are computed by the 5×5 gradient operator[22]. The use of memory and the EC are the same as those for filtering for the columns in the **Gaussian filter**, except that the size of a block is 16×16.

ALoG-CD filter: Two filters are applied to the scale space pyramid to detect local regions through feature points. First, we apply an approximated LoG (ALoG) filter to all the scales. The ALoG filter reduces the computational cost for contrast detection using only the pixel values at the center and on the circle corresponding to the center and circular extrema of the LoG function, respectively. Secondly, fast corner detectors (CD)[23] with various scales are used to remove feature points on edges that may cause aperture problems. The corner detector also requires only the pixel values at the center and on the circle. The memory usage and the EC are the same as those for the **Gradient filter**.

Feature point: Feature points and their characteristic scales[15] are obtained by identifying $3\times3\times3$ extrema in the scale space pyramid generated by the **ALoG-CD filter**. We search for extrema in each octave. Feature points are selected by thresholding for the responses of the ALoG filters and the corner detectors. The thresholds are 10 and 100, respectively. All the processes are performed in the global memory. The size of a block is 16×32 and threads are assigned to all the pixels in a block. The positions and scales of feature points are the corresponding values for local regions. We organize the positions and scales as a feature list.

Edge sampling: We sample the pixels in the scale space pyramid generated by the gradient filter by extracting spatial 3×3 extrema of gradient magnitudes. A threshold of 10 is used to select the extrema. The global memory is used, and the EC is the same as that for the **Feature point**. The positions and scales of the extrema are added to the feature list.

Orientation map: Multiresolutional orientation maps with the eight quantized directions shown in Fig.3 are calculated from the scale space pyramid obtained by the gradient filter. The orientation maps are generated in the global memory. The EC is the same as that for the **Feature point**. The scales of Gaussian filters applied to the multiresolutional orientation maps which correspond to the sizes of the cells in local regions are determined as explained in Appendix A. The memory usage and the EC are the same as those for the **Gaussian filter**.

Dominant orientation: The dominant orientations[2] of the local regions in the feature list are computed for rotation invariance. The size of each local region is obtained by multiplying its characteristic scale by a factor of five. The global memory is used. We group 30 local regions as a block and assign threads for all the regions. Multiresolutional orientation maps are used for the dominant orientations, in the same manner as the histogram computation for the descriptors.

Descriptor: We calculate the multi-size local descriptors in the local regions with the different sizes obtained by changing the scale factor as follows:

$$s = s_0 \left(0.1L + 1.0\right), \ -N_s \leq L \leq N_l \tag{1}$$

where, s_0 is a base factor and N_s and N_l are the number of smaller and larger scale factors than the base factor, respectively. Local regions are divided into 4×4 cells as shown in Fig.2. The global memory is used. The EC is the same as that for the **Dominant orientation**. We shift the histograms of gradient orientations based on the dominant orientations of local regions for rotation invariance. The descriptors are obtained by normalizing vectors that consist of the histograms of the cells for illumination invariance.

In the next section, we present the measurements of the computational times for the implementations with and without orientation maps on the GPU and the CPU.

Table 1. The computational times of the implementations for the Tour de France image. The units are microseconds. We measured the mean time for 100 trials, because a non-realtime OS (Linux) was used. The Total times include the computational times for processes other than the tasks that were examined, e.g., memory allocation.

Image	Tour de France			
Image size	720×480			
Task/Implement	CPU–1	GPU–1	GPU–5	GPU–C1
Image transfer	N/A	0.721	0.722	0.722
Y component	3.417	0.106	0.107	0.106
Down sampling	0.764	0.140	0.143	0.135
Gaussian filter	253.320	4.386	4.409	4.412
ALoG-CD filter	299.756	6.081	6.083	6.065
Gradient filter	202.901	2.971	2.974	2.968
Feature point	159.468	6.701	6.744	6.755
Edge sampling	8.433	6.933	6.983	6.913
Dominant orientation	68.232	1.850	1.866	4.563
Orientation map	1239.976	22.749	23.995	N/A
Descriptor	41.328	2.807	14.303	51.532
Total	2279.042	56.851	69.734	85.581
#descriptors	5656	5736	5786×5	5731

5. EXPERIMENTAL RESULTS

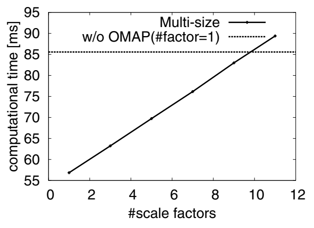
We measured the computational times for feature extraction for the images "Tour de France" shown in Fig.1[13] and the first image in the set called "Graffiti" obtained from the feature detector evaluation sequences[24]. The resolution of the Tour de France image was 720×480. The resolution of the Graffiti image was reduced to permit the measurement of computational time for the QVGA size. The numbers of octaves was five for the Tour de France image and four for the Graffiti image. As the number of scales in an octave was three, we applied to the Gaussian filters to 120 and 96 multiresolutional orientation maps, respectively, with the eight quantized gradient orientations.

We used an Intel Quadcore Xeon (3.16GHz/12MBL2) CPU and an NVIDIA GeForce GTX 480 GPU for the experiments. The local invariant features were calculated on the Xeon and GeForce, and another GPU, an NVIDIA Quadro FX4600, was used solely for the purpose of display. The GPU for computing has 480 CUDA cores, and its compute capability is 2.0[25]. A single core was used in computing on the CPU. Single precision floating-point (float) arithmetic was used in all the implementations. The base factor s_0 in Eq.(1) was 20.

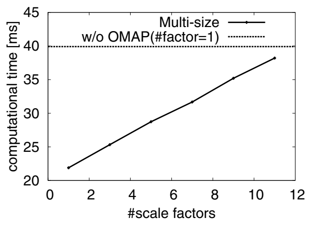
Table 1 lists the computational times for the tasks explained in Section 4 for the Tour de France image. In this table, "CPU-1" is the implementation using the single base factor on the CPU, whereas "GPU-1" and "GPU-5" stand for the corresponding implementations using the single base factor and five scale factors ($N_s = N_l = 2$ in Eq.(1)) on the GPU. "GPU-C1" is the implementation without using orientation maps and using the single base factor on the GPU, in which all the pixels in the cells should be accessed to calculate the histograms. By comparing the computational times for "Dominant orientation", "Orientation map" and "Descriptor" among the implementations, we can see that both the introduction of the use of the GPU and orientation maps are very effective in terms of fast feature extraction. Although the effectiveness depends on the contents of the scene, the introduced computational technique is more than 30 times as fast as the implementation on the CPU even if we use the five scale factors. Similar results were obtained for the Graffiti image.

Figure 5 shows the changes in the total computational times for feature extraction on the GPU as a function of the number of the scale factors. We set $N_s = N_l$ in Eq.(1) and used the value in $[1, 5]$. The dotted horizontal line in the graphs show the computational time of the implementation without using the orientation maps and using the single base factor. These results clearly demonstrate the advantage of the method using the orientation maps for fast extraction of the multi-size local descriptors.

In summary, we have shown that the introduction of GPU computing and orientation maps is very promising to calculate the multi-size local descriptors. The extracted features can be applied to applications such as object localization as demonstrated in Fig.6. The similarities between the five descriptors obtained from each local region were computed to find matching points, which was useful to gain the robustness against for occlusions.



(a)



(b)

Fig. 5. The changes in the total computational times as a function of the number of scale factors. (a) Tour de France(720×480). (b) Graffiti(320×240).

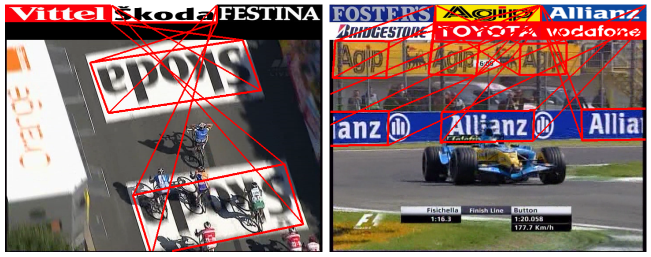


Fig. 6. Examples of logo localization[26]. The local invariant features for localization were extracted using the five scale factors.

6. CONCLUSIONS

In this paper, we have considered GPU computing with orientation maps for calculating the multi-size local descriptors consisting of the histograms of gradient orientations. The fast implementation for a large number of convolution operations that is required for aggregating gradient magnitudes for quantized gradient orientations through the use of Gaussian filters can be realized by a GPU. The usefulness of the introduction of the GPU and orientation maps was confirmed by the experimental results. We believe that the considerations on GPU computing presented here will facilitate the use of multi-size local descriptors for many applications.

7. REFERENCES

[1] C. Schmid and R. Mohr, "Local greyvalue invariants for image retrieval," *IEEE Trans. PAMI*, vol. 19, no. 5, pp. 530–535, 1997.

[2] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comp. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[3] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. PAMI*, vol. 27, no. 10, pp. 1615–1630, 2005.

[4] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A comparison of affine region detectors," *Int. J. Comp. Vis.*, vol. 65, no. 1/2, pp. 43–72, 2005.

[5] C. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Proc. Workshop on Statistical Learning in Computer Vision*, 2004, pp. 1–22.

[6] N. Dalal and B. Triggs, "Histograms of orientated gradients for human detection," in *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, 2005, vol. 1, pp. 886–893.

[7] L. Fei-Fei and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, 2005, vol. 2, pp. 524–531.

[8] X. Ma and W. E. Grimson, "Edge-based rich representation for vehicle classification," in *Proc. Int. Conf. Comp. Vis.*, 2005, vol. 2, pp. 1185–1192.

[9] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *Proc. European Conf. Comp. Vis.*, 2006, pp. 490–503.

[10] K. Mikolajczyk, B. Leibe, and B. Schiele, "Multiple object class detection with a generative model," in *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, 2006, vol. 1, pp. 26–36.

[11] C. H. Lampert, M. B. Blaschko, and T. Hofmann, "Efficient subwindow search: A branch and bound framework for object localization," *IEEE Trans. PAMI*, vol. 31, no. 12, pp. 2129–2142, 2009.

[12] E. Tola, , V. Lepetit, and P. Fua, "DAISY: An efficient dense descriptor applied to wide-baseline stereo," *IEEE Trans. PAMI*, vol. 32, no. 5, pp. 815–830, 2010.

[13] The image was obtained from the broadcast of the Tour de France.

[14] F. Jurie and B. Triggs, "Creating efficient codebooks for visual recognition," in *Proc. Int. Conf. Comp. Vis.*, 2005, vol. 1, pp. 604–610.

[15] T. Lindeberg, "Feature detection with automatic scale selection," *Int. J. Comp. Vis.*, vol. 30, no. 2, pp. 79–116, 1998.

[16] H. Cheng, Z. Liu, N. Zheng, and J. Yang, "A deformable local image descriptor," in *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, 2008.

[17] SiftGPU: http://www.cs.unc.edu/˜ccwu/siftgpu/.

[18] N. Cornelis and L. V. Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," in *Proc. Workshop on Computer Vision on GPU's (in conjunction with CVPR08)*, 2008.

[19] V. A. Prisacariu and I. Reid, "fastHOG – a real-time GPU implementation of HOG –," Tech. Rep. 2310/09, Department of Engineering Science, Oxford University, 2009.

[20] K. Mikolajczyk, A. Zisserman, and C. Schmid, "Shape recognition with edge-based features," in *Proc. Britich Machine Vis. Conf.*, 2003, vol. 2, pp. 779–788.

[21] CUDA Zone: http://www.nvidia.com/object/cuda_home_new.html.

[22] S. Ando, "Consistent gradient operators," *IEEE Trans. PAMI*, vol. 22, no. 3, pp. 252–265, 2000.

[23] M. Trajkovic and M. Hedley, "Fast corner detection," *Image and Vision Computing*, vol. 16, pp. 75–87, 1998.

[24] Feature detector evaluation sequences: http://lear.inrialpes.fr/people/mikolajczyk/Database/.

[25] *NVIDIA CUDA Programming Guide, Version 3.2, p.14*, 2010.

[26] The right image was obtained from the broadcast of F1.

A. DETERMINING THE SCALES OF GAUSSIAN FILTERS FOR ORIENTATION MAPS

Here we describe a method for determining the scales of Gaussian filters for orientation maps. Although the 4×4 cells in Fig.2 are used, the following method can be adapted to other arrangements of cells.

The scale of an orientation map is denoted by σ. The factor s is multiplied by σ to compute the size of a local region. Because the local region is divided into 4×4 squares, the width of a square is $s\sigma/4$. The scale of a Gaussian filter σ_o is determined by setting the value of the independent variable of the Gaussian function at the position of a circumscribed circler cell shown in Fig.2. The value should be chosen carefully, because the scale σ_o directly governs aggregation for the gradient magnitudes in a cell. We use the value $3\sigma_o$, and thus the diameter of the circler cell is $6\sigma_o$. Using the width of the square and the diameter, we have the following scale:

$$\sigma_o = \sqrt{2}s\sigma/24\,. \tag{2}$$

In measuring the computational times shown in Tab.1, the maximum scale factor obtained from Eq.(1) was used to compute σ_o.