RAPPORT PROJET CSC_51073_EP

Solveur de puzzle automatique

4 septembre 2025

Martin Drieux, Adrien Goldszal

TABLE DES MATIÈRES

1 INTRODUCTION

1.1 Enjeu & Motivation du projet

L'enjeu de la résolution de puzzles est un sujet qui occupe la communauté scientifique travaillant sur la vision par ordinateur depuis des dizaines d'années. En effet, ce 'problème', et les méthodes employées pour le résoudre, sont souvent présentées comme première étape d'une extension possible vers des sujets plus sérieux, comme la reconstruction d'artefacts archéologiques par exemple.

C'est une approche plus ludique qui est à l'origine de ce projet. Voyant la possibilité d'utiliser de la vision par ordinateur pour aider à la résolution des puzzles, ce projet se propose de développer une méthode de résolution de puzzle 'automatique', en direct, observant les pièces avec l'utilisateur et lui indiquant où les placer.

C'est donc un projet mettant en oeuvre de nombreux outils d'analyse d'image et qui cherche une certaine forme de robustesse nécessaire de par son déroulement en direct.

1.2 Etat de l'art

La résolution numérique de puzzles a été abordée de nombreuses fois dans la littérature scientifique, et peut se décomposer aujourd'hui en deux grandes approches :

- Une approche reposant sur la forme des pièces
- Une approche utilisant le contenu des pièces et des outils de feature detection

H. Freeman et L. Gardner [1] furent les premiers à se pencher sur le sujet, pionniers de la résolution algorithmique par les formes. Leur méthode, améliorée depuis et représentant la majorité du travail sur le sujet, a montré ses preuves sur des puzzles de plus de 100 pièces. Ces méthodes algorithmiques nécessitent cependant d'avoir l'ensemble des pièces en entrée dès le début bien qu'elles ne demandent de connaitre le puzzle final. Elles

obtiennent de bons résultats. [2].

Plus récemment, certains, dont des étudiants de Stanford [3], se sont penchés sur de la feature detection et du matching entre les pièces et l'image finale du puzzle. Un intérêt était notamment porté sur la possibilité de résolution rapide du puzzle avec une application web pour aider l'utilisateur à la résolution. Cependant, cette approche a montré des résultats assez mitigés dès que le nombre de pièces dépassait la vingtaine.

Notre méthode s'inspire des travaux de [3], et se propose de fonctionner itérativement en direct, permettant d'accompagner l'utilisateur à la résolution du puzzle. Cette approche, accompagnée d'une meilleure détection de features, permet une résolution plus robuste et avec un nombre de pièces plus important, dépassant tous les benchmarks.

2 MÉTHODE

2.1 Setup

La fonctionnement global du solveur de puzzle est le suivant :

1. Un téléphone, attaché à une potence, permet d'observer en direct un certain nombre de pièces (environ 10 à 15) sur un fond blanc. Ce téléphone est connecté à l'ordinateur par une application, pour récupérer le feed de la caméra. En outre, l'image globale du puzzle (sur la boite) est donnée en entrée.
2. Une image est prise par la caméra à un intervalle régulier donné et traité par l'algorithme.
3. Sur l'écran de l'ordinateur, une interface affiche la vidéo du téléphone, ainsi que le puzzle en construction, indiquant à chaque fois la pièce à déplacer et son emplacement sur le puzzle final.

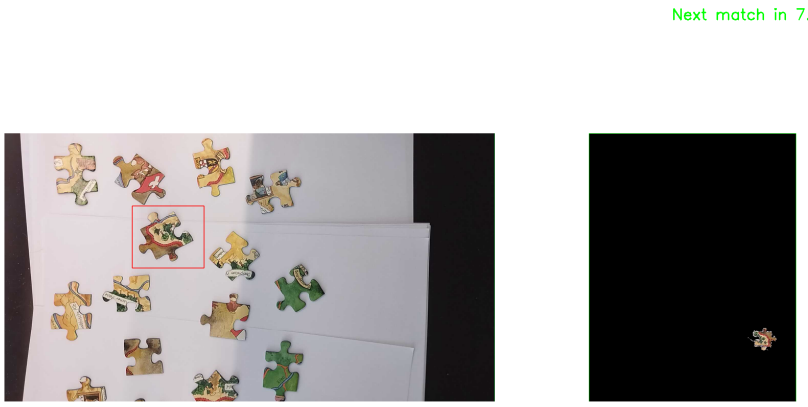


Figure 1 – Première pièce posée sur le puzzle

2.2 Détection des pièces

Nous avons effectué de nombreuses experiences afin d'affiner la pipeline permettant une détection de pièces de puzzle suffisament robuste pour notre cas d'étude. Si la luminosité est inégale, qu'il y a des reflets trop

importatns, ou que la caméra n'est pas parallèle avec les pièces, les résultats sont empirés. Les différents élements de la pipeline ont été ajustés pour limiter l'impact de ces paramètres.

Blur On applique tout d'abord un filtre gaussien pour supprimer le bruit de l'image initiale, qui empêcherait une bonne détection des pièces par la suite. Cette étape a un impact considérable (**Figure 1**)

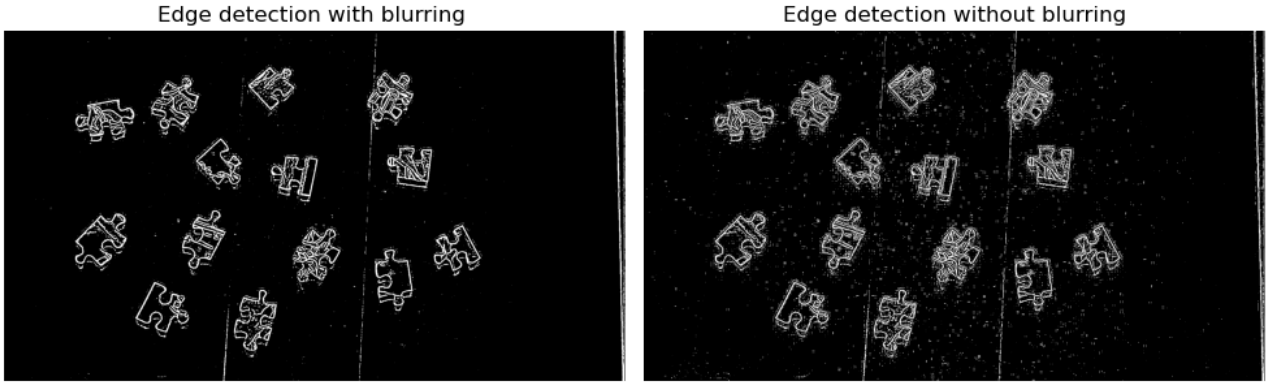


Figure 2 – Impact du blurring sur la edge detection

Première étape de détection Pour une première détection des pièces, nous avons testé deux approches : du thresholding, séparant les pièces du fond en trouvant un seuil qui les différencie, et de la edge detection. Le tresholding s'est avéré moins robuste que la edge detection dans certains cas où il y avait quelques reflets ou une luminosité un peu différente, comme l'en témoigne l'**annexe 1**.

Pour la edge detection, trois méthodes ont été testées, avec des résultats similaires.

— **Adaptive Thresholding** avec un seuil adapté à chaque pixel :

$$\text{seuil}(x, y) = \text{gaussian\_mean}(x, y, \text{window}) - C$$

Cette méthode fonctionne très bien pour détecter les contours des pièces. Il y a deux paramètres à préciser : la *BlockSize*, qui correspond au nombre de pixels considérés autour du pixel en cours, et C, une constante à soustraire du threshold calculé pour éviter de détecter les pixels de fond.

— **Canny** : La méthode de Canny est également efficace, mais elle nécessite de tester deux seuils en amont, ce qui n'est pas pratique pour notre cas.

— **Sobel** : L'étude des gradients fonctionne pour la majorité des cas, mais, lorsque certaines parties d'une pièce puzzle sont très claires et le plan de l'image légèrement incliné, des contours ne sont ainsi pas correctement détectés, comme le montre l'image ci-dessous.

Nous avons donc décidé de poursuivre avec le seuil adaptatif comme méthode de détection de contours.

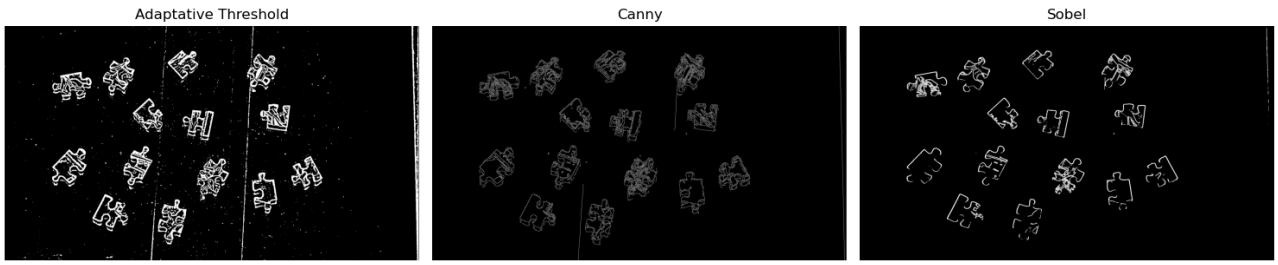


Figure 3 – Résultats de edge detection sur l'image initiale

Détection du contour Pour récupérer les pièces individuelles, nous détectons d'abord les contours (cv2.findContours), cette fonction repose sur *l'algorithme de tracé de contour* vu en cours. Nous remplissons les contours pour avoir des pièces blanches sur fond noir.

Nous avons observé que des contours étaient parfois détectés à l'intérieur des pièces de puzzle. Cela est dû aux imperfections de la détection des contours, qui a du mal à distinguer les véritables bords des motifs dessinés sur les pièces. Par conséquent, la détection des contours avec la fonction de la librairie OpenCV ne fonctionnait pas correctement. Ainsi, nous appliquons en amont des opérations morphologiques (`cv2.morph`) de fermeture pour réduire au maximum la taille des trous et assurer une meilleure détection des contours. Cette méthode fonctionne par dilatation puis érosion. Nous la réalisons avec des noyaux de différentes tailles, adaptés à l'échelle des images que nous traitons. Cette étape intermédiaire permet d'obtenir des résultats bien meilleurs, comme le montre la **Figure 3** ci-dessous.

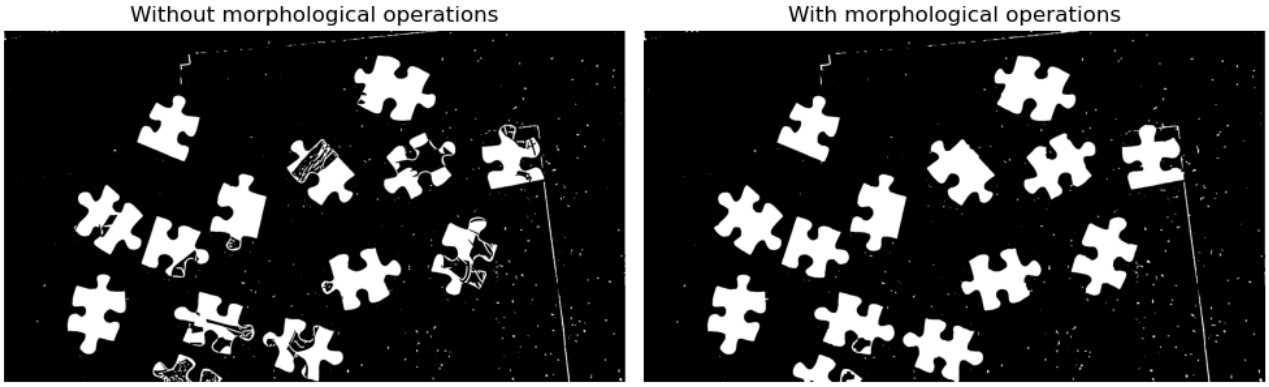


Figure 4 – Importance de la fermeture morphologique pour la détection des contours.

Extraction des pièces Une fois l'image des contours obtenue, nous extrayons les pièces avec la méthode des composantes connexes d'OpenCV (`cv2.connectedComponentsWithStats`). Cet algorithme détecte les pixels blancs adjacents de proche en proche. Les pièces sont ensuite filtrées pour éliminer les très petites composantes correspondant à du bruit détecté involontairement. On obtient ainsi une liste de pièces avec un masque associé représentant les pixels exacts de chaque pièce sur un fond blanc. Ce masque nous sera utile dans la partie suivante.

Approche alternative : YOLOv5 Cette méthode demande d'ajuster plusieurs paramètres et est sensible aux ombres ainsi qu'aux pièces proches les unes des autres. Nous avons donc également testé une méthode reposant sur le fine-tuning d'un modèle préentrainté : yolov5. Grâce à un dataset contenant des pièces de puzzle annotée (trouvé en ligne), nous avons pu entraîner le modèle et obtenir des bounding boxes de très bonne qualité. Il aurait également été possible d'extraire les contours en entraînant un modèle, mais nous aurions du produire les données d'entraînement nous même. Nous avons finalement choisi de conserver l'approche ci-dessus, la jugeant plus simple et intéressante.

2.3 Feature detection & Matching

Feature detection L'image du puzzle total étant donnée en entrée, les features du puzzle complet sont analysées dès le début. Il suffit d'analyser les pièces. Plusieurs détecteurs ont été testés, ORB, SIFT et AKAZE, et SIFT s'est avéré le plus performant de manière générale.

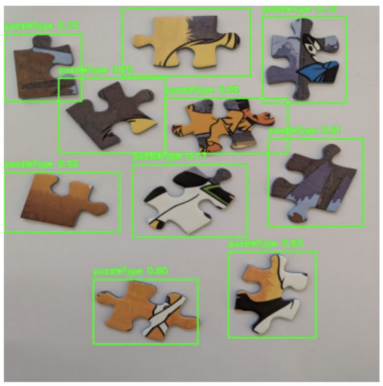


Figure 5 – Bounding boxes identifiées par le modèle yolov5 finetuné

Nous réutilisons pour cela le masque obtenu précédemment, qui ne capture cependant pas parfaitement le contour de la pièce, en particulier à cause d'une résolution d'image limitée et d'un parallélisme pas tout à fait respecté entre le téléphone et les pièces de puzzle. Nous avions donc des problèmes de détection de features sur les bords de la pièce, parasites, qui polluaient le feature matching par la suite. Ainsi, en appliquant une érosion sur le masque et en filtrant les points clés, nous parvenons à supprimer les bords.

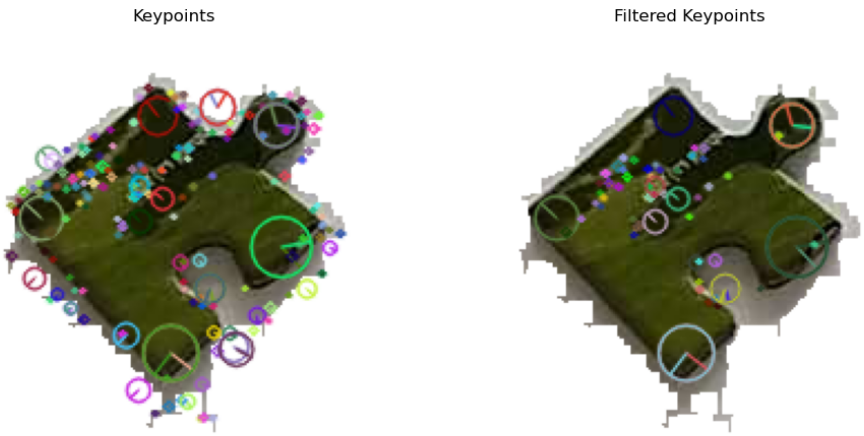


Figure 6 – Impact of keypoint filtration on piece detection

Feature matching Nous utilisons ensuite BFMatcher pour faire un matching des keypoints de la pièce avec le puzzle. Comme la résolution de la pièce n'était pas toujours très bonne, certains matchs devaient être filtrés. Nous gardons les 50 meilleurs matchs puis nous calculons la plus grande enveloppe de points tous distants les uns des autres d'au plus environ une taille de pièce pour ne garder que les matchs "cohérents".

Nous avions également testé le *Lowe's ratio test*, en analysant la proximité entre les deux meilleurs appariements proposés par `BFMatcher` pour le puzzle. L'objectif était de supprimer les appariements où ces deux points étaient trop proches, comme cela est parfois proposé dans la littérature. Toutefois, cette approche s'est révélée contre-productive ici. Certains points, comme ceux situés sur des branches de sapin par exemple, avaient leurs deux meilleurs appariements très proches, sans pour autant devoir être supprimés.

2.4 Homographie

Certaines pièces ne comportent que peu de features, nous avons donc adopté une stratégie demandant le moins de matchs possible pour le calcul des homographies. La caméra étant fixe et parallèle aux pièces, nous ne considérons que des similarités de la forme :

$$H = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

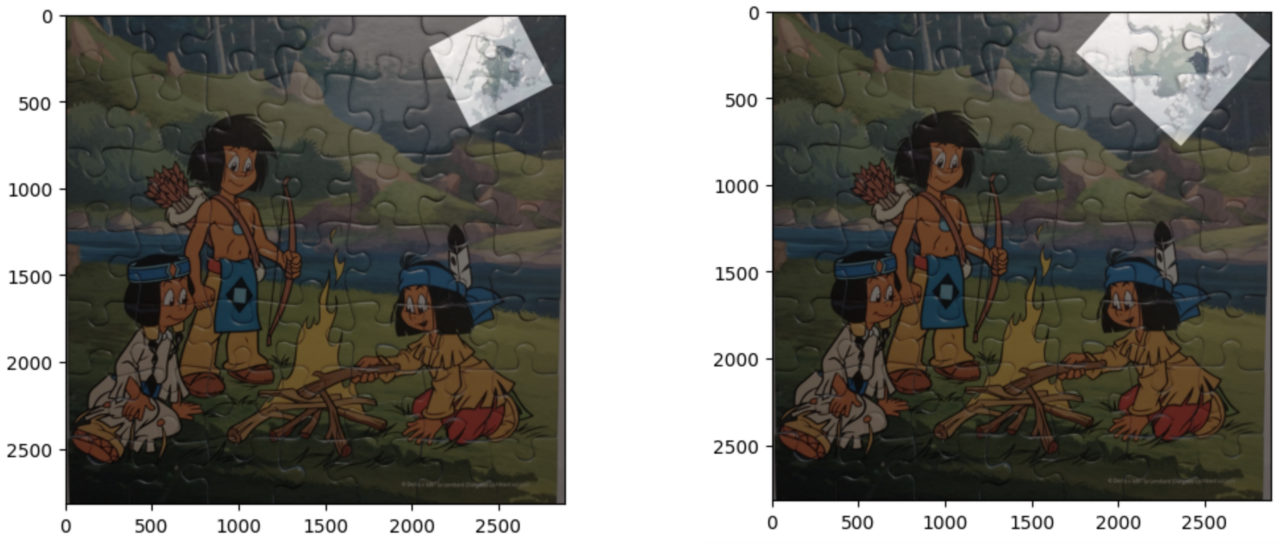
Ce qui correspond à une estimation de 4 paramètres, faisable a priori avec seulement 2 matchs. Nous avons essayé d'implémenter un algorithme de ce type directement (pseudo code détaillé dans l'annexe 2). Nous estimons les scaling grâce aux écarts types (

$$s \approx \frac{\sigma_{target}}{\sigma_{source}}$$

). Nous estimons ensuite la matrice de rotation (2×2) en réalisant une SVD sur la matrice de covariance (adaptée de l'algorithme de Kabsch), et nous en déduisons l'angle de rotation θ. Cependant cette méthode est très sensible au bruit et les méthodes de type RANSAC ne règlent pas vraiment ce problème. La méthode ne fonctionne que sur des images très propres et est inutilisable en pratique.

Nous avons donc adopté une autre approche : nous commençons par calculer les matchs de bonne qualité, les utilisons pour estimer le scaling "s", puis calculons directement des transformations rigides en conservant ce facteur de scaling. La première similarité est calculée avec la fonction cv2.estimateAffinePartial2D (qui requiert au moins 3 matchs). Les transformations rigides ne sont pas disponibles dans `cv2`, nous avons donc recodé l'algorithme correspondant, en utilisant encore une fois une méthode proche de l'algorithme de Kabsch (qui est plus robuste cette fois grâce à une meilleure estimation du facteur de mise à l'échelle). Nous avons également recodé la méthode RANSAC pour fiabiliser les résultats de cette transformation solide.

Cette méthode permet non seulement de calculer les transformations avec moins d'appariements, mais aussi d'éviter des erreurs de calcul de la mise à l'échelle, comme dans l'exemple suivant.



(a) homographie estimée avec scaling inconnu

(b) homographie estimée avec scaling connu

Figure 7 – Illustation de l'amélioration des résultats en stockant le scaling

2.5 Choix de la meilleure pièce et vérification

Nous avons choisi de fonctionner de la façon suivante à chaque mise à jour de l'image :

1. les pièces sont triées par ordre de nombre de matchs décroissants
2. Ensuite, nous itérons à travers la liste et calculons la transformation associée.
3. Nous calculons un score **ZNCC** entre la pièce et la partie du puzzle où elle a été placée à l'aide de `cv2.WarpPerspective`. Des masques sont utilisés pour s'assurer que seules les zones pertinentes sont comparées.
4. Si ce score est suffisamment élevé (au-dessus d'un seuil de **0.55** fixé empiriquement), la pièce est validée et retournée ; sinon, nous passons à la pièce suivante.

.
Le nombre de matchs n'étant pas forcément toujours une très bonne métrique pour juger de la qualité de la détection de la pièce, ainsi que de son emplacement calculé par homographie, cette vérification permet de filtrer les mauvaises pièces.

Pourquoi ZNCC ? La Zero-mean Normalized Cross Correlation permet de comparer simplement, mais de manière assez robuste, deux images grayscale entre elles. En soustrayant par l'illumination moyenne de l'image et en renormalisant par l'écart-type, on obtient une comparaison indépendante de l'illumination. Ce sont finalement les variations d'illuminations qui sont comparées.

$$\text{ZNCC} = \frac{\sum_{x,y} \left(I(x,y) - \mu_I\right)\left(J(x,y) - \mu_J\right)}{\sqrt{\sum_{x,y} \left(I(x,y) - \mu_I\right)^2 \sum_{x,y} \left(J(x,y) - \mu_J\right)^2}}$$

— $I(x,y)$ et $J(x,y)$ étant les intensités des pixels des deux images respectives
— μ_I et μ_J : les intensités moyennes

ZNCC donne des valeurs entre -1 et 1, 1 pour une corrélation parfaite des variations d'intensité sur les deux images, -1 pour une anti-corrélation parfaite. Des pièces bien détectées et matchées étaient autour de 0.65-85 sur nos puzzles, et autour de 0 lorsque ce n'était pas le cas. Nous avons ainsi utilisé cet outil pour filtrer les mauvaises homographies avec un seuil fixé expérimentalement à 0.55. Si le score ZNCC correspondant à l'homographie est faible, cela indique une erreur d'appariement.Cette approche rend les estimations plus robustes, mais limite légèrement le solveur vers la fin du puzzle, où certaines homographies, peu précises mais correctes, sont rejetées. (L'idéal serait de diminuer le seuil au fil des itérations, alors que la qualité des homographies diminue).

2.6 Pour aller plus loin : des features basées sur la couleur

Motivation L'utilisation de features SIFT est très performante sur les parties du puzzle ayant des détails remarquables, mais fonctionne mal sur les parties plus unies. Nous avons donc souhaité utiliser des descripteurs basés sur la couleur pour traiter ces pièces une fois que la feature detection avec SIFT ne fonctionne plus.

Première méthode : histogrammes Notre première tentative consistait à décomposer le puzzle cible en rectangles correspondant à chaque pièce (en utilisant une division uniforme, qui correspond bien à l'emplacement des pièces), puis à calculer un histogramme pour chaque rectangle et chaque pièce (nous avons utilisé cv2.logPolar qui crée un histogramme des couleurs en coordonnées polaires logarithmiques, qui a pour particularité d'être invariant par translation). Pour comparer les histogrammes, nous avons utilisé la distance emd (earth mover distance). L'objectif était d'affecter la pièce au rectangle présentant la plus faible distance emd avec lui. Cependant, cette méthode reste peu précise et a du mal à discriminer entre des cases de couleur similaire. Elle n'est pas assez précise en pratique pour être ajoutée au solver.

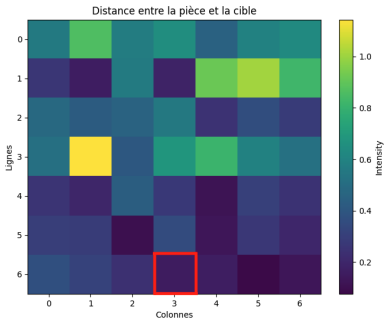


Figure 8 – Heatmap représentant les distances entre l'histogramme d'une pièce et les carrés correspondant dans le puzzle (le carré rouge correspond au match correct, mais n'est pas l'endroit où la distance est la plus faible).

Seconde méthode : décomposition de l'image Notre seconde approche consistait à décomposer chaque pièce en carrés plus petits et à calculer un descripteur simple sur chacun de ces carrés (nous calculons la moyenne des couleurs des pixels sur chaque carré, dans la représentation LAB). Nous parcourons ensuite l'image cible (elle aussi décomposée en carrés, de taille adaptée grâce à la mise à l'échelle stockée) et calculons la distance euclidienne des descripteurs de couleurs. En testant plusieurs orientations, nous retenons la position présentant la plus faible distance, qui correspond à l'emplacement de la pièce. Lorsque cette méthode fonctionne, elle permet de localiser précisément la pièce, mais elle échoue souvent à identifier un minimum clair. De plus, elle demande des calculs importants pour parcourir toute la grille (la complexité augemente de manière quadratique lorsque l'on cherche à réduire la taille des carrés, pour capturer de détails plus fins). Bien qu'elle fonctionne partiellement, cette méthode n'était pas assez robuste pour être intégrée au solveur final.

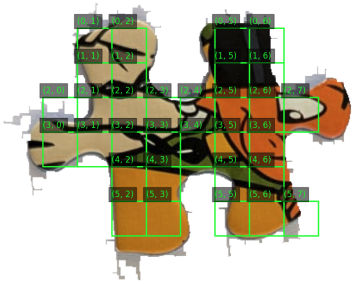


Figure 9 – Décomposition de l'image en plus petits descripteurs

3 RÉSULTATS

Le solver fonctionne très bien en pratique et résout la plus grande partie des puzzles. Il devient moins performant sur les dernières pièces, qui comportent peu de détails, mais accélère considérablement la complétion du puzzle.

Le code associé à ce projet, pour pouvoir tester le solveur sur vos propres puzzles, ce trouve sur lien ci-joint : https://github.com/adriengoldszal/INF573_Puzzle

Nous avons testé notre solveur sur deux puzzles différents dont les images sont en **annexe 3** :

— Un puzzle de 49 pièces à petites pièces : Yakari
— Un puzzle de 54 pièces à grandes pièces et donc plus de features à détecter : Chateau

Puzzles	**Successful Matches**
Yakari	44/49
Château	49/54

Table 1 – Puzzle completion statistics

Pour la suite, il serait intéressant de tester le solver sur des puzzles plus grands (comme celui présenté en figure 12). Au vu des méthodes que nous déployons, nous nous attendons à de bons résultats (le matching de features SIFT est assez robuste et renforcé par le ZNCC, ce qui devrait permettre le passage à l'échelle). Cependant, il est problable que les zones unies restent plus difficiles à reconstituer.

4
ANNEXES

ANNEXE 1 : THRESHOLDING SUR LES FRAMES DE LA CAMÉRA

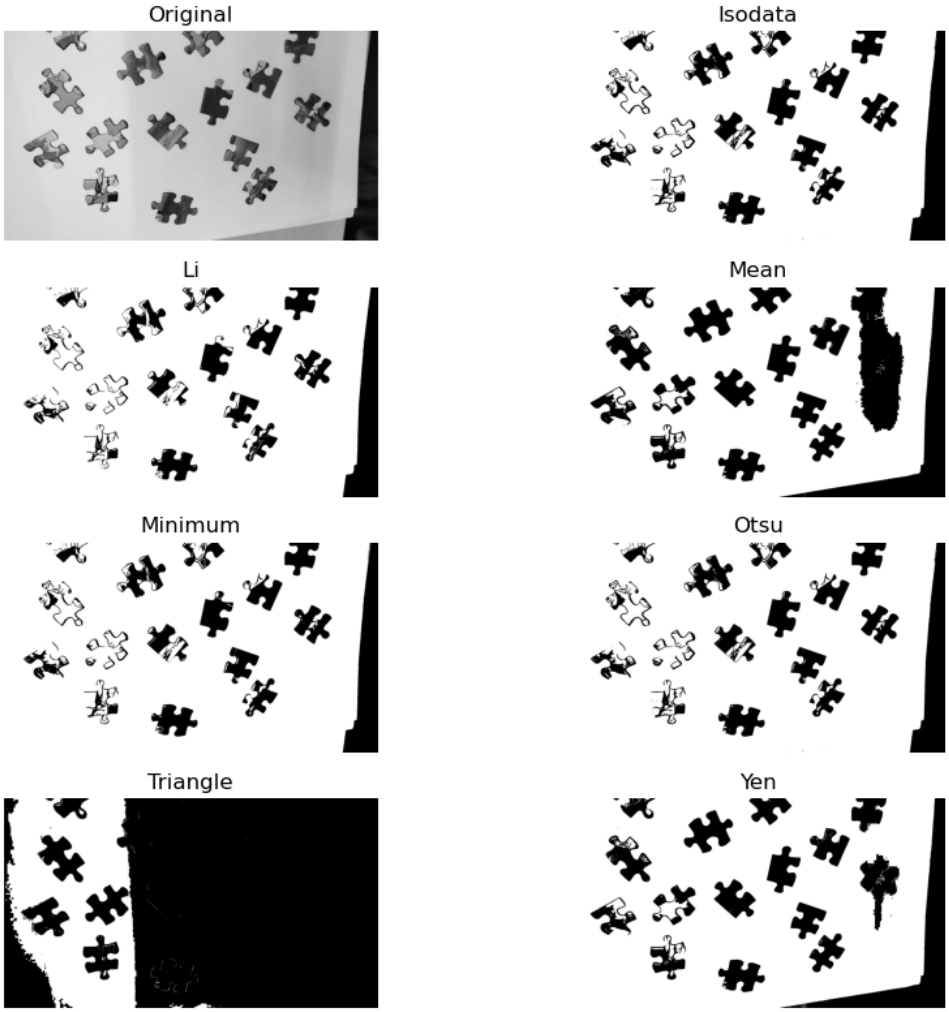


Figure 10 – Différents thresholds sur une image de caméra

ANNEXE 2 : ESTIMATION DE LA TRANSFORMATION DE SIMILARITÉ

Input: src_points : ensemble de points source $(n \times 2)$,
dst_points : ensemble de points destination $(n \times 2)$
Output: $scale$, θ, t : paramètres de la transformation (échelle, rotation, translation)
Étape 1 : Calcul des centroids
$src\_centroid \leftarrow \frac{1}{n} \sum_{i=1}^{n} src\_points[i]$
$dst\_centroid \leftarrow \frac{1}{n} \sum_{i=1}^{n} dst\_points[i]$
Étape 2 : Centrage des points
$src\_centered \leftarrow src\_points - src\_centroid$
$dst\_centered \leftarrow dst\_points - dst\_centroid$
Étape 3 : Filtrage des vecteurs de longueur nulle
$valid\_mask \leftarrow (\|src\_centered\| \neq 0) \wedge (\|dst\_centered\| \neq 0)$
$src\_centered \leftarrow src\_centered[valid\_mask]$
$dst\_centered \leftarrow dst\_centered[valid\_mask]$
Étape 4 : Calcul de l'échelle (scale)
$src\_rms \leftarrow \sqrt{\sum_{i=1}^{n} \|src\_centered[i]\|^2}$
$dst\_rms \leftarrow \sqrt{\sum_{i=1}^{n} \|dst\_centered[i]\|^2}$
if $src\_rms = 0$ **then**
  $scale \leftarrow 1.0$
end
else
  $scale \leftarrow \frac{dst\_rms}{src\_rms}$
end
Étape 5 : Mise à l'échelle des points destination
$dst\_centered\_scaled \leftarrow \frac{dst\_centered}{scale}$
Étape 6 : Calcul de la matrice de covariance croisée
$H \leftarrow src\_centered^T \cdot dst\_centered\_scaled$
Étape 7 : Extraction de la rotation avec SVD
$U, S, V^T \leftarrow \mathrm{SVD}(H)$
$R \leftarrow U \cdot V^T$
if $\det(R) < 0$ **then**
  $U[:, -1] \leftarrow -U[:, -1]$
  $R \leftarrow U \cdot V^T$
end
Étape 8 : Extraction de l'angle de rotation
$\theta \leftarrow \arctan 2(R[1, 0], R[0, 0])$
Étape 9 : Calcul de la translation
$t \leftarrow dst\_centroid - scale \cdot R \cdot src\_centroid$
return $scale, \theta, t$

ANNEXE 3 : PUZZLES DE TEST

Figure 11 – Yakari 49 pièces

Figure 12 – Allemagne 200 pièces

Figure 13 – Chateau 54 pièces

RÉFÉRENCES

[1] H. Freeman and L. Garder. Apictorial jigsaw puzzles : The computer solution of a problem in pattern recognition. IEEE Transactions on Electronic Computers, EC-13(2) :118–127

[2] Travis V. Allen. Using Computer Vision to Solve Jigsaw Puzzles

[3] Liang Liang and Zhongkai Liu . A Jigsaw Puzzle Solving Guide on Mobile Devices