# Reconnaissance d'images en réalité augmentée sur mobile

## Étude du passage à l'échelle

13.01.2023

CHAPELLE Rose
MIOLLANY Yanis
SAINT-LOUIS-AUGUSTIN Elaura
WITZEL Florette

# Structure



# 1. Introduction

Ce travail est fait dans le cadre du cours « Initiation à la recherche » de Fadila Bentayeb.

Le sujet de recherche a été proposé par Serge Miguet et est en rapport avec son sujet de recherche appelé Projet Augmented Artwork Analysis. Il a été proposé par des spécialistes en histoire de l'art et permettrait aux visiteurs de musées d'afficher, sur le côté, les données complémentaires de l'œuvre. Ces données sont : le contexte de sa production, les œuvres qui ont pu inspirer l'auteur pour la production, autrement dit la généalogie de l'œuvre.

Pour reconnaître l'image que filme l'utilisateur de l'application, il faut tout d'abord trouver les points d'intérêt de l'image filmée, ces points d'intérêts sont des groupements de pixels qui vont, suivant une méthode de détection, être transformés en une matrice et devenir des descripteurs. Ces descripteurs pourront ensuite être comparés et les images ayant des descripteurs similaires pourront être considérées comme étant les mêmes images par le programme [1].

Ces travaux s'inscrivent dans la recherche de l'avancée de la réalité augmentée dans l'art. La réalité augmentée s'est associée au domaine artistique aux environs des années 70, nous connaissons tout d'abord l'association entre le numérique et l'art au travers de diverses expositions d'art vidéo avec l'artiste Nam June Paik [2], de photographie ou d'art digital avec récemment l'arrivée des NFT. La représentation d'une version numérique du cadre de vie réel n'a cessé de s'accroître, impliquant des expositions ayant pour thème la réalité augmentée [3] et des œuvres en interaction avec celle-ci via ordinateur ou casque de réalité virtuelle [4].

C'est grâce à ses avancées que nous pouvons continuer à chercher de plus en plus loin afin de non seulement participer au développement artistique, mais aussi de celui de la réalité augmentée.

*Projet Augmented Artwork Analysis* analyse une œuvre d'art, elle fonctionne sans accès à internet.

Pour répondre à notre problématique, nous avons étudié : le principe de la reconnaissance d'objet avec *SIFT* et divers algorithmes permettant de comprendre ces principes et les utiliser.

## 2. État de l'art

Ce chapitre présente les bases nécessaires à la réalisation des objectifs de recherche. Tout d'abord, la section 2.1 traite les bases de la reconnaissance d'objets. Le paragraphe 2.2 explique l'algorithme de détection et de description de caractéristiques « Scale-invariant feature transform ». Enfin, la bibliothèque « Fast Library for Approximate Nearest Neighbours » est présentée dans la section 2.3.

## 2.1 Reconnaissance d’objets

La reconnaissance d’objet est un domaine de l’intelligence artificielle qui vise à identifier et à classer des objets dans des images ou des vidéos, elle est utilisée dans de nombreux secteurs.

Les descripteurs sont utilisés en reconnaissance d’objets pour décrire, à l’aide de points d’intérêts, une image. Il existe plusieurs types d'algorithmes analysant les descripteurs d’une image qui sont utilisés selon le type d’objet à identifier. Ils sont souvent utilisés avec d’autres techniques de reconnaissance d’objets comme les algorithmes de vision par ordinateurs ou les réseaux de neurones artificiels. Ils permettent de fournir une description précise et unique de l’objet, ce qui facilite son identification et sa classification.

La matrice homographique est une matrice trois par trois qui peut être utilisée pour appliquer une transformation à ces coordonnées et permettant de vérifier la similarité entre deux images. Elle est généralement calculée à partir de correspondances entre les points clés d’images différentes. Par conséquent, elle est déterminée à partir d’au moins quatre correspondances entre les points d’intérêts dans ces dernières. Une fois que la matrice homographique est calculée, elle peut être utilisée pour aligner les images selon la même perspective en transformant les coordonnées de chaque point de l’image à tester [5].

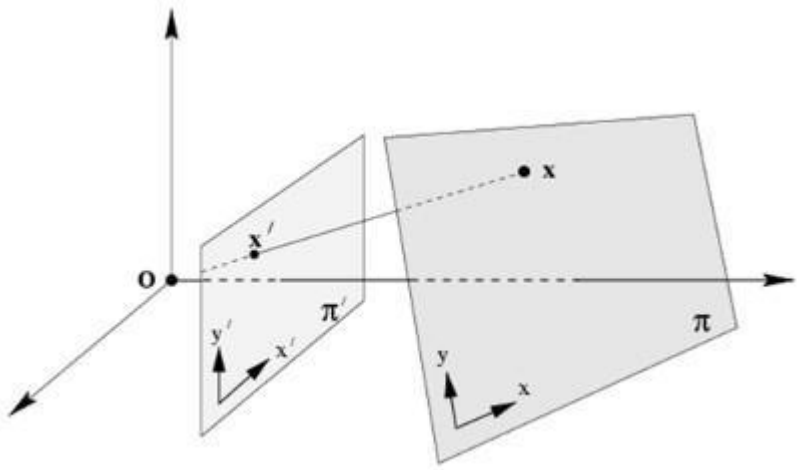


*Image 2.1 Relation entre deux planes [5].*

## 2.2 Scale-invariant feature transform (SIFT)

SIFT (Scale-Invariant Feature Transform ou “transformation de caractéristiques visuelles invariante à l’échelle”) est un algorithme de détection de points d’intérêts et de description de caractéristiques utilisé en traitement d’image et en vision par ordinateur, il permet de trouver des points de repères ou points d’intérêts (“keypoints”) stables dans une image en utilisant des descripteurs de caractéristiques [6].

La détection de points d’intérêts est faite en trouvant les extrema dans l’espace-échelle Gaussien d’une image $I(x, y)$ en utilisant le filtre gaussien $g_\sigma$ et le paramètre d’échelle σ. La définition est comme suit [7]:

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

Où

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

Les descripteurs sont utilisés en reconnaissance d’objets pour aider à identifier et classifier les objets dans une image ou une vidéo. Il existe plusieurs types de descripteurs qui peuvent être utilisés selon le type d’objet à identifier. Ils sont souvent utilisés avec d’autres techniques de reconnaissance d’objets comme les algorithmes de vision par ordinateurs ou les réseaux de neurones artificiels. Ils permettent de fournir une description précise et unique de l’objet, ce qui facilite son identification et sa classification.

SIFT est conçu pour être robuste aux distorsions d’image, aux changements de taille et aux distorsions de l’image [7]. Cela le rend particulièrement utile dans les applications où il est nécessaire de trouver des correspondances entre des images prises sous des angles ou avec des tailles différentes [8]. Il est utilisé dans de nombreuses applications, notamment dans la reconnaissance d’objets et de visages, la géolocalisation d’images, la création de cartes de profondeur et la reconstruction de scènes 3D [9].

Les points d’intérêts sont des points stables et caractéristiques dans une image, ils sont généralement choisis en fonction de leur singularité, de leur robustesse aux perturbations de l’image et de leur capacité à être décrits de manière unique, les points d’intérêts sont souvent

utilisés comme points de référence pour aligner et ajuster des images. Il existe plusieurs algorithmes pour détecter les points d'intérêts dans une image, dont SIFT [9].

Nous étudions SIFT sous Python implémenté par la bibliothèque OpenCV, l'algorithme fonctionne avec une image convertie en niveaux de gris (car SIFT ne peut traiter que les images de cette teinte), on crée ensuite l'objet SIFT qui sera utilisé pour calculer les points d'intérêts et les descripteurs. Les descripteurs sont sous forme d'un tableau de 128 décimaux par point d'intérêt. Ces points d'intérêts sont ensuite affichés sur l'image. Grâce au tableau des descripteurs calculé par la fonction, il est alors possible de le comparer avec celui d'autres images pour trouver des points semblables [10].

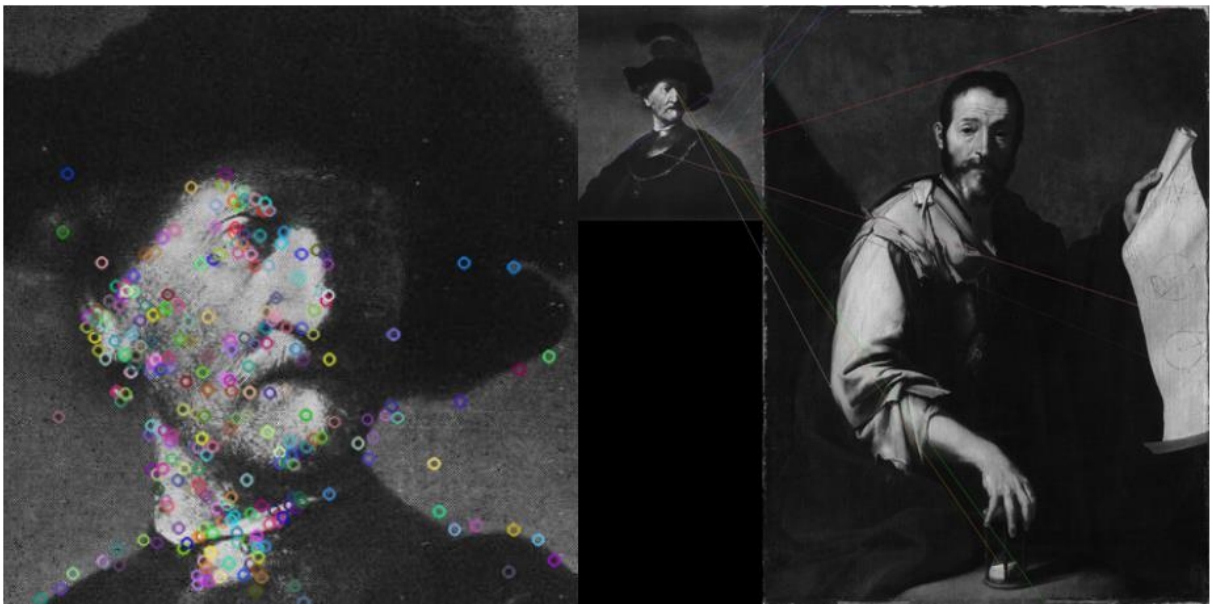

*Image 2.1 Exemple de points d'intérêts détectés et classés par SIFT, puis comparés avec une autre image par BFMatcher.*

À la suite de ces algorithmes, les descripteurs sont reliés par des traits pour illustrer la correspondance [10].

Pour comparer des images, le procédé est le suivant : une fois que l'on a des descripteurs pour les deux images à comparer, on peut utiliser un algorithme de correspondance de points clés comme BFMatcher afin de trouver ceux similaires aux deux images, c'est-à-dire en comparant les descripteurs entre les deux images et en cherchant des correspondances proches. Il existe

plusieurs façons de faire cela, en fonction des descripteurs et de la qualité des images (cf. Image 2.1).

BFMatcher, proposé par OpenCV, est une méthode de matching bruteforce, à savoir qu'il compare tous les descripteurs de l'image de référence avec tous les descripteurs à tester. Cela peut être coûteux en termes de temps de calcul pour des images de grandes tailles ou ayant une grande quantité de points clés. Cependant, il est généralement très précis, car il ne rate pas de correspondances qui pourraient être manquées par des méthodes plus rapides, mais qui sont moins précises [10].

BFMatcher va créer un objet de match, puis va prendre en entrée les descripteurs des deux images afin de les mettre en commun, il va ensuite trier les distances de ces descripteurs.

Pour limiter le nombre de points d'intérêts avec SIFT, nous pouvons limiter le nombre de points clés par images en spécifiant un nombre de points clés à détecter ou limiter la densité de points clés.

Le tri des points d'intérêts peut aider à sélectionner les points les plus importants et/ou les plus représentatifs. En utilisant l'algorithme SIFT, il est possible de trier les points clés en fonction de leur saillance en gardant les points clés les plus distinctifs. Cela peut aider à améliorer les performances de reconnaissances d'images en réduisant le nombre de points clés à traiter.

Il est également possible de trier les points d'intérêts en fonction de leur répartition spatiale dans l'image. Cela peut être utile pour sélectionner les points les plus représentatifs d'une zone donnée de l'image ou pour éviter les points trop proches les uns des autres qui pourraient avoir des caractéristiques similaires.

Au cours de nos expérimentations, nous étudierons divers programmes issus de l'algorithme SIFT notamment en utilisant FLANN (cf. Chapitre 2.3) qui est un algorithme de correspondances de points clés proposé par OpenCV basé sur des structures de données de type arbre pour accélérer la recherche des plus proches voisins.

## 2.3 Fast Library for Approximate Nearest Neighbours (FLANN)

Une des procédures les plus chronophages dans des espaces des hautes dimensions est la recherche du voisinage le plus proche [11]. FLANN (ou « Fast Library for Approximate Nearest Neighbors ») est une bibliothèque pour trouver efficacement les voisins les plus proches. La bibliothèque permet à l'utilisateur d'atteindre une plus haute performance en utilisant seulement un jeu de données et la précision désirée, paramétrée par l'utilisateur [12].

FLANN utilise des structures de données efficaces pour stocker les descripteurs de points clé de l'image de référence. Il peut aussi être utilisé pour gérer des jeux de données volumineux, en utilisant des techniques d'approximation pour trouver des correspondances [12].

*FLANN* prend en entrée les descripteurs de deux images et retourne les indices des descripteurs correspondants dans les deux images. Il permet de choisir différents types d'arbre de recherche pour trouver les correspondances et différents types de distances pour comparer les descripteurs [12].

# 3. Expérimentations

Ce chapitre vise à atteindre l'objectif de recherche défini en introduction. Il présente tout d'abord les fichiers écrits spécifiant les fonctions et bibliothèques utilisées. Dans le paragraphe 3.2 les résultats des expérimentations sont présentés.

## 3.1 Structure des expérimentations

Le fichier *sift_matcher.py* implémente l'algorithme *SIFT* pour trouver des correspondances et obtenir une matrice homographique qui décrit les transformations nécessaires pour aligner les deux images. Nous redimensionnons les images avant d'appliquer ces transformations. Ce programme utilise également l'algorithme *FlannBasedMatcher* pour trouver les correspondances les plus probables entre les points clés. La fonction a un temps d'exécution total et un maximum de quarante-huit fichiers à traiter.

Les premiers tests du programme s'avèrent très lent, pour nos quarante-huit images, on obtient vingt secondes pour les parcourir et les analyser. On s'interroge alors sur la possibilité de redimensionner l'image à nouveau pour obtenir une analyse plus rapide. On sait que, lorsque la

résolution de l'image analysée est trop faible, on trouve plus de correspondance, des faux-positifs. On se demande aussi dans quelle mesure, on peut réduire la taille d'une image en base de données pour s'assurer de la fiabilité du résultat.

On va examiner un autre programme : sift_stat_match_count.py qui utilise également *SIFT* et *FLANN* mais utilise en plus un seuil *match_count* pour décider si une image correspond à l'image de référence. Il lit les descripteurs contenus dans un fichier, transformé avec l'aide de la bibliothèque NumPy, pour chaque image, il exécute ensuite l'algorithme de matching et stocke les résultats dans un fichier csv. Nous faisons varier *match_count* et étudions son temps d'exécution total pour chaque valeur de *match_count*.

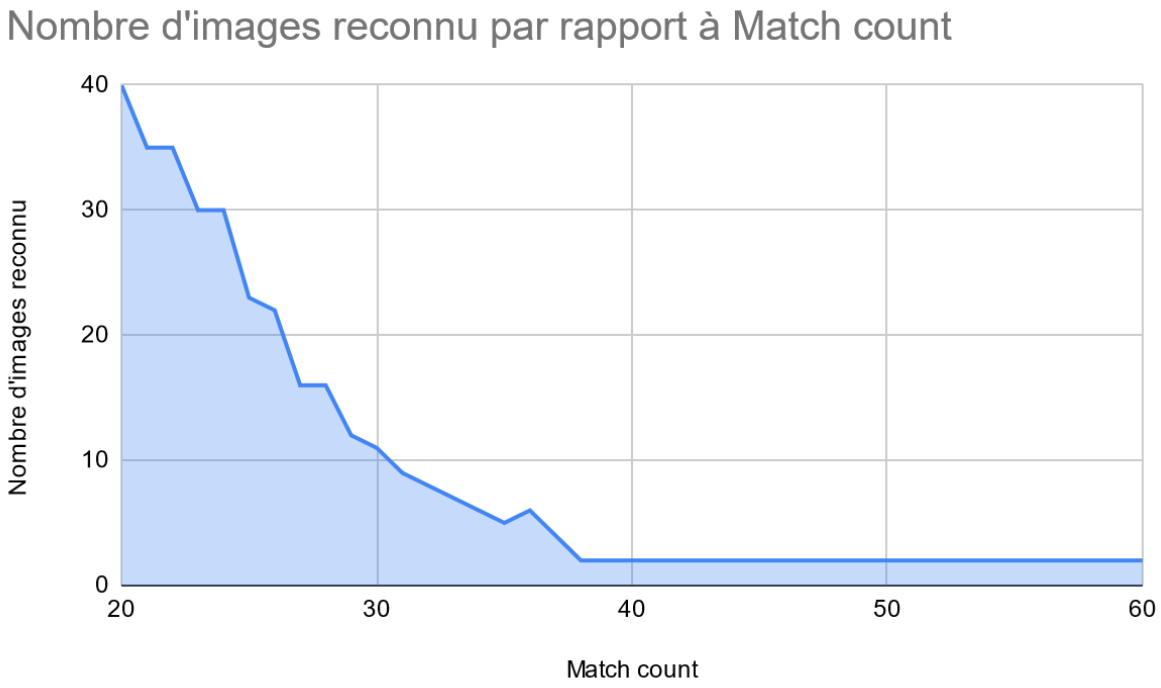


*Image 3.1 Résultats de multiples essayes pour fixer la valeur idéale du nombre de descripteurs*

Après de multiples essais, il semblerait que la valeur idéale du nombre de descripteurs soit aux alentours de 20.

Le programme suivant *sift_stat_check.py* analyse le temps d'exécution en fonction de la valeur de checks. Ici *checks* est un paramètre utilisé pour l'algorithme *FlannBasedMatcher*. Il détermine le nombre de points de recherches effectué lors de la recherche de correspondances entre les descripteurs. Plus le nombre de *checks* sera élevé, plus la recherche sera exhaustive,

ce qui augmentera le temps de calcul et permettra de maximiser les chances de trouver des correspondances.

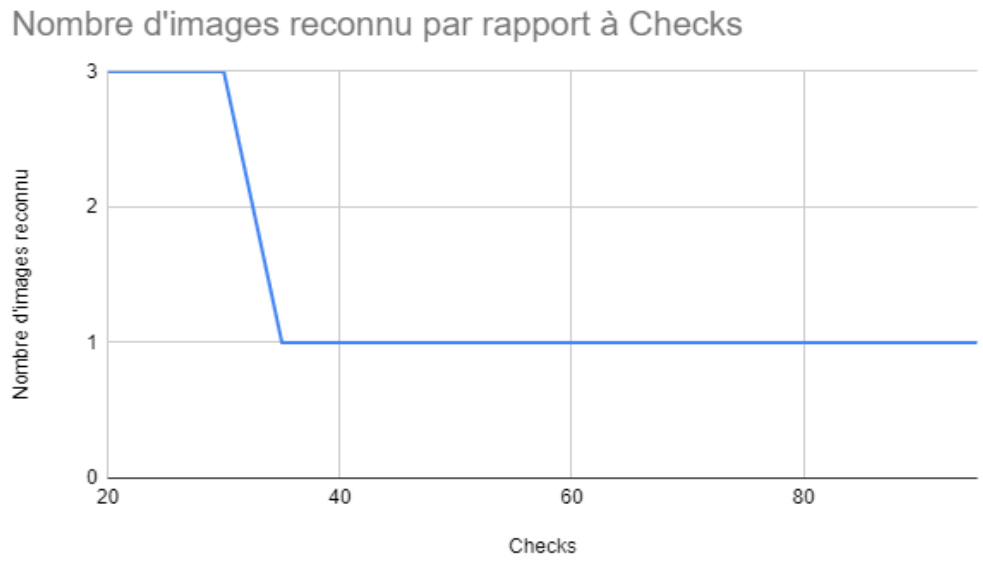


*Image 3.2 Résultats de multiples essayes pour fixer la valeur idéale du paramètre check*

Le paramètre *checks* spécifie le nombre de fois que les arbres de l'index doivent être parcourus récursivement. Des valeurs plus élevées donnent une meilleure précision, mais prennent aussi plus de temps. La valeur idéale semble tendre autour de 35 à 40.


*Image 3.3 Résultats de multiples essayes pour détecter la dégradation d'image possible*

Bien que l'algorithme fonctionne à un certain niveau de dégradation, il est important d'utiliser des images de qualité pour obtenir un résultat exploitable. Lors de la sélection de cette dernière, la résolution doit être suffisamment élevée pour permettre une détection précise des points clés et des descripteurs. Si les images sont trop petites, il peut être difficile de détecter des points clés distincts et de générer des descripteurs de qualité.

La luminosité et le contraste doivent être appropriés pour permettre une détection fiable des points clés. Les images trop sombres ou trop lumineuses peuvent rendre difficile la détection de ces derniers. On utilise aussi des images sans trop de rotation ou distorsion pour permettre une correspondance précise, les images avec des angles de prises de vues ou des distorsions importantes peuvent rendre la correspondance des points clés compliquée. Il est, par ailleurs, important d'utiliser des images avec une variabilité suffisante pour permettre une reconnaissance robuste.

## 3.2 Résultats et analyse des expérimentations

Dans cette étude, nous allons observer le résultat des expérimentations afin d'évaluer les performances de *match_count* et *checks* en utilisant des graphiques à trois dimensions. Nous avons fait varier le nombre d'images utilisées ainsi que le nombre de *checks* effectués. Ces diagrammes permettent de mieux visualiser les relations entre les différentes variables étudiées.

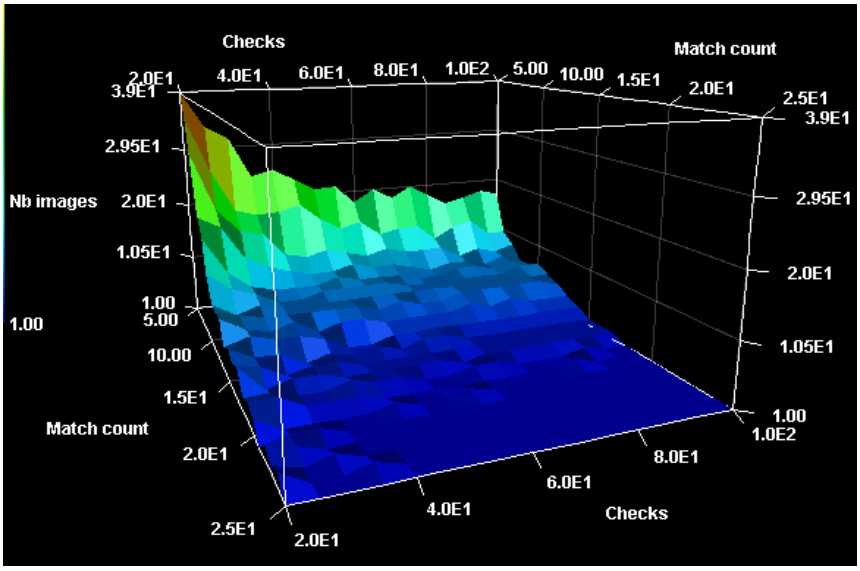


*Image 3.4 Résultats des expérimentations afin d'évaluer les performances de match_count et checks*

Les meilleurs paramètres pour *checks* et *match_count* sont 20 et 60, car ils correspondent à l'emplacement où il y a le plus souvent qu'une seule image reconnue. En calculant les données à nouveau, avec un intervalle moins grand, nous obtenons le graphique suivant :

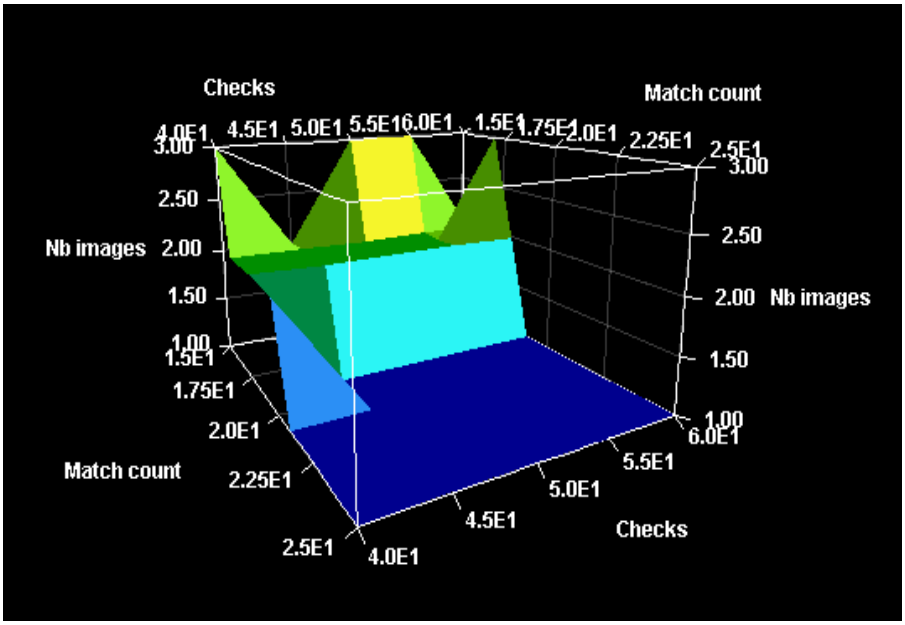


*Image 3.5 Résultats des expérimentations après avoir appliqué les paramètres checks et match_count optimisé*

Ce graphique nous permet d'évaluer les paramètres avec plus de précision. Soit 23 pour *match_count* et environ 50 pour *checks*. Cependant, avec une nouvelle image, nous obtenons :

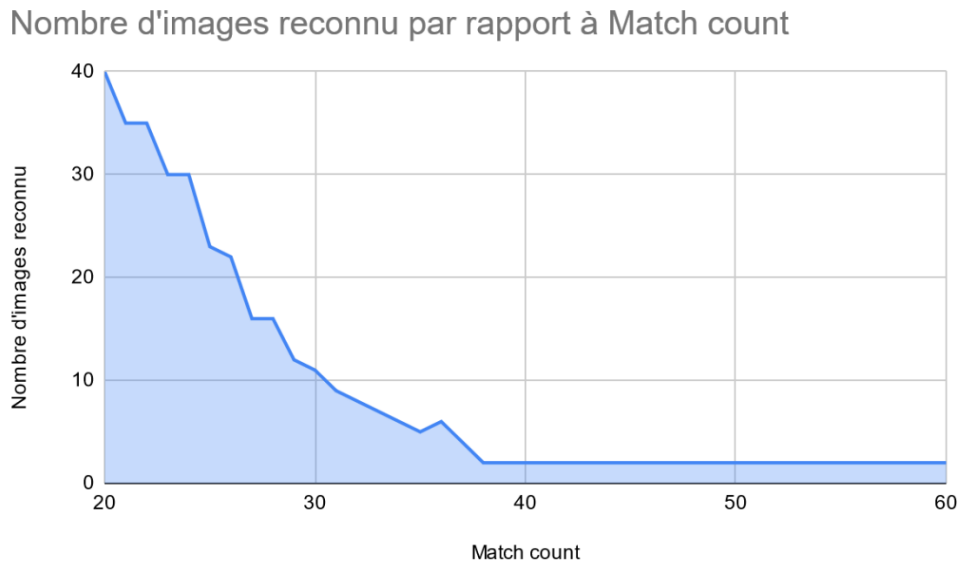


*Image 3.6 Nombre d'images reconnu par rapport à match_count*

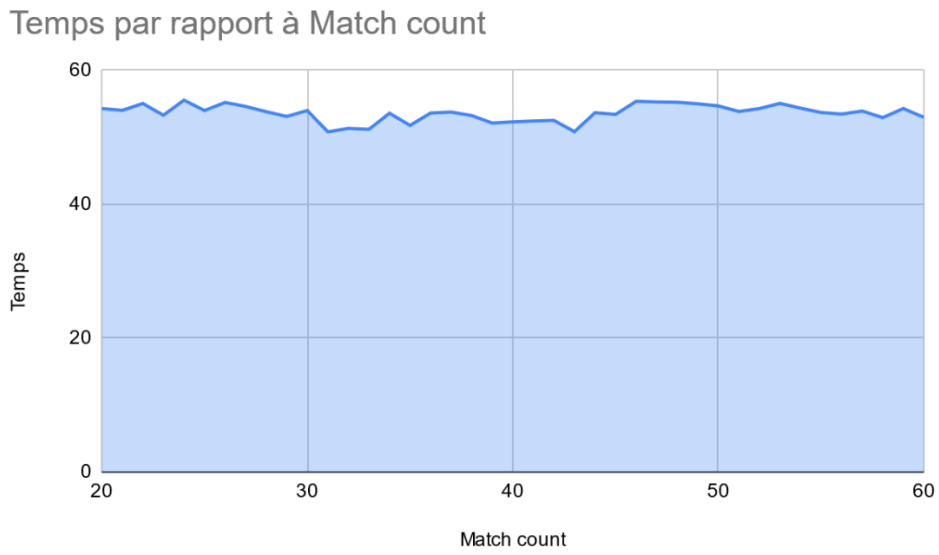


*Image 3.7 Temps nécessaire par rapport au paramètre match_count*

On se retrouve bloqué à deux images reconnues vers 40 de *match_count* avec 2 image reconnue, mais le temps ne varie pas ou varie ou peu. Il faut donc optimiser la valeur de *checks* pour trouver la bonne image parmi les deux trouvées.

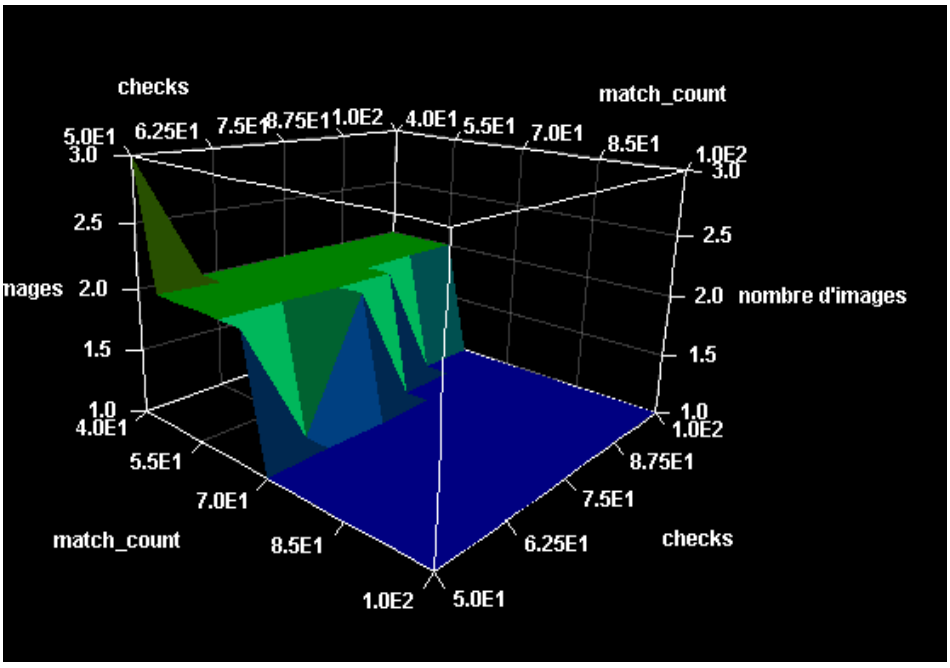


*Image 3.8 Résultats des expérimentations en variant match_count et checks*

En faisant varier *checks* et *match_count*, on parvient à trouver une valeur intéressante vers 60 avec un *checks* supérieur à 75 où seulement une seule image est trouvée.

Cependant, le temps augmente en fonction de *checks*, il est donc intéressant de trouver un optimal où sa valeur n'est pas trop élevée. Nous faisons alors varier *match_count* (en x) et checks (en z) ensemble pour étudier le temps d'exécution (en y) :

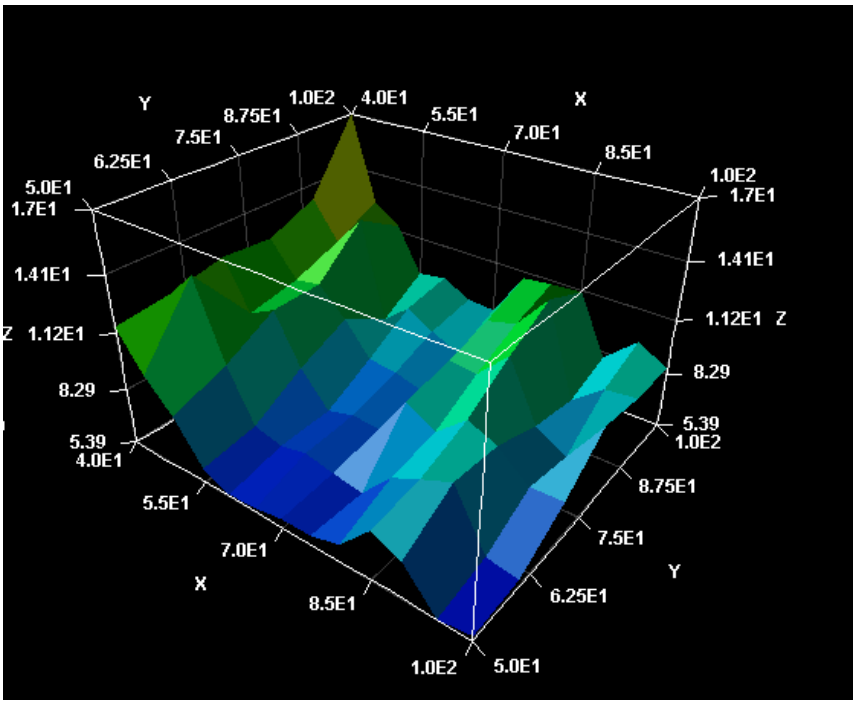


*Image 3.9 Résultats des expérimentations en variant match_count et checks*

Ici, nous observons un fort impact de *checks* sur le temps d'exécution. Nous avons alors décidé que si l'image figurait dans les cinq premières images des descripteurs les plus proches de ceux de notre image témoin, alors le test est validé. Le paramètre *match_count* est ainsi obsolète, n'ayant aucun impact sur le temps. C'est *checks* qui reste impactant (cf. graphique précédent).

En effet, ce dernier prend beaucoup de temps lors de l'exécution du programme. Nous nous sommes rendu compte que le *checks* pouvait être retiré et que cela accélérait le temps de la recherche tout en ne gênant pas la reconnaissance des images.

De plus, lorsque le paramètre *TREES* dans l'appel de la création de l'objet de matching de *FLANN* est bas, cela augmente la vitesse de *FLANN* :

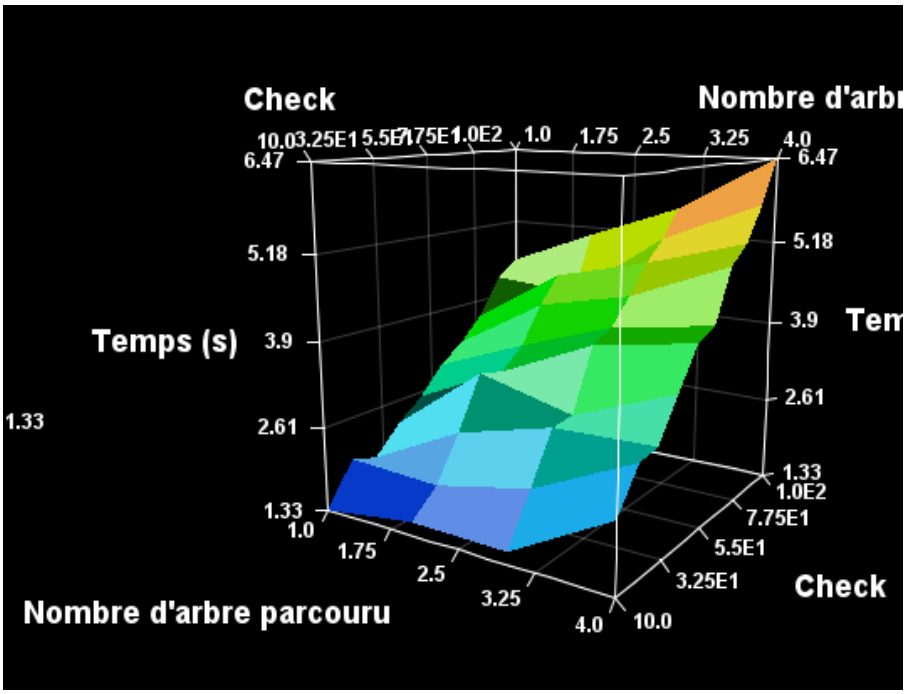


*Image 3.10 Temps passé en rapport avec checks et nombre d'arbre*

Encore une fois, nous voyons l'importance du paramètre *checks* sur le temps. Alors que le temps est beaucoup moins impacté par le paramètre *TREES*.

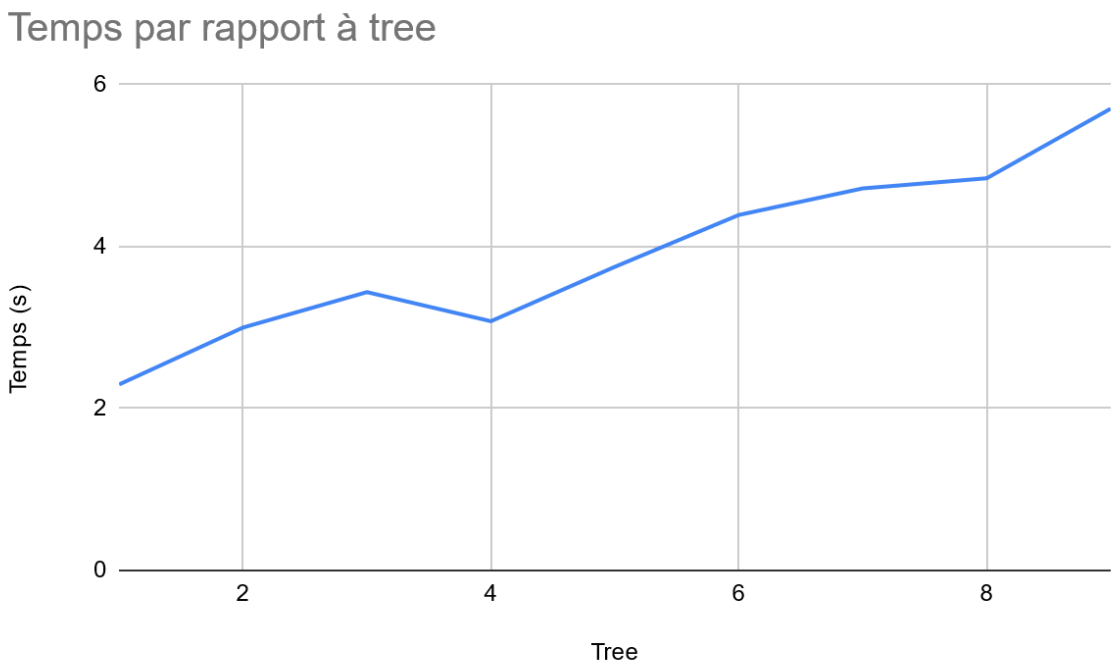


*Image 3.11 Temps passé par rapport à nombre de TREE parcouru*

Il y a une différence notable entre chaque incrémentation de *TREE*. Cependant, l'impact sur le nombre de matches n'est pas tant notable, il n'est pas intéressant dans notre cas de la mettre au-delà de 1 :

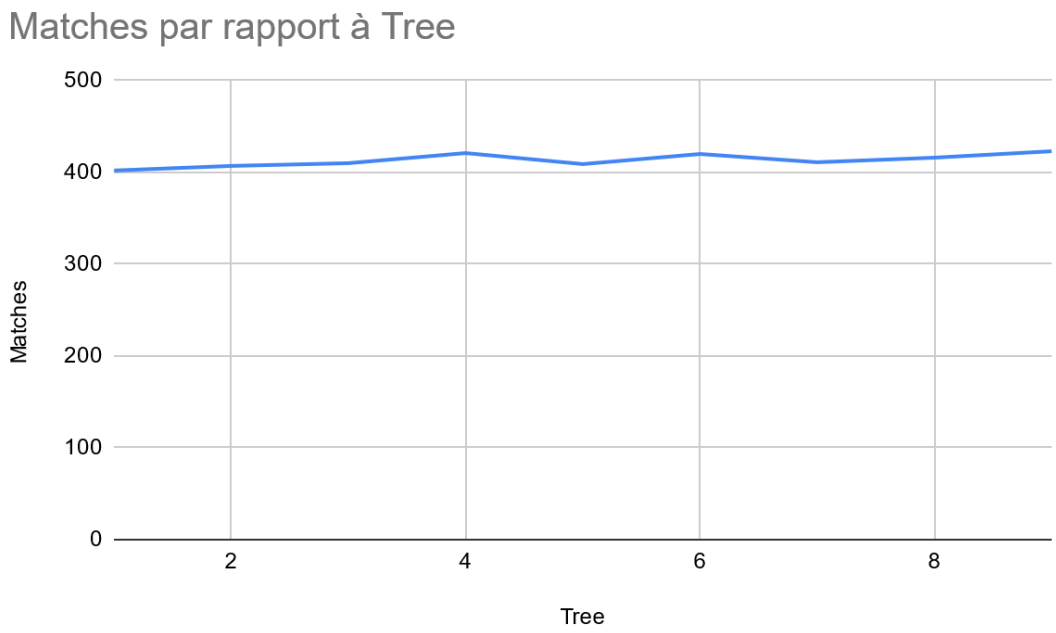


*Image 3.12 Matchs par rapport à nombre de TREE parcouru*

Par rapport à la résolution de l'image, ce n'est pas important de l'étudier, car nous avons un temps infini pour récupérer les descripteurs dans un fichier.

# 4. Discussion

Après l'étude des paramètres *match_count*, *checks* et la taille des images, nous avons pu constater que *match_count* était obsolète puisqu'on va utiliser les cinq premières images ayant le plus de descripteurs en communs. La taille d'image est, elle aussi, obsolète, car nous avons un temps infini pour trouver les descripteurs des images de la base.

D'une part, nous avons pu observer le nombre d'images reconnues en fonction du nombre de descripteurs et comprendre l'efficacité de chaque paramètre. Avec la première image témoin, *match_count* arrivait à faire correspondre au mieux les images à partir de vingt descripteurs et *checks* autour de trente-cinq descripteurs alors que la modification de résolution est devenue inintéressante.

D'autre part, nous avons pu évaluer qu'avec la seconde image témoin, nous avons eu deux images reconnues à quarante descripteurs et qu'en optimisant le paramètre checks pour reconnaître qu'une image, il est devenu inutile.

Au début, nous n'utilisions pas de fichier de descripteur, ce qui augmentait radicalement le temps de recherche, car nous les calculons durant l'exécution. Pour faire correspondre une image entre les quarante-huit de notre base, cela prenait vingt-deux secondes, ce qui est un temps beaucoup trop élevé pour un passage à l'échelle.

Nous avons, tout d'abord, effectué l'enregistrement dans un fichier *JSON*, la lecture et l'écriture étaient très longues : vingt secondes pour créer le fichier et le remplir et trois secondes pour le lire.

Nous avons ensuite sauvegardé les données dans un fichier binaire *NumPy* (.npy), permettant d'enregistrer un tableau ou un dictionnaire dans ce type de fichier. Ce dernier est plus rapide à lire grâce à la conversion en binaire.

Le temps d'écriture est le même, soit vingt secondes, mais est entre une demi-seconde et une seconde pour la lecture.

Pour résumer, il est important de choisir un bon système d'enregistrement de descripteurs, il faut également évaluer combien d'images, nous pouvons analyser en deux secondes, pour savoir si le passage à l'échelle d'un million d'images à analyser est possible (voir fichier *enregistrement_descripteurs.py*).

Par rapport à nos paramètres initiaux : *checks* valant 50 et cinq arbres parcourus, nous pouvons analyser vingt-deux images en deux secondes.

Alors que selon nos paramètres optimaux, pas de valeur pour le paramètre *checks* et qu'un seul arbre parcouru, nous pouvons analyser quarante images en deux secondes.

## 5. Conclusion

L'utilisation des différents programmes (*sift_matcher.py, sift_stat_match_count.py, sift_stat_check.py*) qui utilisent l'algorithme *SIFT*, vont permettre de trouver des correspondances entre les images en se servant de l'algorithme de matching *FLANN* pour trouver les correspondances les plus probables entre les descripteurs.

D'une manière générale, pour accélérer la lecture et l'écriture du fichier contenant les descripteurs de nos images, nous avons trouvé que la méthode la plus efficace était de créer un fichier de descripteurs binaire. Cela se fait en utilisant la bibliothèque *NumPy* qui permet de convertir un tableau en données binaires.

Nous avons également évalué que le paramètre *checks* peut être ignoré dans de futurs travaux de recherche, ce qui raccourcit encore une fois le temps d'exécution.

En addition, nous avons trouvé qu'une augmentation du paramètre *nombres d'arbres parcourus* n'est pas rentable par rapport au nombre de matchs trouvé pour un temps d'exécution optimal.

Pour la question du passage à l'échelle, nous avons analysé les différentes combinaisons de paramètres qui ont permis de diminuer considérablement le temps de recherche d'une image. Cependant, ce temps reste important et dans l'état des choses, le passage à l'échelle est impossible.

# References


1. Basso: Augmented Artwork Analysis. Computer-aided interpretation device for art images (2020)

2. Radau, B., Haunhorst, R.: Biografie Nam June Paik. LeMO-Biografien, Lebendiges Museum Online. https://www.hdg.de/lemo/biografie/nam-june-paik.html

3. Palais Garnier: Exposition, Juin 2021

4. Yacine Aït Kaci: Oeuvres en interaction avec celle-ci via ordinateur ou casque de réalité virtuelle, Galérie 1111 de Lyon

5. Bleser, G., Gava, C.: 2D projective transformations (homographies) (2011)

6. Sonfack, Koreimy, Arama: Reconnaissance d'objets avec le descripteur SIFT, Institut de la Francophonie pour l'innovation. https://www.researchgate.net/publication/331247897_Reconnaissance_d'objets_avec_le_descripteur_SIFT (2018)

7. Boyer: Informatique Visuelle. Détection de points d'intérêts et mise en correspondance, Université Grenoble Alpes. https://morpheo.inrialpes.fr/people/Boyer/Teaching/M2PGI/c4.pdf

8. Hua, S., Chen, G., Wei, H., Jiang, Q.: Similarity measure for image resizing using SIFT feature. J Image Video Proc **2012**(1), 1–11 (2012). doi: 10.1186/1687-5281-2012-6

9. Guo, F., Yang, J., Chen, Y., Yao, B.: Research on image detection and matching based on SIFT features. In: Guo, F. (ed.) 2018 3rd International Conference on Control and Robotics Engineering. ICCRE 2018 : April 20-23, 2018, Nagoya, Japan. 2018 3rd International Conference on Control and Robotics Engineering (ICCRE), Nagoya, 4/20/2018 - 4/23/2018, pp. 130–134. IEEE, Piscataway, NJ (2018). doi: 10.1109/ICCRE.2018.8376448

10. OpenCV. Open Source Computer Vision (2022)

11. Lowe, D.: Distinctive Image Features from Scale-Invariant Keypoints

12. Muja, M., Lowe, D.: Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, University of British Columbia (2009)