Template matching in large scale

Image Analysis and Computer Vision 2019 Project

Supervisor:

Alessandro Concetti
Politecnico di Milano
alessandro.concetti@mail.polimi.it

Giacomo Boracchi
Politecnico di Milano
giacomo.boracchi@polimi.it

Mattia Sanchioni
Politecnico di Milano
mattia.sanchioni@mail.polimi.it

Abstract

Template matching is a technique in digital image processing for finding small parts of an image which match a desired template. This paper aims to present an approach that deals with multi-template scenes with several template instances and overcomes the main challenges in template matching such as occlusion, illumination and scale changes. The proposed algorithm uses SIFT to extract keypoints from the scene and leverages their spacial distribution to identify area where template instances are more likely to be found in order to reduce the complexity of homographies' fitting procedure. Moreover, the multi-template scenario has been treated as an opportunity rather than a complication since several useful information, explained in the following sections, can be gather in this context.

1. Introduction

The common approach to solve simple template matching problem exploits SIFT and RANSAC algorithm in order to extract features and fit homographies. This approach can be summarized as follow: (1) Keypoints and their associated descriptors are extracted through SIFT algorithm from template image and scene image. (2) Each keypoint in the scene is matched with the closest one in the template image w.r.t. SIFT descriptors. (3) These matches are used to fit a homography exploiting RANSAC algorithm.

Nevertheless, this approach shows different problems in scenarios with multiple instances of several different templates. These issues will be better explained in the following paragraphs.

1.1. Issues with multi-instances of the same template

As will be discussed in the following section, RANSAC algorithm fits homographies using 4 keypoints and evaluates consensus of the others. As highlighted in figure 1, the probability of finding four points belonging to the same object in a single instance scenario is way higher respect to a multi instances scenario. This causes several problems in using a traditional approach when multiple instances of the same template are present in the scene. The main issue is that a lot of wrong homographies are found using keypoints belonging to distinct instances. This causes a strong slowdown of the RANSAC algorithm since a lot of time is lost before fitting the right homography.

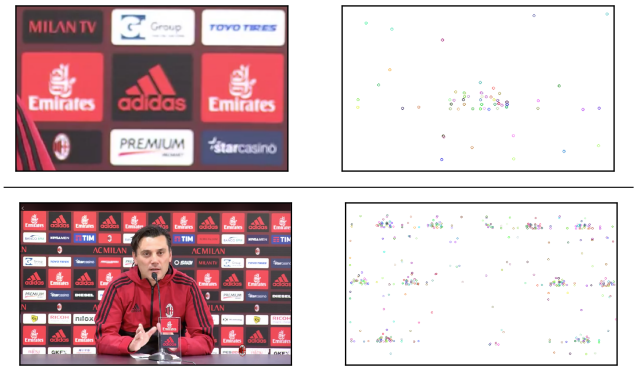


Figure 1. The target template is the *Adidas*© logo. The left columns shows two different test images. The top one contains only one instance of the template, instead the bottom one has multiple instances. On the other hand the right column shows the keypoints extracted from each test image.

In order to speed up the fitting process, the set of

keypoints from which the four candidates are picked should be reduced using a search window whose aim is to filter out keypoints. As can be seen in pictures on the left in figure 1.1 showing the spacial distribution of keypoints, from a human perspective it is trivial to locate areas where an instance is more likely to be found. The proposed approach transforms this intuition into a heuristic used to move the search window to perform a scatter search.

1.2. Issues with multiple templates

Since performing a scatter search is computationally very efficient, multiple searches associated with different templates could be executed simultaneously. Although this approach has brought several efficiency advantages, it is not enough to deal with scenes with several instances of different templates. The main issue in real applications is that different templates sharing portion of the image, such as brand logo, could be present in the scene. In these situations several homographies referred to different templates can be associated to the same area of the test image, as shown in figure 2.

Figure 2. *Multi templates problem*: on the left two different template images, on the right two instances of the templates found.

In order to remove all the false positive homographies a series of checks, explained in following sections, have been exploited.

2. Related work

2.1. SIFT

The scale-invariant feature transform (SIFT) is a feature detection algorithm to detect and describe local features in images. It takes as input an image and returns a set of keypoints. Each keypoints is also associated with a descriptor, which is a highly distinctive vector partially invariant to variations such as illumination, 3D viewpoint, etc. The algorithm is extensively explained in *Lowe, D.G. (2004)* [2].

2.2. k-NN

The nearest-neighbor (NN) search problem is defined as follows: given a set $\mathcal{S}$ of points in a space $\mathcal{M}$ and a query point $q \in \mathcal{M}$, find the closest point in $\mathcal{S}$ to q. A direct generalization of this problem is a k-NN search, where we need to find the k closest points. In this project, the algorithm *knnMatch* provided by the library FLANN (Fast Library for Approximated Nearest Neighbor) of OpenCV has been exploited in order to match keypoints in the scene image and template image based on the euclidean distance between their SIFT descriptors.

2.3. RANSAC

Random sample consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates. The algorithm was first published by Fischler and Bolles at SRI International in 1981 *(Fischler, Martin A. & Bolles, Robert C. 1981)* [1]. In the proposed algorithm RANSAC has been used to fit homographies based of the initial set of matches. The following pseudo code summarizes how RANSAC algorithm can be used for fitting a homography starting from a set of matches $\mathcal{M}$.

Algorithm 1 RANSAC

$H = null$
$nBest = 0$
for $i = 0...N$ **do**
  $p4 = SelectRandomSubset(\mathcal{M})$
  $H_i = ComputeHomography(p4)$
  $nInliers = ComputeInliers(H_i)$
  if $nInliers > nBest$ **then**
    $H = H_i$
    $nBest = nInliers$
  end if
end for

3. Proposed approach

The main idea behind the algorithm is to find for each template all possible homographies and merges them together performing a global analysis in order to discard false-positive homographies.

Therefore, the algorithm is intricately divided into two parts: the former called *single-template analysis* and the latter *multi-template analysis*.

In this section, they will be largely argued, whereas all the implementation details will be omitted and discussed in section 4.

3.1. Single template analysis

This phase aims to find all the possible homographies given a single template image. The first step of the single-template analysis consists in feature extraction using SIFT algorithm, see 2.1. This algorithm takes an image as input and produces a set of keypoints with their associated SIFT descriptors. The features have been extracted both in the scene image and the template image producing two sets of keypoints, respectively called $\mathcal{K}_S$ and $\mathcal{K}_T$.

The keypoints in $\mathcal{K}_S$ and in $\mathcal{K}_T$ are matched together using 2-NN algorithm [2.2] and ratio test [4.1], and only keypoints in $\mathcal{K}_S$ having a match with a keypoint $\mathcal{K}_T$ will be kept.

As already briefly mentioned, we have leveraged the space distribution of keypoints over the test image to find areas where the considered object is more likely to be found. Keypoints could be represented in a sparse fashion using a binary image in which 1 is assigned to pixels corresponding to keypoints. Now, keypoints are clustered using a series of morphological transformations, see section 4.2.

These clusters identifies areas where there is a high density of keypoints. The bigger the cluster area the higher is the probability to find an object in a neighborhood of the cluster. These clusters, called *hotpoint* so far, are identified by their centroids and characterized by their areas.

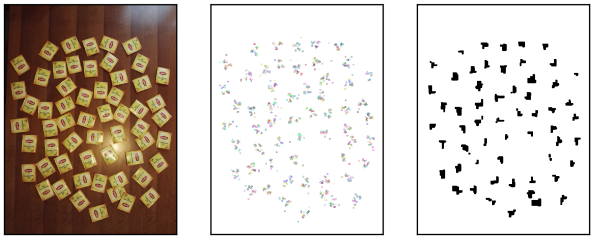


Figure 3. *Hotpoints*: On the left the test image. The picture in the center shows the keypoints after the ratio test. The image on the right represents the result of the explained morphological transformations.

At this point the common approach is to use RANSAC algorithm [2.3] to fit homographies searching in the whole scene. As previously explained in section 1.1, searching homographies without spacial filters becomes computational expensive and inefficient in case of several instances of the same template. The proposed approach exploits hotpoints to understand where template instances are placed in the scene.

The idea of the algorithm is to explore each hotpoint, sorted by decreasing area, using a smaller window in order to reduce the search space thus the complexity of the problem. See paragraph 4.3 for more details about the hotpoints exploration.

For each homography returned by the RANSAC algorithm a series of checks has been performed to validate the homography just found: (1) The homography must not be degenerate. (2) The polygon associated with homography must be convex. (3) All the inliers must lay inside the polygon. (4) The inliers must be at least five. (5) The homography must be coherent with homographies validated so far.

The last check based on the concept of *coherence of homographies* is worth to be argued. All the sides' lengths and angles of polygons associated with each homography are saved. Any time a new homography is found, as shown in figure 4, it is evaluated whether this homography is coherent with the others or not.

In this case *coherent* means that the sides' lengths and the angles should not be too different from the nearest homographies validated so far in the scene.

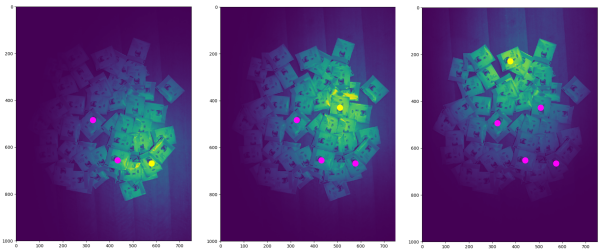


Figure 4. *Homography coherency:* these images show how new homographies are evaluated. The pink points are the centers of the template instances found so far, while the yellow points identify the homography to be evaluated. That is compered with all the instances found weighted based on an *attention Gaussian* as shown in the pictures

Moreover, each time the algorithm finds a homography, it is fitted again with different keypoints.

First of all the polygon describing the template instance is drawn transforming the rectangular border of the template with the inverse of the homography found. Then, only keypoints inside this polygon are used to fit the homography again. This is useful in situations in which a homography straddling the border of the search window is found. In these cases the first homography is fitted with few keypoints since most of them are outside the boundary of the search window. This could lead to a not so well estimated homography. The second homography, instead, will be fit using only the keypoints inside the first homography polygon ending up with a better estimate of the homography.

3.2. Multi template analysis

This phase aims to collect all the homographies found by each single template analysis and merge them together verifying the coherence of the homographies.

As seen in the section 1.2, different templates sharing portion of the image such as the brand logo, can fit various homographies using the same keypoints of the test image. This involves that an object in the scene could be matched by two or more different templates. Trivially, at most one template could be the right one. Hence, the first check of the multi template analysis aims to delete all the overlaps in the scene keeping only the right homographies. In order to evaluate which homography matches better the scene, as shown in figure 6, it is assigned to each homography a so called *histogram error*. This value represents the average pixel-wise difference between the template image and the rectified portion of the scene containing the instance found.

After all the overlaps have been removed, the following three checks are performed:

Number of inliers normalized by area: the lower the inliers spacial density the higher is the probability to have been fitted a wrong homography.

Histogram error: the higher the histogram error the higher is the probability to have been fitted a wrong homography.

Ratio cm/pixel: Since the real dimensions of templates are known, for each template instance a ratio cm/pixel could be estimated using the root square of the ratio between the real area and the area in the scene. Homographies whose ratio cm/pixel deviates too much from the mean ratio are more likely to be wrong.

It is important to notice that all this checks do not provide a hard evidence that a homography is actually a false-positive because every test could be deceived in some special cases. The first one fails in its estimate in cases where a very far template instance is found. The second one is deceived when an instance is partially covered by some other objects. The last one fails in case of a very strong perspective. Hence a scoring system has been designed in a way that each check assigned a positive score if the homography is likely to be correct and a negative score otherwise. At the end all the homographies with a positive overall score are kept.

4. Implementation details

In this section all the implementation details will be extensively argued.

The template matching algorithm has been designed leveraging a multi-processing approach. This design decision has brought several advantages as shows the experiments results, see paragraph 5.2.

At the beginning of the algorithm for each template a process is launched and performs the single-template analysis [3.1] on the assigned template. The main process waits for the end of all these subprocesses and once all of them ends all results are collected and the multi-template analysis [3.2] is performed.

4.1. Ratio test

The ratio test is used in the algorithm to filter out keypoints keeping the most distinguishable ones. The features have been extracted using SIFT both in the scene image and the template image producing two sets of keypoints, respectively called $\mathcal{K}_S$ and $\mathcal{K}_T$, as already mentioned. The keypoints of the test image and template image are matched together using 2-NN algorithm [2.2] that returns for each keypoint in the scene the two nearest keypoints in the template image w.r.t. the SIFT descriptors. Note that a match is defined in the domain $\mathcal{M} \subset \mathcal{K}_S \times \mathcal{K}_T$. Hence, the set produced by k-NN algorithm as output is defined as $\mathcal{M}_{flann} \subset \mathcal{M} \times \mathcal{M}$. Finally, only the matches passing the ratio test (see algorithm 2) are kept. Note that the symbol $\|m\|$ is used to identify the euclidean distance between SIFT descriptors of keypoints belonging to match m. While $\mathcal{M}_{good} \subset \mathcal{M}$

identifies the set of matches passing the test.

Algorithm 2 Ratio test

for $(m_1, m_2) \in \mathcal{M}_{flann}$ **do**
  if $\|m_1\| < 0.8 * \|m_2\|$ **then**
    $Add\ m_1\ to\ \mathcal{M}_{good}$
  end if
end for

The ratio test removes all those matches in which the corresponding keypoint in the scene is sufficiently close to two completely different points in the template image hence it could be considered ambiguous.

4.2. Hotpoint clustering

Hotpoint clustering is the procedure thanks through which hotpoints are created. The main idea is to represent keypoints in a sparse-fashion and cluster them together using a series of morphological transformation.

In the algorithm below the following notation will be used: $\mathbb{1}(n)$ to indicate a unary nxn kernel. $\mathcal{C}(n)$ to indicate a circular nxn kernel. $\mathcal{I}_k$ to indicate the binary image representing the keypoints location. Finally, $\oplus$ is the dilation operator while $\ominus$ is the erosion operator .

Algorithm 3 Keypoints clustering with morphological transformation

$\mathcal{I}_k \leftarrow \mathcal{I}_k \oplus \mathbb{1}(15)$
$\mathcal{I}_k \leftarrow \mathcal{I}_k \ominus \mathbb{1}(10)$
$\mathcal{I}_k \leftarrow \mathcal{I}_k \ominus \mathbb{1}(8)$
$\mathcal{I}_k \leftarrow \mathcal{I}_k \oplus \mathcal{C}(5)$

The first dilation is used to join all the near keypoints in a single cluster, then two erosions are perfomed: the first one in order to remove outliers, and the second to reduce even more the dimension of clusters' area. Finally, the last dilation with a circular kernel is used to smooth the contours of remaining clusters.

4.3. Hotpoint exploration

The homographies fitting procedure is performed using RANSAC algorithm in a smaller region of the image spotted by a so called search window placed in hotpoints neighborhood. The dimension of the search window is initialized using the dimensions of the bounding box of the biggest hotpoint, increased by three times to be sure to include at least one instance of the desired template. After the first iteration, the window dimensions are updated to a value equal to 1.3 times the dimensions of first instance found. In order to explore the whole hotpoint neighborhood the following search strategy is performed.

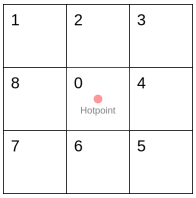


Figure 5. *Grid search strategy*

A 3x3 grid is placed over the center of the hotpoint, as shown the figure 5. Each cell of this grid has the same dimension of the search window previously computed. The search starts placing the window in the central cell and proceeds moving the window in clockwise order.

Algorithm 4 Hotpoint exploration

$hotpoints \leftarrow sort(hotpoints)$
$window\_dims \leftarrow 3 * hotpoints[0].bounding\_box$
$first\_hom \leftarrow find\_first\_homography(window\_pos, window\_dims)$
$window\_dims \leftarrow 1.3 * get\_dimension(first\_hom)$

$homographies\_found = [\,]$

for h **in** $hotpoints$ **do**
  for $window\_pos$ **from** 0 **to** 8 **do**
    $hom\_list \leftarrow find\_homographies(window\_pos, window\_dims, test\_keypoints, template\_keypoints)$

    $homographies\_found.add\_items(hom\_list)$
  end for
end for
return $homographies\_found$

As explained in section 2 homographies are fitted using the RANSAC algorithm [2.3] that takes as input keypoints of the test image and of the template image. In the following paragraphs more details about the function *find_homographies* mentioned in the Algorithm 4 are provided.

Given a window, defined by a position and a di-

mension in the pseudo code, the function keeps searching as many homographies as it can inside that window and finally returns a list of homographies found.

At implementation level, the RANSAC algorithm [2.3] has not been implemented from scratch but the function *find_homography* of the OpenCV library has been exploited.

4.4. Histogram error

The histogram error represents the average pixel-wise difference between the template image and the rectified portion of the scene containing the instance found. The procedure to calculate this value is summarized by the psuedo code 5 and follows these guidelines: (1) The test image $\mathcal{I}_S$ is rectified using the inverse of the homography in exam $\mathcal{H}$ and finally cropped in order to obtain the rectified image of the template instance $\mathcal{I}_I$. (2) The histogram of $\mathcal{I}_I$ is adapted to reduce eventual color differences due to various brightness conditions respect to the template image, $\mathcal{I}_T$. More details about this step will be provided later. (3) The histogram error is a function of the difference between the two images, called $\mathcal{D}$ in the algorithm.

Algorithm 5 Histogram error

$h \leftarrow height\ of\ \mathcal{I}_T$
$w \leftarrow width\ of\ \mathcal{I}_T$
$\mathcal{I}_I \leftarrow \mathcal{H}^{-1} * \mathcal{I}_S$
$\mathcal{I}_I \leftarrow \mathcal{I}_I[0...h][0...w]$

$\mathcal{I}_I \leftarrow match\_histogram(\mathcal{I}_I, \mathcal{I}_T)$

$\mathcal{D} \leftarrow |\mathcal{I}_T - \mathcal{I}_I|$
$\mathcal{D} \leftarrow Compute\ second\ order\ norm\ along$
$channel\ dimension\ to\ obtain\ a\ 2D\ image$

$m, n \leftarrow respectively\ height\ and\ width\ of\ \mathcal{D}$

$e \leftarrow (\sum_{i=1}^{m} \sum_{j=1}^{n} \mathcal{D}_{ij})/(m \times n)$

return e

It is important to notice that the function *match_histogram* in algorithm 5 adjusts the pixel values of the instance image $\mathcal{I}_I$ such that its histogram matches that of the template image. This procedure is based on the normalized cumulative histograms of both $\mathcal{I}_I$ and $\mathcal{I}_T$, respectively called $NCH_I(\bullet)$ and $NCH_T(\bullet)$. These functions are defined in the following discrete domain $\mathcal{D}$:

Figure 6. *Histogram error:* In this image in the first column the real template images $\mathcal{I}_S$ are shown. In the second column the rectified instance images $\mathcal{I}_I$. While in the third column the pixel-wise normalized differences between the two images $\mathcal{D}$. Error1, associated with the template in the first row, is equal to 56.83 while Error2, associated with the one in second row, is equal to 39.84. These values show how the histogram errors can be used in comparison of two homographies

$$\mathcal{D} = \{x | x \in \mathcal{N} \wedge 0 \leq x \leq 255\}$$

Given a pixel value x as input they return the relative frequency of pixel values less than or equal to x.

Each pixel value x in $\mathcal{I}_I$ is replaced by $x_{new} = NCH_T^{-1}(NCH_I(x))$, i.e. the pixel value of $\mathcal{I}_T$ having the same relative frequency of x, since the formula can be derived by the equation $NCH_T(x_{new}) = NCH_I(x)$.
In cases where $x_{new} = NCH_T(f_{new})^{-1}$ is not defined for some frequency values $f_{new} = NCH_I(x)$, the new pixel value x_{new} is calculated as $x_{new} = NCH_T(\ \tilde{f}\ )^{-1}$ where $\tilde{f}$ is the frequency closest to f_{new}.

5. Experiments

5.1. Dataset

The dataset mainly contains images and it is divided in two main components:

Test images dataset, containing images of possible application scenes. A batch of 38 images has been used during testing procedure. Most of those have been used in the first phase of testing to evaluate the algorithm in single-template scenarios and then they have not been used anymore since not suitable for multi-template testing. The final algorithm has been tested with 16 images from different scenarios such as super market shelves and advertising walls.

Template images dataset, containing all the template images. The total number of template images is 68. These images represent templates that the algorithm has to looking for in the scene.

Both datasets are accessed by the algorithm through JSON files containing metadata about images.

The JSON file associated with the *template images dataset* contains the path where these images are placed and a list of data structures representing template images with additional information such as the real dimensions (width and height in cm) of the template and the label used at run-time to identify each template.

```
{
"image_path": "./images/template",
"templates": [
    {
        "name":"COCO_POPS",
        "path": "cereali_coco_pops.jpg",
        "size": [13.2, 14.3]
    },
    ...
    ]
}
```

The structure of the JSON file of the *test images dataset* is similar and it stores for each test images the path and the name of the test image:

```
{
    "image_path": "./images/test",
    "test_images": [
        {
            "name":"MILAN_ADS",
            "path": "pressAds.png"
        },
        ...
        ]
}
```

5.2. Efficiency experiments

In order to test the efficiency of the proposed algorithm and to highlight the speed-up in term of execution time, we propose a comparison with the single-thread version of our algorithm. Both the algorithms (single-process version and multi-process version) has been tested using three different scenes and steadily increasing the number of templates that has to be searched, ranging from the actual number of templates in the scene to whole template database (68 images). As can be seen from the experiment results from figure 7 the complexity of both algorithms seems to be linear, at least for a treatable number of templates. Although the asymptotic complexity of the algorithm remains the same ($\Theta(n)$) in the multi-process version, constants are significantly more favourable allowing a better scalability of the algorithm.

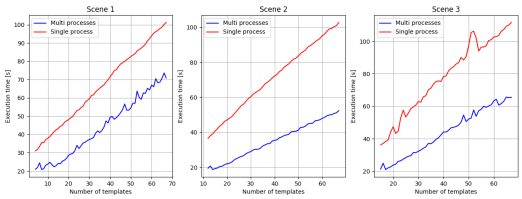


Figure 7. Efficiency between single process and multi processes

5.3. Effectiveness experiments

The effectiveness of the algorithm is declared in term of *precision* and *recall* computed with five different test images. Images associated with each scene are shown in the appendix of this paper. Let define as $\mathcal{F}_P$ the set false positive matches, as $\mathcal{T}_P$ the set of true positives, as $\mathcal{F}_N$ the set of false negatives and as $\mathcal{T}_N$ the set true negatives. Moreover let call $\mathcal{O}_f$ the total number of template found (both wrong and correct) and $\mathcal{O}_c$ the total number of correct template that should have been found. We can define **precision** $\mathcal{P}$ and **recall** $\mathcal{R}$ as follow:

$$\mathcal{P} = \frac{|\mathcal{T}_P|}{\mathcal{O}_f} = \frac{True\ Positive}{Number\ of homographies\ found}$$

$$\mathcal{R} = \frac{|\mathcal{T}_P|}{\mathcal{O}_c} == \frac{True\ Positive}{Correct\ number\ of homographies}$$

$$\mathcal{P} = \frac{|\mathcal{T}_P|}{|\mathcal{T}_P| + |\mathcal{F}_P|} = \frac{|\mathcal{T}_P|}{\mathcal{O}_f}$$

$$\mathcal{R} = \frac{|\mathcal{T}_P|}{|\mathcal{T}_P| + |\mathcal{F}_N|} = \frac{|\mathcal{T}_P|}{\mathcal{O}_c}$$

Scenes	$\lvert\mathcal{T}_P\rvert$	$\mathcal{O}_f$	$\mathcal{O}_c$	$\mathcal{R}$	$\mathcal{P}$
Scene 1	26	29	30	86.7%	89.7%
Scene 2	10	10	10	100%	100%
Scene 3	18	18	19	94.7%	100%
Scene 4	15	15	15	100%	100%
Scene 5	11	12	12	91.7%	91.7%

In order for a match to be labeled as *true positive* its IOU (Intersection Over the Union) w.r.t the ground truth must be greater than 80%. In some examples like Scene 1, this causes some good homographies to be labeled as *false positives* since they do not matched precisely the ground truth and this make IOU to be less then 80%.

6. Conclusion

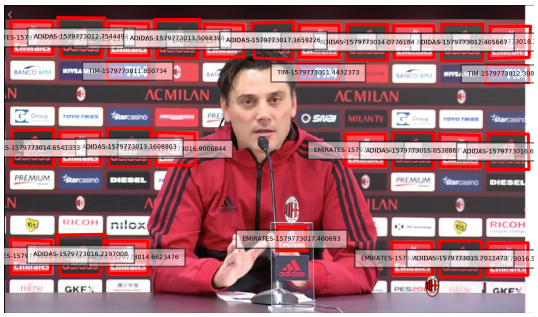
Drawing all these results together, the proposed algorithm is able to match rectangular templates in images very efficiently. Nevertheless, the algorithm is still not suitable for real time applications, like for example marketplace shelves monitoring where template instances should be found in the scene in at most 2-3 seconds to keep up with customers' behaviors. Anyway, the algorithm could be used in applications where scenes change not too fast and a delay for the templates matching can be acceptable, like warehouses of marketplaces and pharmacies.

Possible future extensions could focus on making the algorithm more robust with non-rectangular templates and on making it able to deal with non rigid transformation of templates in the scene.

Acknowledgements

Last but not the least it is important to highlight that this project gave us the possibility to practice with most important tools of computer vision and computer science. On a hand we have had the opportunity to examine in depth algorithms like SIFT and RANSAC and exploit them for practical purpose. On the other hand we got a chance to deal with implementation issue like algorithm optimization and multiprocessing programming.

A. Effectiveness tests



(a) template matched

(b) ground truth

Figure 8. Scene 1

(a) template matched

(b) ground truth

Figure 9. Scene 2

(a) template matched

(b) ground truth

Figure 10. Scene 3

(a) template matched (b) ground truth

Figure 11. Scene 4

(a) template matched (b) ground truth

Figure 12. Scene 5

References

[1] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, 1981. 2

[2] D. G. Lowe. Distinctive image features from scale-invariant keypoints, 2014. 2