PROJET BIBLIOGRAPHIQUE

COMPUTER VISION

Omar MHAIMDAT

Supervisé par Mr BOUTABIA Mohamed

2 ème année Génie Informatique 2018-2019

UNIVERSITÉ INTERNATIONALE DE CASABLANCA

Table des matières

Table des figures

Introduction au projet bibliographique

Tout au long du semestre, j'ai eu la chance de mener un projet bibliographique sur le domaine du « Computer Vision ». Ce projet m'a permis d'acquérir de nombreuses compétences essentielles telles que l'organisation, l'auto-éducation et, surtout, j'ai dû développer un esprit d'analyse en filtrant l'énorme quantité d'informations afin de mieux structurer les idées présentées dans le livre.

La raison qui m'a poussée à choisir le sujet « Computer Vision », c'est que tout d'abord le domaine est en plein changement et surtout en développement rapide avec tous les six mois une avancé majeur. Il s'agit de mon point de vue d'une technologie largement utilisée (Snapchat, Instagram, voitures autonomes …) qui présente de nombreux avantages.

Le présent rapport a pour objectif d'expliquer correctement les piliers du « Computer Vision » avec un soucis d'être le plus clair possible pour tous les lecteurs, même ceux qui n'ont aucune connaissance préalable de certaines notions mathématiques ou computationnelles. Certains éléments complexes sont ajoutés progressivement pour satisfaire la curiosité des lecteurs les plus avancés.

À travers ces pages, j'essayerai de donner un aperçu du domaine de « Computer Vision » avec quelques définitions et des modèles clairs.

1. Présentation du livre

1.1. Le livre

« Computer Vision Algorithms and Applications » est un livre rédigé par Mr Richard Szeliski et publié en 2011. C'est un livre assez exhaustif et qui traite de toutes les parties fondamentales et historique du domaine de la vision par ordinateur.

Il est à noter que ce livre ne traite pas des dernières évolutions dans ce domaine c'est-à-dire les techniques utilisant les Neural Networks, Deep Learning ou le Machine Learning.

1.2. L'auteur

Richard Szeliski a rejoint Facebook à l'automne 2015 en tant que membre fondateur de l'équipe de « Computer Vision », dont la mission est d'offrir et de contenter les utilisateurs grâce à des expériences photo et vidéo innovantes basées sur des technologies de vision et de graphisme de pointe.

Avant d'occuper son poste actuel, Richard a travaillé pendant vingt ans chez Microsoft Research, où il dirigeait le groupe Interactive Visual Media. Il a également travaillé pendant six ans au Cambridge Research Lab de Digital Equipment Corporation, ainsi que pour plusieurs autres organisations de recherche industrielle.

Richard Szeliski est un professeur affilié à l'Université de Washington, où il encadre des doctorants et parfois enseigne des cours. Mr Szeliski est également l'auteur de l'un des manuels les plus populaires sur le domaine du « Computer Vision ». Ses affiliations professionnelles incluent National Academy of Engineering, il est aussi membre de l'ACM et de l'IEEE.

2. Introduction

2.1. C'est quoi le « Computer Vision » ?

En tant qu'êtres humains, nous percevons la structure tridimensionnelle du monde qui nous entoure avec une facilité apparente. Le mieux serait de prendre un exemple, lorsque vous regardez un vase de fleurs assis sur la table à côté de vous (Figure 1)**Error! Reference source not found.**.

Figure 1: Perception d'un modèle tridimensionnel

Vous pouvez distinguer la forme et la translucidité de chaque pétale à travers les subtils motifs de lumière et d'ombrages qui se déplacent sur toute la surface et segmentent sans effort chaque fleur à partir du fond de la scène. En regardant un portrait de groupe encadré, vous pouvez facilement compter et nommer toutes les personnes sur l'image et même deviner leurs émotions par leur apparence au visage. Les psychologues ont passé des décennies à essayer de comprendre le fonctionnement du système visuel et, même s'ils peuvent créer des illusions d'optique pour expliquer certains de ses principes (Figure 2), une solution complète à ce problème reste insaisissable.

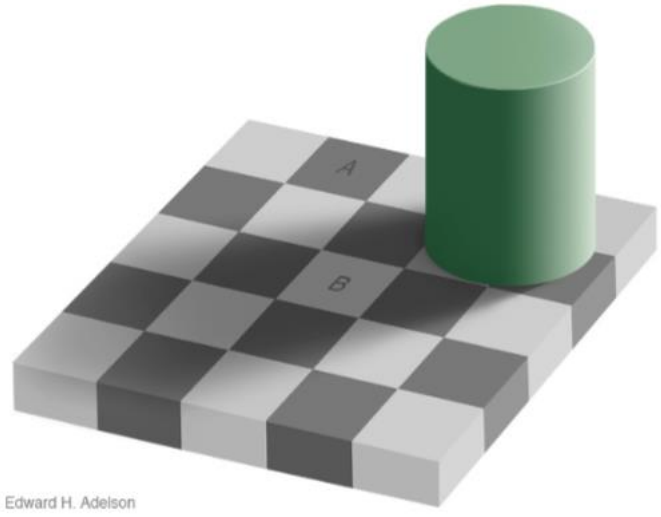

Figure 2: Le carré «blanc» B dans l'ombre et le carré «noir» A dans la lumière ont en réalité la même valeur d'intensité absolue. La perception est dû à la constance de la luminosité, la tentative du système visuel de réduire l'éclairage lors de l'interprétation des couleurs.

Des chercheurs en vision par ordinateur ont mis au point, parallèlement, des techniques mathématiques permettant de retrouver la forme et l'apparence tridimensionnelles d'objets dans

l'imagerie. Nous disposons maintenant de techniques fiables pour calculer avec précision un modèle 3D partiel d'un environnement à partir de milliers de photographies se chevauchant partiellement. Avec un ensemble suffisamment grand de vues d'un objet ou d'une façade spécifique, nous pouvons créer des modèles de surface 3D dense et précise à l'aide de la correspondance stéréo. Nous pouvons suivre une personne qui se déplace sur un arrière-plan complexe. Nous pouvons même, avec un succès modéré, essayer de trouver et nommer toutes les personnes sur une photographie en utilisant une combinaison de détection et de reconnaissance du visage, des vêtements et des cheveux (Figure 3).

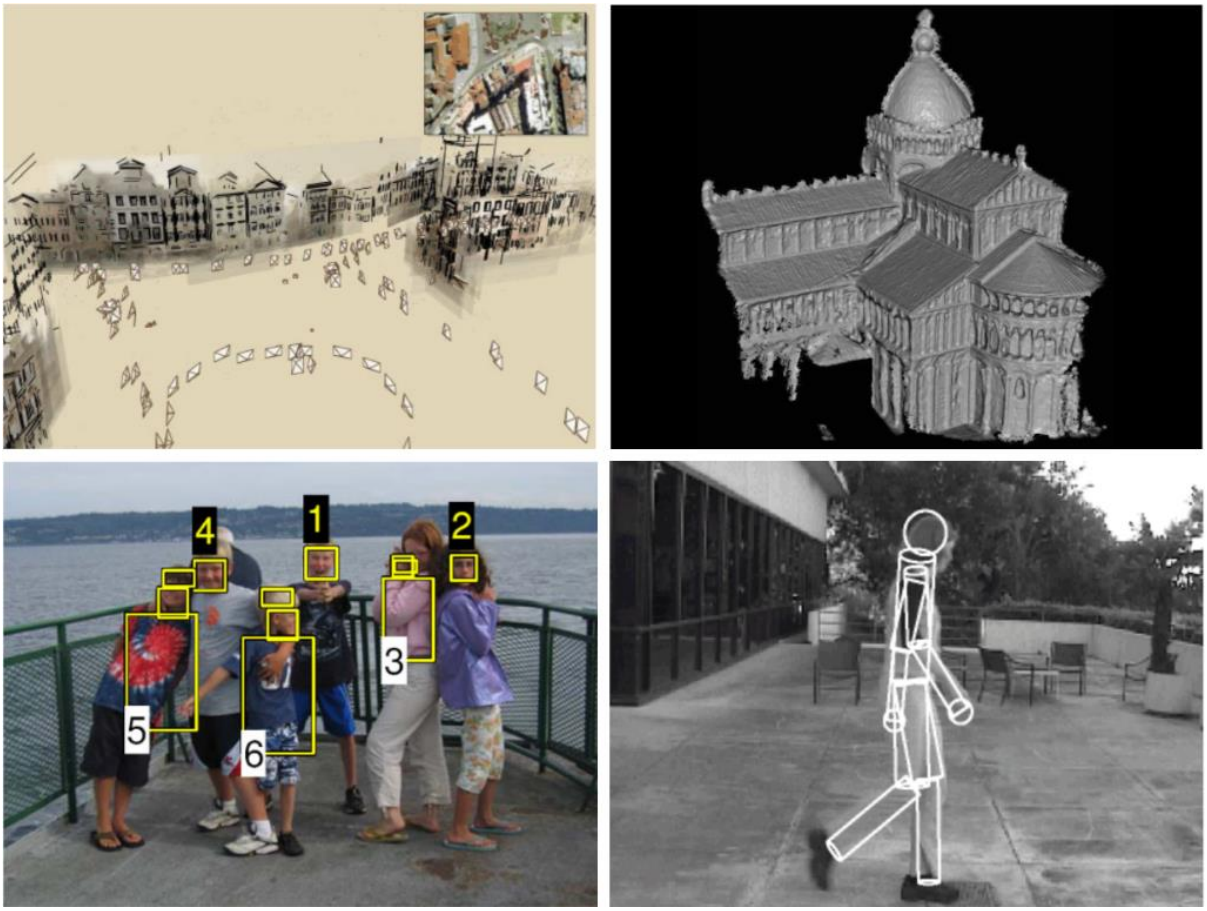

Figure 3: Quelques exemples d'application de la vision par ordinateur

Cependant, malgré toutes ces avancées, le rêve d'avoir un ordinateur qui interprète une image au même niveau qu'un enfant de deux ans (par exemple, en comptant tous les animaux d'une image) reste insaisissable. Pourquoi la vision est-elle si difficile? C'est en partie parce que la vision est un problème inverse, dans lequel nous cherchons à récupérer des inconnues à partir d'informations insuffisantes pour spécifier pleinement la solution. Nous devons donc recourir à des modèles probabilistes et basés sur la physique pour distinguer les solutions potentielles. Cependant, modéliser le monde visuel dans toute sa complexité est beaucoup plus difficile que, par exemple, modéliser le canal vocal qui produit les sons parlés.

Les modèles avancés que nous utilisons en vision par ordinateur sont généralement développés en physique (radiométrie, optique et conception de capteurs) et en infographie. Ces deux champs modélisent la manière dont les objets bougent et s'animent, comment la lumière se reflète sur leurs surfaces, est dispersée par l'atmosphère, réfractée par des objectifs d'appareil photo (ou des yeux humains) et finalement projetée sur un plan d'image plat (ou courbe). Bien que les images de synthèse ne soient pas encore parfaites (aucun film entièrement animé avec des personnages humains n'a encore réussi à traverser le « uncanny valley » qui sépare les humains réels des robots androïdes et des humains animés par ordinateurs), dans des domaines limités, tels que le rendu d'une scène immobile. des objets du quotidien ou des créatures disparues animées telles que les dinosaures, l'illusion de la réalité est parfaite.

En vision par ordinateur, nous essayons de faire l'inverse, c'est-à-dire de décrire le monde que nous voyons dans une ou plusieurs images et de reconstruire ses propriétés, telles que la forme, l'éclairage et la distribution des couleurs. Il est étonnant que les humains et les animaux le fassent sans effort, alors que les algorithmes de vision par ordinateur sont tellement sujets aux erreurs.

2.2. Exemples d'utilisation

- **Optical Character Recognition (OCR) :** lecture de codes postaux manuscrits sur des lettres et reconnaissance automatique de plaques d'immatriculation.
- **Inspection des machines :** inspection rapide des pièces pour l'assurance qualité utilisant la vision stéréoscopique avec éclairage spécialisé pour mesurer les tolérances sur les ailes d'aéronefs ou les pièces de carrosserie ou la recherche de défauts dans les pièces moulées en acier utilisant la vision par rayons X.
- **Imagerie médicale :** enregistrer des images préopératoires et peropératoires ou effectuer des études à long terme sur la morphologie du cerveau des personnes à mesure qu'elles vieillissent.
- **Surveillance :** surveillance des intrus, analyse du trafic routier et surveillance des piscines pour les victimes de noyade.
- **Reconnaissance d'empreintes digitales :** pour l'authentification d'accès automatique ainsi que des applications légales.
- **Voiture autonome :** traitement des images en temps réel émanant de plusieurs capteurs tels que les lidars, radar et les caméras.

2.3. Les sujets abordés dans ce rapport

Lors de la lecture du livre, on se rend compte très vite de la complexité et surtout le nombre de disciplines inclus dans les études qui se rapporte de prêt ou de loin au « Computer Vision ».

Il existe des relations entre les images, la géométrie et la photométrie, ainsi qu'une quantités importante de sujets abordés dans ce livre.

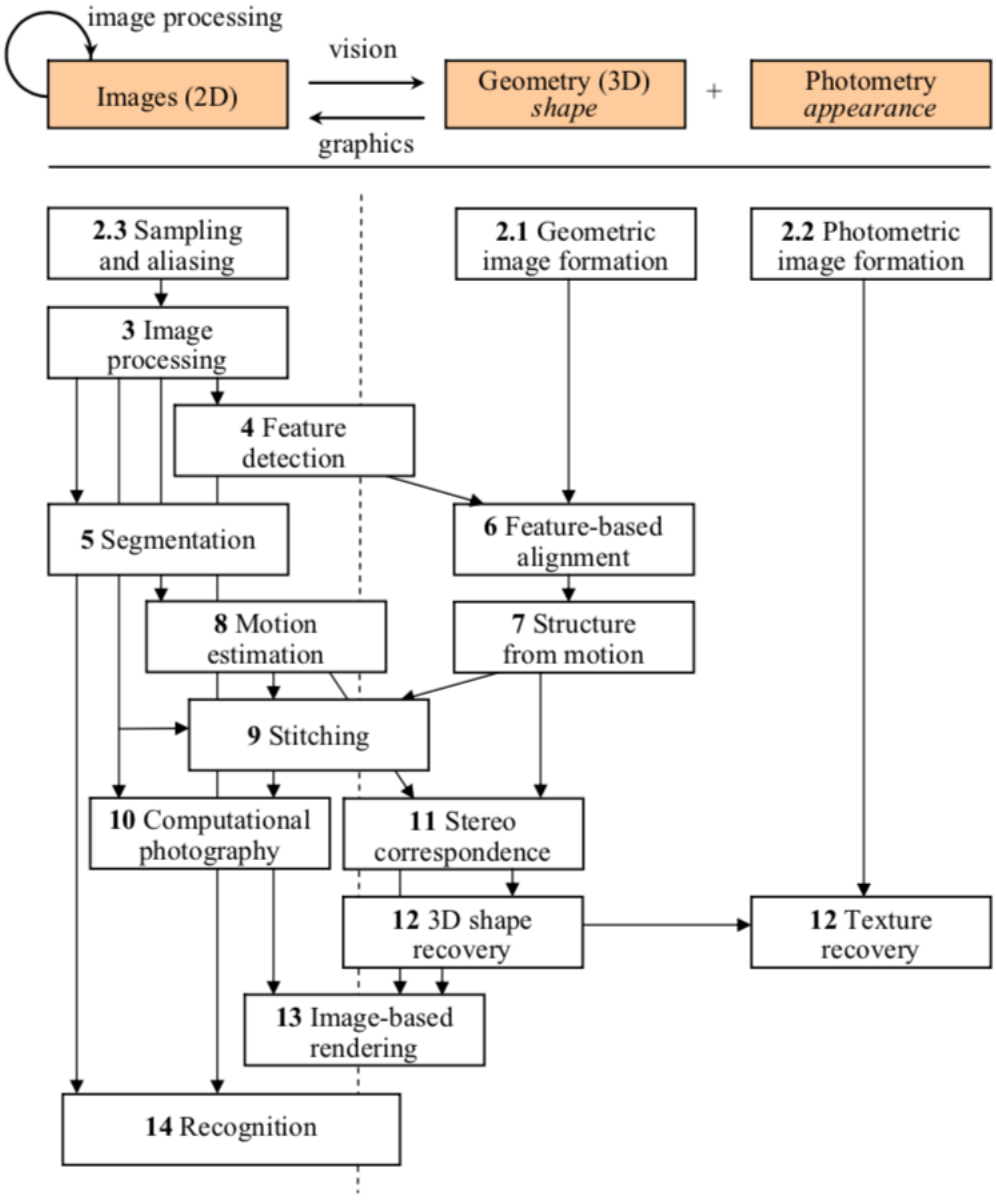

Figure 4: les chapitres abordés dans le livre

Les sujets (**Error! Reference source not found.**) sont grossièrement positionnés le long de l'axe gauche-droite, selon qu'ils sont plus étroitement liés aux représentations basées sur une image (gauche), sur la géométrie (moyen) ou sur l'apparence (droite), et plus on avance sur l'axe vertical on augmente le niveau d'abstraction. La figure n'est pas exhaustive, car il existe de nombreux autres liens subtils entre des sujets non illustrés ici.

Nous allons nous concentrer plus particulièrement sur les chapitres 1, 2, 3, 4 et 14.

3. Formation d'une image

Afin de pouvoir manipuler les images en tant qu'entité immatérielle, il est primordiale de comprendre certaines parties de la formation d'une image. Deux parties essentielles composent une image ; l'intensité de la lumière et les couleurs.

3.1. Formation d'images photométriques

Contrairement à ce qu'on croit les images ne sont pas composées d'entités 2D. Au lieu de cela, elles sont composées de valeurs discrètes de couleur ou d'intensité. D'où viennent ces valeurs? Quel est leur lien avec l'éclairage de l'environnement, les propriétés et la surface de la surface, l'optique de la caméra et les propriétés du capteur? Dans cette section, nous développons un ensemble de modèles pour décrire ces interactions et formuler un processus génératif de formation d'images.

Les images ne peuvent exister sans lumière. Pour produire une image, la scène doit être éclairée avec une ou plusieurs sources de lumière. Les sources de lumière peuvent généralement être divisées en sources de lumière ponctuelles et surfaciques.

Une source de lumière ponctuelle prend naissance à un seul endroit dans l'espace (par exemple, une ampoule électrique), potentiellement à l'infini (par exemple, le soleil). En plus de son emplacement, une source de lumière ponctuelle a une intensité et un spectre de couleurs, c'est-à-dire une distribution sur les longueurs d'onde L (λ). L'intensité d'une source de lumière diminue avec le carré de la distance entre la source et l'objet éclairé, car la même lumière est répartie sur une zone plus grande (sphérique). Une source de lumière peut également avoir une atténuation directionnelle (dépendance), mais nous l'ignorons dans notre modèle simplifié.

Les sources de lumière sont plus compliquées. Une source lumineuse à surface simple, telle qu'un plafonnier fluorescent muni d'un diffuseur, peut être modélisée comme une zone rectangulaire finie émettant de la lumière de la même manière dans toutes les directions Lorsque la distribution est fortement directionnelle, un champ lumineux à quatre dimensions peut être utilisé à la place.

Une distribution de la lumière plus complexe qui se rapproche, par exemple, de l'éclairage incident sur un objet assis dans une cour extérieure peut souvent être représentée à l'aide d'une carte d'environnement. Cette représentation cartographie les directions de la lumière incidente v sur les valeurs de couleur (ou longueurs d'onde, λ),

3.2. Les couleurs

L'espace colorimétrique RGB (rouge, vert et bleu) est l'un des espaces colorimétriques les plus utilisés, spécialement pour les images numériques 8 bits. Ce modèle est généralement

utilisé pour représenter des couleurs dans des appareils électroniques tels que des moniteurs de télévision et d'ordinateurs, des scanners et des appareils photo numériques. La théorie de la vision des couleurs trichromatique de Young – Helmholtz et du triangle de Maxwell est à la base du modèle RGB.

Le RGB est un modèle additif dans lequel les couleurs rouge, vert et bleu sont combinées sur différentes quantités ou portions pour reproduire d'autres couleurs. Les pixels d'une image représentée dans le modèle RGB ont généralement une profondeur de 8 bits, ce qui donne 256 intensités possibles, c'est-à-dire la plage de [0, 255] pour chaque couleur.

Une couleur dans le modèle RGB peut être décrite, indiquant la quantité de rouge, de vert et de bleu. Chaque couleur peut varier entre la valeur minimale (totalement sombre) et la valeur maximale (totalement intense). Lorsque toutes les couleurs ont la valeur minimale, la couleur résultante est le noir. Au contraire, lorsque toutes les couleurs ont la valeur maximale, la couleur résultante est le blanc.

Ce modèle est appelé cube de couleur RGB, car il repose sur le système de coordonnées cartésien et son sous-espace de couleur intéressant est un cube. Les couleurs primaires et secondaires sont aux coins du cube. La couleur noire est à l'origine et la couleur blanche est à son coin opposé. La diagonale entre le noir et le blanc est l'échelle de gris.

3.3. Caméra digitale

Après avoir démarré à partir d'une ou de plusieurs sources de lumière, réfléchi par une ou plusieurs surfaces dans le monde et passé à travers l'optique de la caméra (objectifs), la lumière parvient enfin au capteur d'imagerie. Comment les photons arrivant à ce capteur sont-ils convertis en valeurs numériques (R, G, B) que nous observons lorsque nous regardons une image numérique? Dans cette partie nous développons un modèle simple qui prend en compte les effets les plus importants tels que l'exposition (gain et vitesse d'obturation), les mappages non linéaires, l'échantillonnage et le repliement du spectre, et le bruit. La (Figure 5), basée sur des modèles de caméras montre une version simple des étapes de traitement des caméras numériques modernes. Chakrabarti, Scharstein et Zickler (2009) ont mis au point un modèle sophistiqué à 24 paramètres qui correspond encore mieux au traitement des caméras d'aujourd'hui.

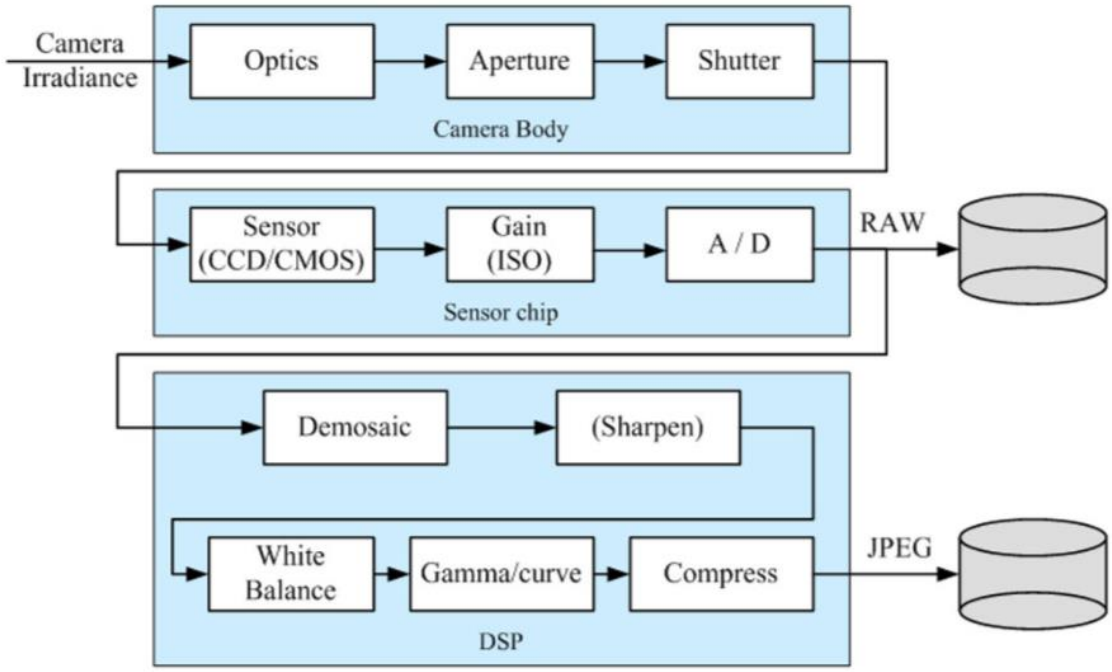

Figure 5: Les étapes de détection d'images montrant les différentes sources de bruit ainsi que les étapes typiques du post-traitement numérique.

La lumière qui tombe sur un capteur d'imagerie est généralement captée par une zone de détection active, intégrée pendant la durée de l'exposition (généralement exprimée en vitesse d'obturation en une fraction de seconde), puis passé à un ensemble d'amplificateurs de détection. Les deux principaux types de capteurs utilisés dans les appareils photo numériques et les caméras vidéo actuels sont des dispositifs à couplage de charge (CCD[1]) et des oxydes de métaux complémentaires sur silicium (CMOS[2]). Dans un CCD, les photons s'accumulent dans chaque puits actif pendant le temps d'exposition. Puis, dans une phase de transfert, les charges sont transférées de puits en puits dans une sorte de "bucket brigade" jusqu'à ce qu'ils soient déposés au niveau des amplificateurs de lecture, qui amplifient le signal et le transmet à un convertisseur analogique-numérique (ADC). Les capteurs CCD plus anciens étaient sujets à la surexposition, lorsque les charges d'un pixel surexposé se déversaient dans les pixels adjacents, mais la plupart des CCD récents ont une technologie anti-efflorescence (des «creux» dans lesquels l'excès de charge peut se répandre).

Avec la technologie CMOS, les photons frappant le capteur affectent directement la conductivité (ou le gain) d'un photo détecteur, qui peut être sélectivement réglé pour contrôler la durée d'exposition, et explicité localement avant d'être lu à l'aide d'un schéma de

[1] CCD : Charged-Coupled Device
[2] CMOS : Complementary Metal Oxide on Silicon

multiplexage. Traditionnellement, les capteurs CCD surpassaient les performances du CMOS dans les applications sensibles à la qualité, telles que les reflex numériques, alors que le CMOS était préférable pour les applications à faible consommation, mais aujourd'hui, le CMOS est utilisé dans la plupart des appareils photo numériques.

Les principaux facteurs affectant les performances d'un capteur d'image numérique sont la vitesse d'obturation, la hauteur d'échantillonnage, le facteur de remplissage, la taille de la puce, le gain analogique, le bruit du capteur et la résolution et la qualité du convertisseur analogique-numérique. La plupart des valeurs réelles de ces paramètres peuvent être lues à partir des balises EXIF[3] (une spécification de format de fichier pour les images utilisées par les appareils photographiques numériques) intégrées aux images numériques. d'autres peuvent être obtenus à partir des feuilles de spécifications du fabricant de l'appareil photo ou des sites Web de révision ou d'étalonnage de l'appareil.

4. Image processing

Les types les plus simples de transformation de traitement d'image sont les opérateurs de point, où la valeur de chaque pixel de sortie dépend uniquement de la valeur de pixel d'entrée correspondante (plus éventuellement de certaines informations ou paramètres collectés de manière globale). Des exemples de tels opérateurs incluent les réglages de luminosité et de contraste, ainsi que la correction des couleurs et les transformations.

4.1. Color transforms

Bien que les images couleur puissent être traitées comme des fonctions vectorielles arbitraires ou des ensembles de bandes indépendantes, il est généralement logique de les considérer comme des signaux hautement corrélés, fortement liés au processus de formation d'images, à la conception de capteurs, et la perception humaine (subjective). On peut envisager, par exemple, d'éclaircir une image en ajoutant une valeur constante aux trois canaux. En fait, l'ajout de la même valeur à chaque canal de couleur augmente non seulement l'intensité apparente de chaque pixel, mais peut aussi affecter la teinte et la saturation du pixel.

Les coordonnées de chromaticité ou même des rapports de couleurs plus simples peuvent tout d'abord être calculés, puis utilisés après avoir manipulé la luminance Y pour recalculer une image RGB valide avec même teinte et saturation.

De la même manière, l'équilibrage des couleurs (par exemple, pour compenser l'éclairage incandescent) peut être réalisé en multipliant chaque canal avec un facteur d'échelle différent ou par le processus plus complexe de mappage vers l'espace colorimétrique XYZ, en modifiant

[3] EXIF : Exchangeable Image File Format

le point blanc nominal et mappage en RGB, qui peut être écrit en utilisant une matrice de transformation par torsion couleur 3×3 linéaire.

4.2. Histogramme

L'histogramme d'une image fait normalement référence à un histogramme des valeurs d'intensité de pixel. Cet histogramme est un graphique indiquant le nombre de pixels d'une image pour chaque valeur d'intensité différente trouvée dans cette image (Une distribution).

Pour une image en niveaux de gris à 8 bits, il existe 256 intensités différentes possibles. L'histogramme affichera donc graphiquement 256 nombres illustrant la répartition des pixels entre ces valeurs en niveaux de gris. Les histogrammes peuvent également être créés avec des images en couleur: des histogrammes individuels des canaux rouge, vert et bleu peuvent être créés, ou un histogramme 3D peut être créé, les trois axes représentant les canaux rouge, bleu et vert et la luminosité. à chaque point représentant le nombre de pixels. Le résultat exact de l'opération dépend de la mise en œuvre - il peut simplement s'agir d'une image de l'histogramme requis dans un format d'image approprié ou d'un fichier de données représentant les statistiques de l'histogramme.

4.2.1. Exemple d'histogramme

Nous allons utiliser une bibliothèque qui fait référence dans le domaine du « Computer Vision » , OpenCV[4] (Open Source Computer Vision Library) est une bibliothèque de logiciels de Machine Learning et de Computer Vison qui est open source. OpenCV a été conçu pour fournir une infrastructure commune aux applications de vision par ordinateur et pour accélérer l'utilisation de la perception de la machine dans les produits commerciaux. OpenCV étant un produit sous licence BSD, il est facile pour les entreprises d'utiliser et de modifier le code.

Il est assez simple de l'installer sur mac (Figure 6), puisqu'il suffit d'utiliser un package manager très connu est assez puissant et qui est uniquement sur MacOS.

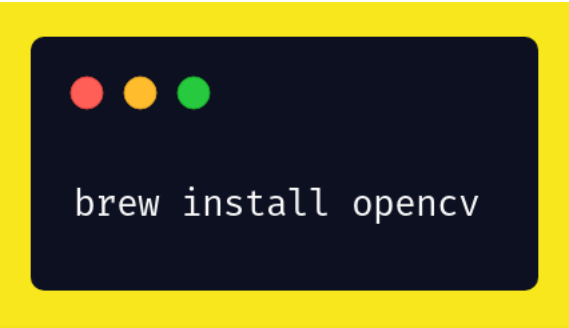

Figure 6: Commande pour installer OpenCV sur MacOS

[4] OpenCV : https://opencv.org/releases.html

Afin de pouvoir compléter l'installation, OpenCV a un certain nombre de dépendances qui faut impérativement installer.

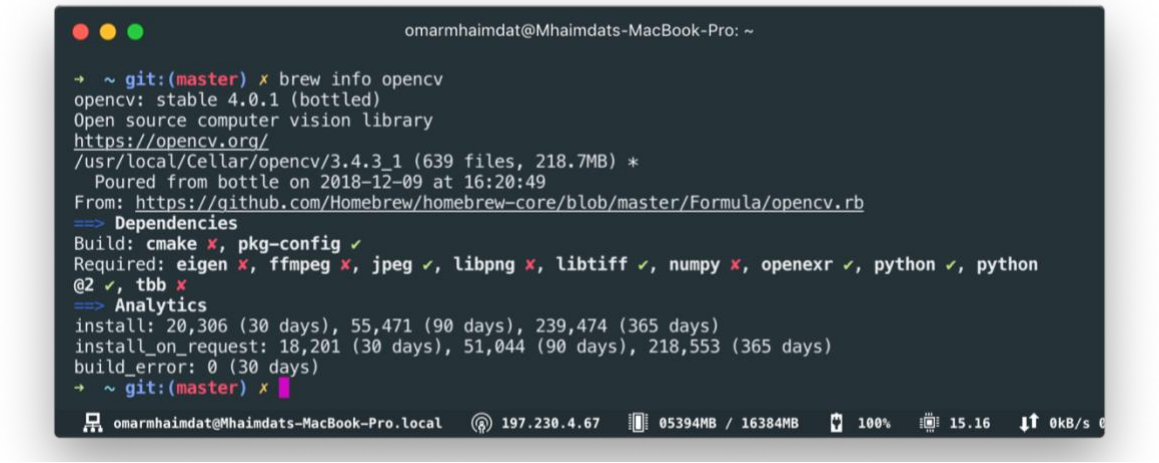

Figure 7: La liste des dépendances de OpenCV

Nous allons prendre une photo (Figure 8) au hasard et faire un certain nombre de transformations et manipulations.

Figure 8: Photo de référence qui va nous servir tout au long de ce rapport

Afin de pouvoir afficher l'histogramme de cette photo (Figure 8), il faut effectuer un nombre d'étapes qui sont explicités dans la (Figure 9).

```cpp
//initialiser les variables
    int histogramSize = 256;
    float range[] = { 0, 256 };
    const float* histogramRange = { range };
    bool uniforme = true;
    //Est ce qu'on souhaite mettre à jour l'histogramme en temps réel
    bool accumulate = false;

    Mat HistogramRED;
    Mat HistogramGREEN;
    Mat HistogramBLUE;

    calcHist(&RGB_planes[0], 1, 0, Mat(), HistogramRED, 1, &histogramSize, &histogramRange, uniforme, accumulate);
    calcHist(&RGB_planes[1], 1, 0, Mat(), HistogramGREEN, 1, &histogramSize, &histogramRange, uniforme, accumulate);
    calcHist(&RGB_planes[2], 1, 0, Mat(), HistogramBLUE, 1, &histogramSize, &histogramRange, uniforme, accumulate);

    int histogramWidth = 1000;
    int histogramHeight = 500;

    int binWidth = cvRound((double) histogramWidth/histogramSize);

    //Initialiser une image qui recevra les histogrammes
    Mat histogramImage(histogramHeight , histogramWidth, CV_8UC3, Scalar(0, 0, 0));

    normalize(HistogramRED, HistogramRED, 0, histogramImage.rows, NORM_MINMAX, -1, Mat());
    normalize(HistogramGREEN, HistogramGREEN, 0, histogramImage.rows, NORM_MINMAX, -1, Mat());
    normalize(HistogramBLUE, HistogramBLUE, 0, histogramImage.rows, NORM_MINMAX, -1, Mat());

    //Boucle qui trace l'histogramme
    for (int i = 0; i < histogramSize; i++) {
        line(histogramImage, Point( binWidth*(i-1), histogramHeight - cvRound(HistogramRED.at<float>(i-1)) ), Point( binWidth*(i), histogramHeight - cvRound(HistogramRED.at<float>(i)) ), Scalar(0, 0, 255), 2, 8, 0);
        line(histogramImage, Point( binWidth*(i-1), histogramHeight - cvRound(HistogramGREEN.at<float>(i-1)) ), Point( binWidth*(i), histogramHeight - cvRound(HistogramGREEN.at<float>(i)) ), Scalar(0, 255, 0), 2, 8, 0);
        line(histogramImage, Point( binWidth*(i-1), histogramHeight - cvRound(HistogramBLUE.at<float>(i-1)) ), Point( binWidth*(i), histogramHeight - cvRound(HistogramBLUE.at<float>(i)) ), Scalar(255, 0, 0), 2, 8, 0);
    }
```

Figure 9: Code pour créer une image contenant un histogramme

Il est à noter que les images sont représentées en matrice, dans ce cas OpenCV a une classe avec plusieurs constructeur pour une matrice qui n'est autre qu'une image.

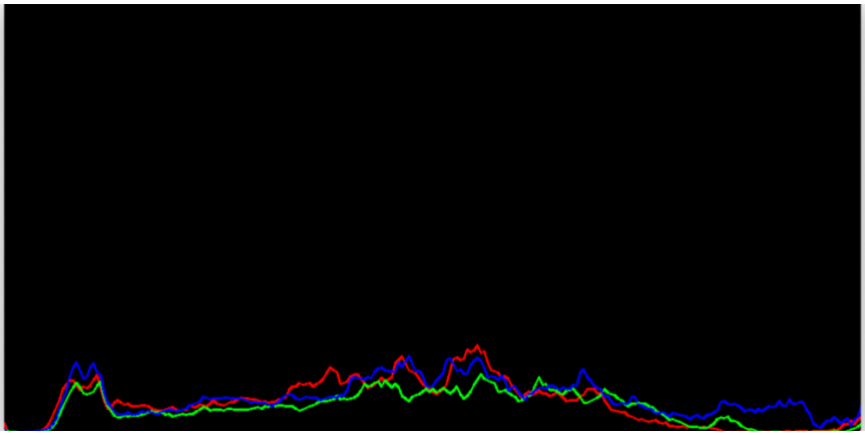

Figure 10: "En rouge" la courbe de distribution de la couleur rouge, "En vert" la courbe de distribution de la couleur verte, "En bleu" la courbe de distribution de la couleur bleu

4.3. Convolution

La convolution est une opération mathématique qui généralise l'idée d'une moyenne (Figure 13) mobile.

Le concept de convolution se généralise à plusieurs dimensions. Par exemple, dans la (Figure 11). Une fonction 2D simple avec des fonctions pointues ressemblant à une étape est « convolutionnée » avec une fonction Gaussienne 2D, produisant une version floue de la fonction d'origine. (Notez que les fonctions de deux variables peuvent être représentées sous forme d'images, la valeur de la fonction pour des coordonnées x et y particulières spécifiant la luminosité du pixel de l'image à cet endroit).

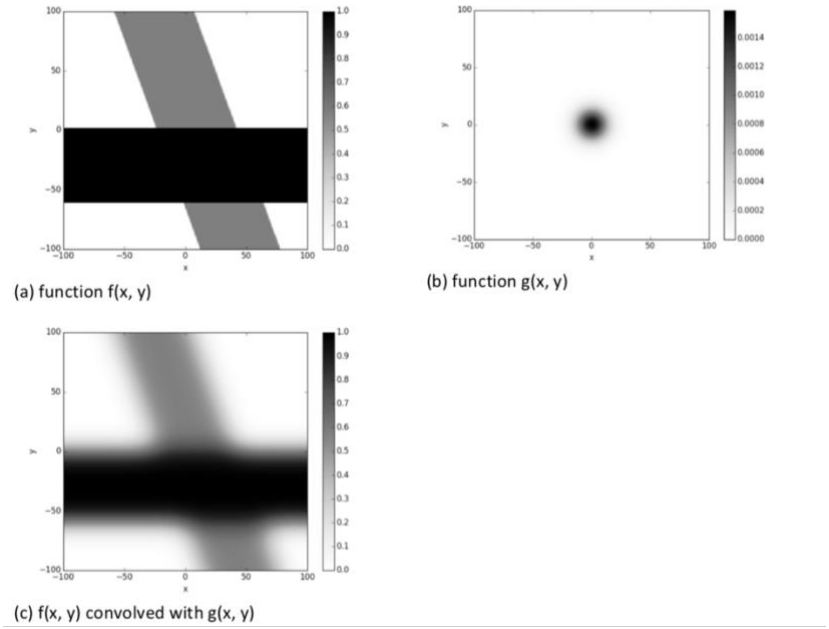

Figure 11: La convolution d'une fonction en 2D (a) avec une fonction gaussienne à deux dimensions (b) donne une version floue de la fonction d'origine (c).

4.4. Filtres

Les filtres ou « Kernel » sont utilisé comme des matrices de convolution afin de faire un traitement sur une image qui elle aussi est une matrice.

4.4.1. Filtre linéaire

La convolution peut nous aider grandement à concevoir des systèmes car au lieu de nous préoccuper des signaux complexes, nous devons nous préoccuper uniquement de la simple fonction δ. Si nous pouvons concevoir la réponse impulsionnelle d'un système, nous savons que le fait de convoluer le signal d'entrée avec une réponse impulsionnelle nous fournirait la réponse correcte pour toute fonction générale.

Prenons le cas de la conception d'un filtre (ou d'une réponse impulsionnelle) qui brouillera tout signal général. Pour concevoir cela, nous devons d'abord penser intuitivement

à quoi ressemblerait un signal delta brouillé. En d'autres termes, que produirait un système linéaire brouillant un signal lorsqu'un delta est fourni en entrée. Pour cela, considérez la (Figure 12).

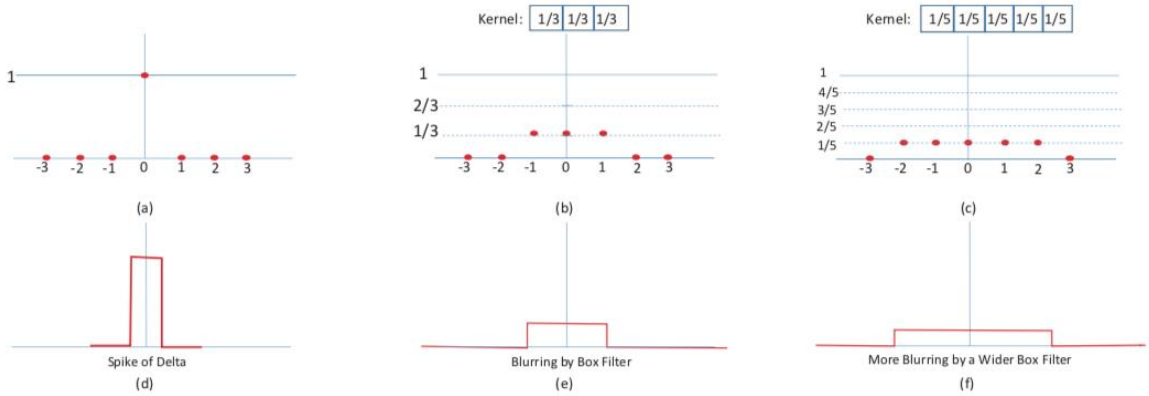

Figure 12: Cette figure montre comment concevoir un filtre de flou en considérant uniquement la simple fonction δ

La rangée supérieure montre le concept à l'œuvre dans une représentation numérique tandis que la rangée inférieure montre la contrepartie analogique. (a) montre un δ. (b) montre la réponse impulsionnelle d'un système flou, c'est-à-dire le destin de δ lorsqu'il est passé à travers un système flou. Au lieu d'avoir une pointe de 1 à 0, il a maintenant une valeur de 1/3 à chacune des valeurs suivantes: -1, 0 et 1. La largeur de « filterv » est également souvent appelée support du noyau ou du filtre. Si la largeur du flou est plus large, on dit que le support du filtre est maintenant plus élevé.

Delta est une fonction qui a un seul échantillon de valeur 1 à 0, qui est essentiellement une pointe. Par conséquent, intuitivement, on peut s'attendre à ce que le flou d'une fonction delta produise une pointe avec une base plus large et une hauteur inférieure. Ceci est représenté par une fonction qui a plusieurs échantillons centrés autour de 0 et dont les valeurs sont inférieures à 1.

4.4.2. Filtre non linéaire

Parlons maintenant de quelques filtres non linéaires qui ne sont pas utilisés pour la détection de caractéristiques (feature detection).

Pour cela, nous allons d'abord explorer un filtre appelé filtre médian. Ce filtre est très similaire au filtre linéaire qui est le filtre de moyenne ou de boîte. Dans un filtre de boîte, la moyenne de toutes les valeurs au voisinage d'un pixel est utilisée pour remplacer la valeur du pixel. Cela permet effectivement un lissage de la fonction et est souvent utilisé pour réduire le bruit. Dans un filtre médian, le pixel est remplacé par la médiane (au lieu de la moyenne) de

toutes les valeurs situées dans son voisinage. En utilisant les marques médianes on filtre un filtre non linéaire.

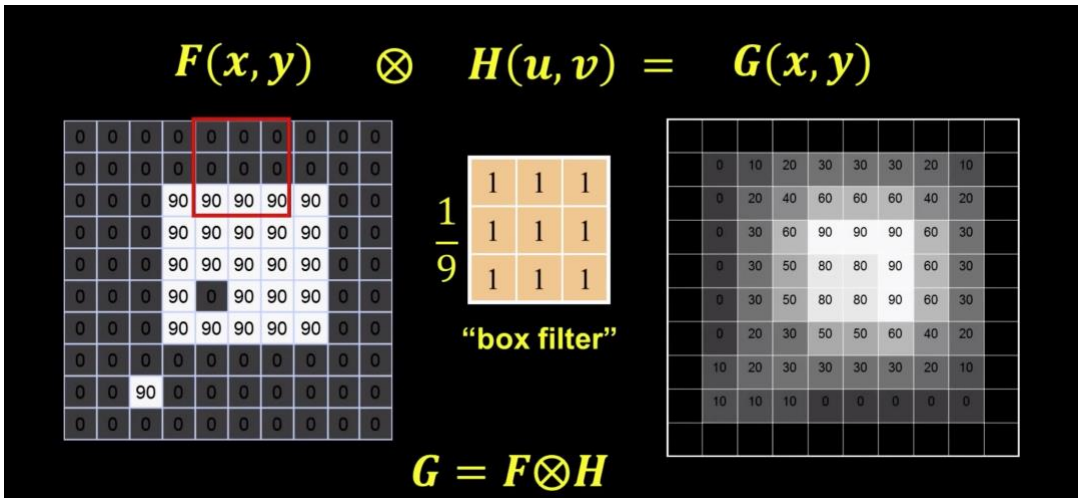

Figure 13: Filtre qui fait la moyenne des pixels proche du pixel en question

Un filtre médian est utilisé pour éliminer les valeurs aberrantes, c'est-à-dire des valeurs radicalement différentes de leur voisinage. Ils peuvent être dus à différentes limitations d'appareils dans différentes applications. Par exemple, un pixel mort dans une caméra peut faire que la valeur de ce pixel soit toujours égale à 1ou 0, ce qui se révélera être une valeur supérieure. Dans le cas des images, ces valeurs aberrantes définissent un bruit que l'on appelle souvent le bruit « salt and pepper ». Les pixels deviennent noirs ou blancs en raison de problèmes liés au système. Contrairement à un filtre médian, un filtre moyen réduit efficacement le bruit gaussien, mais ne fonctionne pas bien pour le bruit du sel et du poivre, car la moyenne tend à étendre la contribution des sites très localisés de sel et de poivre à leur voisinage. Cependant, un filtre médian fonctionne beaucoup mieux car la médiane n'est généralement pas affectée par la variation des valeurs dans le voisinage. La (Figure 14) montre un exemple.

Figure 14: Salt and pepper noise

4.5. Exemple de filtres

4.5.1. Niveaux de gris

Figure 15: À gauche la photo originale, à droite la photo avec uniquement les niveaux de gris

Ci-dessous le code qui permet d'effectuer cette transformation :

```
1 //Transformer l'image en noir et blanc
2     cvtColor(image, image, COLOR_BGR2GRAY);
```

Figure 16: Le code afin de convertir une image en niveaux de gris

4.5.2. Différents canaux de couleurs

Figure 17: À gauche (Rouge), Au milieu (Vert), À droite (Bleu)

Ci-dessous le code qui permet d'effectuer cette transformation (Figure 18):

```
    Mat imageChannel = imread("test_1.jpg", CV_LOAD_IMAGE_COLOR);

    Mat RGB[3];
    split(imageChannel, RGB);

    //S'assurer que l'image s'est bien chargée
    if (!image.data) {
        return -1;
    }

    namedWindow("Photo originale", CV_WINDOW_AUTOSIZE);
    imshow("Photo originale", image);
    imshow("Bleu.png", RGB[0]);
    imshow("Vert.png", RGB[1]);
    imshow("Rouge.png", RGB[2]);


    // Creer une variable de type vector pour tous les canaux et spliter l'image
originale en trois canaux
    //BRG
    vector<Mat> spl;
    split(src,spl);

    // Creer une image qui est une variable de type matrix

    Mat empty_image;
    empty_image = Mat::zeros(src.rows, src.cols, CV_8UC1);

    //Creer trois images pour recevoir les trois canaux
    Mat result_blue(src.rows, src.cols, CV_8UC3);
    Mat result_green(src.rows, src.cols, CV_8UC3);
    Mat result_red(src.rows, src.cols, CV_8UC3);

    // Creer une channel de la couleur bleu
    Mat in1[] = { spl[0], empty_image, empty_image };
    int from_to1[] = { 0,0, 1,1, 2,2 };
    mixChannels( in1, 3, &result_blue, 1, from_to1, 3 );

    // Creer une channel de la couleur verte
    Mat in2[] = { empty_image, spl[1], empty_image };
    int from_to2[] = { 0,0, 1,1, 2,2 };
    mixChannels( in2, 3, &result_green, 1, from_to2, 3 );

    // Creer une channel de la couleur rouge
    Mat in3[] = { empty_image, empty_image, spl[2]};
    int from_to3[] = { 0,0, 1,1, 2,2 };
    mixChannels( in3, 3, &result_red, 1, from_to3, 3 );

    imshow("blue channel",result_blue);
    imshow("green channel",result_green);
    imshow("red channel",result_red);
```

Figure 18: Code qui sépare une photo en trois photos, chacune avec canal de couleurs différent (selon le modèle RGB)

4.5.3. Filtre de « Moyenne standard »

Figure 19:(À gauche) l'image originale, à droite l'image avec un filtre de moyenne standard

Ci-dessous le code qui permet d'effectuer cette transformation (Figure 20) :

```
1    blur(image, H, Size(11,11));
2    namedWindow("Photo Smoothing Moyenne Standard", CV_WINDOW_AUTOSIZE);
3    imshow("Photo Smoothing Moyenne Standard", H);
```

Figure 20: Code qui applique un filtre de moyenne standard sur une image

4.5.4. Filtre gaussien

Figure 21: À gauche l'image originale, à droite l'image avec un filtre gaussien

Ci-dessous le code qui permet d'effectuer cette transformation (Figure 22):

```
1    int hsize = 11;
2    int sigma1 = 5;
3    Mat H;
4    GaussianBlur(image, H, Size(hsize,hsize), sigma1);
5    namedWindow("Photo Smoothing - filtre gaussien", CV_WINDOW_AUTOSIZE);
6    imshow("Photo Smoothing - filtre gaussien", H);
```

Figure 22: Code qui applique un filtre gaussien sur une image avec l'affichage de l'image avec un filtre gaussien

5. Feature detection

5.1. Feature detector

Dans le traitement d'images et de vision par ordinateur, nous devons représenter l'image à l'aide des caractéristiques qui en sont extraites. L'image brute est parfaite pour que l'œil humain puisse extraire toutes les informations; Cependant, ce n'est pas le cas avec les algorithmes informatiques. Il existe généralement deux méthodes pour représenter les images, à savoir les fonctionnalités globales et les fonctionnalités locales. Dans la représentation globale des caractéristiques, l'image est représentée par un vecteur caractéristique multidimensionnel, décrivant les informations dans l'ensemble de l'image. En d'autres termes, la méthode de représentation globale produit un seul vecteur avec des valeurs qui mesurent divers aspects de l'image, tels que la couleur, la texture ou la forme. Pratiquement, un seul vecteur de chaque image est extrait, puis deux images peuvent être comparées en comparant leurs vecteurs caractéristiques. Par exemple, quand on veut distinguer les images d'une mer (bleue) et d'une forêt (verte), un descripteur global de couleur produirait des vecteurs très différents pour chaque catégorie.

Figure 23: Représentation des caractéristiques globales et locales

Dans ce contexte, les caractéristiques globales peuvent être interprétées comme une propriété particulière de l'image impliquant tous les pixels. Cette propriété peut être un histogramme de couleur, une texture, des arêtes ou même un descripteur spécifique extrait de

certains filtres appliqués à l'image. D'autre part, l'objectif principal de la représentation locale est de représenter distinctement l'image en fonction de certaines régions saillantes tout en restant invariant des changements de point de vue et d'éclairage. Ainsi, l'image est représentée sur la base de ses structures locales par un ensemble de descripteurs de caractéristiques locales extraits d'un ensemble de régions d'image appelées régions d'intérêt (c'est-à-dire, points clés) comme illustré sur la (Figure 23). La plupart des caractéristiques locales représentent une texture dans l'image. pièce.

Tuytelaars et Mikolajczyk cité dans le livre définissent une caractéristique locale comme « un motif d'image qui diffère de son voisinage immédiat ». Ainsi, ils considèrent que le but des caractéristiques invariantes locales est de fournir une représentation qui permette de faire correspondre efficacement les structures locales entre les images. En d'autres termes, nous souhaitons obtenir un ensemble fragmenté de mesures locales qui capturent l'essence des images d'entrée sous-jacentes et codent leurs structures intéressantes.

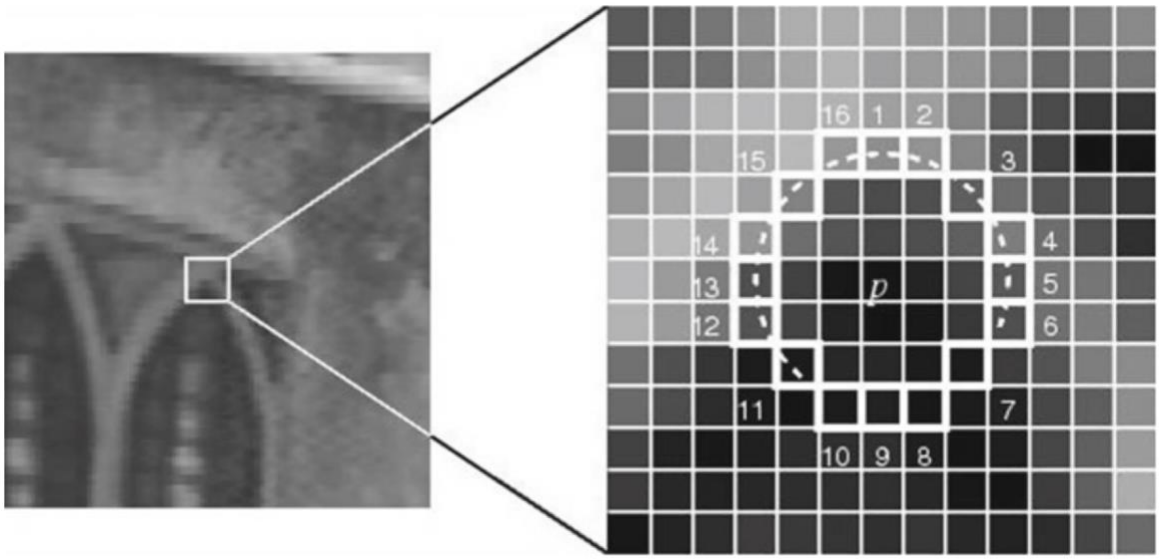

Figure 24: Détection de fonctionnalités dans un patch d'image à l'aide du détecteur de type FAST[5]

Pour atteindre cet objectif, les détecteurs et les extracteurs de fonctionnalités doivent avoir certaines propriétés, en gardant à l'esprit que leur importance dépend des paramètres réels de l'application et que des compromis doivent être apportés. Les propriétés suivantes sont importantes pour l'utilisation d'un détecteur de caractéristiques dans les applications de vision par ordinateur:

- Robustesse : l'algorithme de détection de caractéristiques doit pouvoir détecter les mêmes emplacements de caractéristiques indépendamment de la mise à l'échelle, de la

[5] FAST : Features from Accelerated Segment Test docs.opencv.org

rotation, du décalage, des déformations photométriques, des artefacts de compression et du bruit.

- Répétabilité : l'algorithme de détection de caractéristiques devrait pouvoir détecter les mêmes caractéristiques de la même scène ou du même objet à plusieurs reprises dans diverses conditions de visualisation.
- Précision : l'algorithme de détection de caractéristiques doit localiser avec précision les caractéristiques de l'image (mêmes emplacements de pixel), en particulier pour les tâches de correspondance d'image, lorsque des correspondances sont nécessaires pour estimer la géométrie épipolaire.
- Généralités : l'algorithme de détection de caractéristiques devrait pouvoir détecter les caractéristiques qui peut être utilisé dans différentes applications.
- L'efficacité : l'algorithme de détection de caractéristiques devrait pouvoir détecter les caractéristiques détecter rapidement les caractéristiques de nouvelles images afin de prendre en charge les applications en temps réel.
- Quantité : l'algorithme de détection des caractéristiques devrait pouvoir détecter la totalité ou la plupart des caractéristiques de l'image. Où, la densité des caractéristiques détectées doit refléter le contenu en informations de l'image afin de fournir une représentation compacte de l'image.

5.2. Feature matching

Une fois que nous avons extrait les caractéristiques et leurs descripteurs de deux ou plusieurs images, l'étape suivante consiste à établir des correspondances préliminaires des caractéristiques entre ces images.

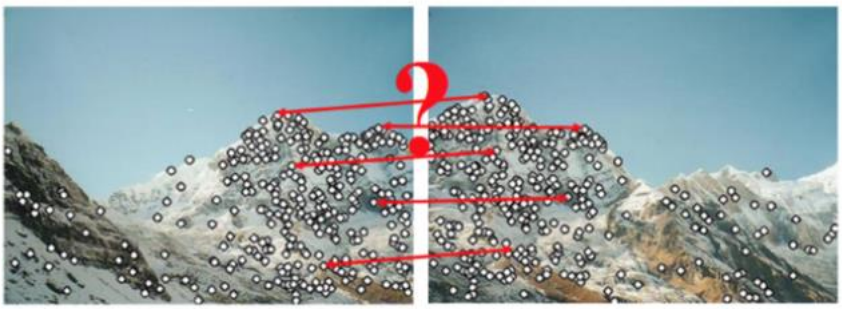

Figure 25: Extraire des descripteurs locaux invariants des variations inter-images et suffisamment discriminants pour établir des correspondances correctes

La détermination des correspondances de caractéristiques qu'il est raisonnable de traiter ultérieurement dépend du contexte dans lequel la correspondance est effectuée. Supposons que deux images se chevauchent (par exemple, pour l'assemblage d'images, comme illustré dans la

(Figure 25), ou pour le suivi d'objets dans une vidéo. Nous savons que la plupart des caractéristiques d'une image sont susceptibles de correspondre à l'autre image, bien que certaines puissent ne pas correspondre parce qu'elles sont obstruées ou que leur apparence a trop changé.

Figure 26: Deux images d'entraînement de la base de données sont affichées à gauche. Celles-ci sont associées à la scène encombrée du milieu à l'aide des fonctions SIFT, affichées sous forme de petits carrés dans l'image de droite. La chaîne affine de chaque image de base de données reconnue sur la scène est représentée par un parallélogramme plus grand dans l'image de droite.

D'autre part, si nous essayons de reconnaître le nombre d'objets connus apparaissant dans une scène encombrée (Figure 26), la plupart des fonctionnalités risquent de ne pas correspondre. En outre, il faut rechercher un grand nombre d'objets potentiellement concordants, ce qui nécessite des stratégies plus efficaces, comme décrit ci-dessous.

Pour commencer, nous supposons que les descripteurs d'entités ont été conçus de manière à ce que les distances euclidiennes (magnitude vectorielle) dans l'espace des entités puissent être directement utilisées pour le classement des correspondances potentielles. S'il s'avère que certains paramètres (axes) d'un descripteur sont plus fiables que d'autres, il est généralement préférable de redimensionner ces axes à l'avance, par exemple en déterminant leur variation par rapport à d'autres bonnes correspondances connues. Un processus plus général, qui implique la transformation des vecteurs de caractéristiques en une nouvelle base à l'échelle, est appelé «whitening».

5.3. Edge detection (détection de contours)

Bien que les points d'intérêt soient utiles pour trouver des emplacements d'image qui puissent être précisément mis en correspondance en 2D, les points de contour sont beaucoup plus nombreux et comportent souvent d'importantes associations sémantiques. Par exemple, les limites des objets, qui correspondent également à des événements d'occlusion en 3D, sont généralement délimitées par des contours visibles. D'autres types d'arêtes correspondent aux

limites d'ombre ou aux arêtes froissées, où l'orientation de la surface change rapidement. Les points de contour isolés peuvent également être regroupés dans des courbes ou des contours plus longs, ainsi que des segments de droite. Il est intéressant de noter que même les enfants n'ont aucune difficulté à reconnaître des objets ou des animaux familiers à partir de dessins au trait très simples.

La détection des contours est une technique de traitement d'image permettant de déterminer les limites d'objets dans des images. Cela fonctionne en détectant les discontinuités de luminosité (intensité de lumière). La détection des contours est utilisée pour la segmentation d'images et l'extraction de données dans des domaines tels que le traitement d'images, la vision par ordinateur.

Figure 27: Segmentation de l'image avec l'algorithme Canny

5.3.1. Algorithme « Canny »

L'opérateur Canny a été conçu pour être un détecteur de conteur optimal (selon des critères particuliers - il existe d'autres détecteurs qui prétendent également être optimaux par rapport à des critères légèrement différents). Il prend en entrée une image en niveaux de gris et produit en sortie une image montrant les positions des discontinuités d'intensité suivies.

L'opérateur Canny travaille dans un processus en plusieurs étapes. Tout d'abord, l'image est lissée par convolution gaussienne. Ensuite, un simple opérateur de première dérivée à deux dimensions est appliqué à l'image lissée afin de mettre en évidence les régions de l'image avec des premières dérivées spatiales élevées. Les bords donnent lieu à des bords dans l'image du dégradé. L'algorithme suit ensuite le haut de ces crêtes et met à zéro tous les pixels qui ne se trouvent pas réellement au sommet de la crête afin de laisser une fine ligne dans la sortie, processus appelé suppression non maximale. Le processus de suivi présente une hystérésis contrôlée par deux seuils: T1 et T2, avec T1> T2. Le suivi ne peut débuter qu'à un point situé

sur une crête supérieure à T1. Le suivi se poursuit ensuite dans les deux directions à partir de ce point jusqu'à ce que la hauteur de la crête tombe en dessous de T2. Cette hystérésis aide à garantir que les bords bruyants ne sont pas divisés en fragments multiples.

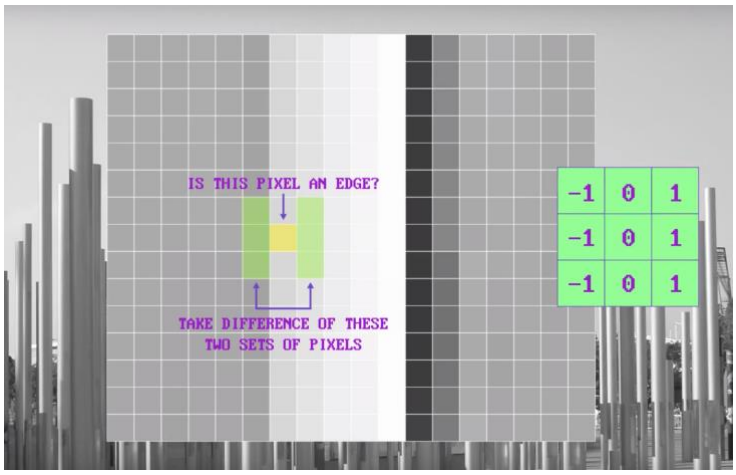

Figure 28: filtre de détection de contours (verticaux)

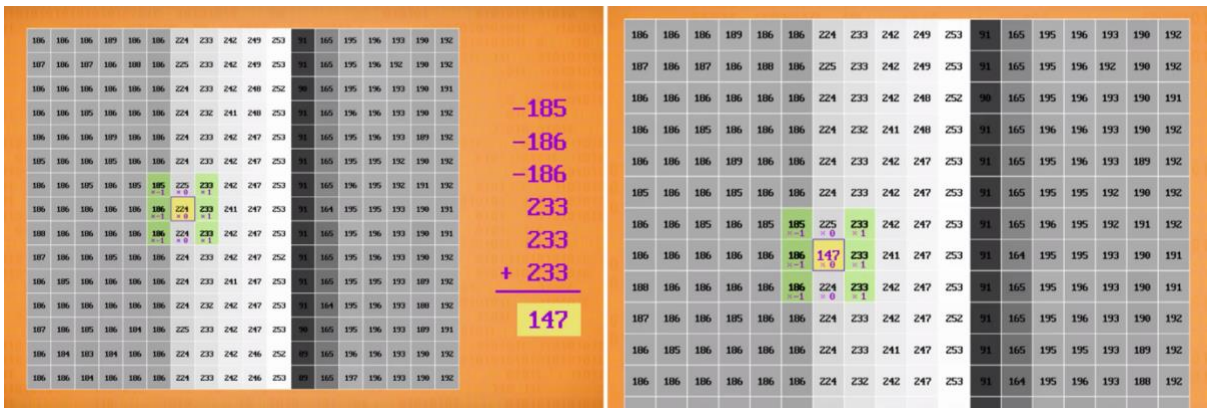

Figure 29: On applique le filtre à la totalité des pixels de la photo, pixel par pixel

Appliquons l'algorithme Canny sur notre image de référence (Figure 8) :

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Figure 30: Gx est le filtre de détection des contours verticaux, Gy est le filtre de détection des contours horizontaux

Figure 31: Filtre Canny appliqué à notre image

Ci-dessous le code qui permet d'effectuer cette transformation :

```
    Mat image = imread("test_1.jpg");
    Mat destination, AvecContours;


    int limiteInferieur = 1;
    int lowThreshold;
    int const limiteSuperieur = 100;
    int ratio1 = 3;
    int tailleFiltre = 3;

void ChangerLimite(int, void*)
{
    // Reduire le bruit avec un filtre 3x3
    blur( image, AvecContours, Size(3,3) );

    // Canny detecteur (algorithme tres performant pour trouver les contours
    Canny( AvecContours, AvecContours, lowThreshold, lowThreshold * ratio1, tailleFiltre );

    // Creer une image avec un fond noir
    destination = Scalar::all(0);

    //Inserer les contours dans destination
    image.copyTo( destination, AvecContours);
    imshow( "Contours", destination );
}

int main(int argc, const char * argv[]) {

    //Transformer l'image en noir et blanc
    cvtColor(image, image, COLOR_BGR2GRAY);

    destination.create( image.size(), image.type() );
    namedWindow( "Contours", CV_WINDOW_AUTOSIZE );
    createTrackbar( "Min:", "Contours", &lowThreshold, limiteSuperieur, ChangerLimite );
    ChangerLimite(0, 0);

    waitKey(0);

    return 0;
}
```

Figure 32: Le code en C++ qui permet de générer les contours de l'image avec l'algorithme Canny

6. Recognition

Si on nous donne une image à analyser, comme le portrait de groupe de la (Figure 33), nous pourrions essayer d'appliquer un algorithme de reconnaissance à toutes les sous-fenêtres possibles de cette image. De tels algorithmes sont susceptibles d'être à la fois lents et sujets aux erreurs. Au lieu de cela, il est plus efficace de construire des détecteurs spéciaux, dont le travail est de trouver rapidement les régions probables où des objets particuliers pourraient se produire.

Figure 33: Résultat de détection de visages dans une photo de groupe

Les détecteurs de visage, qui sont parmi les exemples de reconnaissance les plus réussis. Par exemple, de tels algorithmes sont intégrés à la plupart des appareils photo numériques actuels pour améliorer la mise au point automatique et aux systèmes de vidéoconférence pour contrôler les têtes rotatives.

6.1. Détection de visage en temps réel (Algorithme Viola Jones)

La détection des visages est l'un des problèmes les plus complexes et les plus difficiles dans le domaine de la vision par ordinateur, en raison des fortes variations causées par les changements d'apparence faciale, de l'éclairage et de l'expression faciale. La distribution des visages doit donc être hautement non linéaire et complexe dans tout espace linéaire par rapport à l'espace image d'origine. Dans les applications temps réel telles que la surveillance et la biométrie, les limitations de la caméra et les variations de diverses rendent la distribution des visages humains dans l'espace des fonctions plus complexe que celle des visages frontaux. Cela complique encore le problème de la détection de visage.

De nombreuses techniques ont été créé durant ces vingt dernières années et de nombreux progrès ont été proposés dans ce domaine. La plupart des méthodes de détection se concentraient sur la détection de faces frontales présentant suffisamment de conditions d'éclairage. Dans son enquête cité dans le livre, Mr Yang[6] a classé cette méthode en quatre types: basée sur la connaissance, invariante par rapport aux caractéristiques, correspondant à des modèles et basée sur l'apparence.

- Des méthodes basées sur la connaissance ont modélisé les traits du visage en utilisant un codage humain, tel que deux symétriques, bouche, nez etc.
- Les caractéristiques qui sont invariantes pour la pose et les conditions d'éclairage sont déterminées à l'aide de la méthode invariante.
- La corrélation entre une image test et une image antérieure entre dans la catégorie de correspondance des modèles.
- Cette méthode basée sur l'apparence comprend des techniques d'apprentissage automatique (Machine Learning) permettant d'extraire une caractéristique discriminante d'un ensemble pré-étiqueté.

Paul Viola et Michael Jones ont présenté une méthode rapide et robuste pour la détection des visages, 15 fois plus rapide que n'importe quelle technique au moment de la publication, avec une précision de 95% à environ 17 FPS[7].

L'algorithme de Viola et Jones est utilisé comme base de notre conception. Comme nous savons qu'il existe des similitudes dans tous les visages humains, nous avons utilisé ce concept comme une fonction de Haar pour détecter les visages dans l'image. L'algorithme recherche la caractéristique spécifique Haar d'un visage, si cette caractéristique est trouvée on transfère le candidat à l'étape suivante, c'est-à-dire au Haar suivant. Ici, le candidat n'est pas une image complète, mais une partie rectangulaire de cette image connue sous le nom de sous-fenêtre a une taille de 24 * 24 pixels. Avec cette fenêtre, l'algorithme vérifie l'ensemble de l'image.

6.1.1. Haar features

Les caractéristiques de type Haar sont des caractéristiques d'image numérique utilisées dans la reconnaissance d'objet. Ils doivent leur nom à leur similarité intuitive avec les ondelettes de Haar [8] et ont été utilisés dans le premier détecteur de visage en temps réel.

[6] Yang : Architecte GPU senior travaillant pour le groupe Architecture graphique de NVIDIA Corporation
[7] FPS : Frame Per Second
[8] Ondelette de Haar : Il s'agit d'une fonction constante par morceaux

Historiquement, le fait de ne travailler qu'avec des intensités d'image (c'est-à-dire les valeurs de pixel RGB pour chaque pixel de l'image) rendait la tâche du calcul des caractéristiques coûteuse en calculs. Viola et Jones propose la possibilité de travailler avec un autre ensemble de caractéristiques basé sur les ondelettes de Haar au lieu des intensités d'image habituelles. Viola et Jones ont adapté l'idée d'utiliser des ondelettes de Haar et développé les fonctions dites de type Haar. Une caractéristique de type Haar considère les régions rectangulaires adjacentes à un emplacement spécifique dans une fenêtre de détection, résume les intensités de pixels dans chaque région et calcule la différence entre ces sommes. Cette différence est ensuite utilisée pour classer les sous-sections d'une image. Par exemple, supposons que nous ayons une base de données d'images à visage humain. Il est courant de constater que parmi tous les visages, la région des yeux est plus sombre que la région des joues.

Par conséquent, une caractéristique commune à Haar pour la détection des visages est un ensemble de deux rectangles (Figure 34) adjacents situés au-dessus de l'œil et de la région des joues. La position de ces rectangles est définie par rapport à une fenêtre de détection qui agit comme un cadre de sélection pour l'objet cible (le visage dans ce cas).

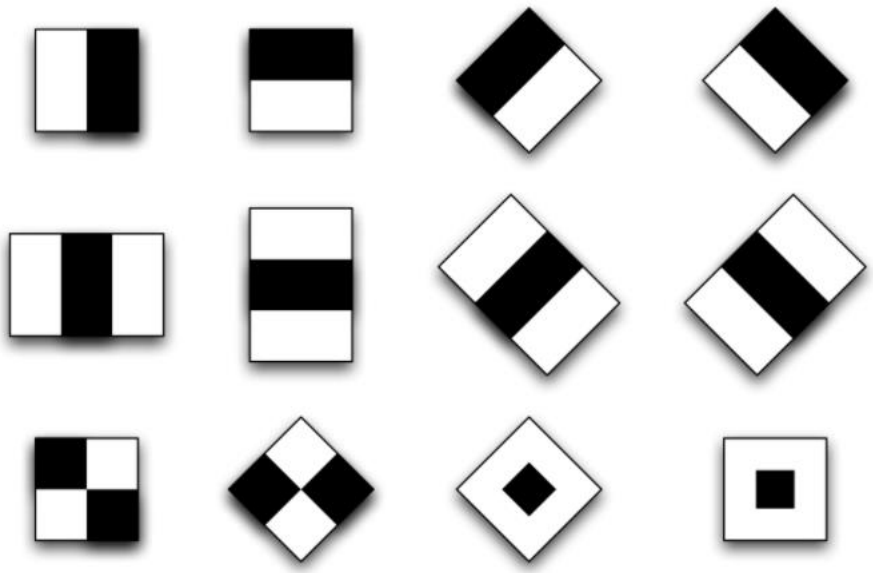

Figure 34: Exemple de "Haar features"

Dans la phase de détection du cadre de détection d'objet Viola et Jones, une fenêtre de la taille de la cible est déplacée sur l'image en entrée et, pour chaque sous-section de l'image, la fonction de type Haar est calculée. Cette différence est ensuite comparée à un seuil appris qui sépare les non-objets des objets. Comme une telle caractéristique de Haar n'est qu'un apprenant ou un classificateur faible (sa qualité de détection est légèrement meilleure que celle de deviner de manière aléatoire), un grand nombre de caractéristiques de type Haar sont

nécessaires (exemple Figure 35) pour décrire un objet avec une précision suffisante. Dans le cadre de détection d'objets Viola et Jones, les fonctionnalités de type Haar sont donc organisées en une cascade appelée classificateur en cascade pour former un apprenant ou un classificateur puissant.

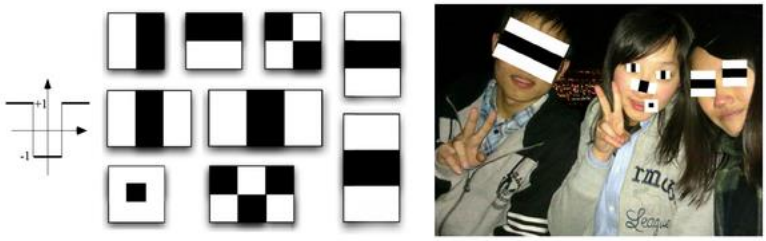

Figure 35: Exemple de "Haar features" appliqués sur une image

Le principal avantage d'une fonctionnalité de type Haar par rapport à la plupart des autres fonctionnalités est sa vitesse de calcul.

6.1.2. Integral image

Étant donné qu'il est évident qu'un nombre considérable de ces caractéristiques « Haar » rectangulaires doivent être évaluées chaque fois que Viola Jones propose une technique soignée pour réduire le calcul, ainsi au lieu de faire la somme de toutes les valeurs de pixels sous les rectangles noir et blanc.

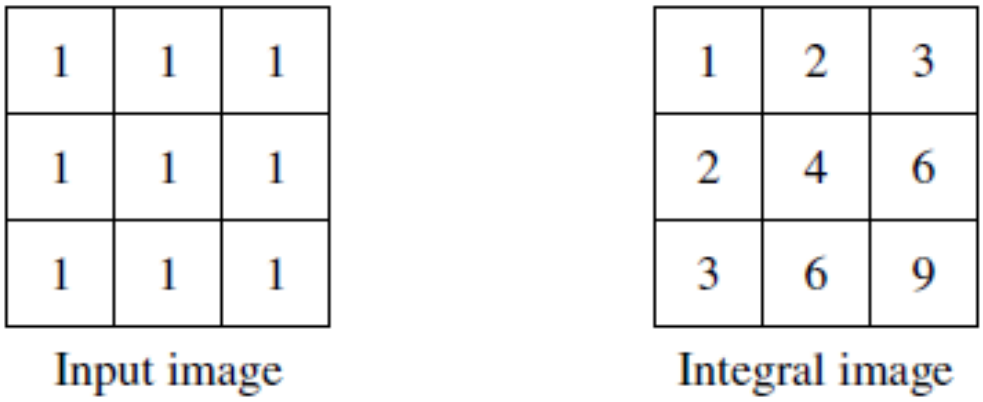

Figure 36: Transformation en Image intégrale

Ils ont introduit le concept d'image intégrale pour trouver la somme de tous les pixels sous un rectangle avec seulement 4 valeurs de coin de l'image intégrale (Figure 36).

6.1.3. Adaboost

Comme indiqué précédemment, il peut y avoir environ 160 000 valeurs de caractéristiques dans un détecteur à une résolution de base de 24 x 24 qui doivent être calculées.

Mais il faut comprendre que seuls quelques ensembles de fonctionnalités seront utiles parmi toutes ces fonctionnalités pour identifier un visage.

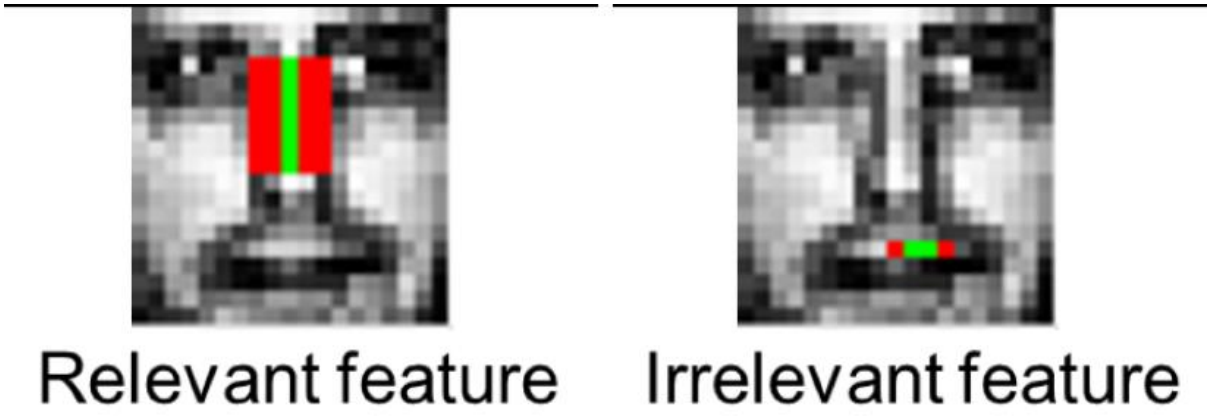

Figure 37: À gauche une caractéristique utile, à droite une caractéristique inutile

Adaboost est un algorithme d'apprentissage automatique qui aide à trouver uniquement les meilleures fonctionnalités parmi toutes ces 160 000 fonctionnalités. Après avoir trouvé ces caractéristiques, une combinaison pondérée de toutes ces caractéristiques est utilisée pour évaluer et décider si une fenêtre a un visage ou non. Chacune des caractéristiques sélectionnées est considérée comme acceptable si elle peut au moins exécuter mieux que la valeur aléatoire (détecte plus de la moitié des cas).

Ces fonctionnalités sont également appelées classificateurs faibles. Adaboost construit un classificateur puissant sous la forme d'une combinaison linéaire de ces classificateurs faibles.

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots$$

Figure 38: F(x) est un classificateur puissant, les f(x) sont des classificateurs faibles

6.1.4. Cascading

Le classificateur en cascade est composé d'étapes contenant chacune un classificateur puissant d'AdaBoost. La tâche de chaque étape est de déterminer si une sous-fenêtre donnée n'est pas un visage ou peut-être un visage. Lorsqu'une sous-fenêtre est classée comme une non-visage à un stade donné, elle est immédiatement supprimée.

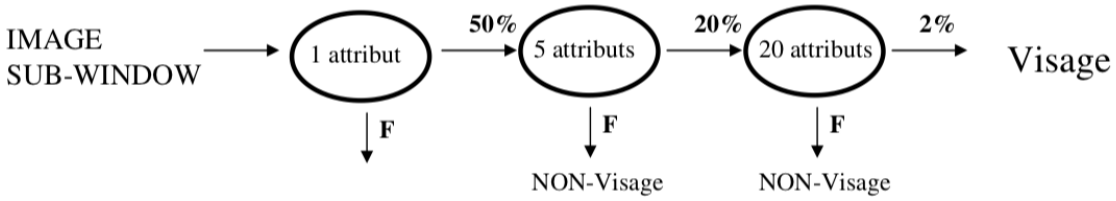

Figure 39: Processus de casacading

Inversement, une sous-fenêtre classée comme un visage peut-être transmise à la prochaine étape de la cascade. Il s'ensuit que plus une sous-fenêtre donnée passe par plusieurs étapes, plus il y a de chances que la sous-fenêtre contienne un visage.

6.1.5. Exemple de détection de visage en temps réel

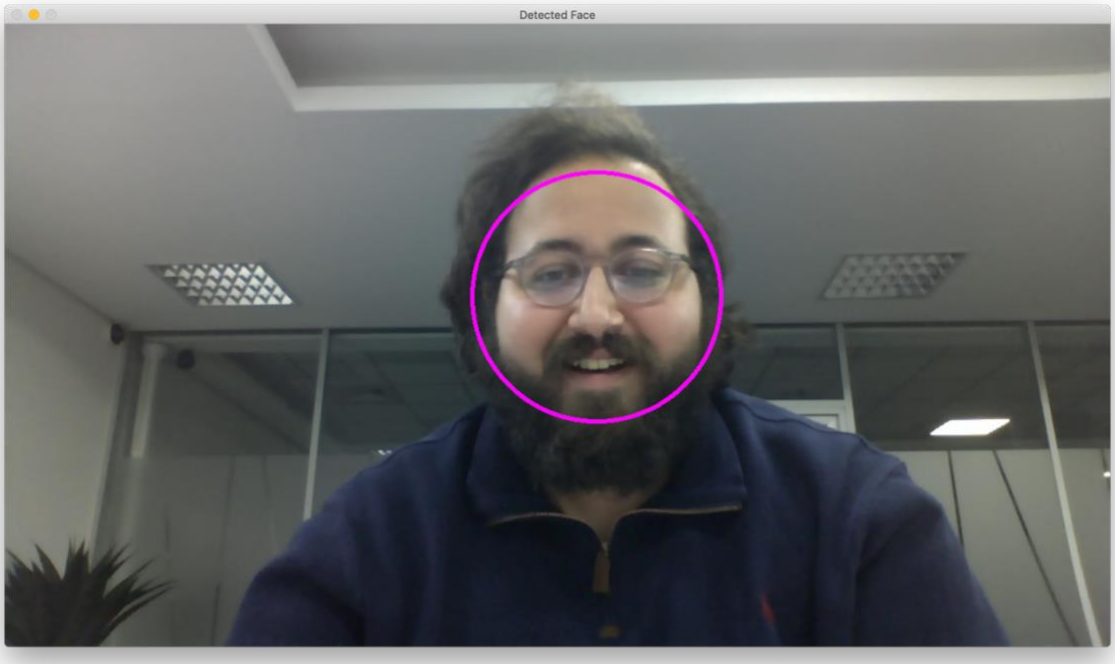

Figure 40: Détection de visage - position frontale

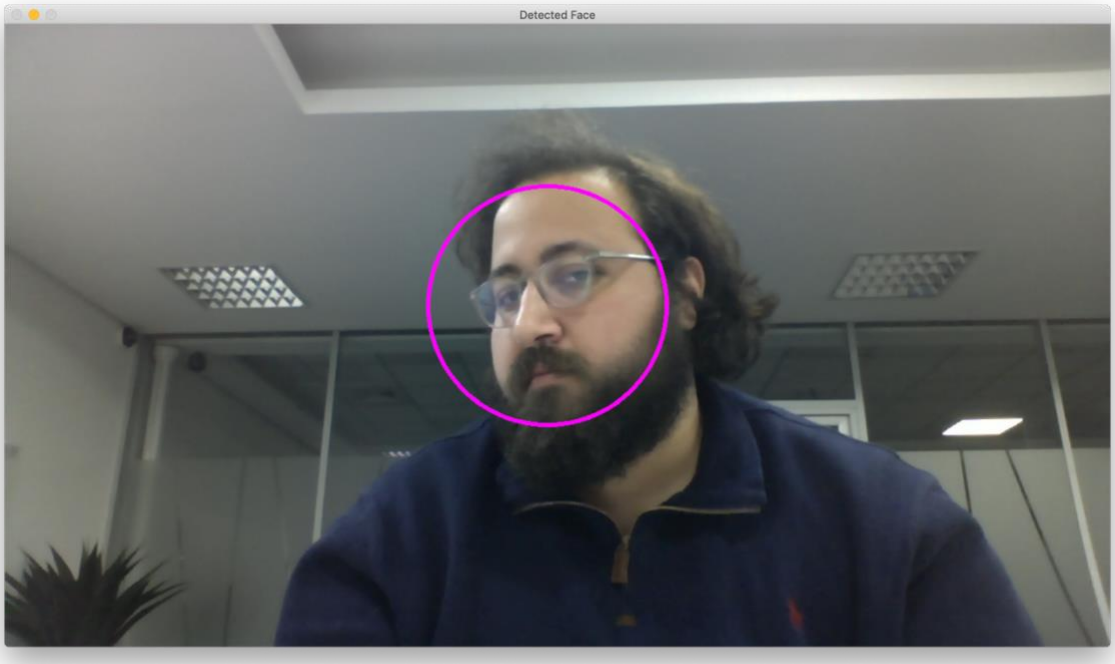

Figure 41: Détection de visage - position latérale

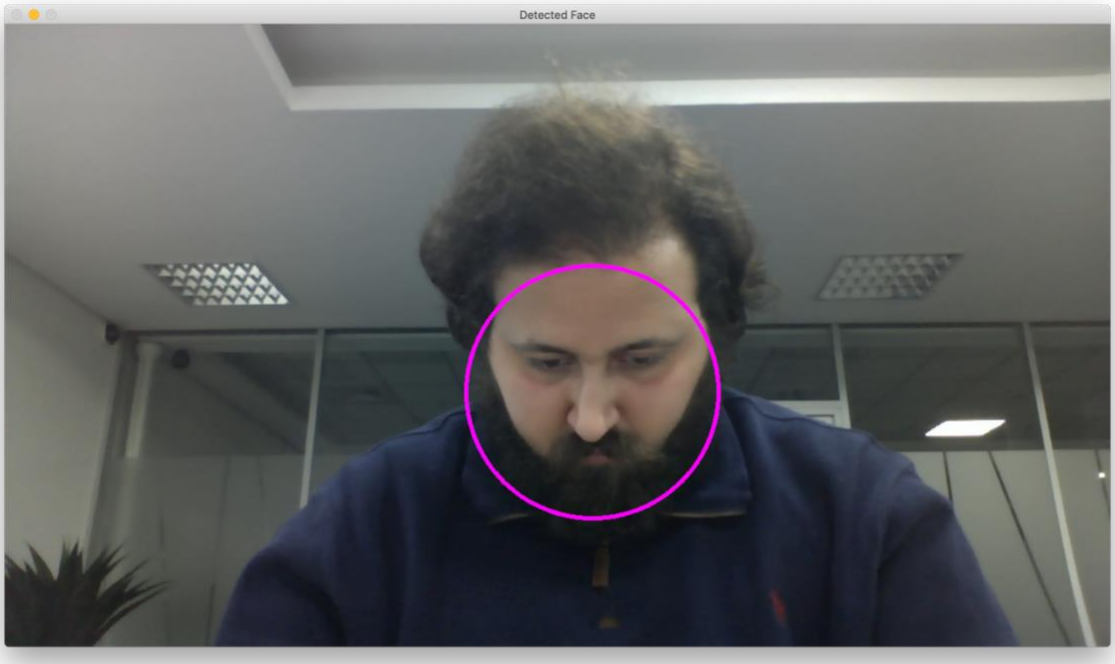

Figure 42: Détection de visage - position en bas

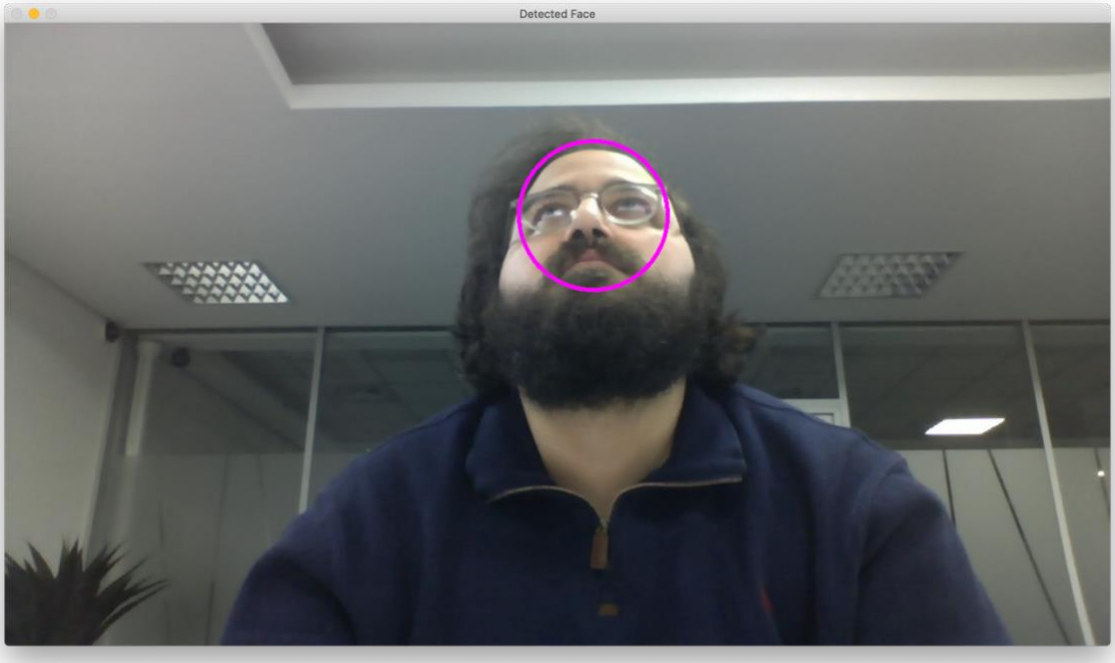

Figure 43: Détection de visage - position haut

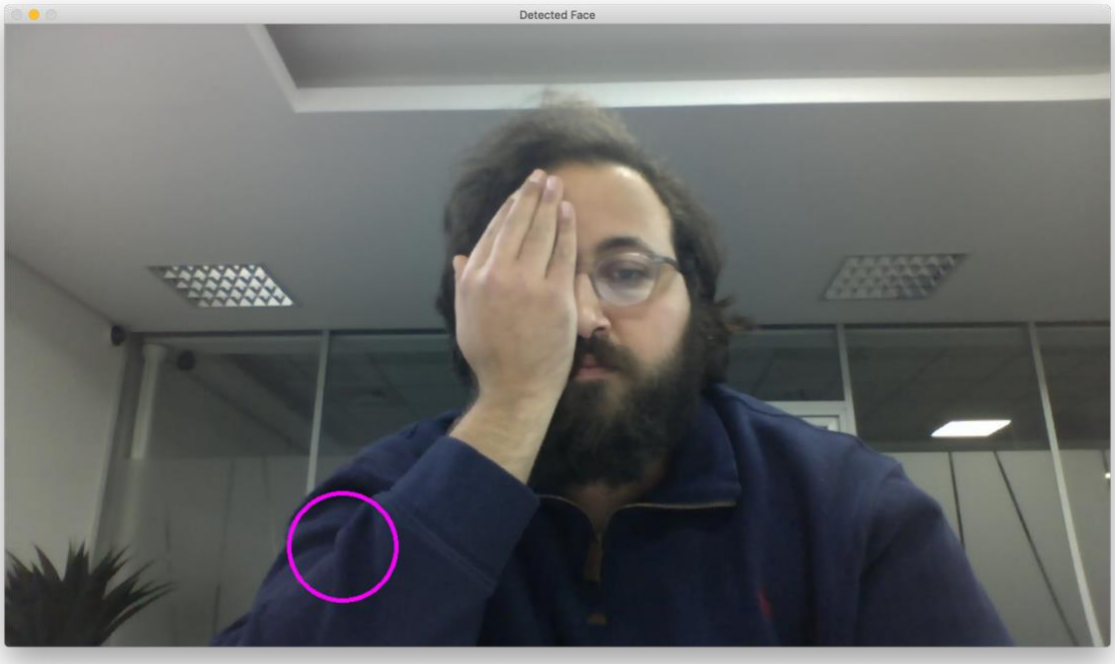

Figure 44: Limitation de l'algorithme de Viola et Jones

Figure 45: Détection simultané de plusieurs visages

Ci-dessous le code qui permet de détecter les visages en temps réel :

```cpp
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

int main( )
{
    VideoCapture capture(0);
    if (!capture.isOpened())
        throw "Error when reading file";
    namedWindow("window", 1);
    for ( ;; )
    {
        Mat image;
        capture >> image;
        if (image.empty())
            break;

        // Charge le classificateur en cascade qui est déjà entraîné (un fichier xml)
        CascadeClassifier face_cascade;
        face_cascade.load("haarcascade_frontalface_default.xml" );
        if(!face_cascade.load("haarcascade_frontalface_default.xml"))
        {
            cerr<<"Erreur, un problème avec le fichier xml"<<endl;
            return 0;
        }

        // Détecter les visages
        std::vector<Rect> faces;
        face_cascade.detectMultiScale( image, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE,
Size(30, 30) );

        // Dessiner un cercle sur les visages détecter
        for( int i = 0; i < faces.size(); i++ )
        {
            Point center( faces[i].x + faces[i].width*0.5, faces[i].y +
faces[i].height*0.5 );
            ellipse( image, center, Size( faces[i].width*0.5, faces[i].height*0.5),
0, 0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );
        }

        //Charger une fenêtre afin d'afficher le output de la webcam
        imshow( "Détection de visages", image );
        waitKey(1);
    }

    return 0;
}
```

Figure 46: Code C++ permettant de détecter les visages en temps réel

Conclusion

Ce projet a été une excellente occasion pour moi d'apprendre de nouveaux concepts et surtout cela m'a permis de découvrir un domaine aussi important que le Computer Vision. Il m'a aussi permis d'étendre mes connaissances et mes compétences en rédaction et en présentation.

À la suite de ce travail, j'ai eu la chance d'étudier un sujet extrêmement intéressant et vaste allant de notions déjà connues aux concepts nouveaux et complexes de la vision par ordinateur et des technologies informatiques en général jamais étudiés auparavant dans notre programme actuel.

Grâce à cette initiative, j'ai acquis de nombreuses compétences générales telles que la gestion du temps, le développement d'un esprit analytique et même lors de la rédaction de ce rapport, j'ai pu perfectionner mes compétences rédactionnelles techniques. Enfin, j'ai également amélioré mes compétences en prise de parole en public grâce aux présentations que j'ai effectuées tout au long de ce semestre.

La prochaine étape serait certainement d'aller au-delà des techniques et algorithmes exposés dans ce rapport, et je pense que l'un des domaines les plus intéressant qui semble être très promettant est le Convolutional Neural Network. Ce sera certainement mon prochain sujet pour le semestre à venir.

Je voudrais remercier notre professeur, Mr. Mohammed BOUTABIA pour ses directives et ses réponses à toutes nos questions concernant ce projet.

Bibliographie

- "Canny Edge Detector." *Spatial Filters - Gaussian Smoothing*, homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm.
- "Cascade Classifier." *Cascade Classification - OpenCV 2.4.13.7 Documentation*, docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.
- *COMPUTER VISION, PATTERN RECOGNITION, IMAGE PROCESSING, AND GRAPHICS: 6th*. SPRINGER, 2018.
- Khalil KHATTAB. *Détection De Visages En Temps Réel Avec La Méthode De Viola Jones*.
- Efendi, Jacky, et al. "Real Time Face Recognition Using Eigenface and Viola-Jones Face Detector." *JOIV : International Journal on Informatics Visualization*, vol. 1, no. 4, 2017, p. 16., doi:10.30630/joiv.1.1.15.
- "FAST Algorithm for Corner Detection." *Cascade Classification - OpenCV 2.4.13.7 Documentation*, docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html.
- Hassaballah, M., et al. "Image Features Detection, Description and Matching." *Image Feature Detectors and Descriptors Studies in Computational Intelligence*, 2016, pp. 11–45., doi:10.1007/978-3-319-28854-3_2.
- Majumder, Aditi. *Introduction to Visual Computing: Core Concepts in Computer Vision, Graphics, and Image Processing*. Taylor & Francis, a CRC Title, Part of the Taylor & Francis Imprint, a Member of the Taylor & Francis Group, the Academic Division of T&F Informa, Plc, 2017.
- "OpenCV Modules." *Cascade Classification - OpenCV 2.4.13.7 Documentation*, docs.opencv.org/4.0.1/.
- Szeliski, Richard. *Computer Vision: Algorithms and Applications*. Springer, 2011.

- Viola, P., and M. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features.” *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, doi:10.1109/cvpr.2001.990517.
- Zhu, Jie, and Zhiqian Chen. “Real Time Face Detection System Using Adaboost and Haar-like Features.” *2015 2nd International Conference on Information Science and Control Engineering*, 2015, doi:10.1109/icisce.2015.95.