Mise en correspondance stéréo

Romain Froger & Thomas Leymonerie

10 juin 2023

Résumé

Dans le cadre de l'UV SY32 et afin d'appliquer les notions vues en cours et en TD, nous avons implémenté plusieurs algorithmes de mise en correspondance stéréoscopiques d'images. Ces méthodes nous permettent d'obtenir des cartes de disparité qui permettent l'estimation de profondeurs. Nous allons présenter les méthodes de *Block Matching* et de*Semi Global Block Matching*. Ce rapport décrit le processus de recherche et de développement de nos algorithmes.

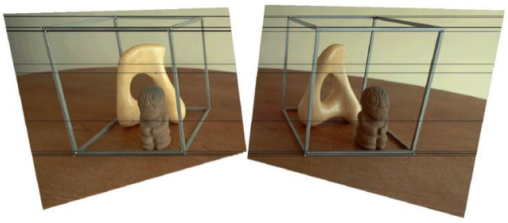


FIGURE 2 – Images rectifiées
Source: Loop, C., and Zhengyou Zhang. Computing rectifying homographies for stereo vision. Vol. 1, 1999.

1 Préliminaires

L'objectif de ce projet est de faire une mise en correspondance stéréovision de deux images représentant la même scène sous des angles de vues différents. Les caméras ayant photographié la scène sont donc disposées dans deux positions différentes. On se ramènera au cas le plus simple (avec des homographies) où les deux caméras sont dans une configuration de translation simple comme décrite sur la figure 1.

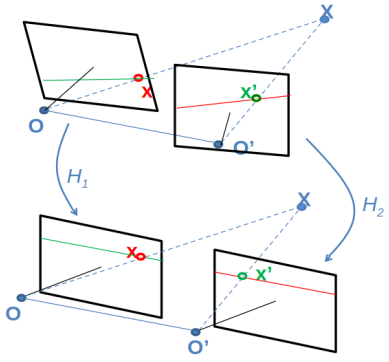


FIGURE 1 – Rectification stéréoscopique
Source: Stéréovision, géométrie épipolaire, 3D avec rectification stéréoscopique, Julien Moreau, 2023

Les deux plans images alignés, les images sont projetées sur le même plan contenant les épipôles afin d'aligner les lignes épipolaires (fig. 2). On peut alors chercher la correspondance d'un pixel donné en faisant une recherche le long de la ligne épipolaire conjuguée.

Dans toute la suite de l'étude des algorithmes de mise en correspondance stéréo, on se placera dans le cas où les images sont rectifiées. En particulier, les algorithmes seront exécutés et évalués sur deux paires d'images de référence : *Teddy* et *Cones* (cf fig. 3). Pour chaque paire d'images, les cartes d'occlusions et de *Ground Truth* sont fournies pour l'évaluation, qui se fera toujours sur un mise en correspondance de la gauche vers la droite. On fera donc attention à optimiser les paramètres des modèles pour maximiser une performance globale sur les images *Teddy* et *Cones*.

L'évaluation prendra en compte :
— Erreur moyenne pixellique
— Pourcentage de pixels avec une erreur de disparité $> s$ pour $s \in \{1, 2\}$ notés `ps1` et `ps2`.
— Rapidité : temps d'exécution en ms

(a) (b)

FIGURE 3 – Teddy gauche(a) et Teddy droite (b)

2 Algorithme Block Matching

La première méthode que nous avons souhaité explorer est une méthode locale : le block matching. Elle est dite locale car elle optimise un appariement pixel à pixel sans optimiser une disparité globale. Du fait que les images soient rectifiées, et donc que les lignes épipolares appartiennent au même plan, on peut limiter la recherche d'un pixel correspondant sur la ligne conjuguée. On s'assure donc de limiter le coût de calcul des disparités. Afin de sélectionner le meilleur candidat correspondant, on cherchera à minimiser une mesure de dissimilarité sur un région voisine du pixel considéré, comme le montre la figure 4. Il y a deux paramètres à choisir pour optimiser le block matching :

— Le calcul de dissimilarité entre deux blocs.
— La taille du voisinage à observer.

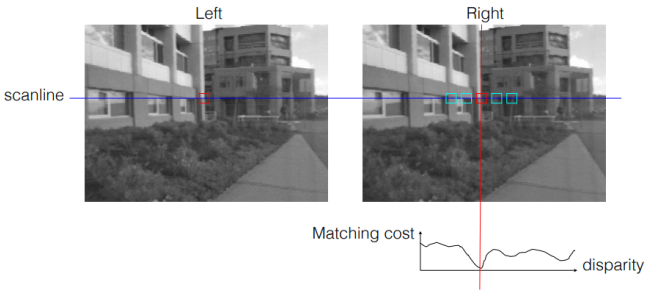


FIGURE 4 – Recherche de la dissimilarité minimale
Source: Carnegie Mellon University, Kris Kitani, 16-385

2.1 Dissimilarités

Il existe plusieurs manières de calculer les dissimilarités entre deux fenêtres et il conviendra d'étudier l'impact de chacune sur les performances finales. Pour une taille de fenêtre fixée, on a relevé les performances présentées ci-dessous sur les images *Teddy* (des metrics du même ordre ont été calculées sur *Cones*).

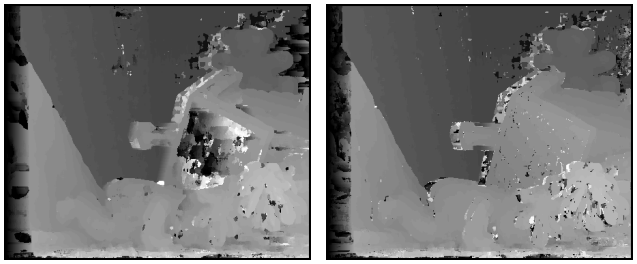
Metric	Exec. (s)	Avg. err	ps1	ps2
SAD	0.93	3.13	38.19%	32.39%
SSD	1.00	2.24	31.06%	27.31%
ZSSD	2.84	1.50	24.99%	21.66%
ZNSSD	6.93	1.50	24.99%	21.66%
NCC	5.53	1.68	25.14%	22.02%
GC	470.11	1.57	25.10%	22.81%

TABLE 1 – Performance de chaque dissimilarité

Dans le souci d'obtenir une méthode à la fois efficace et avec un temps d'exécution correct, il apparait que les dissimilarités de type ZSSD et ZNSSD sont optimales, on choisira donc ZSSD pour son exécution plus rapide. Ci-dessous la formule correspondante :

$$\mathrm{ZSSD}\left(\Omega_2, \Omega_1\right) = \sum \left(\left(\Omega_2 - \widehat{\Omega_2}\right) - \left(\Omega_1 - \widehat{\Omega_1}\right) \right)^2$$

Ainsi ZSSD a l'avantage de fournir une invariance aux changements globaux de luminosité, ce qui rend la mesure de dissimilarité plus robuste dans des scènes complexes comme le montre la figure 5 qui compare à la dissimilarité SAD (Sum of Absolute Difference).



(a) (b)

FIGURE 5 – Carte des disparités SAD (a) vs. ZSSD (b)

On remarquera que dans le tableau 1,le block matching utilisant la dissimilarité *Gradient Correlation* a un temps d'exécution substantiellement plus long que les autres. En effet, on ne peut appliquer de `jit` de `numba` sur des opérations de gradients, ce qui explique sa lenteur par rapport aux autres dissimilarités.

2.2 Taille des blocs

Nous avons trouvé une mesure de dissimilarité qui semble être optimale pour nos scènes *Teddy* et *Cones*. Nous devons désormais optimiser la taille de voisinage à observer. Pour cela, nous avons testé l'impact du changement de cette variable sur les performances globales. En effet, un voisinage trop resserré ne permettra pas une bonne identification du bloc correspondant dans l'image paire, tandis qu'un voisinage trop grand prendra une trop grande marge sur les bords de l'image. C'est ce que l'on peut observer sur la figure 6 ci-dessous. On décide donc pour le reste de l'algorithme de choisir un voisinage de taille 7 car il minimise les erreurs `ps1` et `ps2`. On remarque d'ailleurs que l'erreur pixellique moyenne n'est pas nécessairement la meilleure mesure à observer lors des choix de paramètres optimaux.

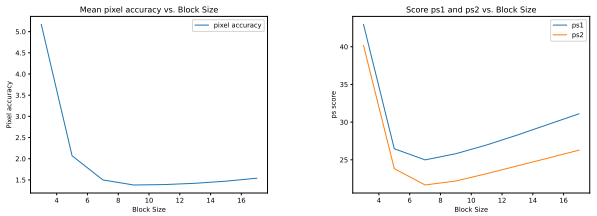


FIGURE 6 – Performances en fonction de la taille du voisinage

2.3 Pre/Post-processing

Les résultats du blockmatching sont pour le moment corrects mais nous pensons pouvoir obtenir bien mieux. Nous allons donc explorer des méthodes afin d'optimiser la qualité des résultats obtenus par l'algorithme de blockmatching, notamment en appliquant des filtres de pré et post-traitement. Nous avons donc déterminé que les filtres suivants ont un impact non négligeable sur la qualité des résultats.

N'atteignant toujours pas une précision moyenne sous-pixellique, on s'intéresse à l'ajout d'un filtre de post-traitement. Plusieurs filtres existent comme les flous gaussiens, les flous médians, les filtres bilinéaires et les filtres de mode [1]. Si l'on observe la carte des disparités (fig. 7), on observe des trous au milieu des objets à texture uniforme.

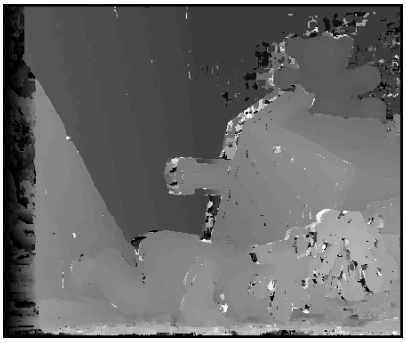


FIGURE 7 – Carte des disparités brutes

Ces imperfections que l'on retrouve sur les surfaces lisses peuvent être corrigées par un filtre de mode. Ce dernier va affecter à chaque pixel la valeur la plus présente dans son voisinage et lissera la carte des disparités. On décidé donc de tester l'impact d'un filtre de mode sur la précision moyenne. Un filtre de mode doit prendre en paramètre la taille du voisinage. Un voisinage trop faible n'aura pas ou peu d'effet car les voisins des pixels "corrompus" peuvent l'être, et un voisinage trop grand aura pour effet de déformer les structures rectilignes. On cherche donc un compromis entre les deux, ci-dessous les résultats d'une recherche de la taille optimale du voisinage :

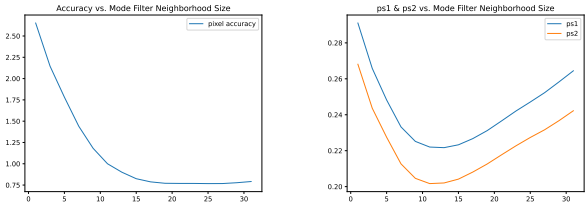


FIGURE 8 – Performances en fonction de la taille du voisinage du filtre de mode

On remarque donc effectivement qu'il faut faire un compromis comme le montre la sous-figure de droite de la figure 8. Encore une fois on favorisera les scores `ps1` et `ps2` à l'erreur moyenne pixellique. On trouve donc une valeur optimale du voisinage égale à *11*, ce qui permet d'obtenir la carte des disparités suivante :

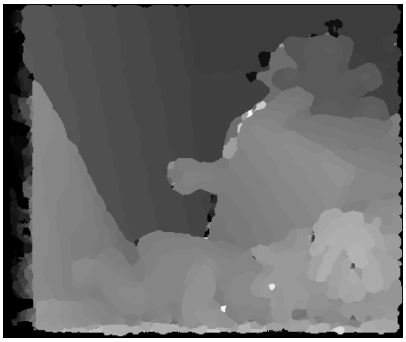


FIGURE 9 – Carte des disparités filtrée

Comme on peut le remarquer, la carte des disparités est beaucoup plus lisse mais les structures locales (formes et features linéaires) qui étaient nets dans la première carte de disparité (fig. 7) ne le sont plus. Un objectif pourrait être de rendre la carte de disparité brute de meilleure qualité afin de limiter les déformations du filtre de mode en postprocessing. Néanmoins les résultats sont probants étant donnés les performances obtenues décrites dans la table 2. Le filtre permet de réduire de 45% l'erreur pixellique moyenne et de 21% de la mesure ps1.

Image	Avg. err	ps1	ps2
Brut	1.50	24.99%	21.66%
Filtre de mode	0.83	19.7%	17.56%

TABLE 2 – Performances induites par le filtre de mode

Finalement, un dernier filtrage peut-être effectué afin d'obtenir des résultats encore meilleurs. En effet, la technique du block matching ne permet pas d'identifier les zones occultées. Néanmoins, en jouant sur un seuil de disparité, on peut essayer de marquer certaines zones comme occultées. En théorie, un point occulté a une disparité égale à 0, cependant les approximations du blockmatching peuvent parfois donner des disparité très faibles, au lieu de marquer le point comme occulté. On peut donc appliquer un filtre en mettant à 0 toutes les disparités en dessous d'un certain seuil ε. Pour cela il faut choisir un seuil adapté en trouvant le seuil optimal.

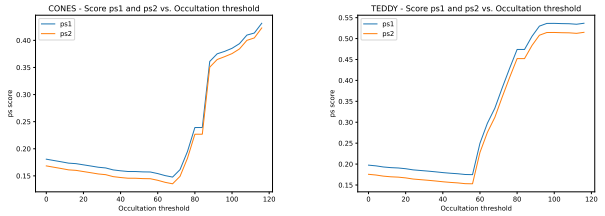


FIGURE 10 – Performances en fonction du seuil ε de filtrage des occlusions

Ainsi, au vu des résultats présentés dans la figure 10, on trouve qu'un seuil de disparité égal à *56 px* est optimal pour minimiser le taux d'erreur moyen pixellique sur les deux scènes. En effet, on remarque un saut des scores `ps1` et `ps2` juste après l'optimum, on choisit donc l'optimum le plus bas. En particulier, en comparant les résultats de l'application de ce filtre aux résultats précédents obtenus avec un filtre de mode, on réduit l'erreur pixellique moyenne de 8% et le score ps1 de 12%.

Image	Avg. err	ps1	ps2
Filtre de mode	0.83	19.7%	17.56%
Filtre de mode + Occl	0.77	17.47%	15.3%

TABLE 3 – Performances induites par les occlusions

Les nuances de gris ne sont pas nécessairement les meilleures manières pour représenter une image sur laquelle on va appliquer des mesures de dissimilarités. Le filtre *census* est un opérateur d'image qui associe à chaque pixel d'une image en niveaux de gris une chaîne binaire, encodant si le pixel a une intensité plus faible que chacun de ses voisins, un bit pour chaque voisin. Ce filtre engendre une robustesse aux variations de luminosité. L'exemple ci-dessous utilise un *kernel* de taille (3x3) mais de plus grands kernels peuvent-être utilisés. Le résultat de la convolution devient la nouvelle valeur du pixel évalué. Le filtre permet notamment de bien identifier les contours des objets comme le montre la figure 11.

$$\begin{bmatrix} 124 & 73 & 32 \\ 124 & 64 & 18 \\ 157 & 116 & 84 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 \\ 1 & x & 0 \\ 1 & 1 & 1 \end{bmatrix} = 11010111$$

(a) (b)

FIGURE 11 – Teddy nuance de gris (a) vs. census (b)

En appliquant le filtre Census et un filtre de mode comme présenté précédemment, on obtient de légers progrès supplémentaires. Il est intéressant de noter qu'un filtre census sans filtre de mode en post traitement est en fait moins efficace que l'image en niveau de gris originale. Il faut donc toujours combiner le filtre Census avec ce dernier.

Image	Avg. err	% > 1px	% > 2px
Teddy Census	0.92	17.10 %	15.06%
Cones Census	0.62	13.54 %	12.88%

2.4 Piste non concluante

Avant de conclure sur les résultats du blockmatching, nous avons souhaité gagner en performance en modifiant l'algorithme de blockmatching original, ce qui nous a mené à une observation non attendue.

En effet, par souci de simplicité, l'algorithme ne prend pas en compte les bords de l'image afin de faciliter la gestion des blocs non entiers sur les bords. Néanmoins cela a la conséquence directe d'avoir des disparités à 0 sur tout le bord de la carte de disparité. Pour résoudre ce problème, nous avons modifié les images originales dont on veut extraire la disparité en leur ajoutant un *padding* de la largeur du voisinage utilisé par le blockmatching. Le padding est composé du miroir du bord de l'image originale comme le montre la figure 13 ci-dessous (marge exagérée pour l'exemple)

Malheureusement le blockmatching résultant de cette image n'est pas plus efficace après filtre de mode qu'une image sans bordure. Cela s'explique par le fait que le filtre de mode appliqué en post processing remplit la

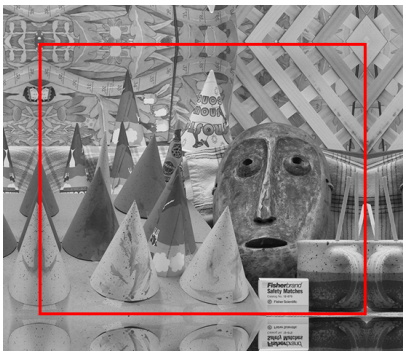


FIGURE 12 – Image Cones avec marge mirroir

marge sur les images non miroir. Les cartes de disparité des images miroirs ne présentent quasi aucune occlusion. C'est cette différence qui explique une moins bonne performance, bien que nous nous attendions à obtenir une image moins bruitée et donc sur laquelle appliquer un filtre de mode moins aggressive pour conserver les structures locales.

2.5 Bilan Block Matching

Pour résumer, la technique du blockmatching qui au premier abord ne semblait pas satisfaisante peut être largement améliorée en choisissant des paramètres optimaux et en appliquant des filtres pré et post traitement. On récapitule donc les choix qui ont été faits pour obtenir les meilleurs résultats :

Paramètre	Valeur
Preprocessing	Census Filter
MAXDISP	64
Block size	7
Dissimilarity	ZSSD
Mode filter	11
Occlusion filter	56

TABLE 4 – Paramètres de Block Matching optimaux

Avec les paramètres cités ci-dessus, on obtient donc les résultats suivants pour les scènes *Teddy* et *Cones* :

Scène	Avg. err	ps1	ps2
Teddy	0.92	17.10 %	15.06%
Cones	0.62	13.54 %	12.88%
Avg.	0.77	15.32%	13.97%

TABLE 5 – Résultats finaux Block Matching

Afin de comprendre plus en détail les temps de calcul, ci-dessous un détail des temps de calcul de chaque étape :

Etape	Teddy	Cones
Pre-Processing	2.87s	2.93s
StereoMatching	3.45s	3.78s
Post-Processing	1.37s	1.46s
Total	7.69s	8.17s

TABLE 6 – Vitesse d'exécution

Au vu des résultats des années précédentes sur le même projet, nous pensons avoir un très bon compromis entre la vitesse d'exécution et les résultats obtenus. Cependant nous souhaitions préciser que les paramètres ont été obtenus en les évaluant les uns à la suite des autres en supposant leur indépendance (ce qui n'est pas le cas). De meilleures performances pourraient être obtenues en effectuant un *gridsearch* sur tous les paramètres, ce qui est très coûteux. Cette méthode de recherche assure une qualité optimale des paramètres pour une image donnée mais pas pour l'ensemble des images. Nous avons donc essayé de faire les meilleurs compromis pour garantir des performances globales optimales.

3 Algorithme Semi Global Matching

Une autre méthode que nous avons voulu mettre en oeuvre pour ce projet est le Semi Global Matching (SGM) [2].
La méthode consiste à réaliser une correspondance stéréoscopique en optimisant les lignes selon plusieurs directions et en calculant un coût agrégé pour chaque pixel avec une disparité donnée.

Le coût se compose d'un terme de correspondance et d'un terme de régularisation binaire. Le terme de correspondance, $D(p, d)$, peut être n'importe quelle mesure de dissimilarité locale d'image, telles que la différence d'intensité absolue ou le ZSSD vu précédemment.

Le terme de régularisation est une fonction en trois parties qui permet une pénalité réduite pour les changements unitaires de disparité, favorisant ainsi des transitions fluides correspondant à des surfaces inclinées, tout en pénalisant les sauts plus importants et en préservant les discontinuités grâce à un terme de pénalité constante.

Ce terme est de la forme :

$$R(d_p, d_q) = \begin{cases} 0 & d_p = d_q \\ P_1 & |d_p - d_q| = 1 \\ P_2 & |d_p - d_q| > 1 \end{cases}$$

pour chaque paire de pixels p et q.

P_1 et P_2 sont des hyperparamètres à choisir pour notre modèle avec $P_1 < P_2$.

Le coût accumulé $S(p,d) = \sum_r L_r(p,d)$ correspond à la somme de tous les coûts $L_r(p,d)$ pour atteindre le pixel p avec une disparité d le long de chaque direction r. Chaque terme peut être exprimé de manière récursive :

$$\begin{aligned} L_r(p,d) = D(p,d) + \\ \min \Big\{ & L_r(p-r,d), \\ & L_r(p-r,d-1) + P_1, \\ & L_r(p-r,d+1) + P_1, \\ & \min_i L_r(p-r,i) + P_2 \Big\} \\ & - \min_k L_r(p-r,k) \end{aligned}$$

Le terme $D(p,d)$ mesure la similarité entre les pixels correspondants dans les deux images, avec ZSSD par exemple.
Le terme $\min\{...\}$ représente la régularisation avec les poids P_1 et P_2 vu précédemment.
Pour finir, le terme $\min_k L_r(p-r,k)$ est le coût minimal au pixel précédent dans la direction r.

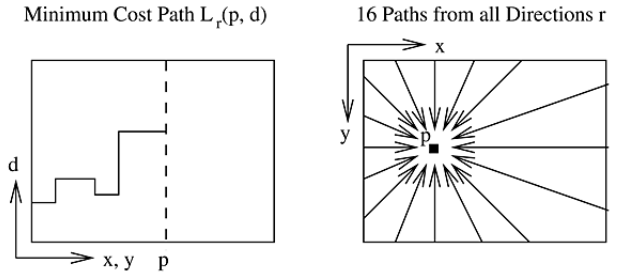


FIGURE 13 – Illustration du calcul des coûts.

Pour finir, La valeur de disparité pour chaque pixel est déterminée en trouvant la disparité qui minimise le coût accumulé.

Plutôt que de parcourir chaque pixel un à un, il est possible d'utiliser une fenêtre afin d'utiliser la moyenne des disparités des pixels contenu dans la fenêtre. Cette technique permet d'améliorer les résultats puisque certains bruits seront déjà lissés et permet de faire gagner du temps de calcul.

3.1 Exemple

Pour mieux comprendre le fonctionnement de la méthode, nous pouvons prendre un exemple.

Supposons que nous travaillions avec une image stéréoscopique et que nous voulions calculer la disparité du pixel situé à la position (x, y) dans l'image de gauche par rapport à l'image de droite avec l'algorithme SGM.

Pour le pixel (x, y), nous allons d'abord initialiser les coûts accumulés pour chaque disparité possible, avec ZSSD par exemple. Ensuite, nous itérerons sur les différentes directions pour calculer les coûts accumulés le long de chaque direction (8 ou 16 généralement).

Pour chaque direction, nous examinerons les pixels voisins dans cette direction et utiliserons les coûts accumulés de ces pixels pour calculer les coûts accumulés du pixel (x, y) lui-même. Cela nous permettra d'évaluer à quel point chaque disparité est cohérente avec les pixels environnants.

Enfin, nous choisirons la disparité ayant le coût accumulé le plus bas comme étant la disparité du pixel (x, y).

En résumé, l'algorithme SGM pour le pixel (x, y) consiste à calculer les coûts accumulés pour chaque disparité possible en utilisant l'information des pixels voisins dans différentes directions. La disparité correspondante est ensuite déterminée en choisissant la disparité ayant le coût accumulé le plus bas. Cela nous permet de trouver la correspondance de ce pixel spécifique entre les deux images.

3.2 Implémentation du SGM

Nous avons choisi de reprendre le pré-traitement avec le filtre census [3] et le post-traitement par le filtre de mode.

Pour cette méthode, il y a plusieurs paramètre à déterminer :

Paramètres
maxdisp
P1
P2
Census kernel
Mode filter kernel
Occlusion filter

TABLE 7 – Paramètres du Semi Global Matching

Pour estimer les disparités initiales entre les images, nous avons utilisé une méthode basée sur le XOR Census et la distance de Hamming. Cette méthode nous permet de mesurer les différences entre les pixels correspondants des deux images en utilisant des opérations de XOR et de comptage des bits différents.

Plus précisément, nous avons suivi les étapes suivantes :

1. Pour chaque valeur de disparité, nous avons extrait les patchs de pixels correspondants de l'image droite en utilisant les décalages appropriés.
2. Ensuite, nous avons effectué une opération de XOR entre les valeurs des pixels de l'image gauche et de l'image droite pour obtenir une représentation binaire des différences.
3. En utilisant cette représentation binaire, nous avons calculé les distances de Hamming, c'est-à-dire le nombre de bits différents, pour chaque pixel.

L'utilisation de cette méthode basée sur le XOR Census et la distance de Hamming nous permet de mesurer efficacement les similitudes et les différences entre les pixels des images.
Grâce au pré-traitement par le filtre de Census, qui transforme la valeur des pixels en 0 ou 1, cette méthode permet de calculer précisément les disparités initiales.

3.3 Résultats

Avant de conclure sur le Semi Global Matching, nous pouvons discuter des paramètres à choisir afin d'avoir les meilleurs résultats.

Comme nous avons pu le voir précédemment, il y a 5 paramètres à régler afin d'obtenir des meilleures résultats.

3.3.1 Réglage du maximum de disparité

Nous pouvons, premièrement, trouver le maximum de disparité à choisir pour obtenir les meilleurs résultats. Ce maximum de disparité, noté maxdisp, doit être un multiple de 16. En testant plusieurs valeurs nous obtenons :

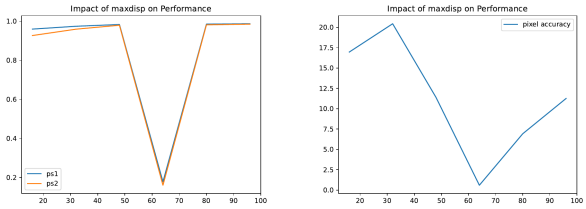


FIGURE 14 – Performances en fonction du maximum de disparité

Comme nous pouvons le voir, le maximum de disparité à choisir est bien 64, ce qui correspond bien à ce qu'on aurait pu s'attendre.

Les paramètres importants à choisir, et qui sont liés à la méthode SGM, sont le choix des paramètres de régularisation P1 et P2.
Pour cela, nous avons testé plusieurs valeurs :

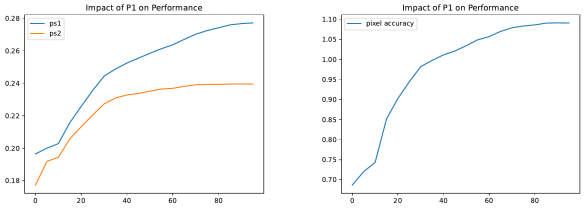


FIGURE 15 – Performances en fonction de P1

On remarque, que pour P1, plus la valeur est faible est meilleurs sont les résultats. Nous avons donc choisi P1 à 0.
Pour P2, la valeur doit être strictement supérieur à P1, nous avons donc les résultats suivants :

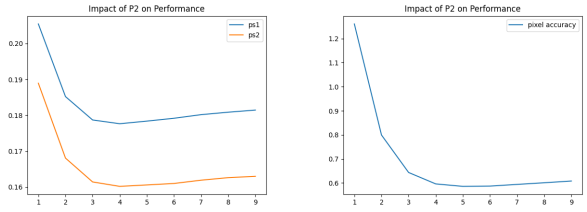


FIGURE 16 – Performances en fonction de P2

Pour P2, nous remarquons que, pour les valeurs 4 ou 5, nous obtenons de meilleurs résultats.
Après avoir comparé avec les deux images, prendre P2 égal à 4 donnait de meilleurs résultats.

Les derniers paramètres à déterminer sont les noyaux du pré-traitement, avec le filtre Census, et du post-traitement, avec le filtre de Mode.

Pour cela, nous avons d'abord testé les valeurs du pré-traitement et nous avons obtenu :

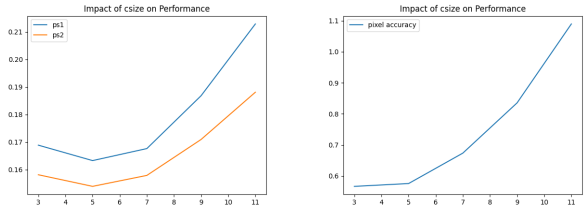


FIGURE 17 – Performances en fonction de csize

Nous avons donc déterminer que prendre un noyau de 5x5 était préférable pour le pré-traitement afin d'obtenir une erreur ps1 minimum.

Nous pouvons choisir la taille du noyau de filtre de Mode :

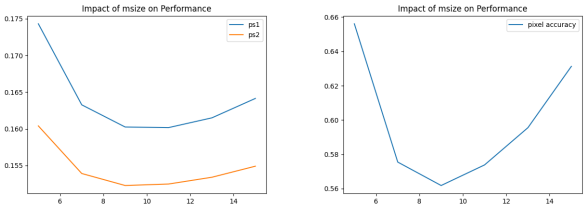


FIGURE 18 – Performances en fonction de msize

Au vu des résultats, prendre un noyau de taille 9x9 permet d'obtenir des meilleurs résultats.

Finalement, en appliquant la même méthode de recherche que pour le block matching, on trouve un seuil d'occlusion optimal à *70*.

3.4 Bilan Semi Global Matching

Pour conclure sur cette méthode du Semi Global Matching, nous avons choisi les paramètres optimaux suivants :

Paramètre	Valeur
csize	5x5
maxdisp	64
P1	0
P2	4
msize	9x9
Occlusion filter	56

TABLE 8 – Paramètres optimaux du Semi Global Matching

Avec ces paramètres, nous obtenons les résultats suivant pour les scènes *Teddy* et *Cones*, avec le temps d'exécution pour chaque scène :

Scène	Avg. err	ps1	ps2	time
Teddy	0.79	14.85%	12.71%	33.98s
Cones	0.55	12.00%	11.30%	32.88s
Avg.	0.67	13.12%	12.01%	33.43s

TABLE 9 – Résultats finaux Semi Global Matching

Cette méthode donne des résultats légèrement supérieurs à la méthode du Block Matching. Tout comme pour le Block Matching, la recherche des paramètres optimaux a été effectuée en supposant l'indépendance entre les paramètres. Cependant, il est bien connu que certains paramètres, tels que les paramètres de régularisation P_1 et P_2, sont étroitement liés, notamment par la relation $P_1 < P_2$.
De plus, cette méthode peut être améliorée en utilisant des "blocks", c'est-à-dire en utilisant une fenêtre pour calculer les disparités, similaire à ce qui est fait dans le Block Matching, plutôt que de les calculer pixel par pixel. Cela pourrait permettre d'obtenir de meilleurs résultats et d'améliorer le lissage des résultats.

4 Conclusion

Pour conclure, au long de rapport d'étude, nous avons pu présenter les différents de mise en correspondance stéréo. Nous avons pu développer deux méthodes dites denses mais n'opérant pas à la même échelle : locale pour le blockmatching et semi-globale pour le SGBM. Bien optimisées, les deux méthodes produisent de bons résultats, mais on observe cependant que la qualité de ces derniers dépend en grande partie des filtres opérées sur l'image. On peut ainsi faire un parallèle avec des méthodes plus modernes utilisant des réseaux de neurones à convolution : la partie encoder du réseau cherchant à optimiser la représentation de l'image d'entrée correspond à notre recherche de filtres optimaux.

Rappel des résultats :

Méthode	Avg. err	ps1	ps2	time
BM	0.77	15.32%	13.97%	7.93s
SGM	0.67	13.12%	12.01%	33.43s

TABLE 10 – Résultats globaux

Références

[1] Deepa and K. Jyothi. A robust and efficient pre processing techniques for stereo images. In *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pages 89–92, 2017.

[2] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2) :328–341, 2008.

[3] M. Humenberger, T. Engelke, and W. Kubinger. A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pages 77–84, 2010.

A Sources

Wikipedia Census Transform

Wikipedia Semgi-Global Block Matching