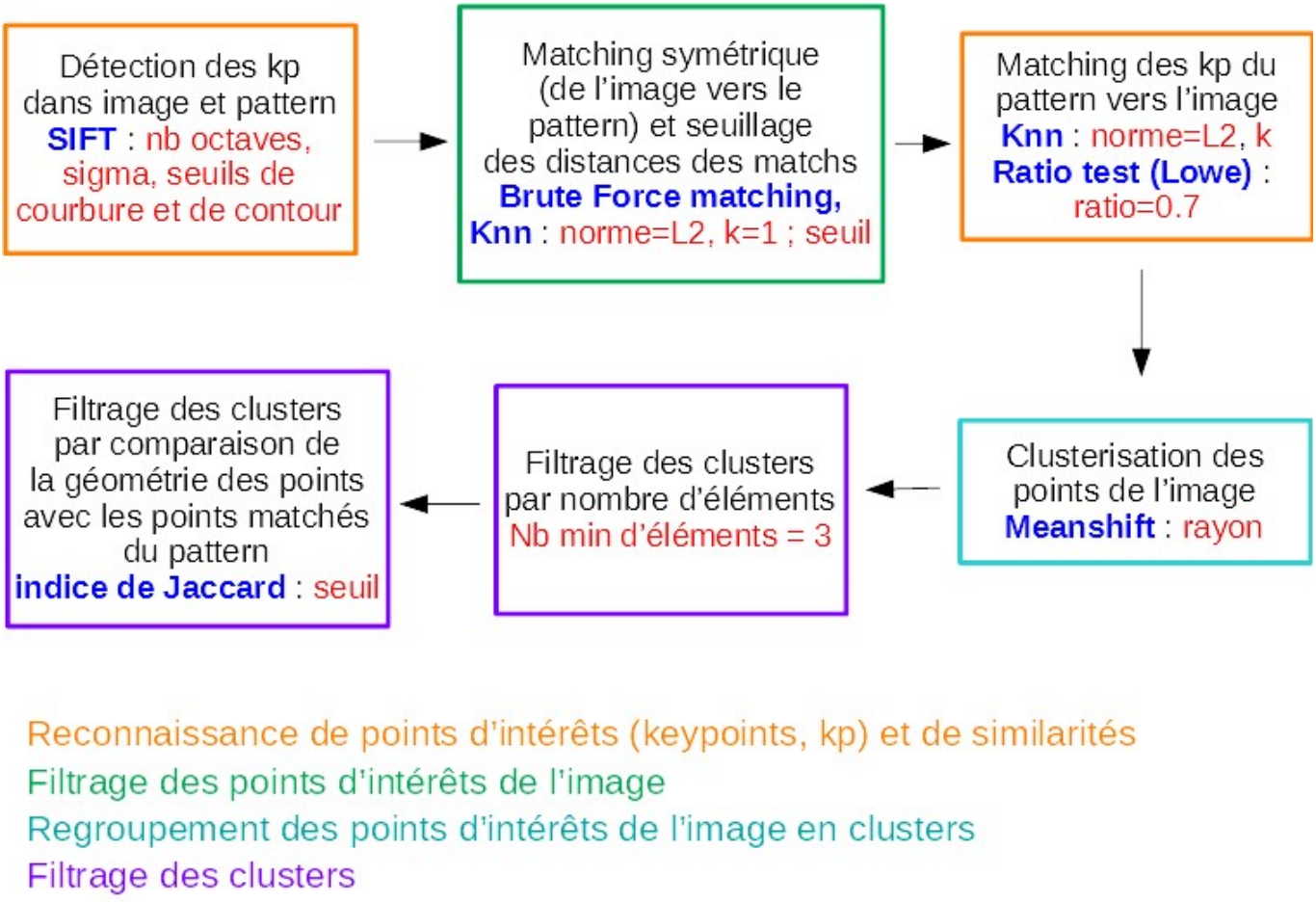
FEATURE MATCHING AVEC SIFT

Introduction : Déroulement de l'algorithme

On dispose de deux images : un pattern issu d'une base de données et une image dans laquelle on veut savoir si l'on peut trouver ce pattern, et si oui à quel endroit. Pour cela, on utilise l'algorithme SIFT afin d'obtenir 2 ensembles de points d'intérêts (ceux du pattern et ceux de l'image). Après avoir enlevé les points d'intérêts qui n'ont rien à voir avec le pattern, on match les points d'intérêt du pattern avec ceux de l'image par un calcul des plus proches voisins. Enfin, on regroupe avec meanshift les points que l'on a réussi à apparier puis on filtre les clusters obtenus pour ne garder que ceux qui ressemblent le plus au pattern.

Le schéma ci-dessous représente le déroulement de l'algorithme de détection, avec en bleu les algorithmes utilisés et en rouge les paramètres (dont certains peuvent être fixés). Chaque étape est détaillée dans la suite de ce rapport.



Reconnaissance de points d'intérêts (keypoints, kp) et de similarités
Filtrage des points d'intérêts de l'image
Regroupement des points d'intérêts de l'image en clusters
Filtrage des clusters

Reconnaissance de points d'intérêts

On détecte les points d'intérêts grâce à l'algorithme SIFT qui nous donne une liste de points auxquels sont associées des informations caractéristiques. Cet algorithme prend plusieurs paramètres : le nombre

d'octaves, \sigma, un seuil de courbure et un de contraste.

Pour ne pas être affecté par des problèmes de redimensionnement, SIFT calcule les points d'intérêts à plusieurs échelles de l'image. Le nombre d'octaves influe sur le nombre d'échelles prises en comptes. \sigma a un impact sur la détection des points d'intérêts et la robustesse de ces points vis à vis d'un changement de point de vue. Enfin, les seuils de courbure et de contraste permettent de filtrer les points d'intérêts trouvés. Ils permettent de filtrer par exemple les coins, où la courbure est la plus élevée, qui ne sont pas des points robustes aux redimensionnements.

Dans notre base de données, les images sont très petites par rapport à l'image où on cherche le pattern. Comme on veut suffisamment de points d'intérêts pour pouvoir comparer des similitudes, on choisit les paramètres en fonction du nombre de points qu'ils génèrent dans les images de la base de données. Ce sont donc des paramètres fixés en fonction de la base de données dont on dispose.

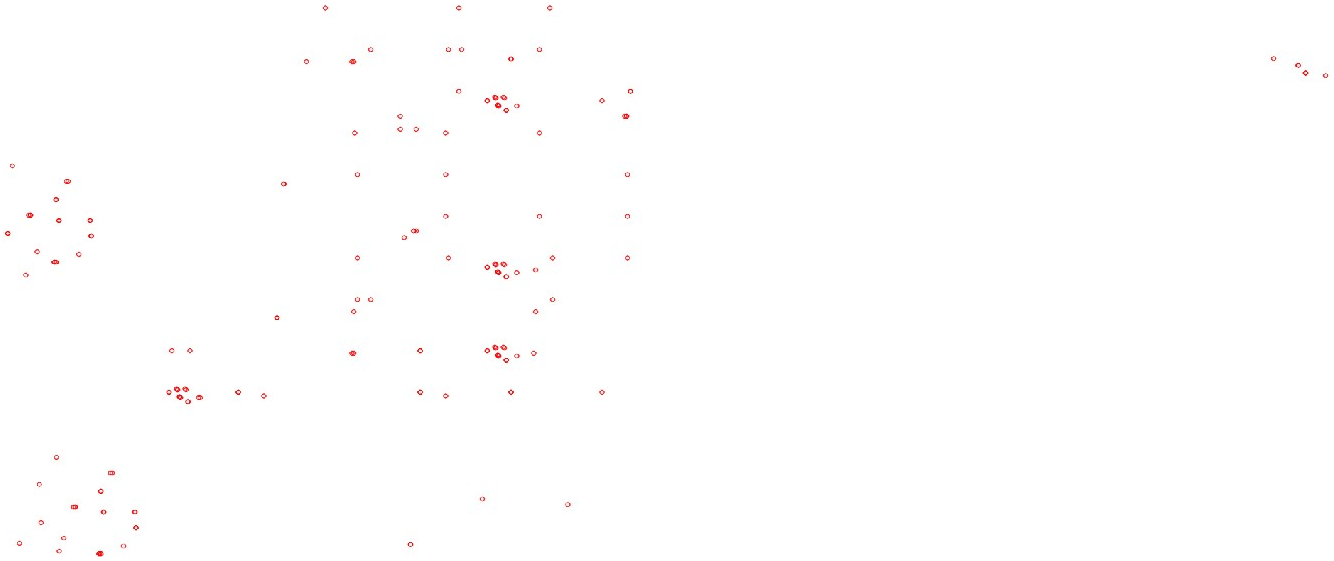
> Points d'intérêts (en rouge) reconnus par SIFT

Filtrage des points d'intérêt de l'image

Comme on peut le voir sur l'image ci-dessus, les paramètres qui génèrent beaucoup de points d'intérêts sur les petites images de la base de données en génèrent beaucoup plus sur les images plus grandes où l'on veut détecter le pattern. Cette étape de filtrage, qui fait passer le nombre de points d'intérêts de plus de 6000 à environ 500, sert entre autres à améliorer les performances aux étapes qui suivront.

On match les points d'intérêts de l'image avec ceux du pattern grâce à l'algorithme des k plus proches voisins, avec k=1. On obtient une liste de distances entre chaque point de l'image et son point le plus similaire dans le pattern. Cette distance est calculée comme une norme L2 entre les descripteurs de chaque point (vecteurs de dimension 128). On filtre ensuite les points de l'image en fonction de leur distance, grâce à un seuil à ne pas

dépasser. Il nous reste alors une liste de points de l'image similaires à un point du pattern.



> Points d'intérêts après le filtrage

Détection des similarités

Maintenant que l'on a dans l'image uniquement des points proches du pattern, on peut chercher des similarités. A l'aide de l'algorithme du brute force matcher, qui parcourt tous les points du pattern et tous les points de l'image pour trouver les k plus proches voisins de chaque point du pattern, on obtient une liste de points appariés. Pour chaque appariement, comme précédemment, on a une mesure de distance entre ces deux points.

Cette fois ci, on choisit k > 1 car, si l'image comporte plusieurs fois le pattern que l'on veut trouver, on veut pouvoir tous les détecter. En choisissant par exemple k = 10, on pourra trouver au maximum 10 fois le pattern dans l'image.

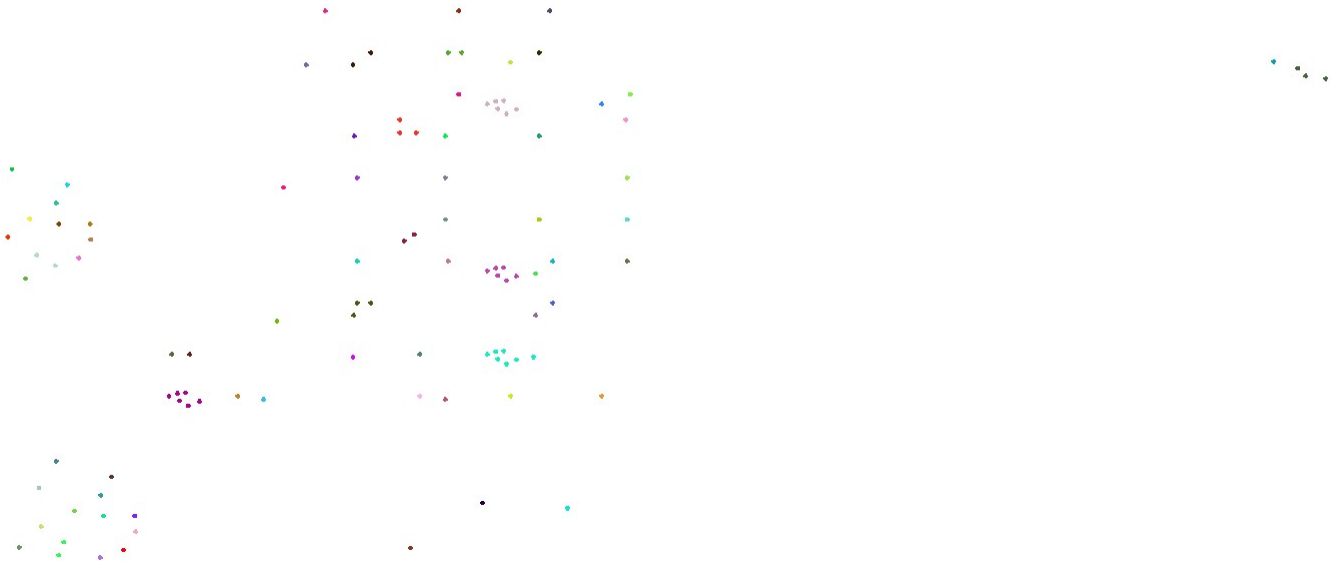
On a donc, pour chaque point du pattern, une liste de k points de l'image appariés triés par ordre croissant de distance. On applique le ratio test de Lowe pour enlevers les points "voisins" distants du pattern. Par exemple, si parmi les 10 voisins trouvés, les 3 premiers ont une faible distance mais que le 4e a une distance trop grande par rapport au 3e (trop grande en fonction du ratio donné en paramètre), alors on supprime les 7 derniers voisins.

> Matching des points d'intérêt du pattern avec ceux de l'image

Partitionnement des points d'intérêt de l'image

Après avoir récupéré les points d'intérêt de l'image qui sont appariés à un point du pattern, et qui ont passé le ratio test de Lowe, on applique l'algorithme de mean shift sur ces points. Cet algorithme nous donne une partition de nos points. Si l'on choisit le bon paramètre de rayon, chaque cluster peut correspondre à un pattern et on n'aura pas plusieurs patterns au sein du même cluster.

Comme cela a été expliqué dans le rapport sur la fenêtre glissante, dans le jeu où l'on applique cet algorithme de détection, les éléments sont agencés suivant une grille. En connaissant les dimensions d'une case de cette grille, on peut donc choisir le rayon de sorte que l'on obtienne des clusters de la taille d'une case.



> Tous les clusters trouvés par mean shift

Filtrage des clusters par nombre d'éléments

Cette étape n'est pas très compliquée. On parcourt tous les clusters et on élimine ceux qui n'ont pas assez d'éléments. Cela permet d'éliminer les clusters associés à des points isolés, et c'est utile pour l'étape de filtrage suivante. En effet, par la suite on voudra filtrer les clusters en fonction de leur géométrie et il nous faudra alors au moins 2 points pour déterminer la transformation (rotation et homothétie) plus au moins un 3e pour vérifier que la géométrie est correcte. On peut donc fixer le nombre minimum d'éléments à 3.

Il faut cependant faire attention aux doublons lorsque l'on compte le nombre d'éléments. En effet, comme l'algorithme SIFT détecte des points d'intérêts sur plusieurs échelles de l'image, la plupart des points trouvés sont en double. A cause de l'imprécision de ces points (coordonnées flottantes) on ne peut pas tester naïvement l'égalité. On applique donc mean shift dans chaque cluster avec un rayon très faible, puis on calcule le nombre de "sous-cluster" trouvé qui correspond au nombre d'éléments sans doublon.

> Clusters après le filtrage par nombre d'éléments

Filtrage des clusters par comparaison de la géométrie

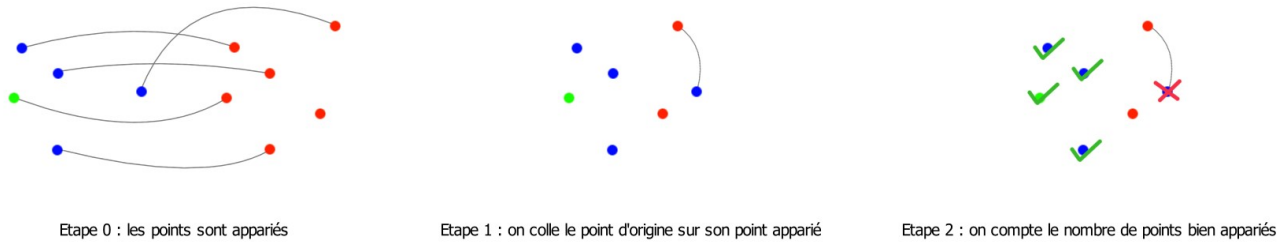
Pour chaque cluster, on définit une transformation affine qui permet de comparer la géométrie du cluster avec la géométrie des points appariés dans le pattern.

On commence par calculer l'échelle entre le cluster et le pattern. Pour cela, on mesure chaque distance entre 2 points du cluster et la distance correspondante des 2 points appariés, puis on fait le ratio de ces deux distances. On trie ensuite la liste des ratios ainsi obtenue et on en prend l'élément médian. On considère cet élément médian comme le ratio entre le pattern et le cluster. Le fait de prendre la médiane des ratios permet d'éviter quelques erreurs. En effet, si au sein du cluster quelques points correspondent mal avec le pattern, mais qu'ils sont minoritaires, alors on aura tout de même une bonne estimation de l'échelle. On pourrait faire de même pour mesurer l'orientation du cluster par rapport au pattern, mais dans le cas étudié, on sait qu'il n'y a pas de rotation.

```
listeRatio := []

Pour chaque (c1, c2) dans l'ensemble des points du cluster
    p1 := match(c1)     // le point apparié à c1
    p2 := match(c2)
    distanceCluster := distance(c1, c2)
    distancePattern := distance(p1, p2)

    listeRatio.push(distanceCluster / distancePattern)

sort(listeRatio)
```

```
return median(listeRation)
```

On peut maintenant calculer la transformation à appliquer à chaque point du cluster. Pour cela, il suffit de définir chaque point du cluster relativement à un point d'origine du cluster (choisit arbitrairement). Il suffit ensuite d'appliquer l'homothétie (et la rotation) à tous les autres points du cluster. On peut alors comparer les points du cluster avec les points du pattern. On déplace le cluster sur le pattern de sorte à confondre le point d'origine du cluster avec son point apparié dans le pattern. On regarde ensuite si chaque point transformé du cluster est proche de son point apparié.



> Schéma de la comparaison des géométries du cluster et du pattern (cluster en bleu, pattern en rouge, point d'origine du cluster en vert)

On mesure le score de correspondance géométrique en fonction du nombre de points correctement appariés (la distance entre l'image du point par la transformation affine et le point du pattern est suffisamment petite), le nombre de points dans chaque ensemble (le cluster et le pattern) et le nombre de points dans l'union de ces deux ensembles (en considérant que 2 points correctement appariés ne font qu'un seul et même point). Avec ces informations, on peut calculer le score de Sorensen-Dice ou l'indice de Jaccard. Contrairement à l'indice de Jaccard, le score de Sorensen-Dice ne pénalisera pas ici les erreurs d'appariement étant donné que le cardinal des deux ensembles est égal.

Une fois que l'on a récupéré les scores de correspondance géométrique de tous les clusters, on filtre les clusters en ne conservant que ceux qui dépassent un score minimal.

> Clusters après le filtrage par correspondance géométrique

Résumé des paramètres

SIFT : nombre d'octaves, sigma, seuil de courbure et de contraste

Matching symétrique : seuil, norme (L2 par défaut)

Matching : k (nombre de voisins), norme (L2 par défaut), ratio (0.7 par défaut)

Mean shift : rayon (déterminé par la taille d'une tuile)

Filtrage par nombre d'éléments : nombre minimum d'éléments (3 par défaut)

Filtrage par correspondance géométrique : seuil