République Algérienne Démocratique et Populaire

Ministaire de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Département d'Électronique

Projet de fin d'etude en vue de l'obtention du Diplome de Master en Électronique

Application de l'algorithme SURF pour le Tracking d'objets

Réalisé par :
Boubezari Rayana

Proposé et Encadré par :
Pr C.Larbes
Pr A.Bouridane

Promotion 2013

ملخص

المغزى من هذا المشروع دراسة خوارزمية لمعالجة الصور ، صممت للكشف عن الأشياء و التتبع في الفيديو . وتتمثل في خوارزميّة SURF

.زيادة على الدّراسة النّظرية ، تطبيق أجري لتتبع جسم متحرّك

هذا التّطبيق أجرى لمتابعة جسم متحرّك باستعمال المكتبة OpenCV أمّا التتبع بواسطة كاميرا.

الكلمات المفتاحيّة: رؤية بواسطة الحاسب، الكشف، واصف ، نقطة التوقف ،تتبع ، فيديو .

Résumé

Ce projet a pour objectif l'étude d'un algorithme de traitement d'image, conçu pour la reconnaissance d'objet, et le suivi sur des vidéos. Il s'agit de l'algorithme SURF.

En plus de l'étude théorique, une application de cet algorithme a été réalisée. C'est un programme de tracking d'objet en mouvement.

L'application a été réalisée à l'aide de la librairie OpenCV, et le suivi se fait à l'aide de la webcam.

Mots clés : Vision par ordinateur, SURF, Reconnaissance, Descripteur, Point d'intérêt, Suivi, Vidéo.

Abstract

This project aims to study an image processing algorithm, designed for object recognition and tracking. The algorithm is called SURF.

In addition to the theoretical study, an application of this algorithm was performed. It is a tracking moving object code.

The application was created using the OpenCV library, and tracking is done using the webcam.

Keywords : Computer Vision, SURF, recognition, descriptor, interest point, tracking, Video.

Remerciements

Je tiens d'abord à exprimer ma gratitude à mes promoteurs, au Pr.BOURIDANE pour m'avoir initiée au domaine du traitement d'image, et au Pr.LARBES pour la disponibilité et l'écoute permanentes dont il a fait preuve tout au long de ce projet.

Je tiens à remercier les membres du jury pour l'intérêt accordé à mon travail.

Enfin, Je remercie tous ceux qui auront contribué à ce travail et à mon cursus universitaire.

Dédicaces

Je dédie ce travail à :

Mes très chers parents, ce travail représente le fruit de vos investissements,

À ma sœur, pour le soutien et l'encouragement prodigués,

À mon frère, mes oncles et tantes,

À mes amis,

Et à tout le corps de l'école nationale polytechnique.

Rayana

Table des matières

Table des figures

Abréviations

API	**A**pplication **P**rogramming **I**nterface
SURF	**S**peeded **U**p **R**obust **F**eatures

Introduction générale

De nos jours, l'imagerie numérique devient de plus en plus présente dans les applications courantes de la vie. Elle permet de localiser, de surveiller, de reconnaitre ou encore de diffuser des informations.

Depuis des dizaines d'années, les scientifiques cherchent et proposent des procédés afin d'acquérir les images, de les interpréter, de les modifier ou d'en extraire les informations nécessaires à diverses applications.

L'objectif de ce projet est l'étude de l'algorithme SURF, conçu pour la caractérisation de l'image à travers des points clés et des descripteurs.

Dans le premier chapitre, nous présentons la partie théorique de cet algorithme ainsi que les différents outils mathématiques utilisés.

Le second chapitre présente une application du SURF à la reconnaissance d'objet sur une vidéo.

Chapitre 1

SURF

1.1 Introduction

SURF ou Speeded Up Robust Features [1], qui signifie caractéristiques robustes accélérées. La recherche des correspondances des images discrètes peut être divisée en trois étapes principales.
Tout d'abord, Il faut sélectionner les points d'intérêt, à différents emplacements sur l'image, comme les coins, les taches et les jonctions en T. L'étape est répétée plusieurs fois afin de tester sa répétabilité, c'est-à-dire, si les mêmes points d'intérêt sont détectés sous des conditions de vision différentes. Puis, le voisinage de chaque point d'intérêt est représenté par un vecteur qu'on appellera descripteur. Ce descripteur doit être distinctif et robuste vis-à-vis du bruit, de la détection d'erreurs ainsi qu'aux déformations géométriques ou photométriques.
A la fin, les vecteurs descripteurs sont mis en correspondance entre différentes images. Cette étape est souvent basée sur la distance entre les vecteurs, la distance Euclidienne par exemple. On utilise aussi la distance de Mahalanobis, qui mesure la corrélation et la similarité entre deux variables.
Le temps de calcul est directement lié aux dimensions des descripteurs calculés.
Le but principal de cette méthode, a été le développement d'un détecteur et d'un descripteur en même temps, qui sont plus rapides à calculer en comparaison avec les méthodes développées précédemment. Toutefois, les performances n'ont pas été sacrifiées. D'ailleurs, l'étude comparative (Juan et Al., 2010) démontre la supériorité du descripteur SURF par rapport à SIFT d'un point de vue de ses performances en temps d'exécution et de sa robustesse aux changements d'illumination.

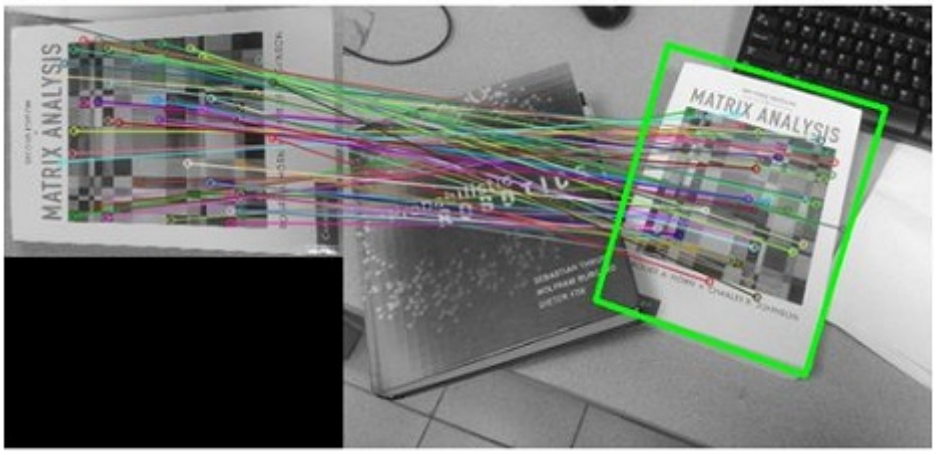

FIGURE 1.1: Mise en correspondance entre deux images, réalisée avec SURF

1.2 Détermination des descripteurs

1.2.1 Détecteur SURF

Le détecteur SURF est basé sur le détecteur de points Fast Hessian, mais en utilisant une approximation très basique. Ce détecteur utilise les images intégrales afin de réduire le temps de calcul, d'où l'appellation 'Fast Hessian Detector '.

1.2.1.1 Image intégrale

Une image intégrale, est un algorithme qui permet de représenter une image sous format numérique, permettant de calculer rapidement des sommes de valeurs dans des zones rectangulaires.
Cette représentation est une image de même taille que l'originale, dont chacun des points contient la somme des pixels situés au-dessus et à gauche de ce point. Cet algorithme est défini par l'expression suivante :

$$ii(x, y) = \sum_{x' \leqslant x, y' \leqslant y} i(x', y') \tag{1.1}$$

Grâce à cette méthode, la somme des valeurs dans une zone rectangulaire peut être calculée avec seulement 4 accès de l'image intégrale (Figure 1.2) ou encore 6 accès pour deux zones rectangulaires contiguës, et donc un temps constant quelle que soit la taille de la zone.

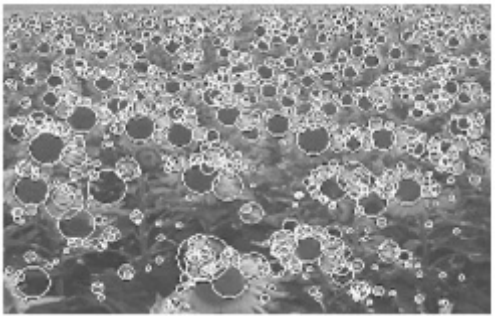

FIGURE 1.2: Le calcul de la somme des pixels dans la zone sombre nécessite uniquement les accès A, B, C et D

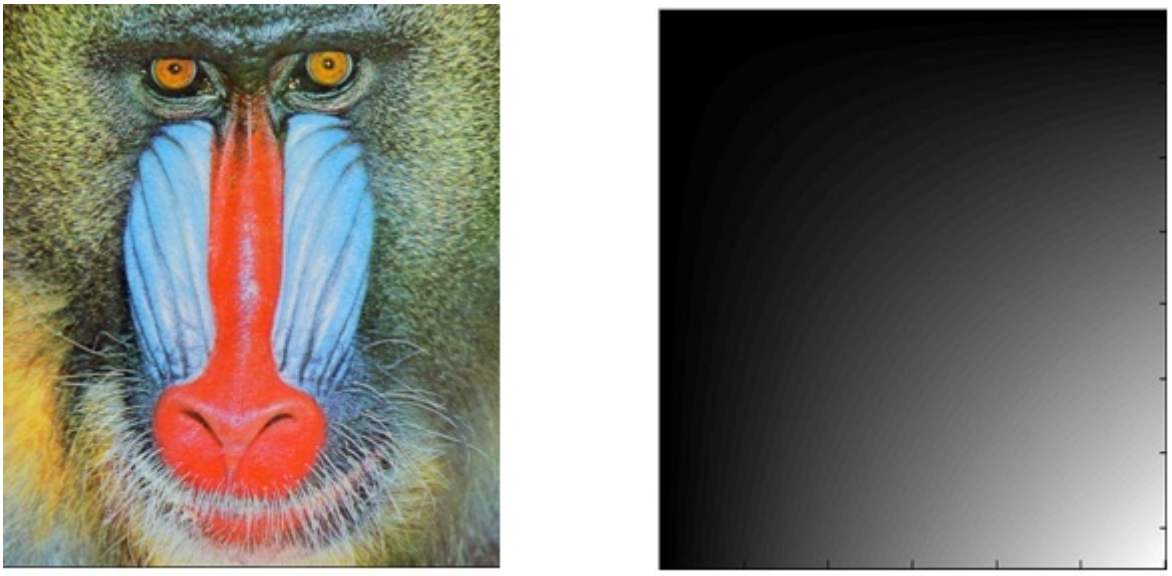

FIGURE 1.3: Une image et son image intégrale

1.2.1.2 Détecteur Fast Hessian

La matrice Hessienne est utilisée dans cette méthode pour ses performances en termes de rapidité et de précision.
Dans le détecteur de 'Hessian-Laplace', il fallait deux opérations distinctes pour la localisation et la détermination de l'échelle. Le détecteur Fast-Hessian se base sur le déterminant de la matrice pour effectuer les deux opérations.
Les zones de fort changement d'intensité des pixels sont recherchées dans l'image.
La matrice Hessienne basée sur le calcul des dérivées partielles d'ordre deux, est utilisée pour ça.

Un point donné $\mathbf{x} = (x, u)$ dans une image I est défini par sa matrice Hessienne $\mathbf{H}(x, \sigma)$ au point $\mathbf{x}$ et à l'echelle σ . La matrice Hessienne est définie comme suit :

$$\mathbf{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{1.2}$$

Où : $L_{xx}(\mathbf{x}, \sigma)$ est la convolution de la dérivée Gaussienne de second ordre 2 $\frac{\partial^2 g(\sigma)}{\partial \mathbf{x}}$ avec l'image I au point $\mathbf{x}$.

Afin de gagner en rapidité, les Gaussiennes sont approximées par des fonctions à paliers appelées box filter. Un box filter est une simple moyenne des pixels d'une région donnée. Si la région est de taille 7x7, le résultat du box filter est la moyenne des intensités de 49 pixels

En effet,convoluer une image plusieurs fois avec un box filter, s'approximerait au traitement avec le filtre de Gauss.

Il a évidemment été prouvé que les performances des box filters sont tout à fait comparables à celles données par les dérivées Gaussienne originales.

Les éléments de la matrice Hessienne ont été renommés de la manière suivante : D_{xx}, D_{xy} et D_{xy}.

Aussi, l'application des box filters ne se fait pas en série, mais en parallèle. C'est-à-dire que nous n'avons pas besoin de la sortie du premier filtre pour lui appliquer le deuxième, mais le traitement se fait sur l'image intégrale.

Le premier masque appliqué a une taille de 9x9 pixels. Il correspond à l'échelle$s = 1.2$ (ce qui correspond à $\sigma = 1.2$ pour le filtre Gaussien). De plus grands masques sont ensuite appliqués : 15x15, 21x21,27x27 etc.

Il existe une certaine cohérence entre le choix des masques et les échelles. Ainsi, un masque de 27x27 correspond à une échelle $s = 3$x1.2.

L'application des différents box filters nous permettra de construire une pyramide.

On a pu aussi établir une nouvelle expression du déterminant de la matrice approximée :

$$Det(H_{approximee}) = D_{xx} * D_{yy} - (0.9D_{xy})^2 \tag{1.3}$$

Si le déterminant de la matrice Hessienne est positif, alors les valeurs propres de la matrice sont toutes les deux positives ou toutes les deux négatives. Ceci signifie qu'un extremum est présent. Les points d'intérêt sont après localisés là où le déterminant est maximal, dans un voisinage de 3x3x3, comme le montre la figure 1.4

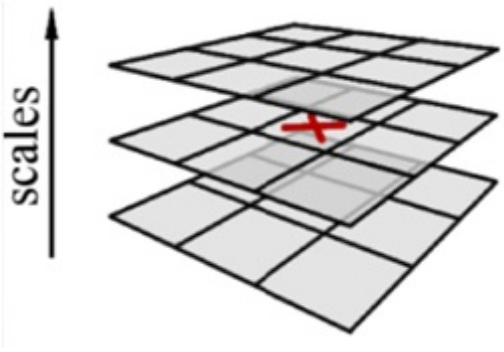

FIGURE 1.4: Localisation des points d'intérêt dans la pyramide

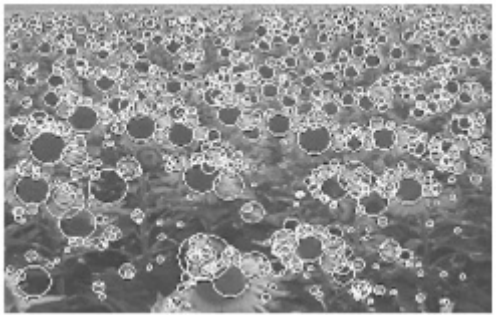

FIGURE 1.5: Exemple de détection des points d'intérêt avec le détecteur Fast Hessian

1.2.2 Descripteurs SURF

Une fois les points d'intérêts extraits, la seconde étape du SURF consiste à calculer les descripteurs correspondants. Cette étape elle-même consiste en deux opérations.

1.2.2.1 Assignation des orientations

Afin d'être invariant par rapport à la rotation, on identifie l'orientation dominante de chaque point d'intérêt. Le descripteur SURF décrit l'intensité des pixels dans un voisinage autour de chaque point d'intérêt. La réponse en x et en y des ondelettes de Haar est calculée dans un voisinage de $6 * \sigma$ où σ est l'échelle à laquelle le point d'intérêt a été trouvé

1.2.2.2 Ondelette de Haar

Dans cet algorithme, on parle d'ondelette de Haar, mais en vérité, il s'agit de l'utilisation des caractéristiques pseudo-Haar. Elles tiennent leur nom de la similarité avec l'ondelette de Haar. La technique qui emploie ces caractéristiques consiste à définir des zones rectangulaires et adjacentes sur l'image. On calcul ensuite la somme des intensités des

pixels de l'image dans ces zones. La différence entre les rectangles noirs et blancs donne la caractéristique pseudo-Haar

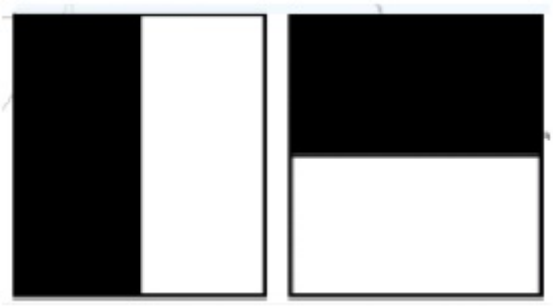

FIGURE 1.6: Ondelettes de Haar pour calculer les réponses en x (à gauche) et les réponses en y (à droite).

Les réponses obtenues en x et en y sont représentées par des vecteurs. A partir des valeurs obtenues, l'orientation dominante de chaque point d'intérêt est calculée de la manière suivante : On fait glisser une fenêtre d'orientation qui recouvre un angle de $\frac{\pi}{3}$, et on calcule la somme de toutes les réponses comprises dans cette fenêtre.
Les deux sommes, résultant de la somme des réponses en **x** et de celle des réponses en **y** , génèrent un nouveau vecteur.
Le vecteur le plus long, représentera l'orientation dominante.

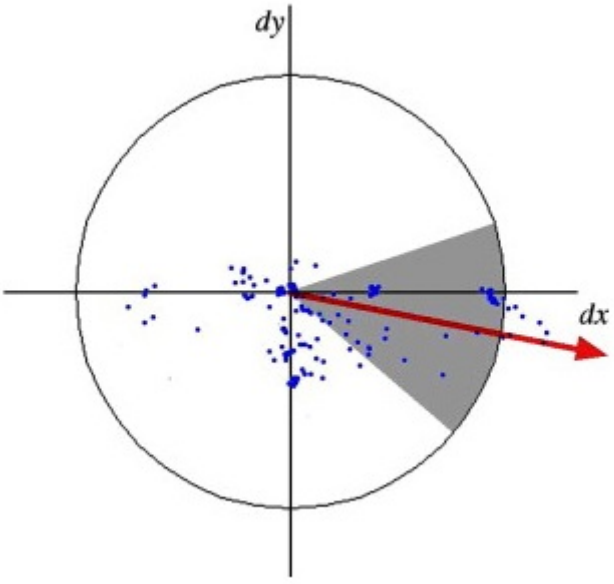

FIGURE 1.7: La fenêtre glissante utilisée pour l'assignation des orientations

1.2.2.3 Composants du descripteur

La première étape de l'extraction des descripteurs, consiste en une construction d'une région carrée autour de chaque point d'intérêt, qui constituera évidemment le centre. Ce carré sera orienté selon l'orientation dominante calculée précédemment. La taille de cette fenêtre est de 20σ. Un exemple est illustré sur la figure 1.8.

FIGURE 1.8: Détails d'un graffiti, présentant les tailles des fenêtres carrées à différentes échelles

Chaque région est subdivisée en $4 * 4$ sous régions. Pour chacune de ces sous régions, les ondelettes de Haar sont calculées sur$5 * 5$ points. On appellera dx la réponse horizontale à l'ondelette de Haar, et dy la réponse verticale. Ces réponses sont pondérées à l'aide d'un filtre Gaussien centré au point d'intérêt.

La somme des réponses verticales et horizontales de chaque sous-région représentera la première partie des caractéristiques du point d'intérêt. Puis, afin d'avoir plus d'informations concernant les changements d'intensité, nous calculerons aussi la sommes des valeurs absolues $|dx|$et $|dy|$.Cette étape est illustrée sur la figure 1.9. Ainsi, chaque sous-région est décrite par 4 caractéristiques. Par conséquent, tout point d'intérêt est représenté par un vecteur de 64 composantes$(4 * 4 * 4)$ La figure 1.10 représente les propriétés de 3 sous-régions, avec des intensités différentes. Ainsi, sur une région homogène comme celle représentée à gauche, toutes les valeurs sont relativement basses. Celle du milieu présente plusieurs fréquences sur $\mathbf{x}$, et donc la valeur $|dx|$ est la seule élevée. Si l'intensité augmente graduellement comme sur la région de droite, dx et $|dx|$ sont toutes les deux élevées.

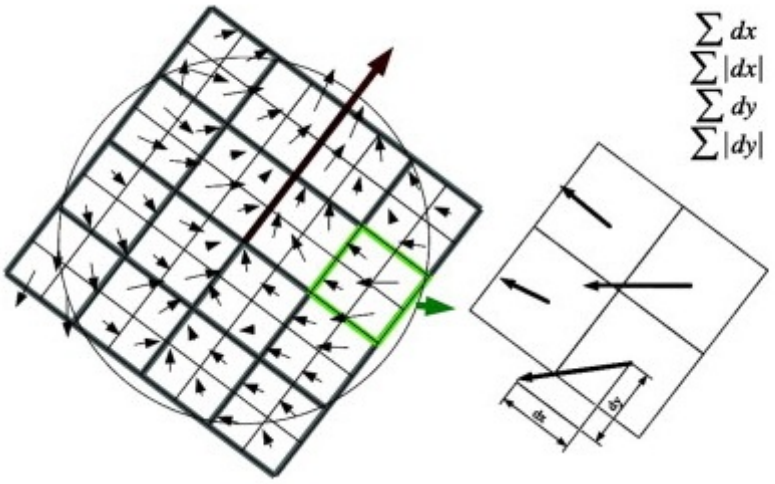

FIGURE 1.9: Calcul des 4 caractéristiques d'une sous-région. Ici, la sous-région est subdivisée en 2x2 pour un but pédagogique

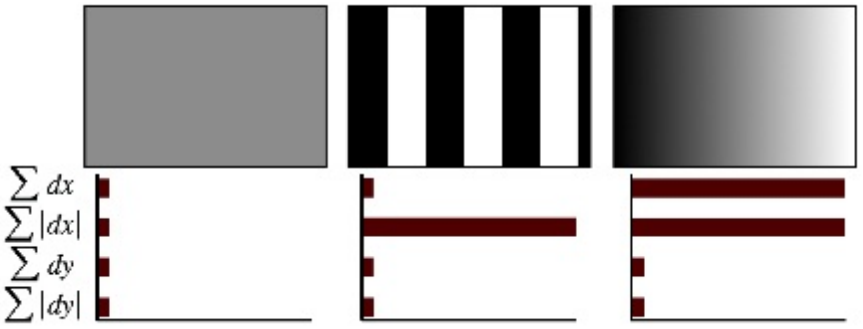

FIGURE 1.10: Comparaison des caractéristiques de 3 sous-régions d'intensités différentes

1.3 Mise en correspondance

Dans cette étape, la mise en correspondance se fait entre les points d'intérêts de deux images, afin d'estimer le degré de similitude entre les deux. A tout point d'intérêt, on associe les deux point d'intérêt les plus proches appartenant à l'image requête. Le plus proche voisin serait suffisant, mais le second sera utilisé pour l'étape de filtrage suivante. Pour trouver les deux PPV (plus proches voisins), on utilise la distance euclidienne entre les deux descripteurs de dimensions 64.

Etant donné qu'une recherche exhaustive des PPV serait longue, une autre méthode a été mise au point. Il s'agit de la méthode des arbres KD, permettant de structurer l'espace de recherche afin d'accélérer la comparaison.

La figure suivante explique la construction d'un arbre KD.

L'espace est récursivement découpé en hyperplans, à la hauteur de la médiane selon x puis selon y.

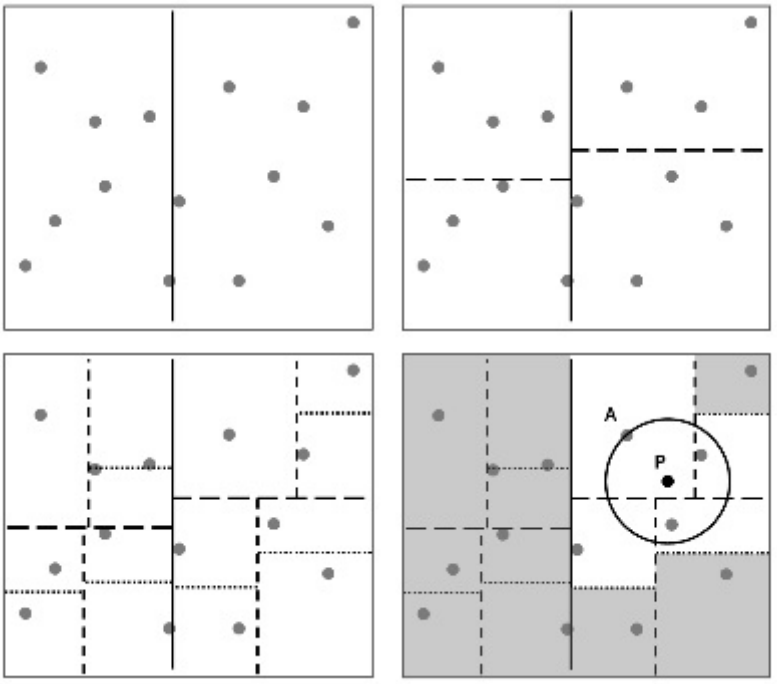

FIGURE 1.11: Construction de l'arbre KD

L'arbre KD est l'arbre binaire correspondant au découpage. Les étapes suivantes explique son utilisation :
La première partie de la recherche consiste à déterminer dans quelle case de l'arbre est le point P dont le PPV est cherché. Le point A présent dans cette case est marqué comme étant l'actuel PPV. Toutes les cases qui sont plus loin que l'hypersphère de centre P et de rayon PA sont éliminées. Il ne reste plus qu'à trouver si un point dans les cases restantes est plus près de P que A.
Puis, on essaie d'éliminer les fausses mises en correspondance, afin de faciliter l'estimation de la transformation. Les étapes suivantes sont effectuées.

- Le premier filtre consiste à supprimer les mises en correspondances dont les deux PPV sont trop proches l'un de l'autre. Cette étape est appelée filtrage par unicité. Il permet d'éliminer les correspondances ambiguës.

- Ensuite, les mises en correspondance sont filtrées en fonction de l'échelle et de l'orientation. Le rapport de l'échelle et la différence d'orientation des mises en correspondance sont calculés. L'espace des angles est découpée par tranche de 20 degrés et l'espace des échelles par facteurs de 1.5. Les correspondances dont l'échelle et la rotation ne correspondent pas au vote de l'échelle et de l'angle majoritaires, sont éliminées.

La figure 1.12 met en évidence l'intérêt de ces filtrages qui permettent d'éliminer facilement un grand nombre de mauvaises mises en correspondance.

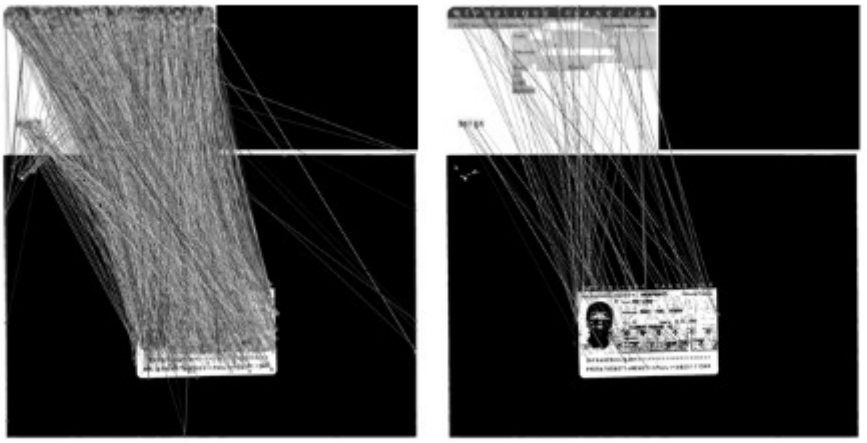

FIGURE 1.12: A gauche 2438 mises en correspondance trouvées. A droite, 78 mises en correspondance retenues

1.4 Conclusion

Le SURF est un algorithme de reconnaissance d'objet. Il adopte une approche locale et se base sur l'extraction des points clés pour décrire l'image. Il s'agit d'une variante du SIFT. Elle est cependant plus adaptée pour les applications en temps réel.
Ce chapitre nous a permis de découvrir cet algorithme, et de nous familiariser avec les différents outils utilisés tels que les ondelettes ou les box filters.

Chapitre 2

Application du SURF pour le tracking d'objets

Introduction

Comme nous l'avons vu dans le chapitre précédent, le SURF est un algorithme basé sur une approche locale du traitement de l'image. Il détecte les points clés porteurs d'informations sur une image et calcule leurs descripteurs.
Les applications de cet algorithme sont nombreuses. Pour en donner quelques exemples, nous pouvons citer la reconstruction 3D. Il s'agit de reconstruire une scène environnant la caméra, à partir d'un certain nombre d'images. Il y a également le concept de la réalité augmentée, qui se définit par la construction d'une scène en 3D à partir d'une image en deux dimensions de cette scène. On peut citer aussi l'identification des objets sur les images ou encore le suivie d'objet sur les vidéos. C'est d'ailleurs le but de l'application développée dans ce chapitre.
La caractéristique principal de ce détecteur est le temps de calcul, d'où le choix de l'application de Tracking.

2.1 Tracking

Le tracking, ou le suivi d'objet est une application temps réelle du traitement d'image [2]. Il est défini par l'étude au fil du temps du déplacement des points d'intérêt, dont l'identification se base sur leurs descripteurs.

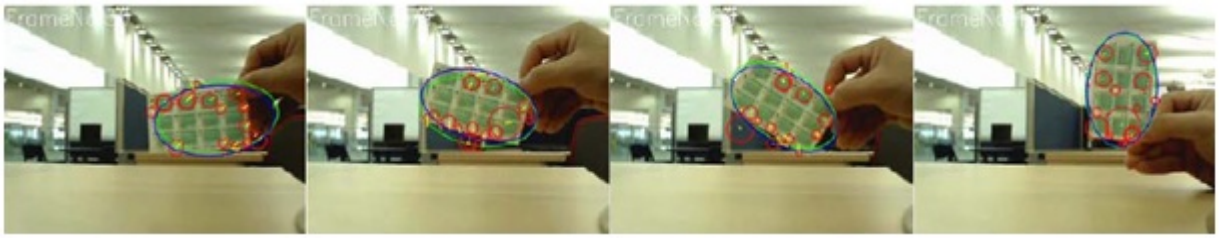

FIGURE 2.1: Illustration du suivi d'un objet en déplacement

2.2 Application du SURF pour le Tracking

Cette application a pour but l'illustration temps réel de l'algorithme SURF, pour le suivi d'objets en déplacement. Le programme a été écrit en C++ sur CodeBlocks. L'implémentation de SURF est disponible sur la librairie OpenCV, qui nécessite une configuration sur CodeBlocks pour permettre son utilisation.

2.2.1 Présentation d'OpenCV

OpenCV (Open Source Computer Vision) est une bibliothèque proposant un ensemble de plus de 2500 algorithmes de vision par ordinateur, accessibles au travers d'API pour les langages C, C++, et Python. Elle est distribuée sous une licence BSD (libre) pour les plate-formes Windows, GNU/Linux, Android et MacOS.
Initialement écrite en C il y a 10 ans par des chercheurs de la société Intel, OpenCV est aujourd'hui développée, maintenue, documentée et utilisée par une communauté de plus de 40 000 membres actifs. C'est la bibliothèque de référence pour la vision par ordinateur, aussi bien dans le monde de la recherche que celui de l'industrie.
Afin de mieux présenter son étendue et ce qu'elle permet de faire, voici les principaux modules accessibles au travers de son API C [3].

- **core :** Les fonctionnalités de base.
  Cette bibliothèque permet de manipuler les structures de base, réaliser des opérations sur des matrices, dessiner sur des images, sauvegarder et charger des données dans des fichiers XML...
- **imgproc :** Traitement d'image.
  Nous entrons dans le cœur du sujet. Les fonctions et structures de ce module ont trait aux transformations d'images, au filtrage, à la détection de contours, de points d'intérêt...

- **features2d :** Descripteurs.
  Ce module concerne principalement l'extraction de descripteurs selon deux approches courantes (SURF et StarDetector), que nous utiliserons d'ailleurs dans l'application.

- **objdetect :** Détection d'objets. Cette bibliothèque permet de faire de la reconnaissance d'objets dans une image au moyen de l'algorithme Adaboost (Viola & Jones, 2001).

- **video :** Traitement de flux vidéo.
  Ces fonctions servent à segmenter et suivre les objets en mouvement dans une vidéo.

- **highgui :** Entrées-sorties et interface utilisateur.
  OpenCV intègre sa propre bibliothèque haut-niveau pour ouvrir, enregistrer et afficher des images et des flux vidéo. Celle-ci contient aussi un certain nombre de fonctions permettant de réaliser de simples interfaces graphiques.

- **calib3d :** Calibration, estimation de pose et stéréovision. Ce module contient des fonctions permettant de reconstruire une scène en 3D à partir d'images acquises avec plusieurs caméras simultanément

2.2.2 Organigramme du programme

L'image de référence choisie pour cette application est la suivante.

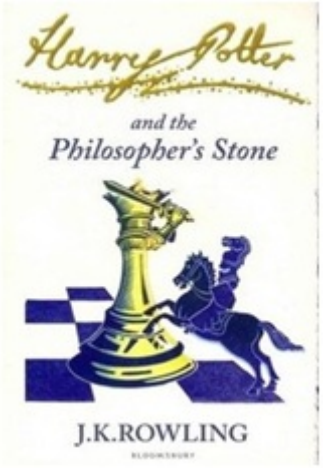

FIGURE 2.2: Image de l'objet suivi

Dans un premier temps, l'image de référence est lue puis traitée avec SURF. Les descripteurs des points clés extraits sont stockés.

Le programme permet ensuite de lancer la webcam, et de capturer ses images. Toutes les captures sont converties en niveau de gris. Puis, les descripteurs SURF sont calculés.
La troisième étape consiste à comparer les descripteurs et à afficher un cercle rouge à sur les points communs entre les deux images.
L'organigramme sur la figure 2.3 permet de mieux expliquer cela et (Les programmes écrits sont en annexe A [4])

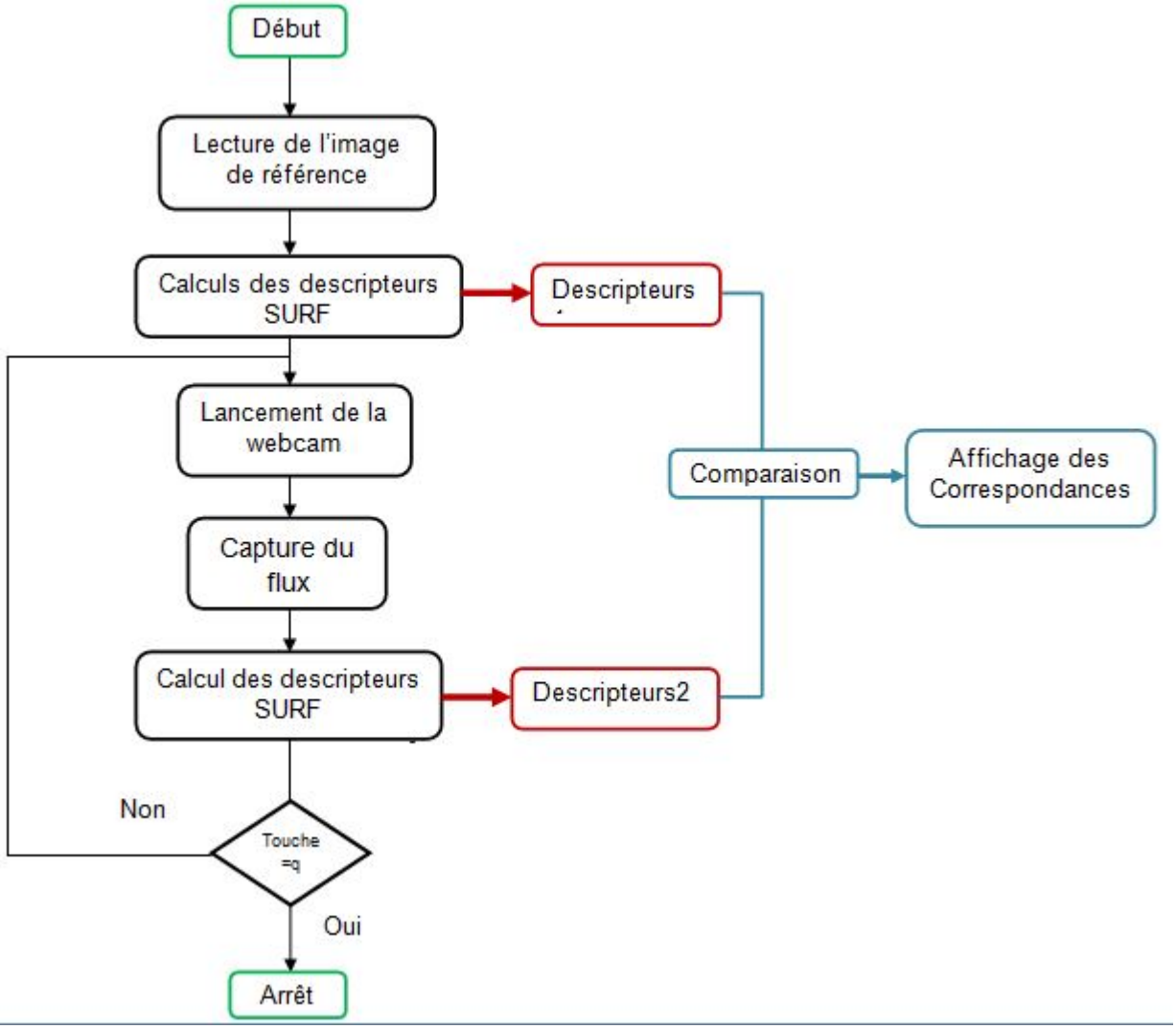

FIGURE 2.3: Organigramme

2.2.3 Résultats

Dans cette partie, nous illustrons l'exécution de l'application décrit précédemment. Les points rouges traduisent les points matchés entre l'image de référence et la capture de la caméra.

FIGURE 2.4: Illustration de la détection d'un objet avec SURF à l'aide de la webcam

Conclusion

A l'aide de l'application développée dans ce projet, nous avons pu mettre en avant les performances du SURF en temps réel.
Nous avons pu voir que l'objet est détecté sous différents points de vue. Cependant, quelques erreurs sont quand même présentes. Elles sont traduites par les points mis en correspondance, mais qui n'appartiennent pas à l'objet.
Aussi, le suivi ne fournit pas une bonne reconnaissance lorsque l'objet est en déplacement rapide. Cela peut être dû à la qualité de la caméra utilisée.
De plus, nous avons pu tester les performances de la librairie OpenCV, la référence des librairies dans le domaine du traitement d'image.

Conclusion générale

La vision par ordinateur est un domaine phare de l'intelligence artificielle. Ses applications sont devenues incontournables. Elles offrent aux ordinateurs et aux machines la capacité de voir l'environnement, d'en capturer des images, de les interpréter, d'en soutirer des informations importantes et de les exploiter.

Ce document a pour objectif la présentation d'un algorithme de traitement d'image, conçu pour la reconnaissance d'objet et le suivi. Il s'agit de l'algorithme SURF. Il suit une approche locale, et il se base sur les points d'intérêt et le calcul de leurs descripteurs pour caractériser l'image.

Il a aussi été possible de développer une application pour l'exploitation du SURF dans le Tracking. Il s'agit là de calculer les descripteurs SURF des captures de la webcam et de les comparer à l'image d'un objet enregistrée sur l'ordinateur. Lorsque l'objet apparait sur le champ de vision de la webcam, il est détecté.

Ce projet, à travers cette application, met en avant les performances du SURF en temps réel. Il permet aussi d'avoir des perspectives quant aux applications plus poussées de cet algorithme.

Annexe A

Programme OpenCV de l'application

```
//*****************************************************************************
//**********************Soutenance de Master en Electronique*******************************
//****************************Tracking SURF Application*************************************
//**********Capture le flux de la Webcam et le compare à l'image de référence***********
//*********************************Nom: BOUBEZARI*****************************************
//*********************************Prénom: Rayana *****************************************
//*****************************email: r.boubezari@gmail.com*********************************
//*****************************************************************************
#include <math.h>
#include <stdio.h>
#include <vector>
#include <ctype.h>
#include <stdlib.h>


#include "opencv/cv.h"
#include "opencv/highgui.h"
#include "opencv/cvaux.h"
#include "opencv/cxcore.h"

#include "opencv2/features2d/features2d.hpp"
#include "opencv2/core/core_c.h"
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/video/tracking.hpp"
#include "opencv2/flann/flann.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/legacy/compat.hpp"

using namespace std;
using namespace cv;
////////////////////////////////////////////////////////////////////////////
//Fonction comparatrice des descripteurs

double
compareSURFDescriptors( const float* d1, const float* d2, double best, int length )
```

```
{
    double total_cost = 0;
    assert( length % 4 == 0 );
    for( int i = 0; i < length; i += 4 )
    {
        double t0 = d1[i] - d2[i];
        double t1 = d1[i+1] - d2[i+1];
        double t2 = d1[i+2] - d2[i+2];
        double t3 = d1[i+3] - d2[i+3];
        total_cost += t0*t0 + t1*t1 + t2*t2 + t3*t3;
        if( total_cost > best )
            break;
    }
    return total_cost;
}
////////////////////////////////////////////////////////////////
//Nearest Neighbor
int
naiveNearestNeighbor( const float* vec, int laplacian,
                      const CvSeq* model_keypoints,
                      const CvSeq* model_descriptors )
{
    int length = (int)(model_descriptors->elem_size/sizeof(float));
    int i, neighbor = -1;
    double d, dist1 = 1e6, dist2 = 1e6;
    CvSeqReader reader, kreader;
    cvStartReadSeq( model_keypoints, &kreader, 0 );
    cvStartReadSeq( model_descriptors, &reader, 0 );

    for( i = 0; i < model_descriptors->total; i++ )
    {
        const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
        const float* mvec = (const float*)reader.ptr;
        CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
        CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
        if( laplacian != kp->laplacian )
            continue;
        d = compareSURFDescriptors( vec, mvec, dist2, length );
        if( d < dist1 )
        {
            dist2 = dist1;
            dist1 = d;
            neighbor = i;
        }
        else if ( d < dist2 )
            dist2 = d;
    }
    if ( dist1 < 0.6*dist2 )
        return neighbor;
    return -1;
}
////////////////////////////////////////////////////////////////////
// Points d'inérêt communs à deux images
void
findPairs( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
           const CvSeq* imageKeypoints,
```

```
            const CvSeq* imageDescriptors, vector<int>& ptpairs )
{
    int i;
    CvSeqReader reader, kreader;
    cvStartReadSeq( objectKeypoints, &kreader );
    cvStartReadSeq( objectDescriptors, &reader );
    ptpairs.clear();

    for( i = 0; i < objectDescriptors->total; i++ )
    {
        const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
        const float* descriptor = (const float*)reader.ptr;
        CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
        CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
        int nearest_neighbor = naiveNearestNeighbor( descriptor,
                            kp->laplacian, imageKeypoints, imageDescriptors );
        if( nearest_neighbor >= 0 )
        {
            ptpairs.push_back(i);
            ptpairs.push_back(nearest_neighbor);
        }
    }
}


////////////////////////////////////////////////////////////////
////// Fonction Pirtncipale
int main(int argc, char** argv)
{
////////////////////////////////////////////////////////////////////
/////// Lecture de l'image de référence et calcul de ses descripteurs
IplImage* test = cvLoadImage( "test.jpg" );
IplImage* graytest =cvCreateImage(cvGetSize(test),8,1);
cvCvtColor(test,graytest,CV_RGB2GRAY);
CvSURFParams params = cvSURFParams(500,1);
CvMemStorage *storage1 = cvCreateMemStorage(0);
CvSeq *descriptors1=0,*keypoints1=0;
cvExtractSURF( graytest, 0, &keypoints1, &descriptors1, storage1, params );
//////////////////////////////////////////////////////////////
//Affichage et traitement des captures de la webcam
char fermeture;
CvMemStorage* storage = cvCreateMemStorage(0);
CvCapture *capture;
cvNamedWindow("Recognition", CV_WINDOW_AUTOSIZE);
capture = cvCreateCameraCapture( CV_CAP_ANY );
IplImage *image=0;
while(fermeture !='q' && fermeture !='Q' ){
        image=cvQueryFrame(capture);
        IplImage* grayscale =cvCreateImage(cvGetSize(image),8,1);
        cvCvtColor(image,grayscale,CV_RGB2GRAY);
        CvSURFParams params = cvSURFParams(500, 1);
        CvSeq *descriptors=0,*keypoints=0;
        cvExtractSURF( grayscale,0, &keypoints, &descriptors, storage, params );

//////////////////////////////////////
// Matching des deux images
```

```
        vector<int> ptpairs;


        findPairs( keypoints, descriptors, keypoints1, descriptors1, ptpairs );
        int n = ptpairs.size()/2;
        printf("%d\n", n);


////////////////////////////////////////////////////////////////
//Encercler les points matchés
        vector<CvPoint2D32f> pt1;
        pt1.resize(n);

                for( int h = 0 ; h < n ; h++ )
                        {

                                CvPoint center;
                                pt1[h] = ( (CvSURFPoint*) cvGetSeqElem(keypoints, ptpairs[h*2]) )->pt;
                                center.x = cvRound(pt1[h].x);
                                center.y = cvRound(pt1[h].y);
                                cvCircle( image,center, 2, cvScalar(0, 0, 255, 0), 5, 1, 0 );

                        }


////////////////////////////////////////////////////
//Affichage des captures
            cvShowImage("Recognition",image);
            fermeture=cvWaitKey(5);


            }
return 0;



}
```

Bibliographie

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3) :346–359, June 2008. ISSN 1077-3142. doi : 10.1016/j.cviu.2007.09.014. URL http://dx.doi.org/10.1016/j.cviu.2007.09.014.

[2] Manuel Grand-brochier. *Descripteurs 2D et 2D+t de points d'intérêt pour des appariements robustes*. PhD thesis, Université Blaise Pascal - Clermont II, 2011.

[3] OpenCV Documentation. *The OpenCV reference Manual, release 2.4.5.0*. OpenCV, 5 avril 2013.

[4] OpenCV Documentation. Opencv tutorials. http ://-docs.opencv.org/doc/tutorials/tutorials.html, Avril 2013.