

Received 9 February 2023, accepted 22 February 2023, date of publication 24 February 2023, date of current version 1 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3249100

RESEARCH ARTICLE

Fast Computation of RFD-Like Descriptors in Four Orientations

ANTON V. TRUSOV^{1,2,3}, (Member, IEEE), ELENA E. LIMONOVA^{2,3}, (Member, IEEE),
AND VLADIMIR V. ARLAZAROV^{2,3}, (Member, IEEE)

¹Moscow Institute of Physics and Technology, 141701 Dolgoprudny, Russia

²Smart Engines Service LLC, 117312 Moscow, Russia

³Federal Research Center Computer Science and Control, Russian Academy of Sciences, 119333 Moscow, Russia

Corresponding author: Anton V. Trusov (trusov.av@smartengines.com)

This work was supported by the Ministry of Science and Higher Education of the Russian Federation, Internal number 00600/2020/51896 (21.04.2022), under Agreement 075-15-2022-319.


ABSTRACT RFD-like binary descriptors have been designed to be fast and demonstrate good quality in image-matching tasks. One of those descriptors, RFDoc, produces state-of-the-art results when applied to document localization systems. However, the computational efficiency of such descriptors strongly depends on their implementation. In this study, we consider the computation of RFD-like descriptors for an 8-bit single-channel image; provide a detailed implementation of the baseline algorithm; demonstrate its weak points; and propose four modifications. In those modifications, we investigate two ways to accelerate the descriptor computations: 1) compute common operations globally for the entire input image instead of computing them locally for every patch; or 2) use lookup tables to replace the most computationally demanding operations and minimize the number of conversions between integer and floating point types. Experiments on the document identification and localization task on the MIDV-2020 dataset have shown that the modifications with lookup tables are noticeably faster than the baseline, achieving a 2-2.6 times acceleration on x86 and ARM CPUs. Based on experimental results, modifications with global operations may outperform the baseline algorithm if there are many intersecting patches that require descriptor computation. We also demonstrated that one can use any of the proposed algorithms without loss of image-matching quality and without necessity to retrain the parameters of RFD-like descriptors. Finally, we propose an efficient way to compute the descriptors for four orientations of a patch, which is an important part for document location systems. The proposed method reduces a common computation part for four orientations to a single run; thus, four descriptors are computed 3 times faster than the direct computation of the descriptors for four patches, as shown experimentally.

INDEX TERMS Approximation, binary descriptors, RFD, RFDoc, efficient computations, local feature descriptors.

I. INTRODUCTION

Local feature descriptors in computer vision are compact vector representations of small parts of images (patches). The distance between descriptors of similar patches is small but is large for dissimilar patches.

Local feature descriptors are widely used in image matching algorithms to align two or more images of a scene or an object taken from different viewpoints. Such algorithms are

The associate editor coordinating the review of this manuscript and approving it for publication was Abdel-Hamid Soliman .

used in structure-from-motion (SfM) and multiview-stereo (MVS) estimations [1], object tracking [2], ID document localization [3] [4], and other practical tasks. The image matching algorithms that use local-feature descriptors are called feature-based image matching [5].

One of the most well-known local feature descriptors in the literature is SIFT (Scale Invariant Feature Transform) [6]. SIFT computes orientation histograms of weighted gradients in 4 regions around the center of the patch stores than in a vector, normalizes this vector and uses the result as a descriptor. Bay et al. proposed a more computationally efficient

descriptor called SURF (Speed-Up Robust Features) [7], which used histograms of Haar wavelet responses instead of orientation gradients.

SIFT, SURF and many other descriptor computing algorithms (e.g., LBP [8], DAISY [9], LIOP [10]) produce real-valued vectors. The similarity of those descriptors is measured by the l_2 norm, which is computationally expensive. To address this drawback binary descriptors (e.g., BRIEF [11] and its rotation-invariant modification ORB [12]) have been proposed. An advantage of binary descriptors is that the distance between them is the Hamming distance, which is easy to compute. The Hamming distance also allows for fast descriptor matching using multi-index hashing [13].

All of these descriptors are based on the algorithms that the authors proposed based on their expertise in the field of image matching. Such descriptors are called handcrafted conversely to learning-based descriptors [14], which are constructed with regard to training set patches (or a set of pairs of matching and mismatching patches). There are many examples of such algorithms, from simple PCA-SIFT [15] to complex deep learning-based descriptors such as TFeat [16] and HardNet [17], which use convolutional neural networks.

However, neural networks require a long computation time. According to the experimental measurements on modern computing devices [18], such descriptors are hundreds of times slower than simple descriptors that rely on intensity comparison (ORB [12], BEBLID [19], BAD [18]) and tens of times slower than descriptors based on gradient computation (SIFT [6], BinBoost [20]). Therefore, they are not appropriate for real-time on-device applications; thus binary learning-based descriptors such as BGM (Boosted Gradient Maps) [21], BinBoost [20] and RFD (Receptive Fields Descriptors) [22] are of interest in this field of study. These descriptors can combine fast binary vector matching with the quality of the learning-based descriptors. Recently RFDoc descriptor [23] was proposed. That descriptor computation is similar to classic RFD and achieves accurate results in ID document location and classification tasks. The highest quality in the considered task, potential for fast feature-matching, and simple inference, suitable for real-time on-device computation, are why we have chosen RFD and RFDoc descriptors as the focus of this study.

The computation of an RFD-like descriptor for an image patch is a complex process that consists of several stages. First, the patch is blurred. Then the discrete gradient for each pixel is estimated and its magnitude is soft-assigned to one of the eight images called the gradient maps. Then, the response over predetermined regions of the gradient maps is integrated. Finally, this integrated response is binarized by comparison to a threshold value [22], [23]. The speed of the descriptor computation markedly depends on the implementation of each of those stages. However, in those studies, the authors of both RFD and RFDoc paid more attention to the theoretical

aspects and training algorithms of their descriptors than to the implementation.

In this paper, we describe the process of RFD-like descriptor computation in detail. We determine the most computationally demanding operations function and propose ways to remove them by precomputing and approximation. We also note that patches may overlap and that some operations, which are performed patchwise in straightforward implementation, may be more efficient if performed globally for the entire image. Overall, we consider five versions of RFD, including straightforward baseline implementation. We compare them in terms of their computational efficiency and quality of image matching on photos of identity documents from the MIDV-2020 [24] dataset.

Then, we propose a fast method to compute RFD-like descriptors in four orientations. Using several orientations for descriptors is an important approach in document location systems. For example, in [3], the descriptors for an image rotated by 0, 90, 180, and 270 degrees are computed to find the correct orientation of a rectangular ID document. Surprisingly, this trick is not widely used in other image matching tasks.

Straightforward implementation of RFD/RFDoc in four orientations increases computation time fourfold, which may be a problem of this approach for real-time tasks. However, due to the nature of RFD-like descriptors, it is easy to compute four descriptors for four orientations of the patch using the same gradient maps. In this study, we propose a fast algorithm for computing RFD-like descriptors for four orientations and experimentally evaluate its computational efficiency.

To sum it all up, the main contributions of our work are as follows:

- We consider RFD computation in detail and describe five variants: baseline; the one with global blur; the one with global gradient maps; the one with approximate arctangent; and the one with precomputing of features for all values of the gradient.
- We measure the running time of the proposed algorithms on the MIDV-2020 dataset and derive application areas for them. We also show that regardless of their differences, the proposed algorithms produce comparable quality on matching tasks.
- We also present the fast algorithm for computing RFD-like descriptors for four orientations of an image and measure its computational efficiency on the same dataset.

The remainder of this paper is organized as follows. In Section II we describe different methods and data structures in literature that are used to accelerate local feature descriptor computing. Then, we discuss an algorithm for RFD-like descriptor computing and introduce its baseline implementation in Section III. In Section IV, we propose four modifications of that algorithm and discuss their strong sides and shortcomings. In Section V, we introduce a method for fast computation of the RFD-like descriptor in

four orientations. Section VI describes the experimental measurements of running time and quality of the proposed implementations. In Section VII, we discuss the experimental results and limitations of the considered implementations. Finally, Section VIII concludes this article.

II. RELATED WORK

Increasing the computational efficiency of local feature descriptors while preserving their quality has always been a concern for developers.

For example, the SIFT [6] algorithm relied on a scale pyramid to provide scale invariance of its detector-descriptor pair. The *scale pyramid* [25] is a data structure that is widely used in the areas of image processing and computer graphics, and consists of the input image and its downsampled copies. However, the computation and processing of such a pyramid required a significant amount of time, which is why SURF [7] was proposed. SURF has a scale invariant detector based on a box filter and a descriptor based on Haar wavelet responses. Box filter and Haar wavelets require sums over rectangular image regions. Such sums can be computed quickly with the help of an *integral image* [26], which is a 2D generalization of the prefix sum. Integral images are also used in RFD [22] and RFDoc [23] descriptors to compute sums over rectangular pooling regions and in covariance-based descriptor [27] for fast covariance computation.

In efficient implementations of CARD (compact and real-time descriptors) [28] and Zernike moments-based descriptors [29], the authors demonstrated another approach to accelerate computations that used *lookup tables* (LUTs). LUT is an array of precomputed values that are used to replace complex computations with a simple indexing operation. A classic example of applying LUTs is the method of Four Russians [30]. However, LUTs can be used anywhere where computationally expensive functions can be represented with their values at a finite number of points without loss of quality.

There is always a trade-off between computational efficiency and quality. For example, the best quality in image matching tasks is usually demonstrated by deep learning-based descriptors such as HardNet [17] and DOAP [31]. To accelerate such algorithms, it is possible to use the *quantization* approach to replace floating-point operations in neural networks with integer operations [32]. However, it is unlikely to make them suitable for real-time applications because deep learning-based descriptors are slower than the simple ones by several orders of magnitude, as demonstrated by the developers of BAD (Box Average Difference) descriptor [18].

Conversely, fast and simple descriptors usually produce markedly lower quality. One of the fastest descriptors known in the literature is BRIEF [11], which is not robust even to small rotations. This drawback is fixed in the ORB [12] algorithm, which combines the BRIEF descriptor with the FAST [33] keypoint detector, which compensates for rotation. That algorithm computes quickly, particularly its

modification for single instruction multiple data (SIMD) architecture, which is supported by most modern CPUs [34]. However, in terms of quality, it noticeably falls behind other binary descriptors such as BEBLID [19] or RFDoc [23], which are only marginally more complex in terms of computation.

In this study, we consider fast computation of RFD-like (RFD [22] and RFDoc [23]) descriptors because they balance relative simplicity and high matching quality. Therefore, computationally efficient implementation is required to meet the real-time requirements of practical applications. We aim to provide such an implementation.

To achieve this goal, we use the techniques and data structures described above as follows:

- *Scale pyramid*: This technique may be used to provide scale invariance by computing the local feature descriptor at the scale corresponding to its keypoint and is used in the SIFT descriptor. The scale pyramid additional memory to store the downsampled images.
- *Integral images*: These components are useful when fast computation of many sums over rectangles is required, as is the case for RFD and RFDoc descriptors. Integral images also require memory and take time to be computed.
- *Lookup table*: This component is a widely used data structure that replaces computationally expensive operations with indexing over precomputed arrays. In descriptor computing, a lookup table is used in CARD and Zernike moments descriptors. However, the array must be stored in memory, which limits the applicability of large LUTs.
- *Quantization*: This technique allows for replacement of floating-point operations with integer operations, which are more memory- and computationally efficient. Quantization is popular in neural network computing but introduces some level of error because it is an approximation.

III. RFD-LIKE DESCRIPTOR COMPUTATION

Before considering specific aspects of RFD-like descriptor computation, we provide a brief overview of the entire process. The purpose of the considered process is to present the areas around each keypoint on the image as a binary vector of fixed length n . Given the input image I , the list of coordinates of n pooling regions $\gamma_1, \gamma_2, \dots, \gamma_n$, the list of n thresholds t_1, t_2, \dots, t_n , and the coordinates of keypoint k , the computation of the descriptor consists of the following stages:

- **Patch extraction**: The region of fixed size around the keypoint k is extracted from the image I and is referred to as *patch*. The patch size s is the hyperparameter of the descriptor computation algorithm.
- **Patch smoothing**: The patch is blurred to reduce the noise, and the smoothing parameter σ is another hyperparameter.
- **Gradient computation**: The discrete gradient of the patch is computed.

- **Gradient maps computation:** The patch gradient is mapped into eight images according to its orientation. The gradient magnitude for each pixel is mapped by bilinear soft assignment to the two nearest bins with orientations $0, \pi/4, 2\pi/4, \dots, 7\pi/4$.
- **Feature pooling:** For each receptive field γ_i , the values of one of the gradient maps are integrated over the specified receptive area. The result is called a *response* over the receptive field, and the response is then normalized by the sum of the corresponding responses over all receptive fields.
- **Binarization:** The normalized responses are binarized using corresponding thresholds t_i .

These computations are shown in Figure 1 and illustrated in Figure 2, and produce an n -bit binary vector, which is the local feature descriptor. This vector can be used later to estimate the similarity of keypoints in the image-matching algorithm.

The considered algorithm is general for all RFD-like descriptors. However, it lacks many important implementation details, such as data types, used on each step of computation, exact process of gradient orientation computation, shape of receptive fields, etc. We now describe each step in detail. In this study, we consider an 8-bit single channel image as an input because it is one of the most widely used formats in practical applications. However, the algorithm with other input datatypes can be implemented by replacing the types of intermediate variables so that they do not overflow.

A. PATCH EXTRACTION

During patch extraction, a rectangular patch of fixed size $s \times s$ pixels around a given keypoint k is extracted.

To understand this stage better, we consider a keypoint from the local feature descriptor perspective. Keypoints are selected by a keypoint extraction algorithm prior to descriptor computation. The descriptor receives the coordinates of the keypoint (x_p, y_p) , the scale s_p , which determines the upscale or downscale rate for the area around the keypoint, and the orientation θ , which determines the rotation of the patch [6], [7]. Another optional characteristic of a keypoint is its score, which is a value that shows the algorithm’s “confidence” that the point is indeed a keypoint [35]. Score may be used to limit the maximum number of keypoints to process if there were too many detections on the source image.

Given the coordinates (x_p, y_p) of the keypoint, its scale s_p , and rotation angle θ , the coordinates of the patch image (u, v) are mapped to the coordinates of the source image (x, y) via affine transformation (combination of scale transformation, rotation and translation):

$$\begin{aligned} x &= s_p \left(\left(u - \frac{s}{2} \right) \cos \theta - \left(v - \frac{s}{2} \right) \sin \theta \right) + x_p, \\ y &= s_p \left(\left(u - \frac{s}{2} \right) \sin \theta + \left(v - \frac{s}{2} \right) \cos \theta \right) + y_p. \end{aligned} \quad (1)$$

Calculating the affine transformation (1) for each pixel of the patch is computationally demanding. However, if $s_c = 1$

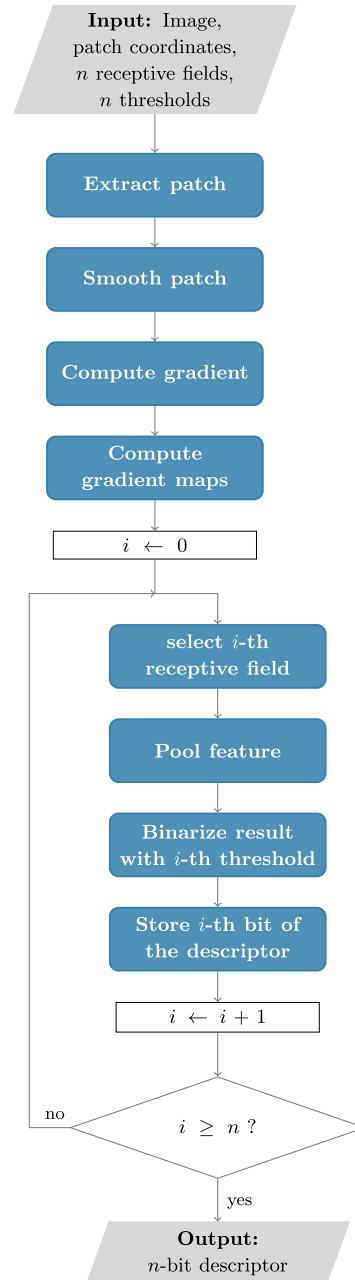


FIGURE 1. Computation of the n -bit RFD-like descriptor for a single patch of an image.

and $\theta = 0$, we have a simple translation instead:

$$\begin{aligned} x &= u - \frac{s}{2} + x_p, \\ y &= v - \frac{s}{2} + y_p. \end{aligned} \quad (2)$$

In this case, we simply copy a region of the source image that appears significantly faster than (1) because there is no additional computation.

Fortunately, we can use a simple scheme (2) instead of a complex scheme (1). To use unit scale $s_c = 1$ we apply a scale pyramid [25]. We construct it before the keypoint

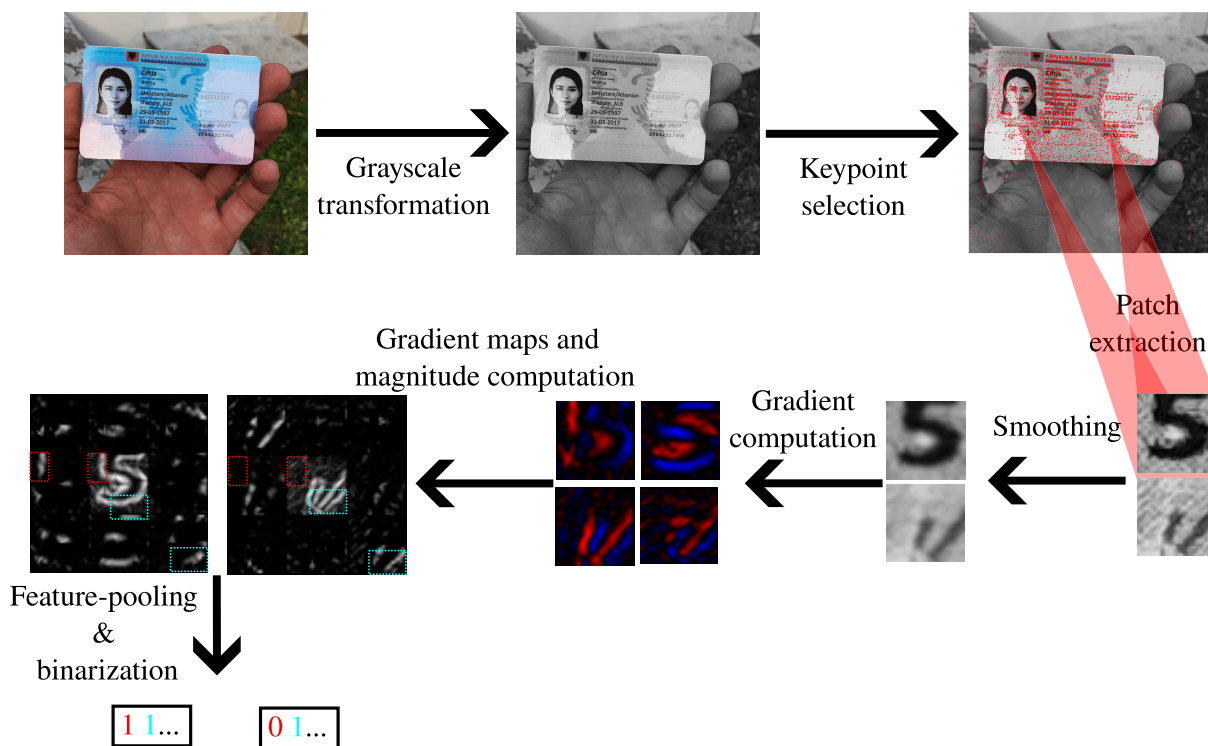


FIGURE 2. Descriptor computation for feature-based image matching. The top row shows image preprocessing and keypoint extraction, and the bottom row shows the steps of RFD-like descriptor computation. The image is a photo of an ID from the MIDV-2020 dataset.

extraction and reuse it for descriptor computation. Then, for each keypoint, we select the nearest layer of this pyramid in terms of scale s_c to copy the patch from it. To address keypoint orientations and set θ to 0, we use keypoints without orientations. For example, in the method proposed in [3], the global rotation of the image is estimated based on straight line segments and compensated before orientation-less feature-based matching of the input document image to a document template.

Thus, we consider the following problem:

- 1) 8-bit single-channel image pyramid as the input for patch extraction;
- 2) Keypoints do not have orientations (θ is set to 0);
- 3) Keypoint scales specify the layer of the scale image pyramid for patch extraction; this layer will be referred to as the source image in this work.

Therefore, the patch extraction is a simple copy of the $s \times s$ region from the source image.

B. PATCH SMOOTHING

Because RFD relies on the directions of gradients, noise of the input patch may produce large errors in the partial derivatives, which is why we introduce a patch smoothing stage before gradient computation. The blur reduces the impact of noise on the gradient.

Another way to estimate gradients with lower noise-caused errors is to use Sobel, derivative-of-Gaussian, or other similar operators.

However, these operators would require two convolutions with such filters (for horizontal and vertical partial derivatives). Considering efficiency, it is better to compute a single convolution with a separable blurring filter (e.g., Gaussian filter) and then use a simple difference scheme twice to find partial derivatives [36].

In this study, we consider that smoothing is performed by a Gaussian filter, and its parameter σ is a hyperparameter of the descriptor computing algorithm. The output of this filter is also an 8-bit image as an input.

In the original RFD [22] and RFDoc [23] papers, the smoothing stage is omitted, but the authors do not describe specifically how the gradient is computed.

C. GRADIENT COMPUTATION

This stage takes the blurred patch as an input and computes vertical and horizontal partial derivatives. This operation can be performed using the simple difference scheme:

$$\frac{\partial}{\partial n} f(n) = (f(n + 1) - f(n - 1))/2, \tag{3}$$

where $f(n)$ is a function of a discrete argument n .

However, if we directly apply (3) to an 8-bit integer patch, the result would either be floating-point or inaccurate when integer division with rounding toward zero is used. Conversion to floating-point data type is time-consuming and unnecessary; instead, we double the value of partial derivatives of

integer patch $P(x, y)$:

$$\begin{aligned} \frac{\partial}{\partial x} P(x, y) &\approx P(x + 1, y) - P(x - 1, y), \\ \frac{\partial}{\partial y} P(x, y) &\approx P(x, y + 1) - P(x, y - 1). \end{aligned} \quad (4)$$

Although the gradient doubles, its orientation is preserved, and the orientation is essential for RFD-like descriptors.

Because the difference of two unsigned 8-bit integers may overflow the 8-bit type, the result should be stored in the signed 16-bit integer type. We propose to convert blurred patches to the signed 16-bit type and then compute the partial derivatives as the difference of two signed 16-bit integers according to (4).

D. GRADIENT MAPS COMPUTATION

Gradient maps are images of the patch that represent the intensity of the gradient in a given direction.

Given partial derivatives of the patch $P_x(x, y)$ (for the horizontal direction) and $P_y(x, y)$ (for the vertical direction), the orientation $\Theta(x, y)$ is defined as:

$$\Theta(x, y) = \text{atan2}(P_x(x, y), P_y(x, y)), \quad (5)$$

where $-\pi \leq \text{atan2}(x, y) \leq \pi$ is a two-argument arctangent function. The function $\text{atan2}(x, y)$ measures the angle between the (x, y) vector and the positive direction of the x -axis.

To compute gradient maps, we also need gradient intensity. In the original RFD paper, there are no specific formulae or instructions. However, the intensity (or magnitude) of the gradient $M(x, y)$ is typically calculated as the l_2 norm:

$$M(x, y) = \sqrt{P_x(x, y)^2 + P_y(x, y)^2}. \quad (6)$$

The authors of the RFDoc descriptor suggested using the l_1 -norm instead and claim that it makes the descriptor more robust and quick to compute:

$$M(x, y) = |P_x(x, y)| + |P_y(x, y)|. \quad (7)$$

Now, we have the orientation $\Theta(x, y)$ and magnitude $M(x, y)$. Then, eight gradient maps $F_0(x, y), \dots, F_7(x, y)$ are computed:

$$\begin{aligned} \phi &= \frac{4(\Theta(x, y) + \pi)}{\pi}, \\ w_1 &= \phi - \lfloor \phi \rfloor, \\ n_0 &= \lfloor \phi \rfloor \bmod 8, \\ n_1 &= (n_0 + 1) \bmod 8, \\ F_{n_1}(x, y) &= \lfloor w_1 M(x, y) \rfloor, \\ F_{n_0}(x, y) &= M(x, y) - F_{n_1}(x, y), \\ F_{i; i \in \{0, \dots, 7\} \setminus \{n_0, n_1\}} &= 0, \end{aligned} \quad (8)$$

where $\lfloor \cdot \rfloor$ denotes rounding down, and $\lceil \cdot \rceil$ denotes rounding to the nearest integer. That operation selects two nearest orientation bins (F_{n_0} and F_{n_1}) for a given pixel and uses bilinear soft assignment to map the gradient magnitude to them.

The computation of gradient maps is a pixelwise operation with integer inputs (P_x and P_y) and integer outputs (F_0, \dots, F_7). Therefore, to prevent unnecessary memory allocation, it may be implemented as a separate subprogram. That subprogram is presented in Algorithm 1. `norm` denotes the vector norm, which is either l_2 (6) or l_1 (7), `floor` is rounding down and `round` is rounding to the nearest integer. The subprogram requires s^2 `norm` operations and up to s^2 `atan2` operations per $s \times s$ patch. Because `atan2` and l_2 -norm computation is markedly more computationally complex than simple additions and multiplications, the gradient map computation is potentially time-consuming.

Algorithm 1 Gradient Map Computation

Input: P_x, P_y are partial derivatives of the patch, $s \times s$ signed 16-bit images

Output: F_0, \dots, F_7, M are gradient maps and gradient magnitude, $s \times s$ unsigned 16-bit images

// zero initializing of the result

for i from 0 to 7 **do**

$F_i \leftarrow 0$

end

$M \leftarrow 0$

// sector size for one bucket

$q \leftarrow 4.0/\pi$

for y from 0 to $s - 1$ **do**

for x from 0 to $s - 1$ **do**

$m \leftarrow \text{norm}(P_x(x, y), P_y(x, y))$

if $m > 0$ **then**

$\phi \leftarrow q * (\text{atan2}(P_x(x, y), P_y(x, y)) + \pi)$

$\psi \leftarrow \text{floor}(\phi)$

$w_1 \leftarrow \phi - \psi$

$n_0 \leftarrow \psi \bmod 8$

$n_1 \leftarrow (n_0 + 1) \bmod 8$

$F_{n_1}(x, y) \leftarrow \text{round}(m * w_1)$

$F_{n_0}(x, y) \leftarrow m - F_{n_1}(x, y)$

end

end

end

In the proposed implementation, the gradient maps are unsigned 16-bit images. We also propose storing the magnitude on the gradient as an unsigned 16-bit image that will be used later during pooling and binarization.

E. FEATURE POOLING

During the feature pooling stage, the responses over rectangular receptive fields are computed. The authors of RFD also propose Gaussian pooling regions, but their experiments have shown that they do not produce descriptors with noticeably higher quality [22]. They also favored rectangular pooling regions because sums over rectangles can be efficiently computed with the help of integral image [26]. In this study, we consider rectangular receptive fields only.

The position of the pooling region γ is determined by five integer variables (x_0, y_0, w, h, c) , where (x_0, y_0) is the position of the top-left corner of the rectangle, w is its width, h is its height, and $0 \leq c < 8$ is the number of gradient maps. The rectangle lies within the $s \times s$ patch.

The response to the pooling region $\gamma = (R(x_0, y_0, w, h), c)$ is:

$$g(\gamma) = \frac{\sum_{x,y \in R} F_c(x, y)}{\sum_{i=0}^7 \sum_{x,y \in R} F_c(x, y)}, \quad (9)$$

where R denotes rectangle (x_0, y_0, w, h) . This response equation is common for RFD and RFDoc descriptors.

We now consider an integral image S for image F :

$$S(x, y) = \begin{cases} \sum_{i=0}^{y-1} \sum_{j=0}^{x-1} F(j, i), & x, y > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

The integral image can be easily computed dynamically because $S(x, y) = S(x - 1, y) + S(x, y - 1) + F(x - 1, y - 1)$ for positive x and y . This image has size $(s + 1) \times (s + 1)$ and allows for simple computation of sums over rectangles on F :

$$\sum_{y=y_0}^{y_1-1} \sum_{x=x_0}^{x_1-1} F(x, y) = S(x_1, y_1) - S(x_0, y_1) - S(x_1, y_0) + S(x_0, y_0) \quad (11)$$

Because gradient maps are constructed according to (8), the sum of all gradient maps in each pixel is the magnitude of the gradient. Therefore, Equation (9) can be simplified:

$$g(\gamma) = \frac{\sum_{x,y \in R} F_c(x, y)}{\sum_{x,y \in R} M(x, y)}. \quad (12)$$

All the sums in (12) can be computed using integral images for gradient maps and magnitude. We propose computing those once per $s \times s$ patch and storing them in $(s + 1) \times (s + 1)$ unsigned 16-bit images.

F. BINARIZATION

In RFD-like descriptors for each receptive field γ_i , there is a corresponding threshold value t_i . Binarization is a simple procedure that determines the value of the i -th bit of the descriptor:

$$b_i = \begin{cases} 0, & g(\gamma_i) < t_i, \\ 1, & g(\gamma_i) \geq t_i. \end{cases} \quad (13)$$

To avoid time-consuming floating-point division, we can combine response computation (12) with binarization (13):

$$b_i = \begin{cases} 0, & \sum_{x,y \in R_i} F_{c_i}(x, y) < t_i \sum_{x,y \in R_i} M(x, y), \\ 1, & \text{otherwise.} \end{cases}, \quad (14)$$

where R_i and c_i are the rectangle and the gradient map index of the receptive field γ_i .

Each bit of the descriptor is calculated by (14) using the integral images of the gradient maps and the gradient magnitude as an input. Now the process of computation of the

binary RFD-like descriptor for 8-bit single channel patch is finished.

It is worth mentioning that receptive fields γ and corresponding thresholds t are parameters of the RFD-like descriptor. They are determined by training on a task-specific dataset [22], [23].

G. DESCRIPTION COMPUTATION: SUMMARY

All steps of the proposed implementation of RFD-like descriptor computation are shown in Algorithm 2 and in Table 1, which represents the data flow in the considered implementation from the $s \times s$ unsigned 8-bit patch to the n -bit descriptor.

The considered algorithm is applied patchwise. For each $s \times s$ patch all the stages up to integral image computation require $O(s^2)$ operations. Then, the responses over receptive fields are computed according to Equation (11); thus, exactly $6n$ integer additions and subtractions, n floating-point multiplications and n comparisons are required to compute the n -bit descriptor.

TABLE 1. Data flow in n -bit RFD computation algorithm for $s \times s$ patch.

#	Data	Image size	Number of images	Data type
0	Input patch	$s \times s$	1	uint8
1	Smoothed patch	$s \times s$	1	uint8
2	16-bit patch	$s \times s$	1	int16
3	Partial derivatives	$s \times s$	2	int16
4	Gradient maps & magnitude	$s \times s$	9	uint16
5	Integral images	$(s + 1) \times (s + 1)$	9	uint16
6	Descriptor	n	1	1-bit

Because the original implementations of the RFD and RFDoc descriptors are not available, we use the proposed algorithm as a baseline in the experiments of this study.

Table 1 shows that in the proposed implementation, all the data are stored as 8- or 16-bit integers in the intermediate steps of computation. Thus, the proposed algorithm is memory-efficient. However, there are still time-consuming floating-point operations in the gradient map computation step. Additionally, this algorithm processes all the patches separately and does not consider that in the image-matching task, all those patches are extracted from the same image. In the following section, we propose four modifications of Algorithm 2 to address these two drawbacks.

IV. ALGORITHM MODIFICATIONS

We will refer to Algorithm 2 as a baseline or **Base**. As mentioned above, that algorithm has two weak points: 1) it requires many computationally expensive operations (for example, atan2 in the gradient orientation computation), and 2) it processes each patch separately. The latter may not seem to be a drawback, but if there are many intersections between patches, it leads to redundant computations. Additionally, it may be faster to perform a single operation over the entire image than to perform it over multiple patches.

Algorithm 2 Detailed RFD Computation Algorithm

Input: I – Image or pyramid of scaled images
 K – list of keypoint coordinates on I
 σ – Smoothing parameter for Gaussian blur
 γ – list of n pooling regions for descriptor
 t – list of n thresholds for each region
Output: D – list of descriptors for each keypoint

for each keypoint k in K **do**
 Select patch P according to k ;
 // Blur patch image
 $P_s \leftarrow \text{GaussianBlure}(P, \sigma)$;
 // Convert patch to 16-bit image to avoid overflow
 $P_{16} \leftarrow \text{Cast8_16}(P_s)$
 // Compute discrete gradient as 2 16-bit images
 $P_x, P_y \leftarrow \text{Gradient}(P_{16})$
 // Compute gradient maps F_i and gradient magnitude
 M as unsigned 16-bit images
 $F_0, \dots, F_7, M \leftarrow \text{GradientMap}(P_x, P_y)$
 for i from 0 to 7 **do**
 $S_i \leftarrow \text{Integrate}(F_i)$
 end
 $S \leftarrow \text{Integrate}(M)$
 // Set up empty n -bit descriptor
 $d \leftarrow 0$
 for $j = 0; j < n; j \leftarrow j + 1$ **do**
 // Compute the response over $\gamma_j = (R, i)$
 $f \leftarrow \text{Sum}(S_i, R)$
 // Compute the response over M
 $m \leftarrow \text{Sum}(S, R)$
 if $f > m * t_j$ **then**
 // Set j -bit of descriptor
 $d_j \leftarrow 1$
 end
 end
 Pass d to D as a descriptor for k
end

A. GS: GLOBAL SMOOTHING

The first step in baseline implementation is smoothing, which is performed by convolution of the patch image with a Gaussian filter. That filter is separable; thus, the convolution can be performed in two steps: convolution with horizontal and vertical filters. In that case, convolution is performed row-by-row over the input image.

In practice, the rows of an image are usually stored consecutively in memory. Therefore, the separable convolution is a cache-friendly operation: CPU cache and preloading provide faster access to values from memory if they are loaded sequentially, which is why a single application of Gaussian blur to the entire image may be faster than several applications to patches if the number of patches is high. Thus, we propose a modification of the baseline algorithm with global smoothing (**GS**). In that case, Gaussian blur with parameter σ is applied to the input image (or all the layers of

the scale pyramid in the case of multiscale keypoints) prior to the patch extraction stage.

B. GM: GLOBAL GRADIENT MAPS AND MAGNITUDE

We can go even further and compute all the steps up to the gradient maps and magnitude computation for the entire input image. In that case, we would have eight images with gradient maps and one with a magnitude of the same size as the input image. Then, we extract patches from them to perform feature pooling and binarization, and thus finish descriptor computation, which is what we propose in a modification of the baseline algorithm with global gradient maps (**GM**). The primary advantage of that algorithm is that it does not compute the same parts of gradient maps in the patch intersection. Additionally, the same considerations about the efficient use of CPU caches as in the **GS** algorithm are applicable in this study. The primary disadvantage is that it computes the gradient maps outside of patches, and those values are not used in later computations. To overcome this drawback, we may use image-size gradient maps but compute only regions covered by at least one patch. Processing patches sequentially, overlapping areas would be computed only once. However, parallel patchwise computations are faster for practical applications.

C. FP: FULL PRECOMPUTING

As mentioned, one of the primary shortcomings of the **Base** algorithm is the use of time-consuming floating-point operations during gradient maps computation. However, according to Equations (5)-(8), the values of the gradient map of a pixel are determined by the values of the partial derivatives P_x, P_y of this pixel. According to (4), the values of partial derivatives are integers lying in the range $-255 \leq P_x, P_y \leq 255$ because the smoothed patch is an 8-bit image. Therefore, there are only 511 possible values for P_x and P_y , which is why we use lookup tables (LUTs) to accelerate computations. We precompute 8-bit values of n_0, n_1 , and 16-bit values F_{n_0}, F_{n_1} in (8) for all possible combinations of P_x and P_y and store them into four 511×511 LUTs accordingly. Thus, the only remaining floating-point operation is the multiplication by a threshold in (14), and the most time-consuming operations, such as atan2 of sqrt , are not required. We call this modification computation with full precompute (**FP**). The only disadvantage of that algorithm is the size of LUTs: $511^2(2 \cdot 1 + 2 \cdot 2)$ bytes, which is approximately 1.5 Mb. The access speed to the values in such LUTs would be limited, because they normally do not fit the L1-cache size on most modern CPUs. Therefore, we need smaller LUTs to make access to them faster.

D. AP: Atan2 PRECOMPUTING

The most time-consuming operation in the Gradient computation is atan2 and square root in the case of the l_2 norm. Therefore, we propose using precomputations for those functions with a fixed number of angles and a quantization procedure that approximates the direction of the gradient by one

of those angles without computing the angle itself. We now consider integer solutions of the following equation:

$$|x| + |y| = N_a \quad (15)$$

There are $4N_a$ points that satisfy this equation. They lie on the square. We number these points with the index τ counterclockwise starting with $\tau = 0$ for point $(-N_a, N_a)$ (see Fig. 3). For any given vector (P_x, P_y) , we define (\hat{P}_x, \hat{P}_y) as the vector that satisfies (15) and has a minimum angle with (P_x, P_y) . We also denote the index of that vector as $\tau(P_x, P_y)$. That index can be computed with integer-only arithmetic operations:

$$l = |P_x| + |P_y|, \\ \tau_0 = \left\lfloor \frac{N_a(P_x + l) + l/2}{l} \right\rfloor, \\ \tau = \begin{cases} \tau_0, & \text{if } P_y < 0 \text{ or } \tau_0 = 0 \\ 4N_a - \tau_0, & \text{otherwise.} \end{cases} \quad (16)$$

The process of computation of the index τ is shown in Fig. 3.

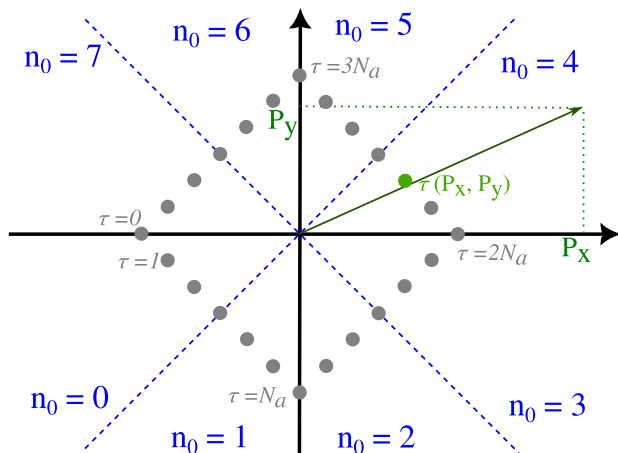


FIGURE 3. Computation of the index τ for a gradient vector (P_x, P_y) .

We compute angles Θ for all the points of (15) and denote them as $\hat{\Theta}(\tau) = \text{atan2}(\hat{P}_y, \hat{P}_x)$. Then, we approximate (5) as:

$$\Theta(x, y) \approx \hat{\Theta}(\tau(P_x(x, y), P_y(x, y))), \quad (17)$$

and (6) as:

$$M = P_x \cos(\Theta) + P_y \sin(\Theta) \approx P_x \cos(\hat{\Theta}) + P_y \sin(\hat{\Theta}), \quad (18)$$

where $\hat{\Theta}$ is an approximation of Θ according to (17). $\hat{\Theta}$, $\cos(\hat{\Theta})$, and $\sin(\hat{\Theta})$ can be precomputed for all $4N_a$ points of the square (15).

Using the proposed approximations and precomputing, we eliminate the most time-consuming functions from the gradient maps computation stage.

We can now eliminate all the floating-point operations from that stage. In (8), n_0 , n_1 and w_1 are determined by angle Θ only; thus, instead of a simple precomputing of $\hat{\Theta}$, $\cos(\hat{\Theta})$, and $\sin(\hat{\Theta})$, we precompute:

- indexes of gradient maps $n_0(\tau)$ and $n_1(\tau)$ as 8-bit values.
- $\hat{w}_1(\tau) = \lfloor w_1(\hat{\Theta}(\tau))N_q \rfloor$ coefficient as an unsigned 32-bit value
- $c_x(\tau) = \lfloor \cos(\hat{\Theta}(\tau))N_q \rfloor$, $c_y(\tau) = \lfloor \sin(\hat{\Theta}(\tau))N_q \rfloor$ coefficients as signed 32-bit values (in case of l_2 norm of gradient only),

where N_q is a predefined integer quantization factor. Now we can determine the magnitude of the gradient for the l_2 norm as:

$$M = \left\lfloor \frac{c_x(\tau)P_x + c_y(\tau)P_y + N_q/2}{N_q} \right\rfloor. \quad (19)$$

In case the of the l_1 norm $M = l = |P_x| + |P_y|$; we have already computed it in (16). The values for the gradient maps are computed as follows:

$$F_{n_1} = \left\lfloor \frac{\hat{w}_1(\tau)M + N_q/2}{N_q} \right\rfloor, \\ F_{n_0} = M - F_{n_1}(x, y). \quad (20)$$

We use N_q equal to a power of 2; thus, the division in (19) and (20) is replaced with a bit-shift, which is simpler to compute.

This modification of the **Base** algorithm approximation and precomputing provides fast integer-only computation of gradient maps. Therefore, we call it modification with **atan2** precomputing (**AP**). The considered modification uses smaller LUTs than **FP**: 3 LUTs of $4N_a$ values for an RFD-like descriptor in the case of the l_1 norm of the gradient and an additional 2 LUTs of the same size in the case of the l_2 norm. N_a and N_q are hyperparameters of the **AP** algorithm: the larger they are, the more accurately **AP** approximates **Base**. However, with the growth of N_a , the size of LUTs increases, which may decrease the computation efficiency. N_q should be sufficiently small such that overflow of integer values does not occur in (19), and (20). N_q should also be a power of 2 to replace divisions with bit-shifts.

E. NOTE ON INEQUIVALENCE

Base, **GS**, **GM**, **AP** and **FP** are modifications of the same algorithm for computing RFD-like descriptors. However, out of the five versions, only **Base** and **FP** are completely equivalent. Thus, using the same image, set of keypoints, and all the parameters as an input for those five modifications, only **Base** and **FP** are guaranteed to produce the same descriptors. We now consider the reason why the others may vary.

GS applies global Gaussian smoothing to the input image and then extracts patches; thus, the values of pixels near the borders of the patches are affected by the values outside of the patches. Therefore, the gradients and the gradient maps may vary, leading to different descriptors. However, this effect is not strong.

The **GM** also uses global Gaussian smoothing; thus, it is different from the **Base**. However, it is also different from the **GS**. If we take a look at (4), we may see that the partial derivatives are not defined on the borders of the patch (top and bottom rows for P_y , and left and right columns for P_x). They

may be set to zero or initialized using the nearest neighbor pixel. The **GM** has no such problem, and the gradient is computed using the input image values lying outside the patch. Those values also vary the value of a descriptor when compared to the **Base** or **GS** but insignificantly.

The **AP** is an approximation of the **Base**; thus, it is not precise. As we increase the values N_q and N_a , the descriptors become increasingly similar.

The inequivalence of the considered versions is shown in Figure 4.

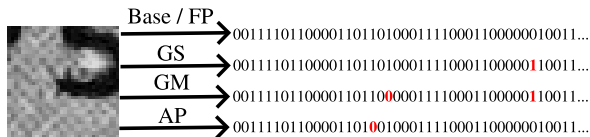


FIGURE 4. The example of proposed RFD modifications demonstrating their inequivalence.

Although the five considered algorithms do not produce exactly the same descriptors, they are all RFD-like descriptor computing algorithms and can all be used for feature-based image matching. We can also use the same receptive fields and thresholds in them and only train them for one algorithm without a loss of quality in image matching. We examine it in detail and demonstrate experimental results in Section VI.

F. NOTE ON PARALLELISM

Parallel computations are standard for high-performance applications because most modern CPUs support multithreading.

In the considered algorithms, it is easy to use parallel computations of the descriptors because the patches are processed separately and they do not use common memory, except the input image as a common source. In the **GS** and the **GM** the rows of global smoothed image, partial derivatives and feature maps can also be computed in parallel.

The importance of the possibility of parallel implementation is the primary reason why we do not consider a version of the **GM** algorithm without excess computations of the gradient maps outside of the patches because that version would require the sequential processing of the patches or accurate synchronization as data races may occur.

G. NOTE ON COMPLEXITY

We now consider the following setup: the computation of the n -bit RFD-like descriptor for m keypoints on the input image of size $h \times w$, with the patch size $s \times s$. As described above, descriptor computation consists of the following stages:

- 1) Smoothing,
- 2) Gradient computation,
- 3) Gradient map computation,
- 4) Integral image computation,
- 5) Descriptor bit computation.

All stages, except the last one, produce an image as a result. The same operations are performed for each pixel of the

result. We denote the complexity of computation of a single pixel as C_1 , C_2 , C_3 , and C_4 for stages 1, 2, 3, and 4, respectively. The computation of a descriptor bit according to Equations (14) and (11) requires exactly six operations of integer addition and subtraction, one floating-point multiplication and one comparison. We also denote its complexity as C_5 , where C_1, \dots, C_5 are not exactly the execution times because the real execution time on the device also depends on the efficiency of loading values from memory through caches. We can use those complexities to show the effects of the proposed modifications.

The complexity of the **Base** algorithm is

$$C = m(s^2(C_1 + C_2 + C_3 + C_4) + nC_5). \quad (21)$$

This algorithm requires $O(s^2)$ of additional memory to store intermediate results of computation, as shown in Table 1.

The **GS** and **GM** computes the first stages for the entire image and not for separate patches. Their complexities are as follows:

$$C_s = hwC_1 + m(s^2(C_2 + C_3 + C_4) + nC_5) \quad (22)$$

and

$$C_m = hw(C_1 + C_2 + C_3) + m(s^2C_4 + nC_5) \quad (23)$$

respectively. These equations show that $C_s < C$ and $C_m < C$ if $hw < ms^2$. Therefore, those modifications are less complex than the **Base**, if the total area of the patches is greater than the total area of the input image, which is unusual in image matching applications. However, we can expect those modifications to work faster because they are more cache-friendly. Those modifications also have one disadvantage: they require $O(s^2 + hw)$ additional memory for intermediate computations. Therefore, if memory allocation or access is slow and the input image is large, they will require a long computation time.

For **AP** and **FP** modifications, their complexities are computed by the Equation (21), as for **Base**, but they are likely to have lower constants C_3 . Because the complexity of `atan2` is significantly higher than that of simpler arithmetic operations. We may also expect noticeable speedup because **AP** and **FP** do not contain any floating-point operations at the gradient maps computation stages. **AP** and **FP** modifications both require $O(s^2)$ memory for intermediate results and some additional memory for LUTs. As calculated above, LUTs require approximately 1.5 Mb of memory for **FP**. For **AP**, the LUT size depends on the number of precomputed points N_a . In case of **AP** with the l_1 norm, there are three tables with a total amount of $4N_a(1 + 1 + 4) = 24N_a$ bytes. For **AP** with l_2 , there are five tables with a total amount of $4N_a(1 + 1 + 4 + 4 + 4) = 56N_a$ bytes. For example, for $N_a = 256$, which we used in the experiments, the sizes of LUTs are 6 Kb and 14 Kb for l_1 and l_2 norms, respectively. Therefore, **AP** is more memory-efficient than **FP**. However, we cannot tell theoretically which one works faster: **FP** has fewer operations, but the LUTs may not fit into an L1 CPU

cache, leading to slower indexing. Therefore, we compare them experimentally in Section VI.

V. FOUR ORIENTATIONS OF A DESCRIPTOR

In [3], the authors propose to use RFD descriptors in ID document location and type identification tasks. They estimate the position before keypoint extraction based on the segments of straight lines that can be found on a document. This process allows for projective distortion to be mitigated, except for the scale and 90-degree rotations. Then, they use a standard feature-based image matching procedure to determine the document type and validate the estimated transformation. The only disadvantage of this method is that feature matching should be performed for four possible rotations of the document (by 0, 90, 180 and 270 degrees).

Because there is a practical requirement to compute the RFD in four orientations, we have investigated this problem. In the case of RFD-like descriptors with rectangular receptive fields, we do not need to rotate a patch to compute the descriptor of the rotated patch but can “rotate” the receptive fields instead (see Fig 5).

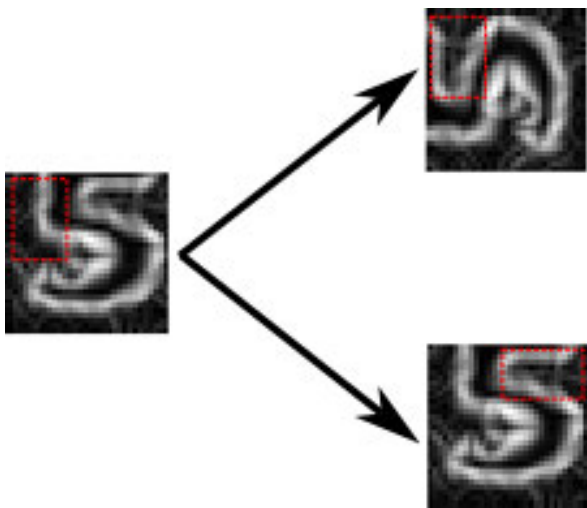


FIGURE 5. Equivalence of patch rotation to receptive field transformation.

More precisely, we propose the following transformation of a receptive field to compute the receptive field for the rotated patch without patch rotation:

$$\begin{aligned} c_1 &= (c_0 + 2) \pmod 8, \\ x_1 &= s - (h_0 + y_0), \\ y_1 &= x_0, \\ w_1 &= h_0, \\ h_1 &= w_0, \end{aligned} \quad (24)$$

where $\gamma_0 = (R(x_0, y_0, w_0, h_0), c_0)$ is a receptive field in the coordinates of the source patch; $\gamma_1 = (R(x_1, y_1, w_1, h_1), c_1)$ is the same receptive field but for the patch that was rotated 90 degrees clockwise in the coordinates of the source patch; and $s \times s$ is the patch size. We need to rotate a rectangle and

to change the index of the gradient map c because when the patch rotates, directions of the gradients change. Because one gradient map corresponds to 45 degrees in the gradient angle space, we increase c by 2.

Given a set of receptive fields for an RFD-like descriptor, we can apply transformation (24) three times sequentially to compute the sets of receptive fields corresponding to orientations of the patch: rotations by 90, 180, and 270 degrees. This approach only requires a small amount of memory to store additional receptive fields: $15n$ additional bytes for the n -bit descriptor because c, x, y, h, w are 8-bit integers. The binary feature descriptors usually consist of 64-512 bits (128 in the proposed experiments), which results in 1-8 Kb of additional memory. Now that the three additional sets of receptive fields are computed, for each patch, we can compute the gradient maps and magnitudes only once and then apply four sets of receptive fields to obtain four descriptors, corresponding to four possible orientations of the patch; this process is markedly faster than computing the descriptors for four orientations of the image.

We now consider the complexity of the computation of the RFD-like descriptor in four orientations using the proposed baseline algorithm described in Section III. If we compute four descriptors by directly rotating the input image, the complexity would be:

$$C_{f_0} \approx 4C = 4m(s^2(C_1 + C_2 + C_3 + C_4) + nC_5), \quad (25)$$

where we use the same notation as in (21). In these calculations, we neglect the complexity of the tree rotations of the input image by 90 degrees, which is a relatively simple and fast operation.

If we use the proposed trick to simultaneously compute four descriptors, the complexity would be:

$$C_{f_1} = m(s^2(C_1 + C_2 + C_3 + C_4) + 4nC_5) + 3nC_r, \quad (26)$$

where C_r is the complexity of recomputation of a receptive field according to transformation (24). Because $C_5 > 0$ and $C_r > 0$, the proposed approach would not work exactly 4 times faster than the naive approach. The speed-up is empirically evaluated in the next section.

VI. EXPERIMENTS

A. EXPERIMENTAL SETUP

1) DATASET

To measure the computational efficiency and quality measurements, we consider the feature-based document location and identification on the *photos* subset of MIDV-2020 dataset [24]. This subset consists of 1000 images of unique mock identity documents of 10 types (100 images per type), each with unique text field values and unique artificially generated faces. Images are taken on smartphone cameras in challenging capturing conditions, which include complex backgrounds (keyboard, text or outdoors scenes), low lighting, high projective distortions, etc. There are two objectives: to identify a document and to find its location. Identification

in that case is a 10-class classification task. The location of the document is determined by its quadrangle because all documents are plane rectangles, and the photo introduces projective distortion to this quadrangle. Examples of two images from the dataset with document quadrangles are presented in Fig. 6.



FIGURE 6. Images from MIDV-2020 dataset with document quadrangles: green for ground truth, and magenta for the detected in our experiments.

2) IMAGE-MATCHING CONFIGURATION

In this study, we focus on descriptor computing. However, to evaluate the proposed algorithms in terms of quality and efficiency in the selected task, we need other modules of the image matching algorithm as well. In the experiments of this study, we use a basic image matching algorithm with the following steps:

- 1) An input RGB image is converted to grayscale.
- 2) 3-layer scale pyramid is constructed, where the first layer is the proposed image, the linear sizes of the second layer are $2/3$ of the input, and the linear sizes of the third layer are $1/2$ of the input.
- 3) For each layer of the pyramid, keypoints are extracted using the YACIPE algorithm [35]. If the number of keypoints is more than $T_{kp} = 50000$, T_{kp} keypoints with the highest scores are selected.
- 4) RFD-like descriptors are computed using one of the algorithms described in Sections III and IV. We use 128-bit descriptors of 32×32 patches with receptive fields and thresholds selected according to the RFDoc [23] training algorithm. For the AP algorithm, we set $N_a = 256$ and $N_q = 2^{20}$ (see Subsection IV-D for details).
- 5) The Hamming distance between all the descriptors of the image keypoints and the descriptors of the template image keypoints is calculated (for all 10 document templates). If the Hamming distance D_h is lower than

$T_h = 32$, we consider that there is a match between the corresponding points on the image and template.

- 6) All the matching points are used in the RANSAC algorithm to estimate the projective transformation H that maps a region of the image to the document template coordinates. The sampling probability of the pair of points is set to be $p_s = (T_h - D_h)/T_h$; thus, the pairs of points that are more likely to match are more likely to be selected in RANSAC. We use 10^6 iterations in the primary loop of RANSAC and fine-tune the result. As in [3], we do not select close points in the RANSAC hypothesis.

The projective transformation H is the answer of the proposed algorithm. In the practical image-matching algorithm, the limit on the number of keypoints T_{kp} and the number of iterations of RANSAC are markedly smaller. For example, in [3], they are 1500 and 8000, respectively. However, in this study, we focus on descriptor computing; thus, we increased T_{kp} to investigate the dependence of the efficiency on the number of keypoints. The number of RANSAC iterations is high to reduce the influence of the transformation estimation stage on image matching quality. We also directly compute the Hamming distance between all the descriptor pairs to achieve reproducible results, as suggested in [23].

To match the image of a document to a template image, we compute the descriptors of the keypoints of the document template in advance. Because there are 10 document types in the MIDV-2020 photos dataset, we use 10 templates. We only use keypoints that lie in the static regions of the document (i.e., those parts that do not contain personal data), as suggested in [3].

The considered algorithm is not the best for solving document identification and location. The algorithms that combine local and global features of an image show better quality than those that rely on local features only [4]. However, in this study, we investigate local feature descriptors, which is why we selected a basic feature-based image-matching approach.

3) METRICS AND HARDWARE

The primary characteristic that we try to estimate experimentally in this work is the computational efficiency of computing RFD-like descriptors on CPUs. Thus, we run the algorithm described above using AMD Ryzen 9 5950X and Amlogic S922X Cortex-A53 CPUs. The first has $\times 86_64$ architecture, which is common for desktop computers, and we denote this CPU as $\times 86$. The architecture of the second CPU is ARM, which is popular on mobile devices, and we denote this CPU as ARM.

We also measure the quality of the feature-based document location and identification. Because identification in the proposed case is a simple 10-class classification, we use the rate of correctly classified documents (accuracy) to measure its quality. In document location, we try to estimate the projective transformation M that maps document quadrangle m on the image to the template rectangle t ($t = M(m)$). We let H denote the estimated projective transformation and q – the

estimated quadrangle ($q = H^{-1}(t)$). As in [24], we use the following scores to measure the location quality:

$$IoU(q, m, t) = \frac{\text{area}(M(q) \cap t)}{\text{area}(M(q)t)}, \quad (27)$$

$$D(q, m, t) = \max_i \frac{\|t_i - H(m_i)\|_2}{P(t)}, \quad (28)$$

where $P(t)$ is the perimeter of the template rectangle t . While (27) computes the Intersection over Union score in the t (template) coordinates, (28) computes the normalized maximum distance between corresponding vertices in the q (estimated) coordinates.

B. SPEED COMPARISON

We experimentally compare the running time of the five considered algorithms for RFD-like descriptor computations: **Base** (see Section III), **GS** (see Subsection IV-A), **GM** (see Subsection IV-B), **AP** (see Subsection IV-D) and **FP** (see Subsection IV-C). We compare the proposed modifications (**GS**, **GM**, **AP** and **FP**) to the baseline algorithm (**Base**) and not to an algorithm from any standard library because there are no publicly available implementations of RFD [22] or RFDoc [23] descriptors.

Thus, we run the image matching algorithm described above in a single thread of ARM and $\times 86$ CPUs. We measure the time of descriptor computing for each of 1000 images of the MIDV-2020 photo dataset. We repeat all measurements twice on the ARM CPU and 10 times on the $\times 86$ CPU to obtain more stable results. Therefore, we have 2000 measurements on ARM and 10000 on $\times 86$ CPUs in total. Different images have different numbers of keypoints; thus, different numbers of patches are extracted, and the descriptor computation time varies.

The dependency of the computation time on the number of keypoints for the proposed algorithms is shown in Fig. 7. According to Fig. 7 for **Base**, **AP**, and **FP**, this dependency is linear. That is because the number of keypoints equals the number of patches. **GS** and **GM** require additional time during the pre-computation stage, after which the dependence is also linear.

Figure 7 shows that algorithms with lookup tables (**FP** and **AP**) show the best performance. When the number of keypoints increases, algorithms with global precomputing stages (**GS** and **GM**) become more efficient in comparison to the baseline. When this number is greater than 10000, **GS** outperforms the baseline, and when it is greater than 20000, **GM** outperforms both **GS** and the baseline. However, as mentioned above in image matching tasks, the limit on the number of keypoints is usually noticeably smaller than 10000; thus, algorithms with global precomputing are not applicable in those tasks.

To compare the efficiency of the considered algorithms, we calculate the mean time of the descriptor computation per image in the proposed setup T_m and the estimated time T_e (using linear estimation via least squares estimation) for 1500 keypoints. This number of keypoints is chosen to be

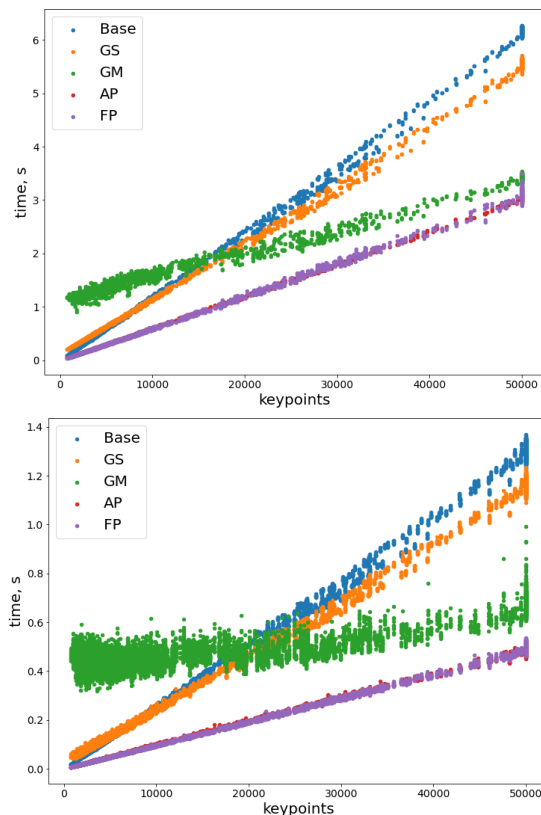


FIGURE 7. Computation time for different RFD algorithms using l_1 norm on ARM (top) and $\times 86$ (bottom) CPUs.

equal to their maximum amount in [3]. We use least squares estimation over images with fewer than 10000 keypoints to estimate T_e , which allows us to consider linear dependency and ignore less stable measurements that appear on the images with the larger number of keypoints (see Figure 7). We also do not consider computing average time per keypoint because **GS** and **GM** include global computation stages; thus, their running time is not directly proportional to the number of keypoints. Table 2 shows T_m and T_e and their errors (standard error for T_m and least squares error for T_e).

The processing time measurements in Table 2 suggest that precomputing significantly accelerates computation of RFD-like descriptors: **FP** is 2 times faster than **Base** on ARM CPU and 2.6 times faster on $\times 86$ CPU. **AP** shows nearly identical efficiency as **FP** for the l_1 norm of the gradient but is approximately 10% slower with the l_2 norm.

Algorithms with a global precomputing stage (**GS** and **GM**) only achieve better efficiency if the number of patches is sufficiently high. If the number of patches is low (i.e., insufficient intersections), **AP**, **FP** and even **Base** algorithms perform better.

From the experimental results, we can conclude that in most cases, the best algorithm to use would be **FP** because it is easy to implement and achieves the best computational efficiency. However, if the l_1 norm is used in the gradient maps (e.g., in the RFDoc descriptor), it is better to use the

TABLE 2. Average descriptor computation time per image (T_m) and per 1500 keypoints (T_e).

Impl.	Norm	ARM	
		T_m , ms	T_e , ms
Base	l_1	1810 ± 40	174.4 ± 0.9
GS		1700 ± 30	276.1 ± 0.8
GM		1801 ± 15	1175 ± 5
AP		910 ± 20	88.6 ± 0.3
FP		910 ± 20	85.8 ± 0.6
Base	l_2	1870 ± 40	181.7 ± 0.8
GS		1770 ± 40	283.4 ± 0.7
GM		1887 ± 15	1274 ± 5
AP		1030 ± 20	101.0 ± 0.4
FP		910 ± 20	85.5 ± 0.6
Impl.	Norm	x86	
		T_m , ms	T_e , ms
Base	l_1	382 ± 4	36.64 ± 0.10
GS		358 ± 3	63.2 ± 0.2
GM		479.9 ± 0.8	443.2 ± 1.2
AP		146.7 ± 1.5	14.60 ± 0.03
FP		145.5 ± 1.5	14.17 ± 0.03
Base	l_2	396 ± 4	38.51 ± 0.08
GS		373 ± 3	65.1 ± 0.2
GM		508.6 ± 0.7	477.8 ± 1.1
AP		160.9 ± 1.6	16.04 ± 0.04
FP		145.4 ± 1.5	14.13 ± 0.03

AP algorithm, which achieves the same computational efficiency but uses markedly less memory. Overall, we recommend using the **FP** algorithm of RFD-like descriptor computing as the default and switching to the **AP** algorithm if low memory consumption is critical (e.g., on embedded devices).

C. FOUR ORIENTATIONS SPEED

In Section V, we described a fast way to compute the RFD-like descriptor for four orientations of a patch simultaneously. We stated that the proposed method is more efficient than computing four descriptors one-by-one.

Now, we experimentally measure the achieved gain. For each image in the dataset, we measure the computation time of RFD-like descriptors (t_1) and the time of simultaneous computing of the descriptors in four orientations (t_4). Table 3 shows the average speed-up of the proposed algorithm $4t_1/t_4$ and the standard deviation of this speed-up over 1000 images of the MIDV-2020 photo dataset.

TABLE 3. Speed-up of fast four RFD orientations with standard deviations over the images of the dataset.

Impl.	Norm	ARM	x86
AP	l_1	3.33 ± 0.02	2.84 ± 0.04
FP		3.32 ± 0.07	2.82 ± 0.04
AP	l_2	3.40 ± 0.02	2.93 ± 0.04
FP		3.32 ± 0.06	2.82 ± 0.04

The proposed method allows us to compute the descriptors approximately 3 times faster (marginally more on ARM CPU, marginally less on x86 CPU) than by simply rotating the input image 3 times with recomputation of descriptors.

D. QUALITY ASSURANCE

We have already mentioned in Section IV that the **Base**, **GS**, **GM**, **AP**, and **FP** algorithms are not equivalent. Now, we must ensure that the choice of the algorithm does not affect the image matching quality. Thus, we evaluated each of the five algorithms on the photos subset of the MIDV-2020 dataset as described above. The receptive fields and thresholds of the RFD-like descriptors have been computed in advance as described in [23]. For each algorithm, we recompute only the descriptors of the keypoints of the templates before matching them to the images of the dataset.

To solve the image identification task (10-class classification), we select the template that has the greatest number of inliers of RANSAC (the keypoint pairs that satisfy the estimated transformation). To evaluate the location quality, we computed IoU (27) and D (28) metrics and compared them to thresholds of 0.9 and 0.2, respectively, as in [24]. Results are presented in Table 4.

TABLE 4. Feature-based document identification and location accuracy.

Impl.	Accuracy, %	$D < 0.02$, %	IoU > 0.9 %
Base	91.1	79.9	81.2
GS	91.3	80.7	81.6
GM	92.4	80.3	81.1
AP	91.5	80.3	80.9
FP	91.1	79.9	81.2

Table 4 shows that all the considered descriptor computing algorithms demonstrate similar quality on the image matching task. The observed quality difference is inconsistent over the metrics and insignificant, considering that there are only 1000 images in a dataset. These results indicate that we can use any proposed descriptor computing algorithm and do not need to recompute receptive fields and thresholds specifically.

VII. DISCUSSION

In this study, we proposed a detailed baseline algorithm for RFD-like descriptor computing, which was not previously reported in the literature or available in open source libraries. Then, we introduced four modifications aimed at speeding up its computation. Those modifications can be divided into two groups.

Algorithms of the first (**GS** from Subsection IV-A and **GM** from Subsection IV-B) perform several stages of descriptor computation for the entire image instead of a single patch. Those algorithms were suggested because they do not compute the values in patch intersections more than once. Such computations are also cache-friendly; thus, they should work faster than patchwise computations. However, the experiments of this study demonstrated that those algorithms work faster than the baseline only if the number of keypoints on an image is high. Therefore, they are not appropriate for typical machine learning tasks.

Another group of algorithms (**FP** from Subsection IV-C and **AP** from subsection IV-D) use LUTs to accelerate the

computation of gradient maps – the part of an algorithm with the most complex operations. We proposed two different algorithms because one is simpler and contains fewer operations (**FP**), and the other is more memory-efficient (**AP**). Based on the experimental results, **FP** is the fastest algorithm, but with the l_1 norm, which is used in RFDoc descriptors, **AP** achieves nearly identical performance; thus, we suggest using the **FP** algorithm for RFD-like descriptor computing by default and choosing the **AP** algorithm if stricter memory restrictions are present.

We have experimentally evaluated all considered modifications and the baseline on a real-world image matching task using a photo subset of the MIDV-2020 dataset. That experiment did not show any noticeable gap in quality between modifications and the baseline, even though no specific adaptation of receptive field coordinates or threshold values for modifications was performed (i.e., we only trained the baseline).

Additionally, we have proposed a fast algorithm for computing RFD-like descriptors in four orientations. Instead of rotating the input image or the patch, it “rotates” the receptive fields, and thus, nearly all the stages of the descriptor computing (except for feature pooling and binarization) become common for all the orientations and are only performed once per patch. The experiments of this study confirmed that the proposed trick allows for approximately 3 times speed-up compared to computing four descriptors. Therefore, the proposed trick should be used if computation of the descriptors in four orientations is required. However, not all applications require an input image to be rotated, so one should not use that method by default.

A. STUDY LIMITATIONS

This study had three primary limitations: the set of computing devices, the dataset size, and the fact that we only considered one image matching application.

We start by considering a set of computing devices. We tested the execution time on two CPUs of two of the most popular architectures: $\times 86$ (Ryzen 9 CPU) and ARM (Cortex-A53 CPUs). The qualitative results of those two CPUs coincide, and the quantitative results only vary marginally. For example, considering simultaneous computation of the descriptors in four orientations, we achieved $\times 2.8$ speedup on $\times 86$ and $\times 3.3$ time speedup on ARM. It is possible that on different computing devices, we could see other results. However, because descriptor computing is usually performed on CPUs and $\times 86$ and ARM are two of the most widely used CPU architectures, we believe the proposed results to be representative.

The other two limitations (the dataset size and the single image-matching task) are important in the context of quality evaluation. We could expect that on a large dataset, **Base** and **FP** would achieve marginally better quality because those descriptors work exactly as they were trained, while the others (**GS**, **GM**, **AP**) use approximations (see Subsection IV-E for more details). However, according to the proposed

experiments, the gap in quality is so small that it is not observable on the considered dataset. We also considered only one image matching application: matching identity document from an image to known template (as did the authors of RFDoc descriptors). That is why we cannot be completely sure that the gap in quality between 1) **Base** and **FP** algorithms and 2) the other considered algorithm would remain negligible in other tasks. Even if additional experiments showed that this gap indeed exists and is critical for practical applications, we could retrain their descriptors according to RFD [22] or RFDoc [23] training algorithms to compensate.

B. FUTURE DIRECTIONS

It is possible to improve the performance of RFD-like descriptor computing on CPUs with efficient use of SIMD instructions, which we did not consider in this work. In the future, the concepts and methods developed in this study could be generalized for other computing devices, including FPGAs and ASICs. We also hope that this study can inspire the creation of new descriptors that would be specifically designed to be fast and memory-efficient.

VIII. CONCLUSION

In this paper, we have proposed five detailed algorithms for RFD-like descriptor computation for 8-bit images. We have empirically shown that any of those algorithms can be used in image-matching tasks without quality loss and without the need to recompute the coordinates of receptive fields and thresholds of descriptors. The modifications that rely on lookup tables in computing the gradient maps achieved the best computational efficiencies among the proposed algorithms: they work 2-2.6 times faster than the baseline. Thus, we recommend using the proposed modifications in image matching applications instead of standard RFD-descriptor computing algorithms.

We have also presented an algorithm that allows for the computation of RFD-like descriptors for four orientations of a patch, which works 3 times faster than the naive method with image rotation. We recommend using this algorithm whenever computation of a descriptor in multiple orientations is required.

REFERENCES

- [1] K. Gao, H. Aliakbarpour, J. Fraser, K. Nouduri, F. Bunyak, R. Massaro, G. Seetharaman, and K. Palaniappan, “Local feature performance evaluation for structure-from-motion and multi-view stereo using simulated city-scale aerial imagery,” *IEEE Sensors J.*, vol. 21, no. 10, pp. 11615–11627, May 2021.
- [2] S. Gauglitz, T. Hoellerer, and M. Turk, “Evaluation of interest point detectors and feature descriptors for visual tracking,” *Int. J. Comput. Vis.*, vol. 94, no. 3, pp. 335–360, Sep. 2011.
- [3] N. Skoryukina, V. V. Arlazarov, and D. P. Nikolaev, “Fast method of ID documents location and type identification for mobile and server application,” in *Proc. ICDAR*, New York, NY, USA, Feb. 2020, pp. 850–857, doi: [10.1109/ICDAR.2019.00141](https://doi.org/10.1109/ICDAR.2019.00141).
- [4] N. S. Skoryukina, V. V. Arlazarov, and A. N. Milovzorov, “Memory consumption reduction for identity document classification with local and global features combination,” *Proc. SPIE*, vol. 11605, pp. 116051G1–116051G8, Jan. 2021, doi: [10.1117/12.2587033](https://doi.org/10.1117/12.2587033).

- [5] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, "Image matching from handcrafted to deep features: A survey," *Int. J. Comput. Vis.*, vol. 129, pp. 23–79, Aug. 2020.
- [6] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2003.
- [7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Comput. Vis. Image Understand.*, vol. 110, no. 3, pp. 346–359, 2008.
- [8] T. Ojala, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 971–987, Aug. 2002.
- [9] E. Tola, V. Lepetit, and P. Fua, "DAISY: An efficient dense descriptor applied to wide-baseline stereo," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 5, pp. 815–830, May 2010.
- [10] Z. Wang, B. Fan, G. Wang, and F. Wu, "Exploring local and overall ordinal information for robust feature description," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 1, pp. 2198–2211, Nov. 2016.
- [11] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2010, pp. 778–792.
- [12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2564–2571.
- [13] M. Norouzi, A. Punjani, and D. Fleet, "Fast exact search in Hamming space with multi-index hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1107–1119, Jun. 2014.
- [14] C. Leng, H. Zhang, B. Li, G. Cai, Z. Pei, and L. He, "Local feature descriptor for image matching: A Survey," *IEEE Access*, vol. 7, pp. 6424–6434, 2019.
- [15] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2004, p. 1.
- [16] V. Balntas, E. Riba, D. Ponsa, and K. Mikolajczyk, "Learning local feature descriptors with triplets and shallow convolutional neural networks," in *Proc. BMVC*, 2016, vol. 1, no. 2, p. 3.
- [17] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas, "Working hard to know your neighbor's margins: Local descriptor learning loss," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–15.
- [18] I. Suarez, J. M. Buenaposada, and L. Baumela, "Revisiting binary local image description for resource limited devices," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 8317–8324, Oct. 2021.
- [19] I. Suárez, G. Sfeir, J. M. Buenaposada, and L. Baumela, "BEBLID: Boosted efficient binary local image descriptor," *Pattern Recognit. Lett.*, vol. 133, pp. 366–372, May 2020.
- [20] T. Trzcinski, M. Christoudias, P. Fua, and V. Lepetit, "Boosting binary keypoint descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2874–2881.
- [21] T. Trzcinski, M. Christoudias, V. Lepetit, and P. Fua, "Learning image descriptors with the boosting-trick," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–16.
- [22] B. Fan, Q. Kong, T. Trzcinski, Z. Wang, C. Pan, and P. Fua, "Receptive fields selection for binary feature description," *IEEE Trans. Image Process.*, vol. 23, no. 6, pp. 2583–2595, Jun. 2014.
- [23] D. P. Matalov, E. E. Limonova, N. S. Skoryukina, and V. V. Arlazarov, "RFDoc: Memory efficient local descriptors for ID documents localization and classification," in *Proc. ICDAR*, in Lecture Notes in Computer Science, vol. 12822, J. Lladós, D. Lopresti, and S. Uchida, Eds. London, U.K.: Springer, 2021, pp. 209–224, doi: [10.1007/978-3-030-86331-9_14](https://doi.org/10.1007/978-3-030-86331-9_14).
- [24] K. B. Bulatov, E. V. Emelyanova, D. V. Tropin, N. S. Skoryukina, Y. S. Chernyshova, A. V. Sheshkus, S. A. Usilin, Z. Ming, J.-C. Burie, M. M. Luqman, and V. V. Arlazarov, "MIDV-2020: A comprehensive benchmark dataset for identity document analysis," *Comput. Opt.*, vol. 46, no. 2, pp. 252–270, 2022, doi: [10.18287/2412-6179-CO-1006](https://doi.org/10.18287/2412-6179-CO-1006).
- [25] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA Eng.*, vol. 29, no. 6, pp. 33–41, 1984.
- [26] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, Jun. 2001, pp. 1–11.
- [27] O. Tuzel, F. Porikli, and P. Meer, "Region covariance: A fast descriptor for detection and classification," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2006, pp. 589–600.
- [28] M. Ambai and Y. Yoshida, "CARD: Compact and real-time descriptors," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 97–104.
- [29] S.-K. Hwang, M. Billingham, and W.-Y. Kim, "Local descriptor by Zernike moments for real-time keypoint matching," in *Proc. Congr. Image Signal Process.*, 2008, pp. 781–785.
- [30] V. L. Arlazarov, Y. A. Dinitz, M. Kronrod, and I. Faradzev, "On economical construction of the transitive closure of an oriented graph," in *Proc. USSR Acad. Sci.*, vol. 194, no. 3, pp. 487–488, 1970.
- [31] K. He, Y. Lu, and S. Sclaroff, "Local descriptors optimized for average precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 596–605.
- [32] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021, *arXiv:2103.13630*.
- [33] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2006, pp. 430–443.
- [34] P. Viswanath, P. Swami, K. Desappan, A. Jain, and A. Pathayapurakkal, "Orb in 5 ms: An efficient SIMD friendly implementation," in *Proc. Asian Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 675–686.
- [35] A. Lukoyanov, D. Nikolaev, and I. Kononenko, "Modification of YAPE keypoint detection algorithm for wide local contrast range images," *Proc. SPIE*, vol. 10696, pp. 1069616-1–1069616-8, Apr. 2018, doi: [10.1117/12.2310243](https://doi.org/10.1117/12.2310243).
- [36] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2006.



network implementations and computer vision algorithms.

ANTON V. TRUSOV (Member, IEEE) was born in Moscow, Russia, in 1997. He received the master's degree from the Moscow Institute of Physics and Technology, in 2021. He is currently pursuing the Ph.D. degree with the Moscow Institute of Physics and Technology, National Research University. Since 2017, he has been working as a Programmer and a Technician with Smart Engines Service LLC, and also with FRC CSC RAS, since 2021. His research interests include device-efficient neural



ELENA E. LIMONOVA (Member, IEEE) was born in Dolgoprudny, Moscow, Russia, in 1993. She received the master's degree in physics, mathematics, and computer science from the Moscow Institute of Physics and Technology, in 2017. She is currently pursuing the Ph.D. degree with FRC CSC RAS. Since 2016, she has been working as a Programmer and a Technician with Smart Engines Service LLC. Her research interests include neural network compression and image recognition on mobile devices.



VLADIMIR V. ARLAZAROV (Member, IEEE) was born in Moscow, USSR, Russia, in 1976. He received the Specialist degree in applied mathematics from the Moscow Institute of Steel and Alloys, in 1999, and the Ph.D. degree in computer science, in 2005. Since 1999, he has been working with the Institute for Systems Analysis, Russian Academy of Sciences (currently the Federal Research Center Computer Science and Control, Russian Academy of Sciences), Moscow, as a Researcher, a Senior Researcher, and the Head of the Laboratory. Since 2016, he has been the General Director of the Smart Engines Service LLC, Moscow. Since 2018, he has been working with the Institute for Information Transmission Problems, Russian Academy of Sciences, as a Senior Researcher. Since 2012, he has also been working with the Moscow Institute of Physics and Technology (National Research University), Moscow, as an Associate Professor. He has published over 150 articles and authored 30 patents. His research interests include computer vision and document analysis systems.

...