Universitetet
i Stavanger

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

IMAGE PROCESSING AND COMPUTER VISION

Feature descriptors
Analysis and comparison of image feature extraction methods: SIFT and SURF

Students :
Nourane BOUZAD
Álvaro ESTEBAN MUÑOZ
Nworkorie ONYEMAUCHE

Teacher :
Kjersti ENGAN

November 14, 2021

Contents

Abstract

The analysis of an image in computer vision is based on the detection and matching of the main characteristics of it, which is very useful in the recovery or reconstruction of images. Algorithms have been implemented to extract the main attributes of an image by comparing several images acquired from different angles or viewpoints by different sensors, for example. Commonly, a robust algorithm is defined when it is stable to transformations or geometric variations. This paper will present two recognized algorithms in feature detection and matching, such as scale invariant feature transform (SIFT) and accelerated robust features (SURF). Widely used, these matching algorithms are known for their simplicity of implementation and speed of computation. It will therefore be interesting to compare these descriptors on their overall performance, i.e. their robustness, their execution time and their ease of implementation.

1 Introduction

The technologies allowing the recovery or reconstruction of images are key points in image processing and computer science. The recovery of an image and thus its appropriate representation is a recurring problem that can be solved by algorithmic applications allowing the analysis and synthesis of an image or a video. These algorithms use tools called descriptors/detectors, which have become indispensable in the field of image representation. They allow, among other things, to characterize the general structure of the appearance of an image as well as its local structure by remaining invariant to a certain number of parameters such as rotation, changes in scale, point of view or illumination of the image. These descriptors are nowadays widely recognized and used to allow the identification of objects or living beings in order to be processed by an artificial intelligence, which from a neural network can compare a set of images and assign a name or characteristics to the objects of study. We will study in this paper the SIFT and SURF [3] [2] descriptors by implementing them respectively on Python and Matlab. We will compare their performances in order to describe their potential advantages and disadvantages.

The document is structured as follows: Section II and III describe the theory around descriptors by explaining how mathematically they work. Section IV discusses the algorithmic implementation of the SIFT and SURF methods, while Section V evaluates the performance of feature detectors and matching techniques. Finally, Section V concludes the study of this paper.

2 Review of SIFT descriptor

2.1 General description

The scale-invariant feature transform (SIFT) developed in 1999 by the researcher David Lowe, which can be translated as "scale-invariant visual feature transform", is an algorithm used in the field of computer vision to detect and identify similar elements between different digital images (landscape elements, objects, people, etc.).

The fundamental step of the method consists in calculating what are called "SIFT descriptors" of the images to be studied. These are numerical information derived from the local analysis of an image and which characterize the visual content of this image in the most independent way possible of the framing, the angle of observation and the luminosity. Thus, two photographs of the same object will have all the chances to have similar SIFT descriptors, and this all the more if the moments of shooting and the angles of view are close. Consequently, two photographs of very different subjects will produce very different SIFT descriptors, which reveals the discriminating power of this method. This robustness is a fundamental requirement for most applications and explains in large part the popularity of the SIFT method.

The SIFT algorithm consists of four main steps, which are scale space extreme detection, key point localization, orientation assignment and description generation.

2.2 Scale space extreme detection

The first step of the algorithm is based on the method of Difference of Gaussian (DoG) to subtract false details. To do this, a spatial scale $L(x, y, \sigma)$ is created from an image $I(x, y)$ which consists of the subtraction of a blurred version of the original image into another less blurred version of the same image. [5]

The scale space is:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Where $G(x, y, \sigma)$ is the Gaussian function, defined by:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2 e^{\frac{(x^2+y^2)}{2\sigma^2}}}$$

The blur is obtained by convolving the original image in gray levels with Gaussian kernels of very close variances. The blurring of an image by applying a Gaussian blur kernel removes the high spatial frequencies. Subtracting one image from the other preserves the spatial information between the two preserved frequency ranges in the two blurred images. Thus, the difference of Gaussian is comparable to a band pass filter that removes all spatial frequencies except those within a well-defined spectrum.

The Difference of Gaussian is preferable to use, as it is faster and more accurate than the Gaussian function and is calculated as follows :

$$DoG = I(x, y) * (G(x, y, k\sigma) - G(x, y, \sigma))$$

$$DoG = L(x, y, k\sigma) - L(x, y, \sigma)$$

2.3 Key point localization

In this second step, low contrast points are eliminated to make the location of key points more affordable. For this, after the generation of images from the difference of Gaussian, we apply the Laplacian to find these key points by detecting the corners and edges of the image. The Gaussian blur is very useful in this case because the second derivative or the Laplacian is very dependent on noise. In terms of computation time and performance, each pixel is subject to the computation of the Laplacian and is therefore very costly. The process can be redundant but still performs well.

2.4 Orientation assignment

In this step of the algorithm, orientations are assigned to each key point based on the gradient magnitude of the image.

2.5 Feature descriptors generation

In the final phase, once the local gradients of the image have been measured for each key point, we proceed to the calculation of the descriptor vectors. This step is performed on the smoothed image with the scale factor parameter closest to that of the key point considered. We consider around the key point, a region of 16 × 16 pixels, subdivided into 4 × 4 areas of 4 × 4 pixels each. On each zone, a histogram of the orientations is computed with 8 intervals. At each point of the area, the orientation and the amplitude of the gradient are calculated as before. The orientation determines the interval to be incremented in the histogram, which is done with a double weighting by the amplitude and by a Gaussian window centered on the key point. Then, the 16 histograms with 8 intervals each are concatenated and normalized to finally provide the SIFT descriptor of the key point, of dimension 128.

3 Review of SURF descriptor

The principle on which SURF is based is quite similar to SIFT. The difference can be seen in the generation of descriptors and their detection, particularly in terms of performance. This efficiency is achieved during the calculation of the hessian matrix, making the algorithm more accurate and fast.

3.1 Feature point detection

The SURF algorithm uses the Hessian matrix and more precisely the Laplacian of Gaussian (LoG) for detection. Thus, for the detection of key point of an image, in the scale space the Hessian matrix is defined as follows:

$$H(x, y, \sigma) = \begin{pmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{yx}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{pmatrix}$$

Where, the scale value σ is defined by the determinant of the Hessian matrix.

In the case of a detection of a contour, this one being located spatially and in frequency, it is clear that the spatial and frequency intervals must be small. Detecting a break in intensity involves an analysis of neighboring pixels, but the neighborhood should not be too large. Filtering the image results in smaller intensity variations, which should be reasonable.

Thus, in order to replace the Gaussian pyramid construction procedure used in SIFT, a process describing an integral image is defined, which is the Laplacian of Gaussian.

3.2 Matching process

The matching process does not require any digital or computer processing, as only the features are needed for this step and have been determined in the previous phases. Thus, the matching phase is quite fast depending on the potential rotations or other geometric variations of the image and then leads to the comparison of the different key points.

3.3 Feature point description

For the description of the key points, SURF first assigns an orientation by identifying, on the circle constructed at the key point, an area to be reproduced. To do this, the algorithm divides these different areas into a defined number of square subregions of dimensions 4 by 4. SURF then introduces a new mathematical tool, which is the Haar wavelet response. This tool allows assigning to each subregion a 4 dimensional vector, thus delivering 64 dimensional vectors (4 by 4 by 4). We then obtain the SURF descriptor.

4 Implementation

For our experiments, we use the SIFT OpenCV implementation and the SURF Matlab implementation. They are both based on the original papers, so the environment or language used to test them does not matter. However, if we want to compare, it is highly recommended using the same one. Indeed, the different use of a software could explain different execution times and therefore taint the results and conclusion.

On the one hand, the implementation of SIFT has some interesting points. The Gaussian Laplacian is very computationally expensive, so we use an approximation called difference of Gaussian, as explained in the previous section. As a reminder, we get it as the Gaussian blur difference of an image with two different σ. When we talk about the storage of descriptors, we have to keep in mind that the SIFT descriptor uses 128-dimensional vectors, which is a lot of memory and therefore makes the calculation slower.

This histogram, which is represented as a 128-dimensional vector, is the SIFT descriptor of the key point and all the descriptors in an image thus establish a true digital signature of the content.[1]

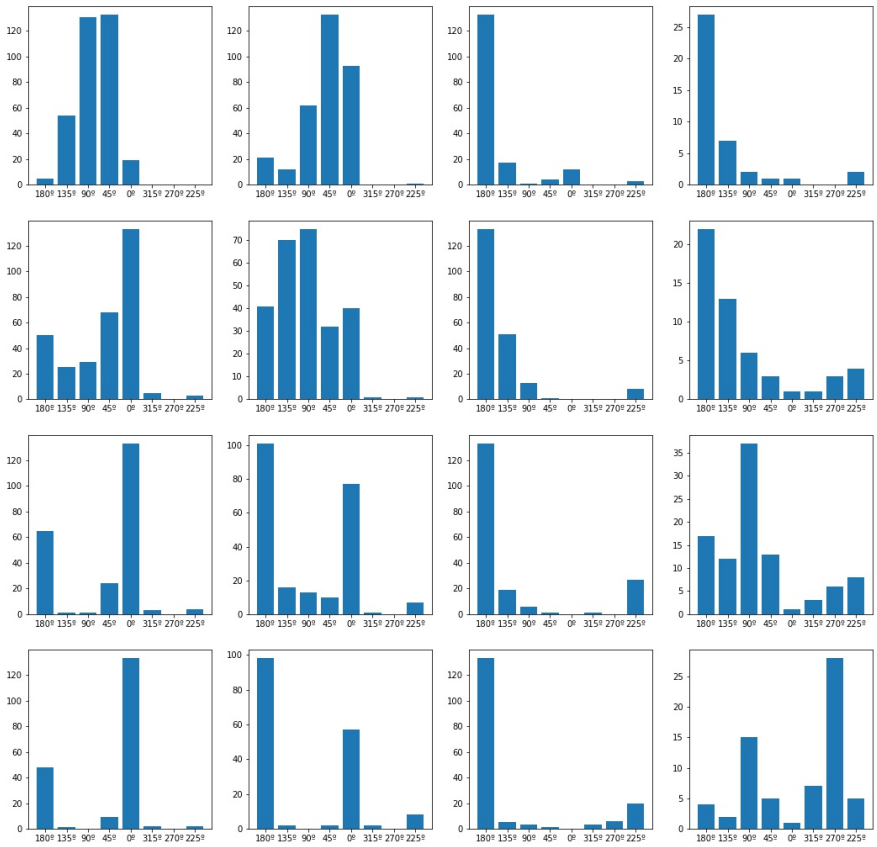


Figure 1: Stored descriptor for SIFT keypoints

On the other hand, the SURF implementation has a lot of tricks that make the computation very fast; the Gauss Laplacian is approximated with a very fast method, it is obtained thanks to the convolution of box filters.

It can be computed very easily with integral images, which makes the Laplacian of Gaussian computation even faster than SIFT. Comparing the memory storage with SIFT, SURF uses, as descriptor, 64-dimensional vectors, this saves a lot of memory and makes the process faster, however, this does not mean that it is less robust or worse. In fact, SURF is better when it comes to accuracy. As feature vectors, we use a neighborhood region of 4x4 subregions and for each subregion, we get the horizontal and vertical Haar wavelet responses, then we compute the sum of these values, as well as the absolute values, over the subregion, these four values will be our subregion features. It is important to note that another advantage of SURF is that we use the sign of the Laplacian (which does not add any computational cost, as it is already computed) for faster matching. The Laplacian sign allows us to classify light and dark blobs, which allows us to match points only if they are of the same blob type.

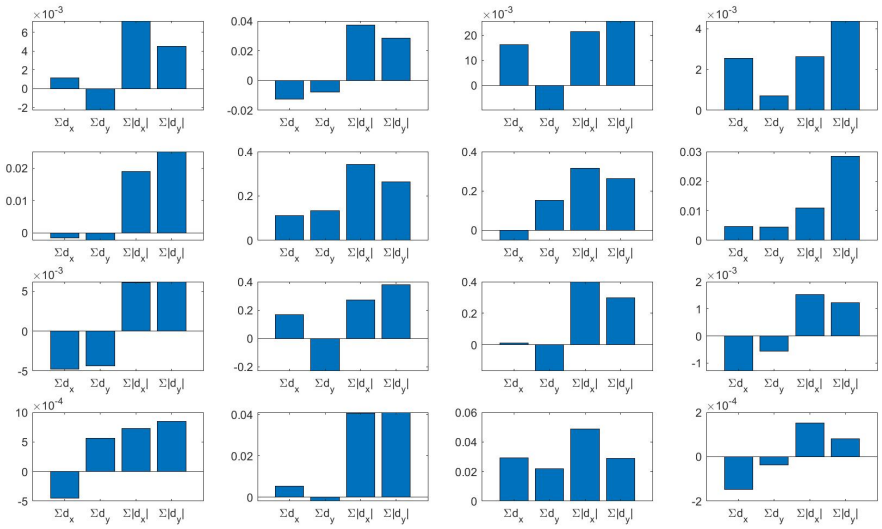


Figure 2: Stored descriptor for SURF keypoints

5 Experiments and results

In order to verify the invariance of these descriptors, the images used were rotated by a certain angle, allowing to test the invariance in rotation. In a second step, the addition of different noises on the images allowed us to analyze the robustness of the descriptors. Indeed, the noise blurs the images, making the details less visible and more particularly the contours of the objects, which can alter the background of the image. In a general way, the chosen images can be studied in different cases of study that we specified later.

It's important to point out that we tested SIFT implementation of OpenCV in a python language programming, while using Matlab's SURF implementation. Experiments were carried out using these implementations, however, the code used for the experiments has been build from scratch and will be available with this report.

5.1 SIFT algorithm

5.1.1 Features detection and description

We simulate here the part of the algorithm that is based on the detection of circular "interesting" areas on the image, centered around the key points.

These areas are characterized by their visual unity and generally correspond to distinct elements on the image. As explained earlier, on each of them, the algorithm detects an intrinsic orientation which is used as a basis for the construction of a histogram of the local orientations of the contours, weighted, thresholded and normalized for more stability.

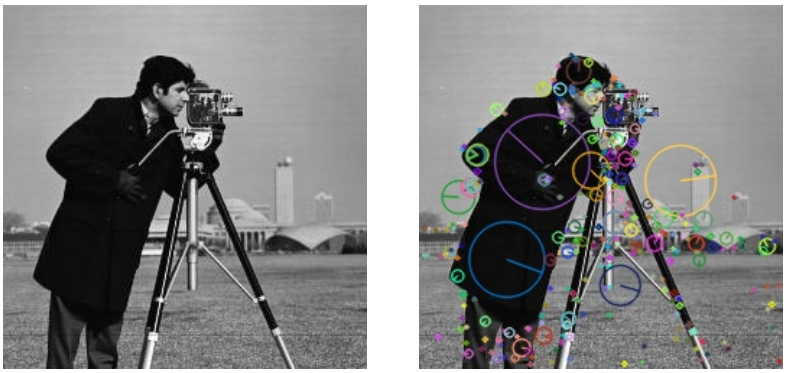


Figure 3: Interest points detected with SIFT algorithm

5.1.2 Matching process

In this section, in order to test the matching process of the algorithm, two pictures were taken with an IDS camera.

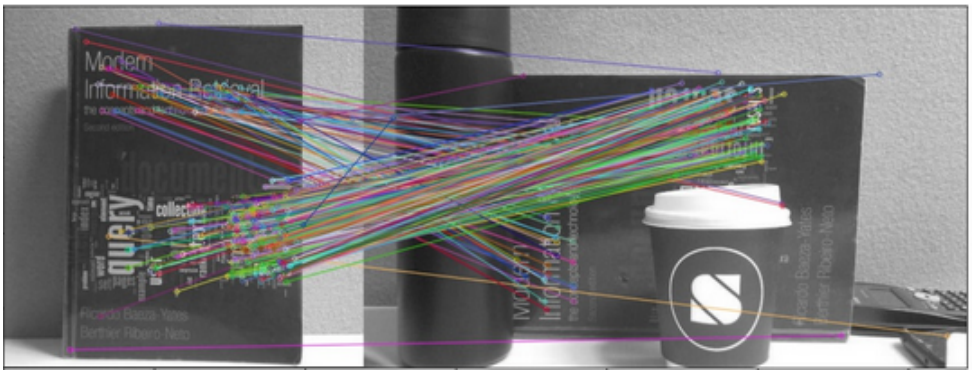


Figure 4: Matching orientation with SIFT algorithm

Generally, the matching algorithm is used when different sensors take images from different angles, obtaining differently oriented images. In order to evaluate this ability of the algorithm, we took these two images in which we have the book is normally arranged and another in which it is turned to 90 degrees, simulating two images taken by different satellites for example. We can see on these two images that the key points are well detected and that the matching is ideally done.

5.2 SURF algorithm

As with the SIFT algorithm, we tested the same images to evaluate how well the code works.

5.2.1 Feature detection and description

Like SIFT, the first step of the experiment is to test the algorithm's ability to extract key feature points using the same images as for the SIFT algorithm.

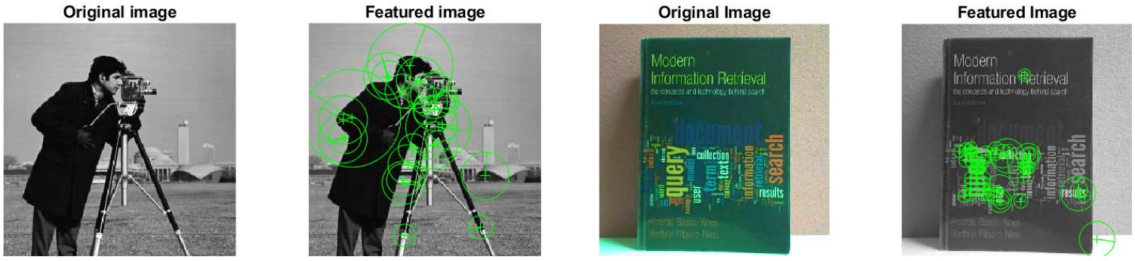


Figure 5: Interested point detected with SURF

5.3 Comparison between SIFT and SURF

In this section, we evaluate the effectiveness of SIFT and SURF descriptors of original and noisy images by detecting the number of feature points and comparing the simulation times of both algorithms.

5.3.1 Set of images used

In order to compare the performance between the two algorithms, we have gathered a set of images in which we are going to apply the algorithm.

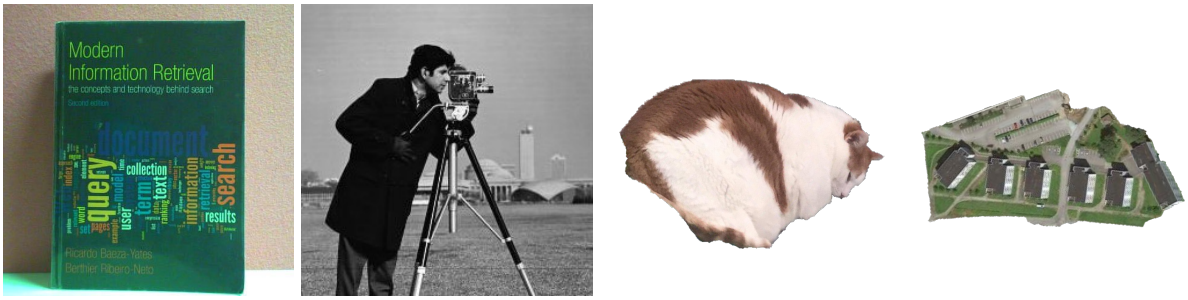


Figure 6: Set of images used for the experiments

To satisfy our experiments, we have chosen four completely different images that can be used for different purposes. Indeed, the first two images can be used for the description of object or individual in full activity, the third is widely used for localization needs in satellite imagery. The last one is an animal (cat) which can be used for motion detection, for example.

5.3.2 Experiment realization

Focusing on the last two images we use for the experiments, you should notice we intentionally extracted them from other images, so we can use the algorithms to check if they are able to match the object in the original image. Image number three is a cat rotated 35º, while image number four is a satellite picture of Sørmarka student boliger rotated -20º (Extracted from Google Maps). Matlab plots feature matching pasting one image (red image) above the other one (blue image) and matching red points with green points, notice how is it done on the following examples.

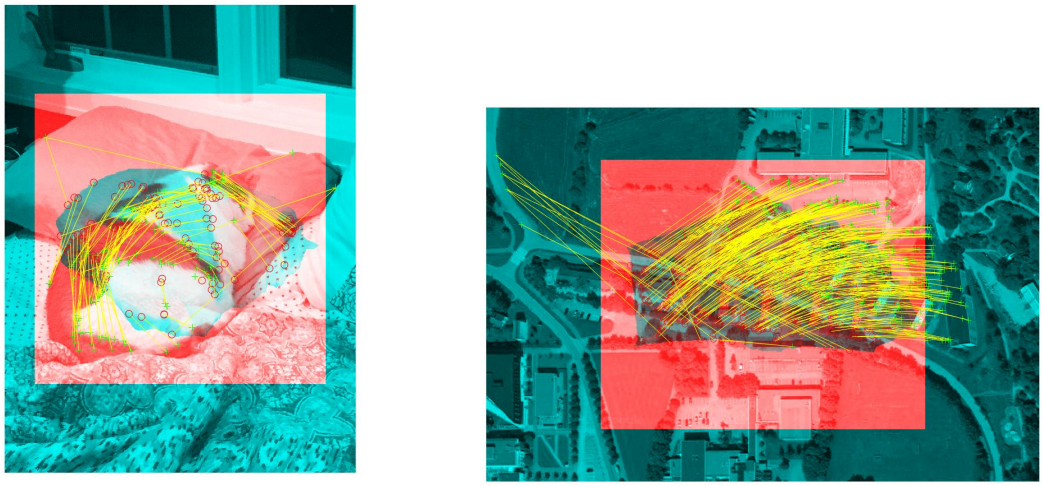


Figure 7: SIFT Matches

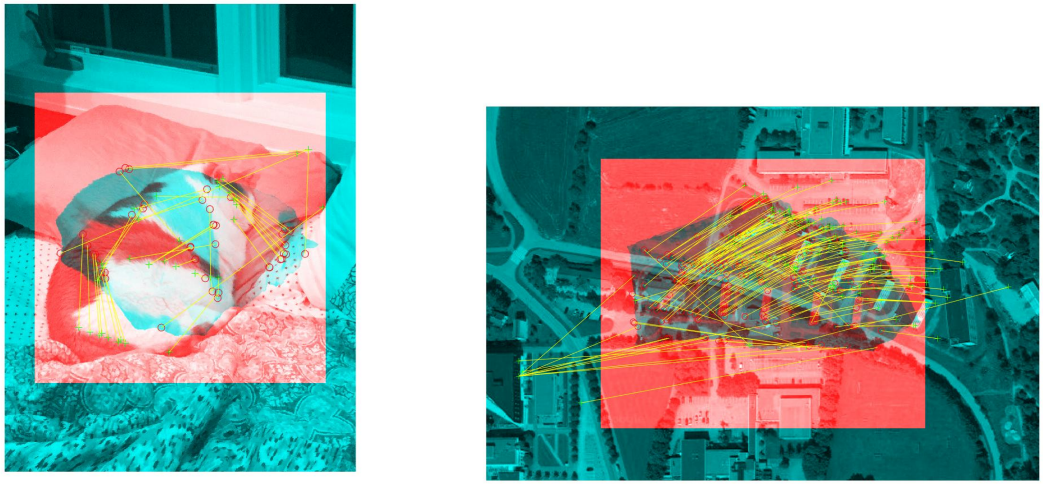


Figure 8: SURF Matches

5.3.3 Algorithm performance on the images

To test the performance of the algorithms in our set of images, we have selected a set of metrics we check on every image for each algorithm. We have simulated both algorithms in Matlab and got the following results:

Image	Criteria	SIFT	SURF
1	Detection Time (s)	0.072	0.041
	Extraction Time (s)	0.127	0.031
	Matching Time (s)	0.230	0.009
	Nº Features Detected	952	457
	Nº Matching Features	100	48
2	Detection Time (s)	0.024	0.017
	Extraction Time (s)	0.051	0.023
	Matching Time (s)	0.018	0.002
	Nº Features Detected	274	182
	Nº Matching Features	123	33
3	Detection Time (s)	0.230	0.079
	Extraction Time (s)	0.346	0.017
	Matching Time (s)	0.702	0.011
	Nº Features Detected	500	206
	Nº Matching Features	144	43
4	Detection Time (s)	0.165	0.056
	Extraction Time (s)	0.257	0.036
	Matching Time (s)	1.464	0.027
	Nº Features Detected	1123	455
	Nº Matching Features	563	154

It can be seen that the number of key feature points extracted for SIFT is significantly higher than SURF for the different images tested.

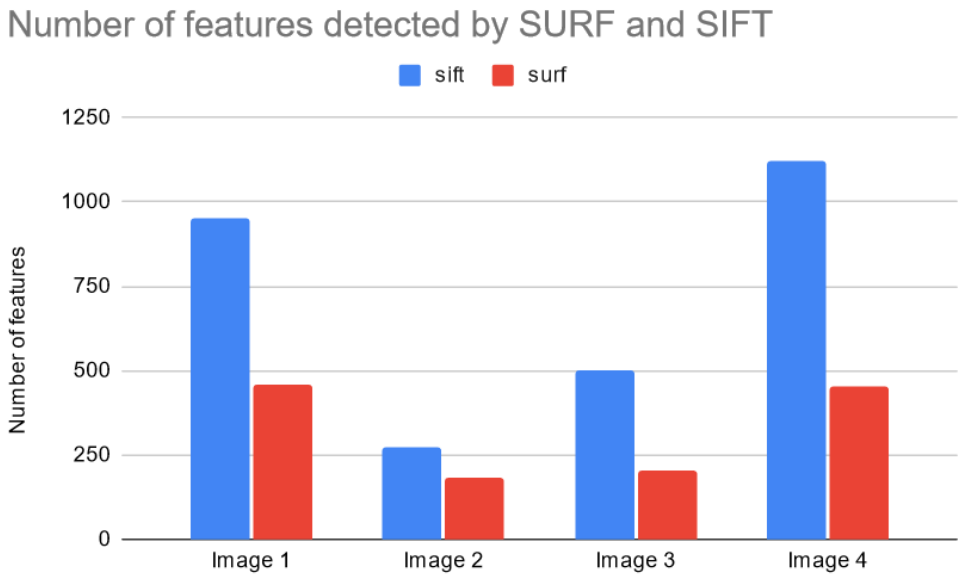


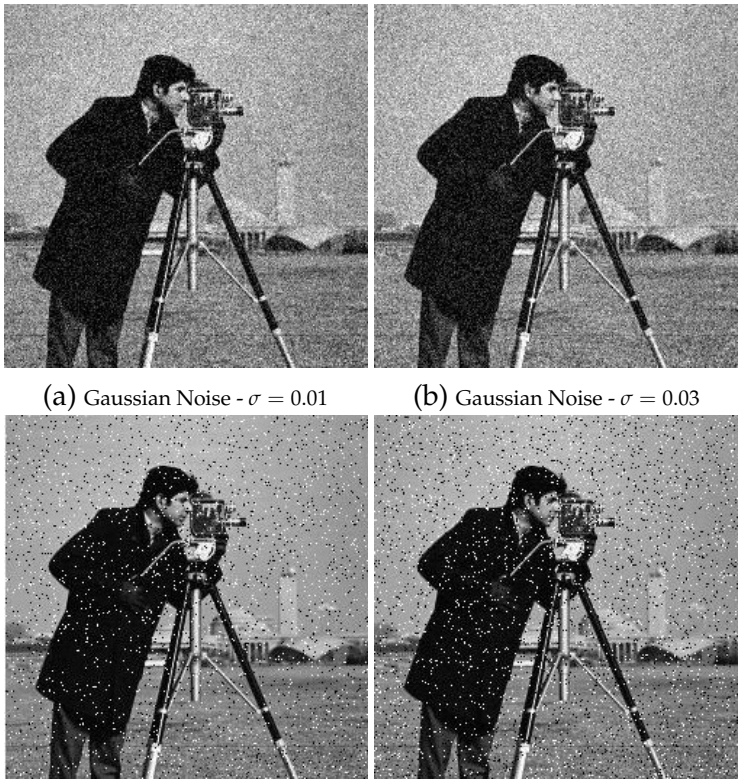
Figure 9: Number of features detected

Therefore, the simulation time is obviously different, making the feature search more time-consuming to SIFT.

5.3.4 Performance of descriptors on noisy images

In order to determine the key point matching between the original and noisy images, all characteristic key points are stored. Intuitively, the higher the number of matches between the original and noisy images, the higher the accuracy. Nevertheless, the recurrent problem that we can encounter is the simulation time, which is greater as soon as the accuracy increases.

For this experiment, we choose to use one image only, which is the cameraman picture being enough to make the conclusions about the robustness of the descriptors.



(a) Gaussian Noise - $\sigma = 0.01$ (b) Gaussian Noise - $\sigma = 0.03$

(c) Salt and Pepper Noise - $\sigma = 0.05$ (d) Salt and Pepper Noise - $\sigma = 0.07$

Figure 10: Set of noise images

The figure above illustrates the noisy derivatives images of the cameraman picture. Indeed, we have implemented two different noises, a Gaussian noise and a SaltPepper noise with 2 different variances each. The interest here is to evaluate if the descriptors remain robust to the addition of noise to an image, and how they react to it in the opposite case. Thus, for the Gaussian noise, we imposed a variance of 0.01 and 0.03 and for the SaltPepper noise variances of 0.05 and 0.07. The results obtained from this experiment are summarized in the following table:

Type of Noise	Variance	Criteria	SIFT	SURF
Gaussian	0.01	Detection Time (s)	0.017	0.019
		Extraction Time (s)	0.033	0.017
		Matching Time (s)	0.013	0.002
		N° Features Detected	330	228
		N° Matched Features	25	32
	0.03	Detection Time (s)	0.016	0.012
		Extraction Time (s)	0.033	0.012
		Matching Time (s)	0.013	0.002
		N° Features Detected	335	240
		N° Matched Features	30	30
Salt & Pepper	0.05	Detection Time (s)	0.017	0.016
		Extraction Time (s)	0.039	0.017
		Matching Time (s)	0.025	0.003
		N° Features Detected	328	257
		N° Matched Features	15	19
	0.07	Detection Time (s)	0.017	0.012
		Extraction Time (s)	0.033	0.014
		Matching Time (s)	0.016	0.002
		N° Features Detected	344	313
		N° Matched Features	7	19

The table illustrates, for different images with a certain type of noise, the performance of the SIFT and SURF algorithms. We can see that the algorithms adopt a similar behavior to the one without noise addition, i.e. SIFT remains more precise as for the number of key characteristic points that it detects compared to the SURF algorithm. Nevertheless, these characteristic points are not said to be determinant in the identification of the major features of the photo, so we can say that SURF remains a robust algorithm thanks to its computational speed and its efficiency on this short simulation time. The images that we treat remain rather simple in the sense that they do not present a lot of details, but we can imagine that when the images are more complex with a more complex background too, the performance of the algorithms is determining. Thus, we can deduce that SURF is a more robust algorithm than SIFT thanks to its efficiency in detecting the main key points of an image in a shorter simulation time.

The conclusions being the same as for the noise-free images, we can conclude on the efficiency of the SURF algorithm, which is more efficient in the detection and description of key features. Indeed, SURF detects fewer features, but the key points detected are major features of the image and are therefore largely sufficient in the description of the image. The SIFT algorithm remains an efficient detection method despite a longer execution time. We have seen in the theory of these methods of image processing that these simulation times are explained from the first stage of detection. Indeed, the Gaussian difference is a mathematical tool which is certainly more precise, but also analytically heavier, whereas the Gaussian Laplacian is a more efficient tool numerically, allowing SURF to be robust and fast in execution and experimentation. Noise is therefore not a source that can alter the respective efficiency of these methods in the analysis of their performances. Also, we could notice that when adding noise despite the fact

that SIFT detects more features it detects less number of characteristic key points than SURF, making SURF a more robust algorithm. This is in line with an aspect that we have studied concerning the lack of robustness of SIFT when the background of an image becomes more complex.

6 Conclusion

In this paper, the SIFT and SURF algorithms have been implemented to test the detection, description and matching of key points and to derive the main performance of each of these descriptors.

Moreover, it has been interesting to evaluate the efficiency of these descriptors on noisy images by determining the points of correspondence between the original and noisy images.

Finally, to overcome the few weaknesses of the SIFT algorithm, improvements have been made to the algorithm to remedy the various points raised in this article. Indeed, algorithms such as B-SIFT, AH-SIFT or WLIB-SIFT allow precision and a greater speed of calculation, making the SIFT method a basis of robustness in image processing.[4]

References

[1] Ankit Agarwal, Devesh Samaiya, and K. K. Gupta. "A Comparative Study of SIFT and SURF Algorithms under Different Object and Background Conditions". In: *2017 International Conference on Information Technology (ICIT)*. 2017, pp. 42–45. DOI: 10.1109/ICIT.2017.48.

[2] Tinne Tuytelaars Herbert Bay and Luc Van Gool. "SURF: Speeded Up Robust Features". In: (2008).

[3] David g. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: (2004).

[4] Zhen-Sheng Ni. "B-SIFT: A Binary SIFT Based Local Image Feature Descriptor". In: *2012 Fourth International Conference on Digital Home*. 2012, pp. 117–121. DOI: 10.1109/ICDH.2012.69.

[5] Sidheswar Routray, Arun Kumar Ray, and Chandrabhanu Mishra. "Analysis of various image feature extraction methods against noisy image: SIFT, SURF and HOG". In: *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. 2017, pp. 1–5. DOI: 10.1109/ICECCT.2017.8117846.

7 Appendix

7.1 Algorithms documentation

Python - OpenCV: SIFT documentation

Matlab SURF documentation

Matlab Feature Extraction documentation

Matlab Feature Matching documentation

Matlab SIFT documentation

7.2 SIFT Code - Python

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# Import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pprint


# # Testing SIFT feature detector on an image

# In[2]:


# Read the image and turn it to gray scale
img = cv2.imread('./images/cameraman.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create() # Create instance of the class SIFT
    feature detector

# FIRST STEP: Feature points detection
feature_points = sift.detect(img_gray, None)

# Draw the points on the image
featured_img = None
featured_img = cv2.drawKeypoints(img_gray, feature_points,
    featured_img,
                                 flags=cv2.
                                     DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
                                     )

# For writing the image in a file
cv2.imwrite('./images/sift_keypoints.jpg', featured_img)

# Plot the results
plt.figure(figsize=(15,15))
plt.subplot(121)
plt.imshow(img_gray, cmap='gray')
```

```
plt.subplot(122)
plt.imshow(featured_img, cmap='hsv')
plt.show()


# In[3]:


# SECOND STEP: Feature descriptor extraction
sift_des = sift.compute(img_gray, feature_points)

# Let's check what we are storing in memory...

"""
The descriptor stores a tuple of two elements, the first one is the array of keypoints, we can check it with
sift_des[0]. The second element is the array of vector descriptors, each vector is the descriptor of the
keypoint with the same index on the keypoints array.
"""

# Array of keypoints (Printed only from the 90th to the 100th point)
for num, keypoint in enumerate(sift_des[0][90:110]):
    print(f"Kp {num+90}:" +
          str([f"Angle: {keypoint.angle:.2f}",
               f"Coords: ({keypoint.pt[0]:.2f}, {keypoint.pt[1]:.2f})",
               f"Size: {keypoint.size:.2f}",
               f"Octave: {keypoint.octave:.2f}"]
              ))


# In[4]:


# Array of feature vectors (each one related to the same indexed keypoint)
pprint.pprint(sift_des[1])


# In[5]:


# Let's have a look at keypoint 100
feature_vector = sift_des[1][100]
bins = 8 # Values used for sampling the gradient orientation

# Histogram values, start on 180   and goes clockwise
```

```
x=["180  ", "135  ", "90  ", "45  ", "0  ", "315  ", "270  ", "225
    "]

# Sift descriptor of the 100th keypoint
plt.figure(figsize=(18,18))
for i in range(len(feature_vector)//bins):
    gradient_h = feature_vector[i*bins:i*bins+bins]
    plt.subplot(4,4,i+1)
    plt.bar(x, gradient_h)
plt.savefig('./images/descriptor.jpg')
plt.show()


# In[6]:


from collections import defaultdict
import pprint

sum_dir = defaultdict(int)
for index, element in enumerate(feature_vector):
    sum_dir[x[index % 8]] += element

pprint.pprint(sum_dir)
print(sift_des[0][100].angle)


# # Matching of feature points using SIFT

# Let's try to use our descriptors to find an object on an image.

# In[7]:


# Read the image of the object
img_object = cv2.imread('./images/book_object.jpg')
gray_object = cv2.cvtColor(img_object, cv2.COLOR_BGR2GRAY)

# Find the feature points and compute the descriptors
fp_obj, des_obj = sift.detectAndCompute(gray_object, None)

# Draw the feature points on the image
featured_img = None
featured_img = cv2.drawKeypoints(gray_object, fp_obj, featured_img,
                                 flags=cv2.
                                    DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
                                    )

# For writing the image in a file
cv2.imwrite('./images/sift_keypoints_book.jpg', featured_img)
```

```
# Plot results
plt.figure(figsize=(15,15))
plt.subplot(121)
plt.imshow(img_object)
plt.subplot(122)
plt.imshow(featured_img, cmap='hsv')
plt.show()


# *Note color image is in BGR color space*

# Now, let's try to find the object using the descriptors we
    computed

# In[8]:


# Read an image where the object appears
img = cv2.imread('./images/book.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# We can use a method that detect and compute the descriptors
    directly

fp, des = sift.detectAndCompute(img_gray, None)

# We create an instance of a BruteForce matcher
bf = cv2.BFMatcher()

# Compute the matches between the descriptors
matches = bf.knnMatch(des_obj, des, k=2)

# Apply a radio test so we don't get many outliers
true_matches = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        true_matches.append([m])

# cv.drawMatchesKnn expects list of lists as matches.
img_matches = cv2.drawMatchesKnn(gray_object, fp_obj, img_gray, fp,
                                 true_matches, None, flags=cv2.
                                    DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
                                    )


# Plot results
plt.figure(figsize=(15,15))
plt.imshow(img_matches)
plt.savefig("./images/matches.jpg")
plt.show()
```

7.3 SURF Code - Matlab

```
%% Testing SURF algorithm
%% Detection of feature points

% Read the image
img = imread("./images/test/cameraman_obj.jpg");
img = rgb2gray(img);

% Detect feature points
points = detectSURFFeatures(img);

% Extract the descriptors
[descriptors, points] = extractFeatures(img, points);

% Plot the results
figure('Position', [0, 0, 1800, 1800]);
subplot(121)
imshow(img)
title("Original image")
subplot(122)
imshow(img); hold on;
plot(points.selectStrongest(25), 'ShowOrientation', true);
title("Featured image")
saveas(gcf, "./images/SURF_features.jpg")
%% SURF descriptor definition

% Let's study how it is stored in memory the SURF descriptor
descriptors
%%
% Notice for SURF we only use 64-dimensional vectors to describe a
    point, half
% of the space we use for SIFT, this makes it faster and it's
    proved that it's
% even more robust than SIFT.
%
% The dimensionality of the vector comes from the same point of
    SIFT, we use
% a neighbourhood region of 4x4 sub regions and for each subregion
    we get horizontal
% and vertical Haar wavelet responses and their sum over over each
    subregion.

% Example of the first point descriptor vector
descriptors(1, :)
descriptor = descriptors(1, :);  % Take the first descriptor vector
subregions = 16;
x = categorical({'\Sigma{d_x}', '\Sigma{d_y}', '\Sigma{|d_x|}', '\
    Sigma{|d_y|}'});
figure('Position', [0, 0, 1800, 1800]);
for i=0:subregions-1
```

```
    y = descriptor((i*4)+1:(i*4)+4);
    subplot(4,4,i+1);
    bar(x, y);
end
saveas(gcf, "./images/SURF_descriptor.jpg");
%% SURF Feature matching

% Read the image of the object
im_obj = imread("./images/test/book_obj.jpg");
obj_gray = rgb2gray(im_obj);

% Detect feature points
fp_obj = detectSURFFeatures(obj_gray);

% Extract the descriptors
[des_obj, fp_obj] = extractFeatures(obj_gray, fp_obj);

% Plot the object image and the featured image
figure;
subplot(121)
imshow(im_obj)
title("Original Image")
subplot(122)
imshow(obj_gray);
hold on;
plot(fp_obj.selectStrongest(50), 'ShowOrientation', true);
hold off
title("Featured Image")
saveas(gcf, "./images/SURF_features_book.jpg")
%%
% Match the object features with an image where the object appears
im = imread("./images/test/book.jpg");
im_gray = rgb2gray(im);

% Detect feature points
fp = detectSURFFeatures(im_gray);

% Extract the descriptors
[des, fp] = extractFeatures(im_gray, fp);

[indexPairs,matchmetric] = matchFeatures(des_obj,des);

matchedPoints_obj = fp_obj(indexPairs(:,1),:);
matchedPoints = fp(indexPairs(:,2),:);

figure;
showMatchedFeatures(im_obj,im,matchedPoints_obj,matchedPoints);
saveas(gcf, "./images/matches_book.jpg")
```

7.4 SIFT VS. SURF - Comparison Script

```
%% Comparing SIFT and SURF feature detectors/descriptors
% It is always interesting to test how two different algorithms perform in a
% set of images, let's do some experiments on a set of pictures comparing SIFT
% against SURF performance.

% Define some variables
im_obj_filepaths = ["./images/test/book_obj.jpg", ...
    "./images/test/cameraman_obj.jpg", "./images/test/fattyCat_obj.png", ...
    "./images/test/sormarka_obj.png"];

im_filepaths = ["./images/test/book.jpg", ...
    "./images/test/cameraman.jpg", "./images/test/fattyCat.jpg", ...
    "./images/test/sormarka.png"];

Descriptor = {'SIFT', 'SURF'};
Metrics = {'Detection Time', 'Extraction Time', 'Matching Time', 'N  Feature Points', ...
    'N  Matched Features'};

num_ims = 4;
num_metrics = length(Metrics);

SIFT_times = zeros(num_ims, num_metrics);
SURF_times = zeros(num_ims, num_metrics);

use_noise = false;
variance = 0.05;
noise = 'salt & pepper';

display_results = false;
save_data = false;
%% SIFT MEASUREMENT

for i=1:num_ims

    im_obj = imread(im_obj_filepaths(i));    % Read object image
    im_obj_gray = rgb2gray(im_obj);          % Turn image into gray scale

    im = imread(im_filepaths(i));            % Read image for object recognition
    im_gray = rgb2gray(im);                  % Turn image into gray scale

     % Apply noise
```

```
    if use_noise
        im_obj_gray = imnoise(im_obj_gray, noise, variance);
        im_gray = imnoise(im_gray, noise, variance);
    end

    % KEYPOINT DETECTION
    % Time testing
    f = @() detectSIFTFeatures(im_obj_gray);    % Handle function
    SIFT_det = timeit(f);                       % Test time of the
       function

    % Detect keypoints
    sift_points_obj_det = detectSIFTFeatures(im_obj_gray);  %
       Detect keypoints of the object
    sift_points = detectSIFTFeatures(im_gray);              %
       Detect keypoints of the background image


    % FEATURE EXTRACTION
    % Time testing
    f = @() extractFeatures(im_obj_gray, sift_points_obj_det);  %
       Handle to function
    SIFT_ext = timeit(f);                                   % Test time
        of the function

    % Extract features
    [des_obj, sift_points_obj] = extractFeatures(im_obj_gray,
       sift_points_obj_det); % Extract feature descriptors of the
       object
    [des, sift_points] = extractFeatures(im_gray, sift_points);
                       % Extract feature descriptors of the
       background image

    % Plot results
    if display_results
        figure;
        subplot(121)
        imshow(im_obj);
        title("Original Image");
        subplot(122)
        imshow(im_obj_gray), hold on;
        plot(sift_points_obj.selectStrongest(50), 'ShowOrientation'
           , true);
        title("Featured Image");
        filename = "./images/Image" + i + "_det_SIFTfp.jpg";
        if save_data
            saveas(gcf, filename);
        end
    end
```

```
    % FEATURE MATCHING
    % Time testing
    f = @() matchFeatures(des_obj, des);    % Handle to function
    SIFT_match = timeit(f);                 % Test the time of the
       function

    % Match features
    [indexPairs, matchmetric] = matchFeatures(des_obj, des);
    matchedPoints_obj = sift_points_obj(indexPairs(:,1),:);
    matchedPoints = sift_points(indexPairs(:,2),:);

    % Plot results
    if display_results
        figure;
        showMatchedFeatures(im_obj_gray, im_gray, matchedPoints_obj
           , matchedPoints);
        filename = "./images/Image" + i + "_SIFTmatches.jpg";
        if save_data
            saveas(gcf, filename);
        end
    end


    % SAVE RESULTS
    SIFT_times(i, :) = [SIFT_det, SIFT_ext, SIFT_match,
       sift_points_obj_det.Count, ...
        length(indexPairs)];

end
%% SURF MEASUREMENT

for i=1:num_ims

    im_obj = imread(im_obj_filepaths(i));   % Read object image
    im_obj_gray = rgb2gray(im_obj);         % Turn image into gray
       scale

    im = imread(im_filepaths(i));           % Read image for object
        recognition
    im_gray = rgb2gray(im);                 % Turn image into gray
       scale

    % Apply noise
    if use_noise
        im_obj_gray = imnoise(im_obj_gray, noise, variance);
        im_gray = imnoise(im_gray, noise, variance);
    end

    % KEYPOINT DETECTION
```

```
    % Time testing
    f = @() detectSURFFeatures(im_obj_gray);    % Handle function
    SURF_det = timeit(f);                       % Test time of the
       function

    surf_points_obj_det = detectSURFFeatures(im_obj_gray);  %
       Detect keypoints of the object
    surf_points = detectSURFFeatures(im_gray);              %
       Detect keypoints of the background image


    % FEATURE EXTRACTION
    % Time testing
    f = @() extractFeatures(im_obj_gray, surf_points_obj_det); %
       Handle to function
    SURF_ext = timeit(f);                               % Test the time
        of the function

    % Extract features
    [des_obj, surf_points_obj] = extractFeatures(im_obj_gray,
       surf_points_obj_det); % Extract feature descriptors of the
       object
    [des, surf_points] = extractFeatures(im_gray, surf_points);
                    % Extract feature descriptors of the background
        image

    % Plot results
    if display_results
        figure;
        subplot(121)
        imshow(im_obj);
        title("Original Image");
        subplot(122)
        imshow(im_obj_gray), hold on;
        plot(surf_points_obj.selectStrongest(50), 'ShowOrientation'
           , true);
        title("Featured Image");
        filename = "./images/Image" + i + "_det_SURFfp.jpg";
        if save_data
            saveas(gcf, filename);
        end
    end


    % FEATURE MATCHING
    f = @() matchFeatures(des_obj, des); % Handle to function
    SURF_match = timeit(f);                 % Test the time of the
       function

    % Match features
```

```
    [indexPairs, matchmetric] = matchFeatures(des_obj, des);
    matchedPoints_obj = surf_points_obj(indexPairs(:,1),:);
    matchedPoints = surf_points(indexPairs(:,2),:);

    % Plot results
    if display_results
        figure;
        showMatchedFeatures(im_obj, im, matchedPoints_obj,
           matchedPoints);
        filename = "./images/Image" + i + "_SURFmatches.jpg";
        if save_data
            saveas(gcf, filename);
        end
    end

    SURF_times(i, :) = [SURF_det, SURF_ext, SURF_match,
       surf_points_obj_det.Count, ...
        length(indexPairs)];

end
%% Plot and save results

% Create table to compare results
for i=1:num_ims
    Image_measures = [SIFT_times(i, :); SURF_times(i, :)];

    Table = array2table(Image_measures, "VariableNames", Metrics, "
       RowNames", Descriptor);

    if save_data
        filename = "./data/results_image" + i + ".csv";
        writetable(Table, filename);
    end

    tableName = "Image " + i;
    Table = table(Table, 'VariableNames', tableName);

    disp(Table)
end
```