École Centrale Lyon

UE ECL C-4
Bureau d'étude
Rapport

Where's Wally?

Students :
Taïga Gonçalves
Allan Belhabchi

Teachers :
Liming Chen
Jean-Yves Auloge

April 6, 2022

Contents

1 Introduction

Object recognition algorithms have been well developed over the years. Nowadays, it can instantly detect some details that even humans cannot notice. But how does these techniques work ?

This Bureau d'Étude aims to study one of the techniques used for this purpose: the SIFT algorithm. This algorithm can detect local features in images and compare the extracted features with the ones of other images. The comparison is performed using the Euclidean vectors of the features vectors. One of the keypoints of SIFT is that the extracted features are invariant to uniform scaling and orientation.

This technique will be used to solve a complex problem, based on the British series *Where's Wally?*. The goal is to find a character named Wally hidden in a crowded scene.

Figure 1: Wally

2 Applying the SIFT on all the images

The first step to solve the problem is to extract the features of the given images. For this study, five images are available, which are named from *Charlie1.jpg* to *Charlie5.jpg* (Charlie is the French version of Wally). The figure 2 represents the image of *Charlie.jpg*. The full image of Wally is also provided in *Charlie.jpg* (figure 1). This image will be used to perform the features comparison.

Figure 2: *Charlie4.jpg*

Given the fact lower part of Wally is almost never represented on the usual illustrations, the default image *Charlie.jpg* need to be truncated to analyze the most interesting part, which is his head. This operation is done with the following code, and its result is showed on figure 3.

```
sample = imread('Charlie.jpg');
charlie_head = sample(1:180,180:320,:);
```

Figure 3: Extraction of Wally's head

Once this is done the SIFT operation can be performed using the function *vl_sift*. However, it has been observed from experience that the results were not satisfying. In fact, the image should go through some others transformations before applying the SIFT.

The need of this pre-processing can be explained by the presence of many noises in the picture. The figure 4 proves that.

Figure 4: Zoomed version of *Charlie4.jpg*

Two approaches have been tested to solve this problem. Firstly, the image have been resized to reduce the number of useless pixels on them. The figure 5 shows the result of this operation.

Figure 5: Result of *imresize* on *Charlie4.jpg*

While the pictures seems to have been improved to extract the features, the final result were still not satisfying. However, resizing the default image (figure 3) by 0.8 proved to be useful. After many tests, it has thus been decided to resize only this image:

```
resize_param = 1; % The images where Charlie is hidden will be resized by this value
resize_param_head = 0.8; % The image of the head of Charlie (sample image) will be resized
```

Another transformation have been tested to solve the noise problem. This is actually the application of the gaussian filter. The figure 6 shows the result of this filter with a std parameter equals to 2. In fact, an image can be blurred using a std greater than 1 in order to reduce the noises on it.

Figure 6: Gaussian filter on *Charlie4.jpg*

Unfortunately, this filter did not permitted to obtain a better result in the end. However, a gaussian filter with a low std proved to greatly improve the final result. This operation does not drastically change the original image since the difference is invisible to the naked eye. However, reducing the std of the pixels on an image could be helpful to extract more precise features on it. The std value of the gaussian filter is initialized with the following variable:

```
gauss_filt = 0.5; % All the images will be filtered by the
    gaussian filter with the std value precised here
```

Finally, to improve the final result, the default image (figure 3) have been rotated using *imrotate()* function. This rotation allows to slightly change the properties of the extracted features, as illustrated in the figure 7.

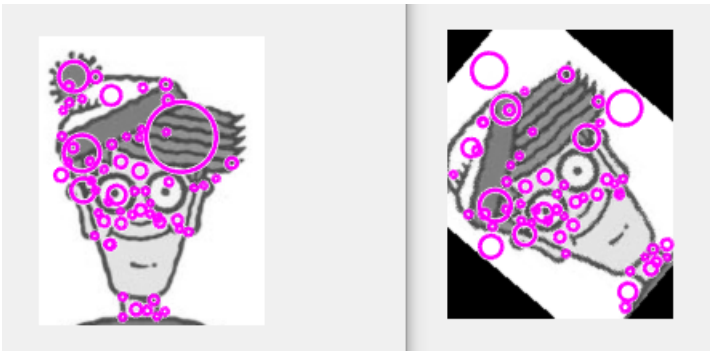

Figure 7: Rotation of the original image

The goal of this transformation is to perform features matching using different orientation of Wally's head, so that multiple "default" images are available. Wally could have then be found in the illustrations by selecting the feature which was matching with most of these "default" images. However, features matching weren't performing well with rotated images. This idea has thus been ditched.

To summarize this section, the default image (Wally's head) has been resized by 0.8 and a gaussian filter with an std equals to 0.5 has been applied:

```
charlie_head = sample(1:180,180:320,:);
I = charlie_head; % Extracting the head of Charlie Use I for
    every images so that no useless variables are created
%I = imrotate(I,70); % Can detect I1 with that, but there's no
%sufficiently enough precision on the result
I = im2single(I);
I = rgb2gray(I);
I = imresize(I,resize_param_head);
I = imgaussfilt(I,0.5);
[f,d] = vl_sift(I);
```

As for the other images, they weren't resized (*resize_param = 1*) but the same gaussian filter has been applied:

```
I = I4; % Use I for every images so that no useless variables
    are created
I = im2single(I);
I = rgb2gray(I);
I = imresize(I,resize_param);
I = imgaussfilt(I,gauss_filt);
[f4,d4] = vl_sift(I);
```

3 Features matching

Once the SIFT operation has been applied, the default image can be compared to the different illustrations using features matching:

```
[matches, scores] = vl_ubcmatch(d,d4,match_threshold);
[scores, sortIdx] = sort(scores,'descend');
matches4 = matches(:,sortIdx);
X4 = f4(1:2, matches4(2,:))';
r4 = f4(3, matches4(2,:))';
```

On the above code, *X4* represent the coordinate of the center of the features matching with the default image (Wally's head) and *r4* corresponds to its radius. The following code use these variables to draw circles on the original illustration, and its result is represented on figure 8.

```
figure;
imshow(I4);
viscircles(X4/resize_param, 50*r4,'color','magenta');
```

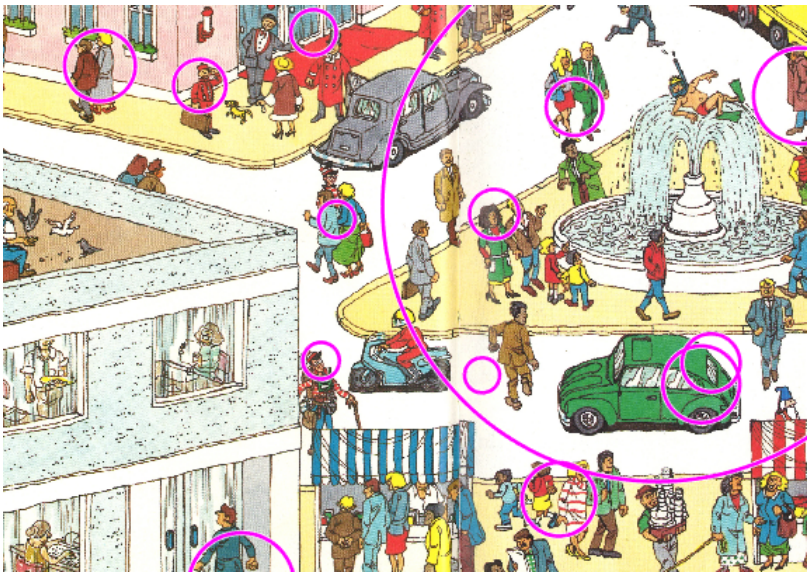

Figure 8: Features matching with a threshold = 1

Wally is found with this algorithm, but others part of the image are encircled without reasons. This is due to the fact that features matching has been applied with a threshold equals to 1, which is not especially high. The number of circles can be reduced by increasing this value to 1.12 for example. The result, represented on figure 9, is better but not still perfect.

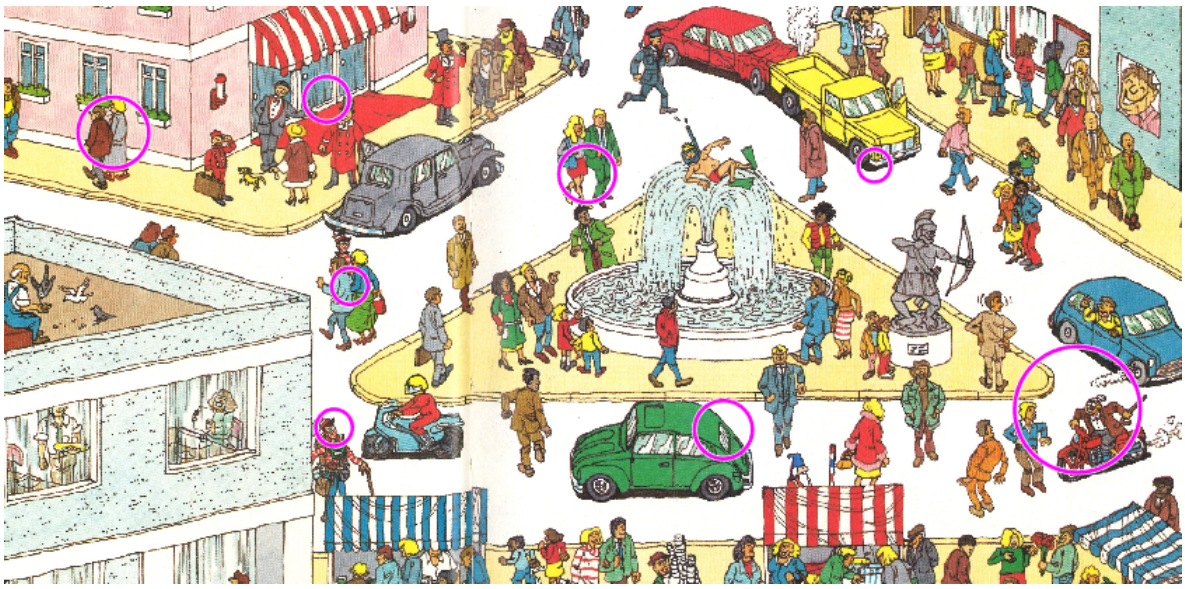

Figure 9: Features matching with a threshold = 1.12

The threshold cannot be increased for *Charlie4.jpg* because the circle pointing out Wally will disappear. In fact, the threshold corresponds to the euclidean distance between the features of *Charlie_head* and *I4*. However, Wally can be perfectly found on *Charlie2.jpg* using a threshold equals to 1.4 as proven by figure 10.

Figure 10: Features matching with a threshold = 1.4

On figure 10, only Wally is encircled. The problem is thus fully solved. This also true for *Charlie5.jpg*. However, other images cannot be solved using with this method. That is why the threshold is left equals to 1, and the unnecessary circles will be removed using a new feature matching method.

4 Apply another feature matching

The idea is to keep only the features of Wally's eyes instead of his full head. In fact, by analyzing the features of Wally in the different illustrations, Wally's eyes and its surroundings seemed to be the most characteristic feature of Wally.

To do so, the variable *Charlie_head* which corresponds to the default image will be truncated and the features will be extracted another time:

```
charlie_head_up = charlie_head(xmin:xmax,ymin:ymax,:);
I = charlie_head_up;
I = im2single(I);
I = rgb2gray(I);
I = imresize(I,resize_param_head);
I = imgaussfilt(I,gauss_filt);
[f_up,d_up] = vl_sift(I);
```

In the code above, *xmin xmax ymin ymax* corresponds to the borders of the new image. To extract the eyes, these parameters have been set to *80 115 25 100*. The figure 11 represents the obtained image.

Figure 11: Extracting the eyes from the default image

Once this is done, feature matching is performed between the features extracted from these eyes and the features which have been kept after the first feature matching. In fact, the idea is to select from the precedent circles (figure 9) the one which corresponds to Wally's eyes. This operation is performed using the following code:

```
[matches, scores] = vl_ubcmatch(d,d5,match_threshold);
[scores, sortIdx] = sort(scores,'descend');
matches5 = matches(:,sortIdx);
X5 = f5(1:2, matches5(2,:))';
r5 = f5(3, matches5(2,:))';
```

This algorithm works well as the figure 12 proves. In fact, only Wally is encircled in *Charlie5.jpg*. However, this methods only works for *Charlie5.jpg* and *Charlie2.jpg*.

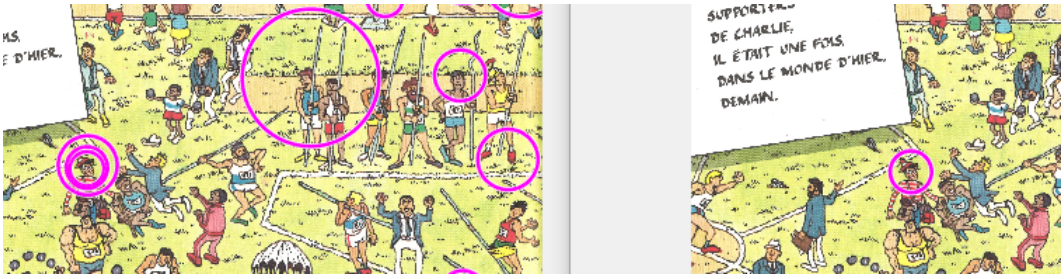

Figure 12: Wally found after two features matching

A way to improve the algorithm is to change truncated default image another time. The new dimensions of the image are the following *80 115 25 80*. The result is represented on figure 13.

Figure 13: New extraction of Wally's eyes from the default image

With this new default image, *Charlie2.jpg*, *Charlie3.jpg* and *Charlie4.jpg* are correctly solved, as proved on the figure 14. However, *Charlie5.jpg* is not solved this time (figure 15), which is regretful as the code was previously working for this image.

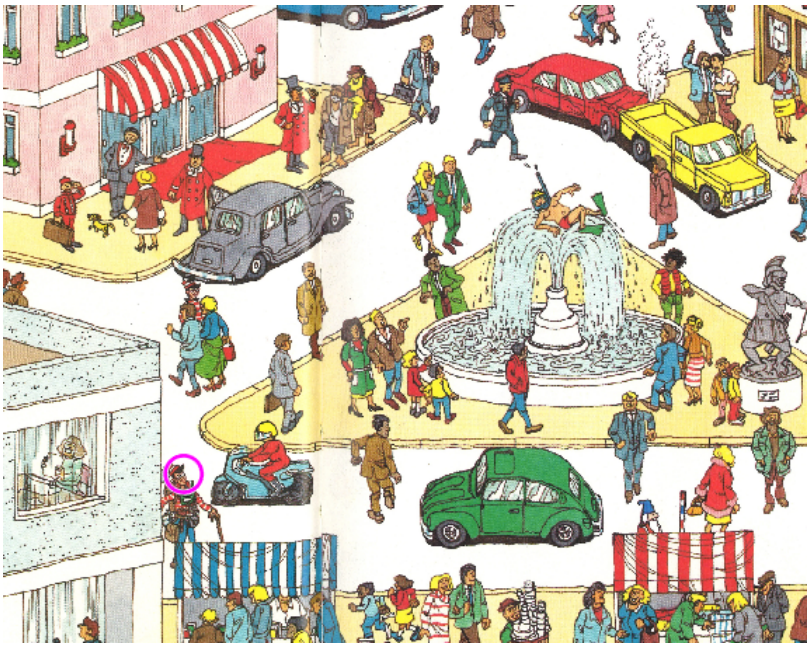

Figure 14: Wally finally found on *Charlie4.jpg*

Figure 15: Wally not found on *Charlie5.jpg*

5 Another way to improve the algorithm

Instead of doing a new feature matching, the problem can be solved by merging some features. The figures 16 17 show that by correctly tuning the hyper-parameters, it is possible to make Wally encircled many times.

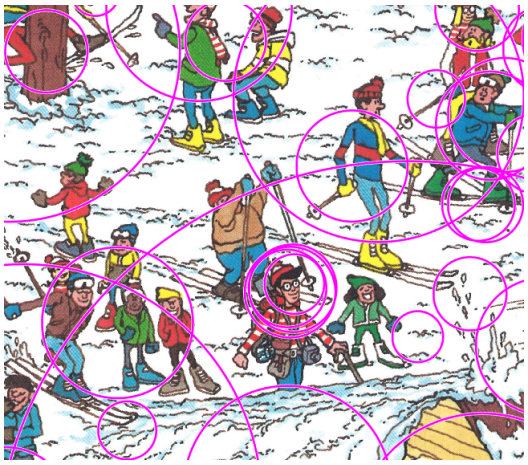

Figure 16: Stacked circles

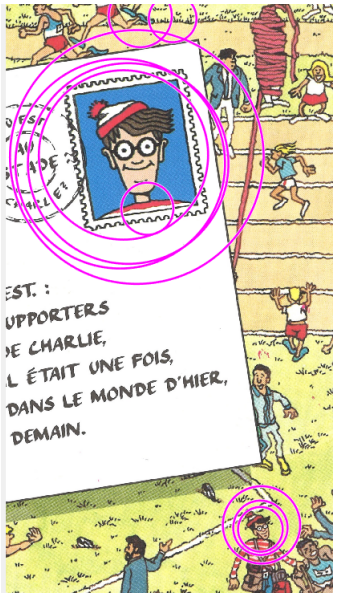

Figure 17: Stacked circles: another example

It is thus possible to code an algorithm which gives a better importance to encircled features which are close together. The degree of importance will be called weight, and the Wally could be found by selecting the feature with the highest weight. This method could make it possible to find Wally on *Charlie5.jpg*. However, this algorithm had not been implemented since some images like *Charlie4.jpg* does not encircle Wally many times (see figure 9).

6 Conclusion

It is probably impossible to provide a perfect solution to this problem using only SIFT computation. In fact, the algorithm can find Wally on *Charlie4.jpg* but fails to find him on *Charlie5.jpg*. On the contrary, it has been explained that another algorithm can make possible to find Wally on *Charlie5.jpg* but not on *Charlie4.jpg*.

A better algotihm which could have been implemented to almost perfectly solve this problem is the one using Convolutional Neural Networks (CNN). Nevertheless, this method would have needed way more than 5 images of Wally in order to be operational.