Published in Image Processing On Line on 2018–07–25.
Submitted on 2017–06–19, accepted on 2018–02–24.
ISSN 2105–1232 © 2018 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2018.213

Automatic Detection of Internal Copy-Move Forgeries in Images

Thibaud Ehret

CMLA, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France
thibaud.ehret@cmla.ens-cachan.fr

Abstract

This article presents an implementation and discussion of the recently proposed 'Efficient Dense-Field Copy-Move Forgery Detection' by Cozzolino et al. This method is a forgery detection based on a dense field of descriptors chosen to be invariant by rotation. Zernike moments were suggested in the original article. An efficient matching of the descriptors is then performed using PatchMatch, which is extremely efficient to find duplicate regions. Regions matched by PatchMatch are processed to find the final detections. This allows a precise and accurate detection of copy-move forgeries inside a single suspicious image. We also extend successfully the method to the use of dense SIFT descriptors and show that they are better at detecting forgeries using Poisson editing.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation and usage instruction are included in the `README.txt` file of the archive.

Keywords: forgery detection; copy-move forgery; PatchMatch; Zernike moment; SIFT descriptor

1 Introduction

Copy-move forgeries correspond to the case where a region is copied from the image and then pasted in another position of the same image. The copy could be modified in the meantime, but the detection presented here is only guaranteed to work when a rotation has been applied. Nevertheless we shall see that this method is also mildly robust to other perturbations such as noise. This type of forgery, which can easily be performed with most image editing software tools, is usually used to hide undesired details in the image, or to add new details. Detecting such modifications can be challenging, especially when carried out by a professional image editor.

[1] https://doi.org/10.5201/ipol.2018.213

Many different methods have been developed to detect such modifications. In [8], Farid presented a large number of such methods: from the simplest ones such as pixel-based methods to much more complex physics-based methods by which, for example, the illumination of the scene is studied. In this study, we focus on a single family of detection methods. These methods, specialized im detecting copy-move forgeries, are organized in the following way. First, descriptors are computed on the suspicious image. These descriptors are chosen to represent well the local behavior of the suspicious image. They are then matched to each other during a matching step to detect abnormal similarities within the image. In the final step, post-processing is applied to refine detections and diminish the number of false positives. The first papers on this subject proposed classic sparse image descriptors such as SIFT [12] or SURF [14]. A recent benchmark [4] of many different methods based on both sparse descriptors and dense descriptors shows that dense descriptors largely outperform sparse descriptors. For this reason the paper under review [5] chose to use dense descriptors. Matching the descriptors, or equivalently finding the nearest neighbor for each descriptor, is a well studied problem but actually quite hard. Many different approaches have been developed but all actually compute approximate matches. The approximation can be computed using partition trees such as the well known KD-trees [3], or VP-trees [17], or adapted hashing functions such as LSH [10, 9]. A recent breakthrough for image specific algorithms is *PatchMatch*, developed by Barnes et al. [1, 2]. This algorithm relies on the structural properties of the image and multiple sampling, which in practice performs extremely well on dense descriptors. It is also the matching algorithm adopted by the reviewed method.

The rest of the paper is organized as follows: the descriptors used for the processing, Zernike moments, are presented in Section 2. The matching algorithm *PatchMatch* and the modifications implemented to improve the matching for our problem are presented in Section 3. Several pre- and post-processing steps presented in Section 4 are then applied to obtain precise and accurate detections. Finally, a couple of experiments where we show success and limits of the method are presented in Section 5, where we successfully extend the method to dense SIFT descriptors..

2 Zernike Moments

The absolute values of Zernike moments are rotation invariant descriptors that are adapted to the task of detecting image repetitions up to a rotation. We address in this section their efficient computation by the Xin et al. [16] method. For a patch u of an image I (defined for now as continuous on $[-1,1]^2$), the Zernike A_{nm} moment of order n and angular dependence m, for $m < n$ and $m \equiv n[2]$, is defined by

$$A_{nm} = \frac{n+1}{\pi} \int\int_{\mathbb{D}} u(x,y)\overline{V_{nm}}(x,y)dxdy, \tag{1}$$

where V_{nm} is the complex Zernike polynomial of order n and angular dependence and $A_{nm} = 0$ otherwise (therefore in the following study, the case $m \not\equiv n[2]$ will be omitted). The polynomials are defined over the unit disk $\mathbb{D}(0,1)$ and usually expressed in polar coordinates ρ and θ. They are fully defined by

$$V_{nm}(\rho,\theta) = R_{nm}(\rho)e^{im\theta}, \tag{2}$$

$$R_{nm}(\rho) = \sum_{s=0}^{\lfloor\frac{n-m}{2}\rfloor} \frac{(-1)^s(n-s)!\rho^{n-2s}}{s!(\lfloor\frac{n+m}{2}\rfloor - s)!(\lfloor\frac{n-m}{2}\rfloor - s)!}. \tag{3}$$

Zernike moments can be interpreted as scalar products between the image u and the Zernike polynomials. The absolute value of these moments is rotationally invariant. Indeed, consider a rotation

of angle α of the given image, or equivalently, a rotation of angle α of the Zernike polynomials

$$V_{nm}^{\alpha} = V_{nm} e^{i\alpha},$$

the studied moment A_{nm}^{α} would then be

$$\begin{aligned} A_{nm}^{\alpha} &= \frac{n+1}{\pi} \int\int_{\mathbb{D}} u(x,y) \overline{V_{nm}^{\alpha}}(x,y) dxdy && (4)\\ &= \frac{n+1}{\pi} \int\int_{\mathbb{D}} u(x,y) \overline{V_{nm}}(x,y) e^{-i\alpha} dxdy && (5)\\ &= \frac{n+1}{\pi} \left(\int\int_{\mathbb{D}} u(x,y) \overline{V_{nm}}(x,y) dxdy \right) e^{-i\alpha} && (6)\\ &= A_{nm} e^{-i\alpha}, && (7) \end{aligned}$$

and therefore

$$|A_{nm}^{\alpha}| = |A_{nm}|, \tag{8}$$

which confirms the rotation invariance of the magnitude of these moments.

In our case, the moments will be computed on each patch of the image (considered as a small image itself). Within the same framework presented previously, one can also define translation invariant descriptors by computing a dense field of descriptors over the entire image. The computation is trickier than it appears at first sight because the image (noted u and so far considered continuous) is sampled over a Cartesian grid whereas Zernike polynomials are more naturally expressed in polar coordinates. The sampled version of I and u with a sampling step $\frac{1}{sp}$ will be noted respectively $\tilde{I}$ and $\tilde{u}$. The coordinates (x_0, y_0) denote the center of $\tilde{u}$ in $\tilde{I}$ and $\tilde{u}$ is defined on $[\![-sp, sp]\!]$. The easiest solution would be to estimate the integral from Equation (1) by sampling over the Cartesian grid by (see wikipedia[2] for the definition of $atan2$)

$$A_{nm} = \sum_{s_x=-sp}^{sp} \sum_{s_y=-sp}^{sp} \tilde{u}(s_x, s_y) \overline{V_{nm}}(\sqrt{s_x^2 + s_y^2}, atan2(s_y, s_x)). \tag{9}$$

Xin et al. have shown in [16] that this approximation suffers from multiple drawbacks and is not properly rotation invariant. The rotational invariance being one of the most important features of the Zernike moments in our case, we shall use the more accurate estimation proposed in [16]. First, we need to resample the image. Indeed the original Cartesian sampling is not adapted to the unit disk. Figure 1a shows that whatever the choice of pixels taken to do the integration on, it will never truly form a disk; therefore leading to approximation in the computations. An adequate image partitioning is required to care for the rotation invariance. First, the unit disk is split into K regions along the radial direction. These regions are then divided into "pixels" with the same area. For this reason each region is split into $(2k+1)L$ subregions. This sampling of the unit disk is presented in Figure 1b. It has the nice property of not losing resolution compared to the original image. In addition, the new pixels have approximately the same area as the old ones. The image is considered piecewise constant over these "pixels". The value over such a pixel, noted $f(\rho_{kl}, \theta_{kl})$, is computed by image interpolation. Xin et al. recommended the usage of a bicubic interpolation. This yields

$$f(\rho_k, \theta_{kl}) = \sum_{s_x=\lfloor \rho_k \cos(\theta_{kl}) \rfloor - 1}^{\lfloor \rho_k \cos(\theta_{kl}) \rfloor + 2} \sum_{s_y=\lfloor \rho_k \sin(\theta_{kl}) \rfloor - 1}^{\lfloor \rho_k \sin(\theta_{kl}) \rfloor + 2} \tilde{u}(s_x, s_y) h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right), \tag{10}$$

[2] https://en.wikipedia.org/wiki/Atan2

(a) Cartesian sampling of the unit disk (b) Polar sampling of the unit disk

Figure 1: Different types of samplings for the Zernike moments.

where

$$h(x) = \begin{cases} \frac{3}{2}|x|^3 - \frac{5}{2}|x|^2 + 1 & \text{if } |x| \leq 1 \\ -\frac{1}{2}|x|^3 + \frac{5}{2}|x|^2 - 4|x| + 2 & \text{if } 1 < |x| \leq 2 \\ 0 & \text{otherwise} \end{cases}. \tag{11}$$

We can now express Zernike moments in polar coordinates by a simple change of variable,

$$A_{nm} = \frac{n+1}{\pi} \int_0^{2\pi} \int_0^1 u(\rho, \theta) \overline{V_{nm}}(\rho, \theta) \rho d\rho d\theta. \tag{12}$$

This leads to the following estimate of A_{nm} by sampling the integral over the newly created regions

$$A_{nm} = \frac{n+1}{\pi} \sum_k \sum_l f(\rho_k, \theta_{kl}) w_{nm}(\rho_k, \theta_{kl}), \tag{13}$$

where w_{nm} corresponds to

$$w_{nm}(\rho_k, \theta_{kl}) = \int \int_{\Omega_{kl}} R_{nm}(\rho) e^{-im\theta} \rho d\rho d\theta \tag{14}$$

$$= \left(\int_{\rho_{k,1}}^{\rho_{k,2}} R_{nm}(\rho) \rho d\rho \right) \left(\int_{\theta_{kl,1}}^{\theta_{kl,2}} e^{-im\theta} d\theta \right), \tag{15}$$

where the support of the pixel Ω_{kl} is parameterized by $\rho_{k,1}, \rho_{k,2}, \theta_{kl,1}, \theta_{kl,2}$ (see Figure 2). Besides the two previous integrals can be computed in closed form

$$\int_{\rho_{k,1}}^{\rho_{k,2}} R_{nm}(\rho) \rho d\rho = \left(\sum_{s=0}^{\lfloor \frac{n-m}{2} \rfloor} \frac{(-1)^s (n-s)! \left(\rho_{k,2}^{n-2s+2} - \rho_{k,1}^{n-2s+2} \right)}{(n-2s+2) s! (\lfloor \frac{n+m}{2} \rfloor - s)! (\lfloor \frac{n-m}{2} \rfloor - s)!} \right), \tag{16}$$

and

$$\int_{\theta_{kl,1}}^{\theta_{kl,2}} e^{-im\theta} d\theta = \begin{cases} \theta_{kl,2} - \theta_{kl,1} & \text{if } m = 0 \\ \frac{i}{m} \left(e^{-im\theta_{kl,2}} - e^{-im\theta_{kl,1}} \right) & \text{otherwise} \end{cases}. \tag{17}$$

Xin et al. have shown that computing the moments using this framework allows for a much better rotation invariance compared to the regular Cartesian approximation. The sums in Equation (13) can be reordered to compute efficiently the entire set of Zernike moments for the entire image $\tilde{I}$.

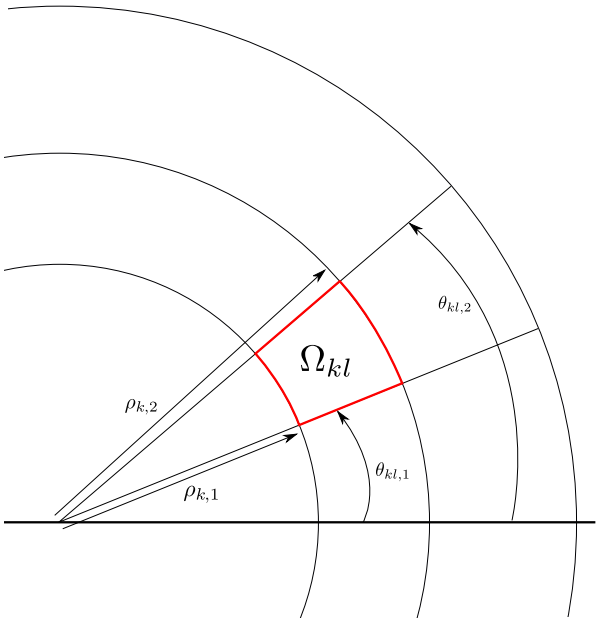


Figure 2: Notations for Equations (15), (16) and (17).

Indeed A_{nm} can be rewritten

$$\begin{aligned}
A_{nm} &= \frac{n+1}{\pi} \sum_k \sum_l f(\rho_k, \theta_{kl}) w_{nm}(\rho_k, \theta_{kl}) \\
&= \frac{n+1}{\pi} \sum_k \sum_l \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right) w_{nm}(\rho_k, \theta_{kl}) \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \sum_k \sum_l \tilde{u}(s_x, s_y) h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right) w_{nm}(\rho_k, \theta_{kl}) \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) \left(\sum_k \sum_l h\left(\rho_k \cos(\theta_{kl}) - s_x\right) h\left(\rho_k \sin(\theta_{kl}) - s_y\right) w_{nm}(\rho_k, \theta_{kl}) \right) \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) F^{nm}_{s_x, s_y},
\end{aligned} \tag{18}$$

which can now be used to define an "image" of zernike moments such that

$$\begin{aligned}
A_{nm}(x_0, y_0) &= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{u}(s_x, s_y) F^{nm}_{s_x, s_y} \\
&= \frac{n+1}{\pi} \sum_{s_x} \sum_{s_y} \tilde{I}\left(x_0 + s_x, y_0 + s_y\right) F^{nm}_{s_x, s_y}.
\end{aligned} \tag{19}$$

Equation (19) shows that the A_{nm}s can be computed efficiently by convolving the image $\tilde{I}$ with the newly defined filter F^{nm}. A null extension is chosen for the convolution for efficiency and simplicity. Indeed, the moments computed on the border won't have any of the properties required for the matching, whatever the extension (in particular invariance to rotation is the hardest to ensure); therefore they should be discarded anyway in the following steps and the choice should be made only for computation efficiency. The pseudocode for the full computation of the dense field of moments for a given image is given in Algorithm 1. It yields both translation and rotation invariant descriptors

with which copy-move forgeries will be detected. The two versions of Zernike moments will be compared for the problem of forgery detection with rotation in Section 5.1.

Algorithm 1: Zernike moments computation

input : An image I, a half-size of patch sp, a maximum order o
output: A dense descriptor map Z
foreach (n,m) *s.t.* $n \in [\![1,o]\!], m \in [\![1,n]\!]$ *and* $n \equiv m[2]$ **do**
  for $s = 0$ *to* $\frac{n-m}{2}$ **do**
    $C_{nms} \leftarrow \frac{(-1)^s (n-s)!}{(n-2s+2)s!(\frac{n+m}{2}-s)!(\frac{n-m}{2}-s)!}$ *Compute the coefficients for Equation (16)*
  end
end
for $\rho = 0$ *to* $sp-1$ **do**
  for $\theta = 0$ *to* $4(2\rho+1)-1$ **do**
    foreach (n,m) *s.t.* $n \in [\![1,o]\!], m \in [\![1,n]\!]$ *and* $n \equiv m[2]$ **do**
      $w \leftarrow 0$ *Initialize the weights*
      for $s = 0$ *to* $\frac{n-m}{2}$ **do**
        $w \leftarrow w + C_{nms}\left(\left(\frac{\rho+1}{sp}\right)^{n-2s+2} - \left(\frac{\rho}{sp}\right)^{n-2s+2}\right)$ *See Equation (16)*
      end
      if $m = 0$ **then**
        $w \leftarrow w \times \frac{2\pi}{4(2\rho+1)}$ *See Equation (17)*
      else
        $w \leftarrow w \times \frac{i}{m}\left(e^{-im\frac{2(\theta+1)\pi}{4(2\rho+1)}} - e^{-im\frac{2\theta\pi}{4(2\rho+1)}}\right)$ *See Equation (17)*
      end
      $F^{nm}_{s_x s_y} \leftarrow 0$
      for $s_x = \left\lfloor \rho\cos\left(\frac{2\theta\pi}{4(2\rho+1)}\right)\right\rfloor - 1$ *to* $\left\lfloor \rho\cos\left(\frac{2\theta\pi}{4(2\rho+1)}\right)\right\rfloor + 2$ **do**
        for $s_y = \left\lfloor \rho\sin\left(\frac{2\theta\pi}{4(2\rho+1)}\right)\right\rfloor - 1$ *to* $\left\lfloor \rho\sin\left(\frac{2\theta\pi}{4(2\rho+1)}\right)\right\rfloor + 2$ **do**
          $F^{nm}_{s_x s_y} \leftarrow h\left(\rho\cos\left(\frac{2\theta\pi}{4(2\rho+1)}\right) - s_x\right) h\left(\rho\sin\left(\frac{2\theta\pi}{4(2\rho+1)}\right) - s_y\right) w + F^{nm}_{s_x s_y}$ *See Equation (18)*
        end
      end
    end
  end
end
foreach *filter* F^{nm} *computed previously* **do**
  Convolve I (with a null extension) with F^{nm} and store its modulus in ZM_{nm}.
end

3 Patchmatch

In order to detect forgeries, we'll have to find the nearest neighbors of the descriptors from Section 2. Usually finding nearest neighbors in high dimension is a rather difficult problem but, for the specific case of images, the *PatchMatch* heuristic is faster and more efficient than non-specific heuristics.

3.1 Original *PatchMatch* Algorithm [1]

PatchMatch was developed to match dense descriptors of a query image A to the dense descriptors of a target image B. Its idea is to exploit the overlap of neighboring patches in both images to accelerate the search. Indeed for two patches that are almost perfectly overlapping, chances are that their most similar patches will be in the same respective overlapping position.

Given an image $B, \mathcal{U}(B)$ represents a uniform distribution over all patches $\mathbf{b}$ of B. U, respectively D, L and R, correspond to functions taking a patch $\mathbf{a}$ and returning respectively the patch just above it, below it, on its left and on its right (always according to the usual sampling grid considered with numerical images). An approximate nearest neighbor field (abbreviated ANNF) is a data structure associated with two images which, for each patch of the first image, associates a patch from the second image, the goal being that the associated patch is the most similar one. An ANNF will be manipulated like a function from the set of patches of the first image to the set of patches of the second image.

The *PatchMatch* algorithm computes an ANNF by leveraging the local similarity property ubiquitous in natural images. This means that if a patch $\mathbf{a}$ in A is matched to a patch $\mathbf{b}$ in B then the neighbors of $\mathbf{a}$ are more likely to be matched to the corresponding neighbors of $\mathbf{b}$. This idea is complemented by a random search for the best possible corresponding patches and is then iterated a fixed number of times. For many applications, a few iterations (between five to ten) provide an acceptable result. The random search step is done at multiple scales. The local random searches find the true minimum when a good match has been found whereas the search at larger scales avoids focusing too much on a non-optimal region of the image by looking also in other regions. When matching with the same image, there's a risk for a patch to find itself (which is obviously the best candidate possible). Because that case is not interesting, matches at a distance smaller than τ_m in the image are rejected. A full region is rejected instead of just the patch itself because the overlapping patches are often really good candidates as well (though also not interesting for the forgery detection).

The pseudocode of the algorithm is presented in Algorithm 2. The extension of *PatchMatch* from image patches (with which the algorithm was originally defined) to dense descriptors is natural and changes nothing in the different steps of the algorithm. Later on in this review, *PatchMatch* (and several modified versions) will be used with descriptors instead of patches, in particular the Zernike moments presented in Section 2. In the following, we will keep referring to the elements as patches even though they can be descriptors as well. The basic propagations are shown in Figure 3a.

3.2 Generalized *PatchMatch*

In a follow-up paper [2], the authors of *PatchMatch* proposed several other versions of the algorithm to either solve slightly different but similar problems or to improve the speed and/or the quality of the matching. In particular, they considered the problem of matching modulo rotations and scaling. While the modification for rotation and scaling yields a slow convergence rate in practice (the space of possible matches becoming far too large for the random search), and is not adapted to rotation invariant descriptors (it explores the entire space of rotations), we will use the modifications suggested when applying *PatchMatch* to the same image. The idea behind this modification is to try to take advantage of the natural symmetric consistency within the matches. Indeed, if $\mathbf{b}$ is the current candidate as nearest neighbor of $\mathbf{a}$, then $\mathbf{a}$ is a good candidate for the nearest neighbor of $\mathbf{b}$.

This will be combined with the modifications suggested by Cozzolino et al. in [5] to take rotation invariant descriptors into consideration and improve their matching. In order to increase the quality of the matches – when taking rotations into consideration – two more propagations such as the one presented in Section 3.1 are added. These two propagations correspond to a propagation

Algorithm 2: Original *PatchMatch* algorithm

input : A, B two images, d a distance function for patches, N a number of iterations, w (usually the width of B) and $\alpha = \frac{1}{2}$ parameters for the random search, a minimum distance threshold τ_m

output: $\mathbf{f}$, an ANNF from A to B

Initialization:

foreach *patch* $\boldsymbol{a}$ *in* A **do**

  $\mathbf{f}(\mathbf{a}) \sim \mathcal{U}(B)$

end

Iterations:

for $n = 1$ *to* N **do**

  if n *is even* **then**

    foreach *patch* $\boldsymbol{a}$ *in* A *from the top-left to the bottom-right* **do**

      Propagation:

      If $R(\mathbf{f}(L(\mathbf{a})))$ is not closer than τ_m from $\mathbf{a}$: $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), R(\mathbf{f}(L(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$

      If $D(\mathbf{f}(U(\mathbf{a})))$ is not closer than τ_m from $\mathbf{a}$: $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), D(\mathbf{f}(U(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$

      Random search:

      $i \leftarrow 0$

      while $w\alpha^i > 1$ *Multiple scale random search*

      do

        $r \sim \mathcal{U}([\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!] \times [\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!])$

        if $\boldsymbol{f}(\boldsymbol{a}) + r$ *is further than* τ_m *from* $\boldsymbol{a}$ **then**

          $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{f}(\mathbf{a}) + r\}} \{d(\mathbf{p}, \mathbf{a})\}$

        end

        $i \leftarrow i + 1$

      end

    end

  else

    foreach *patch* $\boldsymbol{a}$ *in* A *from the bottom-right to the top-left* **do**

      Propagation:

      If $L(\mathbf{f}(R(\mathbf{a})))$ is not closer than τ_m: $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), L(\mathbf{f}(R(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$

      If $U(\mathbf{f}(D(\mathbf{a})))$ is not closer than τ_m: $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), U(\mathbf{f}(D(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$

      Random search:

      $i \leftarrow 0$

      while $w\alpha^i > 1$ *Multiple scale random search*

      do

        $r \sim \mathcal{U}([\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!] \times [\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!])$

        if $\boldsymbol{f}(\boldsymbol{a}) + r$ *is further than* τ_m *from* $\boldsymbol{a}$ **then**

          $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{f}(\mathbf{a}) + r\}} \{d(\mathbf{p}, \mathbf{a})\}$

        end

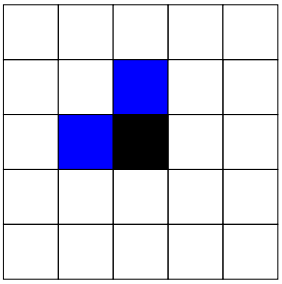
        $i \leftarrow i + 1$

      end

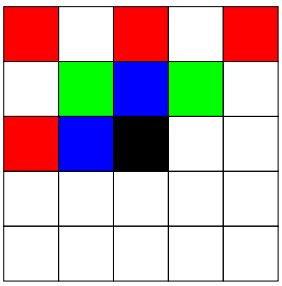
    end

  end

end



(a) Original *PatchMatch* propagation: elements in blue are propagated to the one in black



(b) Improved *PatchMatch* propagation: elements in blue (original propagations), green (completed zero order propagations) and red (first order propagations) are propagated to the one in black

Figure 3: Different types of *PatchMatch* propagations.

from the top-left patch as well as from the top-right one. This means that information is propagated to all connecting patches. These propagations are represented in green in Figure 3b. A second set of propagations, called first-order propagations, are also added to the list of propagations. These propagations are different from the previous ones because they are not simply a displacement propagation. They propagate linearly varying offset modifications. In order to simplify the notations for some of the equations, we also introduce the optimal displacement $\mathbf{s}$ alongside an ANNF $\mathbf{f}$ such that $\mathbf{f}(\mathbf{a}) = \mathbf{a} + \mathbf{s}(\mathbf{a})$. In the following the ANNF and the optimal displacement will be used interchangeably, the one leading to the simplest equations will be chosen. For two functions f and g, the composition of functions is written $f \circ g = f(g(.))$. Using a rough Taylor expansion, an approximation of a function f can be made

$$\begin{aligned} f(x+1) &\approx f(x) + \frac{df(x)}{dx} \\ &\approx f(x) + \frac{f(x) - f(x-1)}{1} \\ &\approx 2f(x) - f(x-1). \end{aligned} \tag{20}$$

This approximation is classic in numerical analysis. Translating Equation (20) to the current problem, A being any composition of L, R, U and D, leads to

$$\mathbf{s}(\mathbf{a}) \approx 2\mathbf{s}(A(\mathbf{a})) - \mathbf{s}(A \circ A(\mathbf{a})). \tag{21}$$

The first-order nodes used for these types of propagations are represented in red in Figure 3b. In the end, a total of eight propagations of the different types is done during every iteration of *PatchMatch*.

Adding all these modifications to the original *PatchMatch* algorithm presented in Section 3.1 gives us the version used to solve the problem at hand. The final modified *PatchMatch* algorithm is summarized in Algorithm 3.

The convergence of this matching algorithm has been proved (for the original one as well as for a modified version like this one) using the framework presented in [7].

4 Forgery Detection

Once the displacement map has been computed using *PatchMatch*, forgeries can be detected. In order to avoid false alarms, multiple pre-processing and post-processing techniques are applied. The different steps for the detection are:

Algorithm 3: Modified *PatchMatch*

Data: A an image, d a distance function for patches, N a number of iterations, w (usually the width of A) and $\alpha = \frac{1}{2}$ parameters for the random search, k the number of nearest neighbors to be computed, a minimum distance threshold τ_m

Result: $\mathbf{f}$, a k-ANNF from A to A

Initialization:

foreach *patch* $\boldsymbol{a}$ *in* A **do**

  $\mathbf{f}(\mathbf{a}) \sim \mathcal{U}(A)$

end

Iterations:

for $n = 1$ *to* N **do**

  foreach *patch* $\boldsymbol{a}$ *in* A *from the top left to the bottom right if* n *is odd, from the bottom right to the top left otherwise* **do**

    Propagation:

    if n *is even* **then**

      For each propagation, reject candidates that are closer than τ_m from $\mathbf{a}$ in the image:

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), R(\mathbf{f}(L(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *Zero-order propagation*

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), D(\mathbf{f}(U(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *Zero-order propagation*

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), D \circ L(\mathbf{f}(U \circ R(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *Zero-order propagation*

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), D \circ R(\mathbf{f}(U \circ L(\mathbf{a})))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *Zero-order propagation*

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(L(\mathbf{a})) - \mathbf{s}(L \circ L(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *First-order propagation*

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(U(\mathbf{a})) - \mathbf{s}(U \circ U(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *First-order propagation*

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(U \circ L(\mathbf{a})) - \mathbf{s}(U \circ L \circ U \circ L(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *First-order propagation*

      $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{a} + 2\mathbf{s}(U \circ R(\mathbf{a})) - \mathbf{s}(U \circ R \circ U \circ R(\mathbf{a}))\}} \{d(\mathbf{p}, \mathbf{a})\}$ *First-order propagation*

    else

      Do the reverse propagations

    end

  end

  foreach *patch* $\boldsymbol{a}$ *in* A **do**

    Symmetrization:

    $\mathbf{f}(\mathbf{f}(\mathbf{a})) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{f}(\mathbf{a})), \mathbf{a}\}} \{d(\mathbf{p}, \mathbf{f}(\mathbf{a}))\}$

  end

  foreach *patch* $\boldsymbol{a}$ *in* A **do**

    Random search:

    $i \leftarrow 0$

    while $w\alpha^i > 1$ *Multiple scale random search*

    do

      $r \sim \mathcal{U}([\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!] \times [\![-\lfloor w\alpha^i \rfloor; \lfloor w\alpha^i \rfloor]\!])$

      if $\boldsymbol{f}(\boldsymbol{a}) + r$ *is further than* τ_m *from* $\boldsymbol{a}$ **then**

        $\mathbf{f}(\mathbf{a}) \leftarrow \text{argmin}_{\mathbf{p} \in \{\mathbf{f}(\mathbf{a}), \mathbf{f}(\mathbf{a}) + r\}} \{d(\mathbf{p}, \mathbf{a})\}$

      end

      $i \leftarrow i + 1$

    end

  end

end

1. Application of a median filter.
2. Computation of an error map and early detections.
3. Remove detections that are too small.
4. Remove detections that are too close.
5. Symmetrize the detections.
6. Dilate the detections.

The full algorithm performing forgery detection is presented in Algorithm 10. All these steps will be developed and explained in the following sections. It is also possible to detect internal copies where the copies have been flipped. It only requires to compute descriptors for the flipped patches; this is easily done by computing descriptors on the flipped image and flip the resulting descriptor map. The distance function used for *PatchMath* in this case is the distance from the regular descriptors to the flipped version. Every other step stays the same. Another possible solution is to use flip-invariant descriptors such the one presented in [18] or [15] so to avoid two computations of the descriptors; this is not studied in this article though.

4.1 Median Filter

The first step is to apply a median filter to the displacement map. It smooths the displacement and improves the detection rate by reducing the overall error computed in the following step. The algorithm used for the median filtering is shown in Algorithm 4.

Algorithm 4: medianFilter algorithm

input : A displacement map D, the radius of the filter ρ_m
output: A filtered displacement map $\tilde{D}$
foreach *pixel* p *in* D **do**
  $\tilde{D}(p) \leftarrow median(\{D(p') \mid p' \in \mathbb{D}(p, \rho_m)\})$
end

4.2 Error Filter and Detection

A copy-move forgery (as its name indicates) copies a region of the image, possibly rotates it, and pastes it at a different image position. Both regions (the original one and the forged one) can then be matched by an affine transform. The idea presented in [5] is then to compare the field of matches created using *PatchMatch* to the one computed using the best affine transform based on the field of *PatchMatch* matches.

Let $P \in \mathcal{M}^{3\times n}(\mathbb{R})$ be the homogeneous coordinates of a set of points. We set

$$P = \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \\ 1 & \dots & 1 \end{pmatrix}. \tag{22}$$

Let Δ be the displacement of these same points (still in homogeneous coordinates). Let $A \in \mathcal{M}^{3\times 3}(\mathbb{R})$ be a matrix representing an affine transform. We are looking for a region such that the transformation error defined by

$$\epsilon(P) = \min_{A \in \mathcal{M}^{3\times 3}(\mathbb{R})} \|\Delta - (AP - P)\|^2, \tag{23}$$

is small. The best affine transform minimizing Equation (23) can be estimated using a simple Least Square Regression, i.e.

$$A = \underset{A \in \mathcal{M}^{3\times 3}(\mathbb{R})}{\operatorname{argmin}} \|\Delta - (AP - P)\|^2. \tag{24}$$

The solution of Equation (24) can be expressed as

$$A = (P + \Delta)P^T(PP^T)^{-1}, \tag{25}$$

therefore the error can be estimated without a minimization step by replacing A by its solution from Equation (25)

$$\begin{aligned}\epsilon(P) &= \|\Delta - (AP - P)\|^2 \\ &= \|\Delta - ((P + \Delta)P^T(PP^T)^{-1}P - P)\|^2 \\ &= \|\Delta - \Delta P^T(PP^T)^{-1}P\|^2 \\ &= \|\Delta(I - H)\|^2, \text{ where } H = P^T(PP^T)^{-1}P.\end{aligned} \tag{26}$$

The set of points P used to estimate this error will be a disk of radius ρ_e and the error will be associated to the center of the disk, which delivers an actual error map. In practice the result is independent of the coordinates in P, as long as Δ is set coveniently. Therefore H can be precomputed for all regions. The error detection map is given by thresholding by τ the error map estimated with Equation (26). The features and the error filter aren't properly defined on the boundary (it would require to define the behavior of these objects outside the image as well), therefore detections on the boundary are not well defined either. This is why the boundary of the image is removed during the detection step so as to avoid any false detection due to a false matching. The algorithm used to compute the error detection map is summarized in Algorithm 5.

Algorithm 5: errorDetectionFilter algorithm

input : A displacement map D, the radius of the filter ρ_e, τ a threshold, *sp* the half-size of the patch used to compute the features

output: An error detection map E

$P \leftarrow \mathbb{D}(0, \rho_e)$ *Define the homogeneous coordinates*

$H \leftarrow P^T(PP^T)^{-1}P$ *See Equation (26)*

$H' \leftarrow I - H$

foreach *pixel p in D* **do**

  Let Δ be the displacements corresponding to $\mathbb{D}(p, \rho_e)$

  $e \leftarrow \|\Delta H'\|^2$ *See Equation (26)*

  $E(p) \leftarrow \begin{cases} 1 & \text{if } e < \tau \\ 0 & \text{otherwise} \end{cases}$

end

Remove any detection at the boundary (of size *sp*) of E

The next steps of the detection correspond to multiple post-processings trying to minimize false alarms and to adjust the detection mask to the actual forged regions.

4.3 Size Filter

The first post-processing step tries to remove "unlucky" matches. Sometimes good matches can be created purely by chance, but in practice these fortuitous regions are very small. The application of a filter which cuts connected components whose area is below a certain threshold removes these "unluckily" matched regions. Such a filter is presented in Algorithm 6.

Algorithm 6: sizeFilter algorithm

input : A detection mask M, area threshold τ_s
output: A filtered detection mask $\tilde{M}$
foreach *4-connected component c in M* **do**
  if *c is smaller than* τ_s *pixels* **then**
    $\tilde{M}(c) \leftarrow 0$ *Assignment for each pixels in c*
  end
end

4.4 Minimum Displacement Filter

As a last step to remove false alarms, candidates that are too close in the image plane are filtered out. If the copy is too close to the original, humans detect easily the forgery. Thus, if a detected forged region is too close to its copy, both are considered false alarms and therefore discarded. This idea is synthesized in Algorithm 7. While this post-processing is necessary for the method to work well, it's actually redundant with the exclusion region from *PatchMatch*. Indeed setting τ_m to the maximum between τ_m and τ_d forces all matches of *PatchMatch* to be at a distance superior or equal to τ_d. Therefore there would be no matches to be rejected with this displacement filter. The deliberate choice to still present the displacement filter was made to describe the originally proposed post-processing steps, without mixing ideas from different sections.

Algorithm 7: minDispFilter algorithm

input : A mask of detections M, a displacement map D, τ_d the minimum match distance
output: A filtered mask $\tilde{M}$
foreach *pixel p in M* **do**
  if *the distance in the image between p and* $D(p)$ *is smaller than* τ_d **then**
    $\tilde{M}(p) = 0$
  end
end

4.5 Symmetrization of Detections

Because the process is not entirely symmetric, it can happen that both regions detected as forged have different shape and size. It may even happen that only one of the forged regions is detected while the other one is missing from the detection mask. To avoid this problem, the detection mask is symmetrized using the displacement map. This process is presented in Algorithm 8. This section is necessary even though *PatchMatch* already has a mechanism favoring the symmetry. Indeed *PatchMatch* only incites it in the sense that if the symmetric match isn't a better candidate then it's rejected. Therefore at the end of the *PatchMatch* step, the displacement can be asymmetric (even though in practice it is already quite symmetric). Because a forgery is necessarily symmetric, the final detection map must be symmetric. Therefore the detection regions in the detection map are symmetrized using the method presented in this section.

4.6 Dilation

Finally, for the last step, each detection is slightly dilated. Indeed both the median filter and the error filter tend to reduce the size of a detected region. This dilation step mitigates these effects.

Algorithm 8: symmetrizationFilter algorithm

input : A detection mask M, a displacement map D
output: A filtered displacement map $\tilde{M}$
foreach *pixel* p *in* M **do**
  $\tilde{M}(D(p)) \leftarrow \max\{M(p), M(D(p))\}$
end

This step is presented in Algorithm 9.

Algorithm 9: dilationFilter algorithm

input : A mask of detections M, the radius of the filter ρ
output: A dilated mask $\tilde{M}$
foreach *pixel* p *in* M **do**
  $\tilde{M}(p) \leftarrow \max\{M(p') \mid p' \in \mathbb{D}(p, \rho)\}$ $\mathbb{D}(p, \rho)$ *is the set of elements at distance smaller or equal than* ρ *from* p
end

The complete algorithm regrouping all the steps presented in the previous sections 2, 3.1 and 4 is summarized in Algorithm 10.

Algorithm 10: Copy-move forgery detection

input : A suspect I, sp half-size of the patches, n_i number of *PatchMatch* iteration, τ_m minimum match distance for *PatchMatch*, the radius ρ_m, respectively ρ_e, for the median, respectively error computation, τ error threshold, τ_s size threshold, distance threshold τ_d
output: A final detection mask F and binary forgery decision

$Z \leftarrow ZernikeMoment(I, sp)$ *See Algorithm 1*
$D \leftarrow Patchmatch(Z, n_i)$ *See Algorithm 2*
$\tilde{D} \leftarrow medianFilter(D, \rho_m)$ *See Algorithm 4*
$E \leftarrow errorDetectionFilter(\tilde{D}, \rho_e, \tau)$ *See Algorithm 5*
$\tilde{M}_1 \leftarrow sizeFilter(E, \tau_s)$ *See Algorithm 6*
$\tilde{M}_2 \leftarrow minDispFilter(\tilde{M}_1, \tau_d)$ *See Algorithm 7*
$\tilde{M}_3 \leftarrow symmetrizationFilter(\tilde{M}_2)$ *See Algorithm 8*
$F \leftarrow dilationFilter(\tilde{M}_2, \rho_m + \rho_e)$ *See Algorithm 9*
return F *and whether* F *is empty or not*

5 Experiments

In this section, results showing both the strengths and the limits of the method are presented. The displacement results presented in this section will have the same visualization. The color scheme used to represent the direction and magnitude of the displacement is shown in Figure 4.

Table 1: Value of the parameters used during the computation of the results in Section 5.1.

Maximum order of the Zernike descriptors o	5
Half-size of patch used to compute the moments sp	8
Number of *PatchMatch* iterations N	8
Minimum distance between two *PatchMatch* matches τ_m	8
Minimum distance between two clones τ_d	50
Error threshold τ	300
Minimum size of a clone τ_s	1200
Radius of the median filter ρ_m	4
Radius of the error filter ρ_e	6

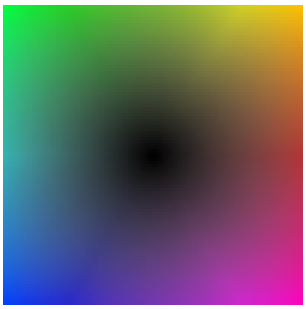


Figure 4: Color scheme representing the direction and magnitude of the displacement

5.1 Using Zernike Moments

The parameters that will be used for the experiments are those suggested in [5]. They are reminded in Table 1. The different images used in this section come from the FAU database[3] presented by Christlein et al. in [4]. The first example, shown in Figure 5, is a straightforward example illustrating the quality of the detections by the method in the simplest case. We can see that every step performs as expected and provides an accurate estimation of the forged regions. It is also important to test the method's resilience to false positives. This is why we applied the exact same method to an intact image containing several instances of a similar object, shown in Figure 6. The result confirms the absence of detection in this case even though some regions could have appeared forged due to their similarity.

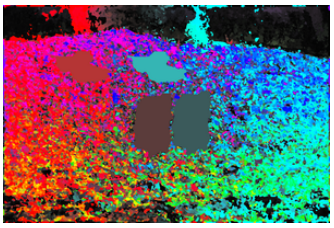
One of the requirements of the method was to be rotation resilient. An example of forgery with a rotation is presented in Figure 7. It actually confirms that the detection succeeds when a rotation has been applied. Even though only a single example is shown, more tests confirming the rotation resilience have been done with varying degrees of rotation. The same experiments have also been done using basic Cartesian Zernike moments in order to see the importance of the resampled version presented in Section 2. In this case, presented in Figure 8, the detection is much worse (only a much smaller region is detected) than when the resampled Zernike moments are used.

We also tested other classic perturbations such as the addition of noise. Figure 9 presents a result with a small amount of noise added after the forgery. The method still works in presence of small noise. The addition of a large noise in a forgery would make the image suspicious anyway. Figure 10

[3] http://www5.cs.fau.de/research/data/image-manipulation/

(a) Forged image

(b) Mask



(c) Displacement map

(d) Error map

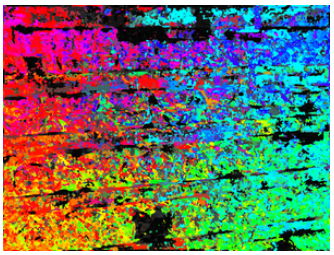
(e) Initial detection mask

(f) Final detection mask

Figure 5: Result of the method on a forged image for a forgery applying only a translation (using Zernike moments).

(a) Forged image

(b) Mask



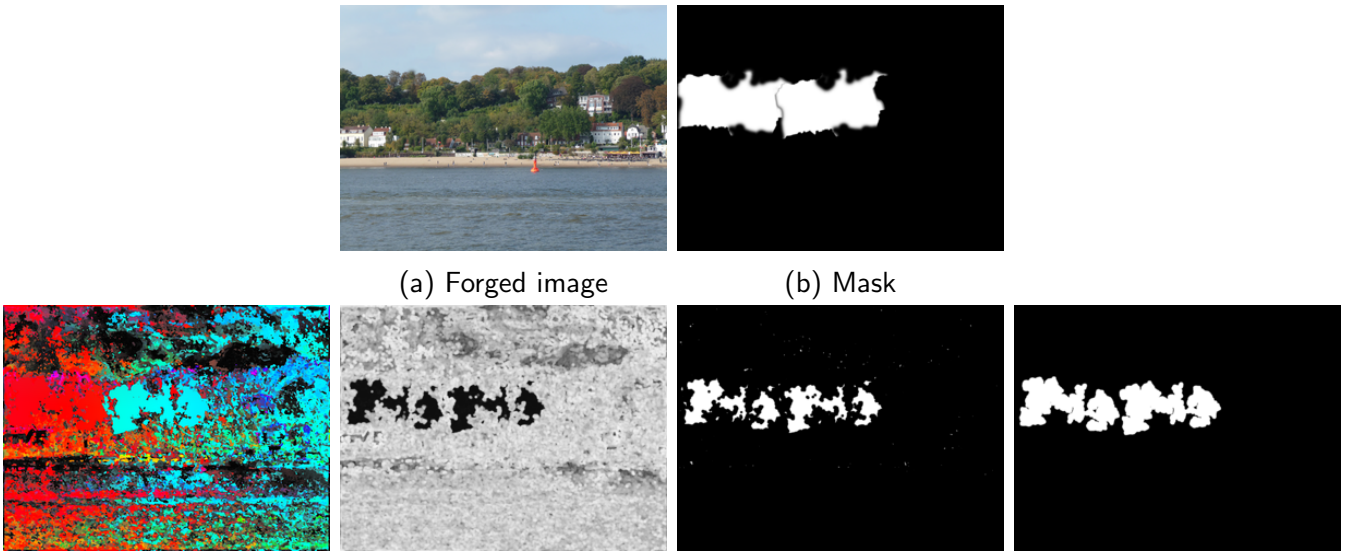
(c) Displacement map

(d) Error map

(e) Initial detection mask

(f) Final detection mask

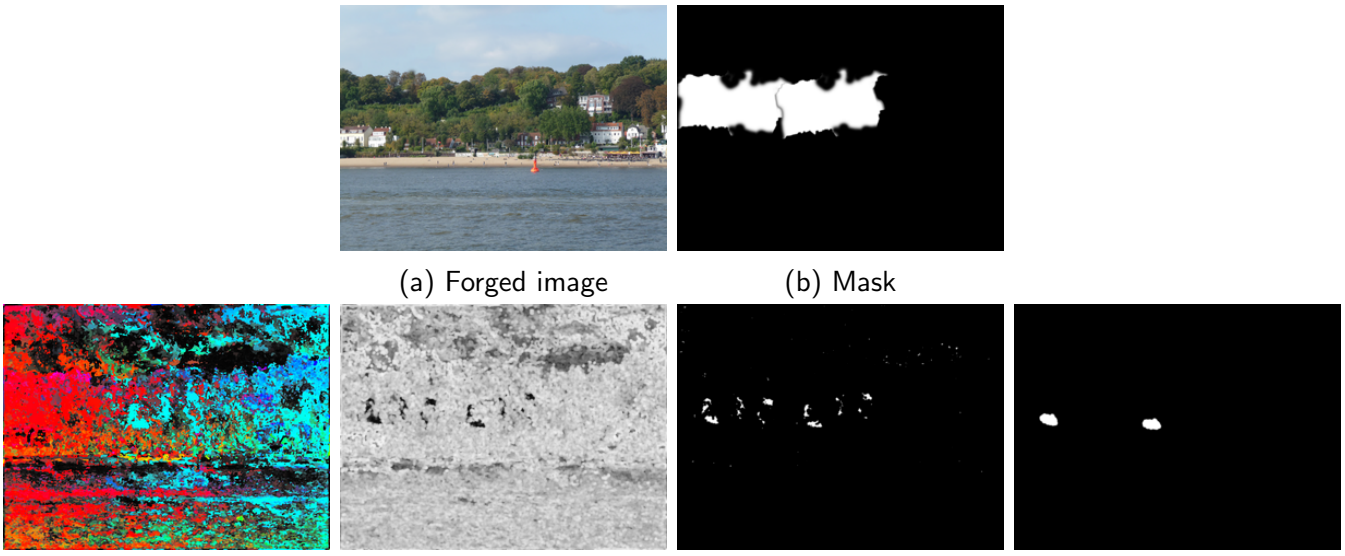
Figure 6: Result of the method on a clean image (using Zernike moments).



(a) Forged image (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

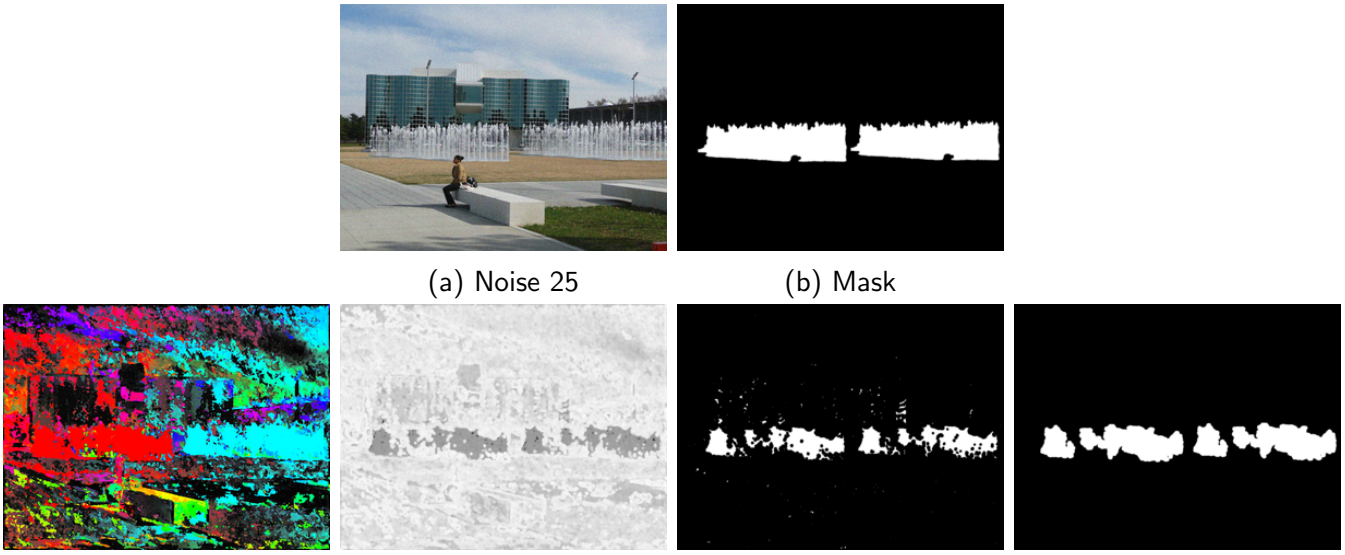
Figure 7: Result of the method on a forged image for a forgery applying a translation and a ten degrees rotation (using resampled Zernike moments).



(a) Forged image (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

Figure 8: Result of the method on a forged image for a forgery applying a translation and a ten degrees rotation (using basic Cartesian Zernike moments).



(a) Noise 25 (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

Figure 9: Result of the method on a forged image for a forgery applying a translation on top of which a noise of standard deviation 25 was added (using Zernike moments).

shows the importance of also taking into account flipped copies.

The example of Figure 11 is no longer a simple copy-move forgery but an actual editing of the image. Indeed the copied section was integrated using the Poisson editing method presented in [13] and implemented in [6]. Poisson editing is quite adapted to forgery as it blends nicely the pasted region in the image. This time, the algorithm actually fails. This shows the limitation not of the method itself but of the descriptors, namely the amplitude of Zernike moments. Those descriptors achieve the wanted invariance but at the same time they are extremely rigid. With appropriate descriptors, the exact same methodology could be applied and should yield good detections. Removing the Zernike moment of order 0 from the list of descriptors used (which actually corresponds to the mean of the local region) didn't improve the results for Poisson editing.

Another limit of the method is the presence of multiple copies. In such a case, the *PatchMatch* displacement field is at risk of splitting between the multiple copies, which increases the error and decreases the chance of an actual detection. Figure 12 shows an example where this problem arises. In that case, parts of each of the copies are actually detected but none is entirely detected for the reasons explained above.

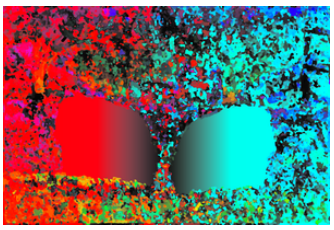
5.2 Using Dense SIFT Descriptors

Although the method performs as expected, as shown in Section 5.1, it is disappointing that forgeries such as Poisson editing are not detected. For this reason, the same experiments have been repeated after changing the Zernike moments by dense SIFT descriptors. SIFT descriptors, presented by Lowe in [11], have been shown to be state of the art descriptors in image matching thanks to their invariance to noise, changes of lighting or to small changes of viewpoint. In particular, they are rotation invariant, just like the Zernike moments. A dense version of these descriptors can be computed by considering all patches instead of those marked by points of interest. The dense version was the one used for the experiments. The VLFeat implementation of dense SIFT was used[4]. The parameters are listed in Table 2. Figures 13, 14, 15, 16 and 17 show the results of the same experiments shown

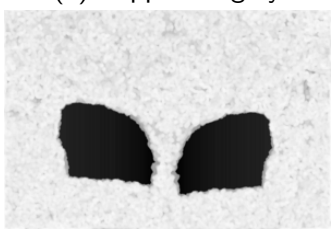
[4]A. Vedaldi and B. Fulkerson, VLFeat: An Open and Portable Library of Computer Vision Algorithms, 2008. http://www.vlfeat.org/

(a) Flipped forgery

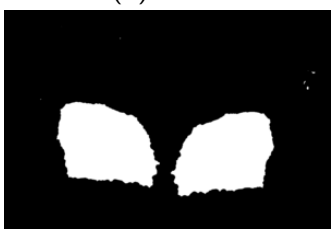
(b) Mask



(c) Displacement map



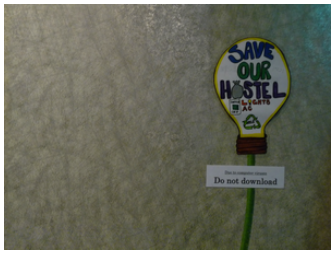
(d) Error map



(e) Initial detection mask

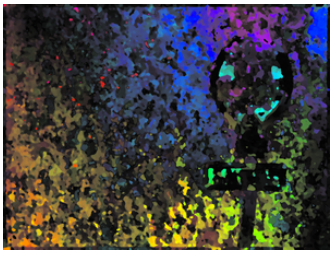
(f) Final detection mask

Figure 10: Result of the method on a forged image for flipped copy (using Zernike moments).



(a) Forged image

(b) Mask



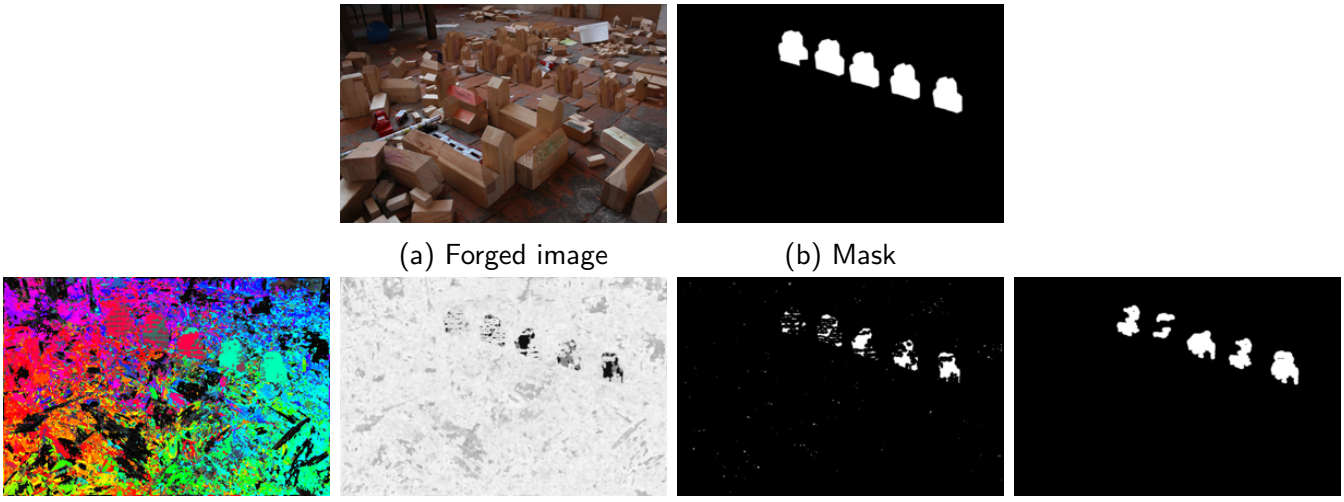
(c) Displacement map

(d) Error map

(e) Initial detection mask

(f) Final detection mask

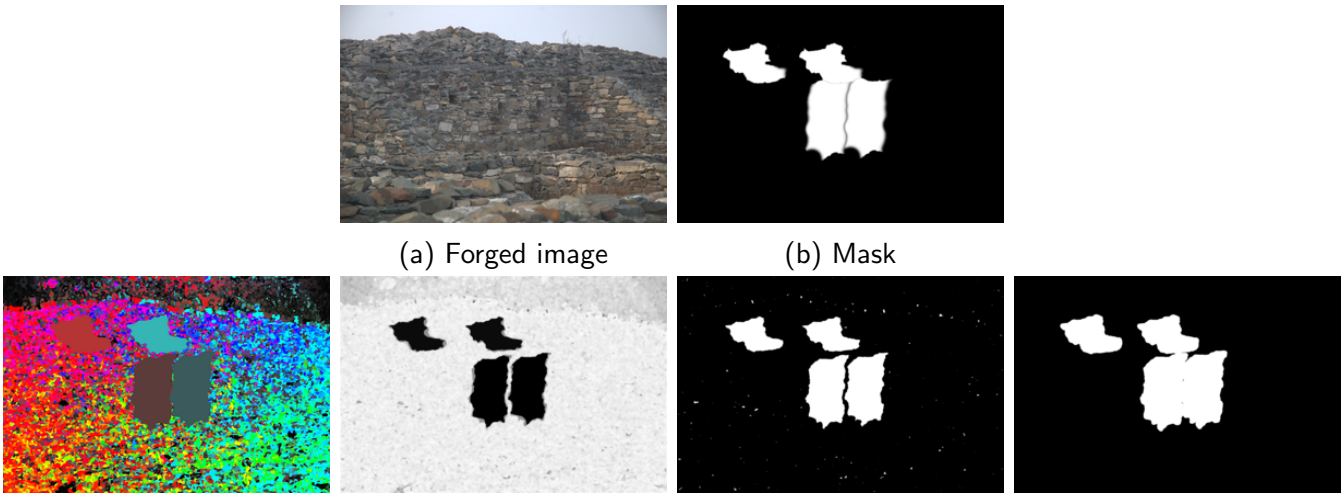
Figure 11: Result of the method on a forged image for a forgery applying a Poisson editing (using Zernike moments).



(a) Forged image (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

Figure 12: Result of the method on a forged image for a forgery applying translations when a same object is copied at multiple different positions (using Zernike moments).



(a) Forged image (b) Mask

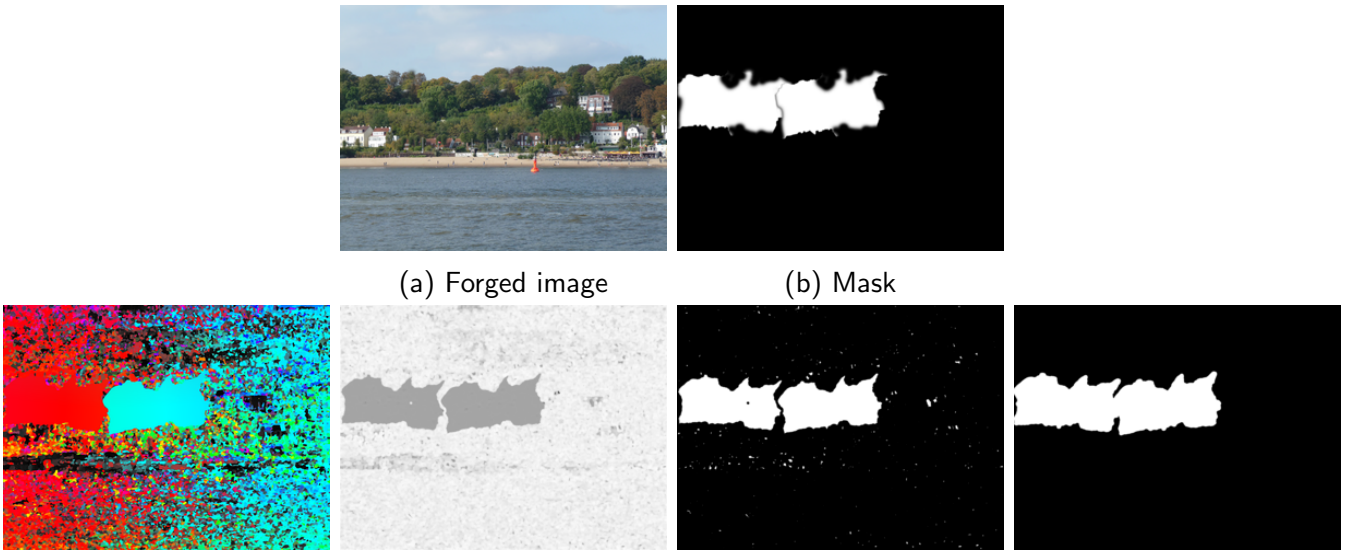
(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

Figure 13: Result of the method on a forged image for a forgery applying only a translation (using dense SIFT descriptors).

in Section 5.1. As one can see, dense SIFT descriptors perform as well as the Zernike moments on the successful experiments but also successfully detect Poisson editing. Nevertheless, dense SIFT descriptors seem to incur into a higher risk of false positives, see Figures 18 and 16.

6 Conclusion

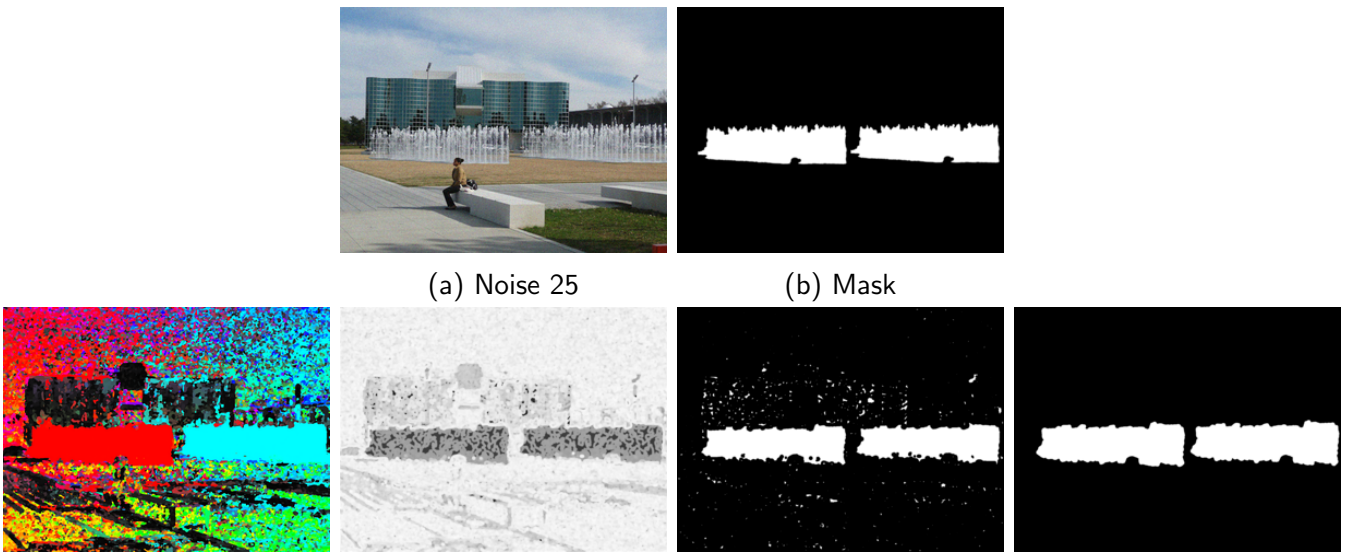
The forgery detection method analyzed in the present paper proves to be an efficient detector. One of its major advantages is its resilience to rotations. It is also robust to image perturbations such as the addition of noise. Nevertheless, it is not deprived of drawbacks. In particular, the results are particularly disappointing when considering multiple copies of an object or when the forgery has been performed by a classic copy-paste method such as Poisson editing. A better choice for the descriptor should be able to fix this deficiency of the method though. Dense SIFT descriptors were considered as an alternative to improve the detection of forgeries performed by Poisson editing. Even though these descriptors allow for better detections, they are more likely to produce false positives.



(a) Forged image (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

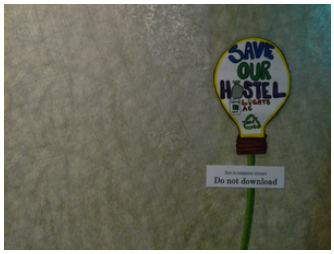
Figure 14: Result of the method on a forged image for a forgery applying a translation and a ten degrees rotation (using dense SIFT descriptors).



(a) Noise 25 (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

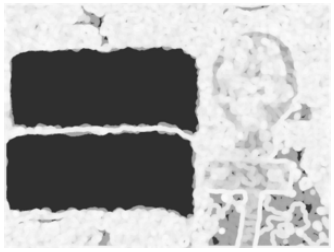
Figure 15: Result of the method on a forged image for a forgery applying a translation on top of which a noise of standard deviation 25 was added (using dense SIFT descriptors).



(a) Forged image

(b) Mask

(c) Displacement map



(d) Error map

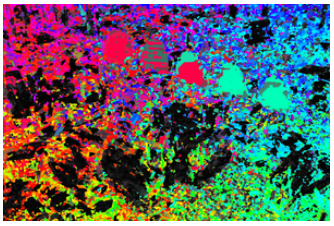
(e) Initial detection mask

(f) Final detection mask

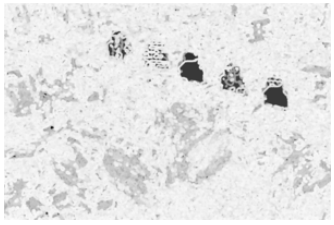
Figure 16: Result of the method on a forged image for a forgery applying a Poisson editing (using dense SIFT descriptors).

(a) Forged image

(b) Mask



(c) Displacement map

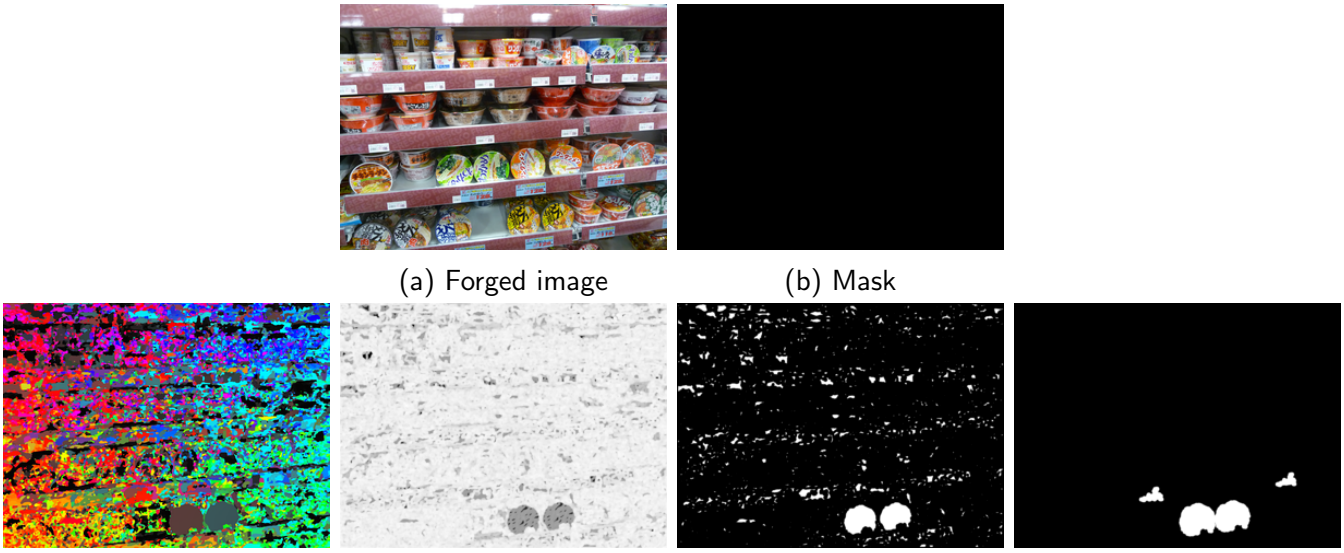


(d) Error map

(e) Initial detection mask

(f) Final detection mask

Figure 17: Result of the method on a forged image for a forgery applying translations when a same object is copied at multiple different positions (using dense SIFT descriptors).



(a) Forged image (b) Mask

(c) Displacement map (d) Error map (e) Initial detection mask (f) Final detection mask

Figure 18: Result of the method on a clean image (using dense SIFT descriptors).

Table 2: Value of the parameters used during the computation of the results in Section 5.2.

Number of SIFT bins	4
Size of SIFT bins	5
Number of *PatchMatch* iterations N	8
Minimum distance between two *PatchMatch* matches τ_m	8
Minimum distance between two clones τ_d	50
Error threshold τ	300
Minimum size of a clone τ_s	1200
Radius of the median filter ρ_m	4
Radius of the error filter ρ_e	6

Nevertheless, this shows that the method can be modified and improved by varying the descriptors. The only drawback found – which is not linked to the descriptors but to the usage of *PatchMatch* – is the inability of the method to detect multiple copies.

Acknowledgment

Work partly financed by IDEX Paris-Saclay IDI 2016, ANR-11-IDEX-0003-02, DGA Defals challenge ANR-16-DEFA-0004-01, MENRT and Fondation Mathématique Jacques Hadamard

Image Credits

All the images in this manuscript are either taken from the FAU database presented by Christlein et al. in [4] or produced by the author.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D.B. Goldman, *Patchmatch: A randomized correspondence algorithm for structural image editing*, ACM Transactions on Graphics-TOG, 28 (2009), p. 24. https://doi.org/10.1145/1531326.1531330.

[2] C. Barnes, E. Shechtman, D.B. Goldman, and A. Finkelstein, *The generalized PatchMatch correspondence algorithm*, in European Conference on Computer Vision, Springer, 2010, pp. 29–43. https://doi.org/10.1007/978-3-642-15558-1_3.

[3] J.L. Bentley, *Multidimensional binary search trees used for associative searching*, Communications of the ACM, 18 (1975), pp. 509–517. https://doi.org/10.1145/361002.361007.

[4] V. Christlein, C. Riess, J. Jordan, C. Riess, and E. Angelopoulou, *An evaluation of popular copy-move forgery detection approaches*, IEEE Transactions on Information Forensics and Security, 7 (2012), pp. 1841–1854. https://doi.org/10.1109/TIFS.2012.2218597.

[5] D. Cozzolino, G. Poggi, and L. Verdoliva, *Efficient dense-field copy–move forgery detection*, IEEE Transactions on Information Forensics and Security, 10 (2015), pp. 2284–2297. https://doi.org/10.1109/TIFS.2015.2455334.

[6] J.M. Di Martino, G. Facciolo, and E. Meinhardt-Llopis, *Poisson Image Editing*, Image Processing On Line, 6 (2016), pp. 300–325. https://doi.org/10.5201/ipol.2016.163.

[7] T. Ehret and P. Arias, *On the convergence of PatchMatch and its variants*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1121–1129.

[8] H. Farid, *Image forgery detection*, IEEE Signal Processing magazine, 26 (2009), pp. 16–25. ISSN:1053-5888.

[9] A. Gionis, P. Indyk, and R. Motwani, *Similarity search in high dimensions via hashing*, in Proceedings of the 25th International Conference on Very Large Data Bases, vol. 99, 1999, pp. 518–529.

[10] P. Indyk and R. Motwani, *Approximate nearest neighbors: towards removing the curse of dimensionality*, in Proceedings of the thirtieth annual ACM symposium on Theory of Computing, ACM, 1998, pp. 604–613. https://doi.org/10.1145/276698.276876.

[11] D.G. Lowe, *Object recognition from local scale-invariant features*, in IEEE International Conference on Computer vision, vol. 2, IEEE, 1999, pp. 1150–1157. https://doi.org/10.1109/ICCV.1999.790410.

[12] X. Pan and S. Lyu, *Region duplication detection using image feature matching*, IEEE Transactions on Information Forensics and Security, 5 (2010), pp. 857–867. https://doi.org/10.1109/TIFS.2010.2078506.

[13] P. Pérez, M. Gangnet, and A. Blake, *Poisson image editing*, ACM Transactions on graphics (TOG), 22 (2003), pp. 313–318. https://doi.org/10.1145/882262.882269.

[14] BL Shivakumar and LDSS Baboo, *Detection of region duplication forgery in digital images using surf*, IJCSI International Journal of Computer Science Issues, 8 (2011).

[15] L. Xie, Q. Tian, J. Wang, and B. Zhang, *Image classification with Max-SIFT descriptors*, in International Conference on Acoustics, Speech and Signal Processing, 2015.

[16] Y. Xin, M. Pawlak, and S. Liao, *Accurate computation of Zernike moments in polar coordinates*, IEEE Transactions on Image Processing, 16 (2007), pp. 581–587. https://doi.org/10.1109/TIP.2006.888346.

[17] P.N. Yianilos, *Data structures and algorithms for nearest neighbor search in general metric spaces*, in Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms, vol. 93, 1993, pp. 311–321.

[18] W-L. Zhao and C-W. Ngo, *Flip-invariant SIFT for copy and object detection*, IEEE Transactions on Image Processing, 22 (2013), pp. 980–991. https://doi.org/10.1109/TIP.2012.2226043.