Réalisation d'un algorithme de mise en correspondance stéréo pour l'estimation de profondeur

Pierre Adorni
Zhixuan Feng

11 juin 2023

Résumé

Afin d'appliquer les notions vues en cours et en TD, nous avons réalisé un algorithme de mise en correspondance stéréo pour estimer la profondeur d'une scène. Nous avons exploré plusieurs méthodes différentes vues en cours, dont deux méthodes locales : le *block matching* classique et le *block matching* avec des poids adaptatifs, ainsi qu'une méthode globale : le *semi-global block matching*. Nous exposons dans ce rapport notre processus de recherche ainsi que nos résultats et notre algorithme final.

1 Introduction

L'estimation de profondeur est un sujet qui suscite un grand intérêt dans de nombreux domaines, comme la robotique et la vision par ordinateur. Par exemple, l'estimation de profondeur est utile pour la modélisation 3D et pour les systèmes d'aide à la navigation ou l'interaction utilisateur-machine. La mise en correspondance stéréo est l'un des principaux moyens de calculer la profondeur d'une scène. Cette méthode consiste à trouver des correspondances entre des pixels de deux images prises par des caméras parallèles. Une fois que les correspondances sont connues, la profondeur peut être calculée à partir de la différence de position entre les pixels correspondants.

D'autres méthodes existent pour calculer une carte de profondeur, basées notamment sur du *deep learning*, utilisant soit une seule image soit une paire d'images stéréo. Les méthodes deep learning sont systématiquement plus performantes que les méthodes classiques que nous explorons dans ce rapport, et occupent depuis leur apparition le haut du classement de Middlebury. Nous illustrons un de ces modèles en annexe.

2 État de l'art

Il existe deux grands types de méthodes de mise en correspondance stéréo : les *méthodes globales*, qui consistent à trouver pour chaque pixel une disparité minimisant une mesure de l'erreur sur toute l'image, et les *méthodes locales*, qui limitent l'optimisation de la disparité à une fenêtre de recherche autour du pixel considéré. Si les méthodes globales sont souvent plus précises, elles sont également plus coûteuses [9][10].

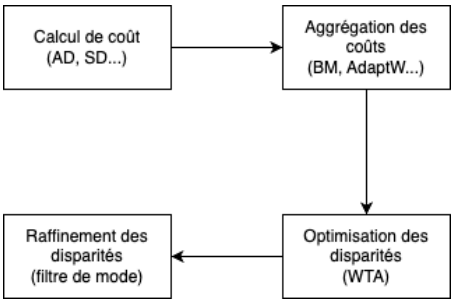


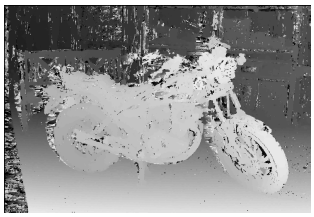
FIGURE 1 – étapes d'un algorithme de mise en correspondance locale

Les méthodes locales sont en majorité composées de quatre parties distinctes, décrites en Figure 1 : (a) Le calcul de coût par pixel, effectué par différence absolue ou plus robustement par différence au carré centrée et normalisée, (b) l'aggrégation des coûts, par block matching simple, fenêtre à poids adaptatifs ou optimisation linéaire, (c) l'optimisation des disparités, classiquement effectuée par la stratégie Winner Takes it All : la meilleure disparité est choisie, les autres sont éliminées, et enfin (d) le raffinement des disparités, par exemple avec un filtre de mode.

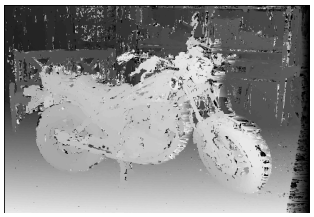
En utilisant une fenêtre à poids adaptatifs pour l'étape d'agrégation, des résultats similaires à ceux des méthodes globales ont été obtenus par [3][4], mais le temps d'exécutions de tels algorithmes sont extrême-

ment coûteux, en ceci qu'il augmente de façon quadratique avec la taille de la fenêtre d'agrégation. Plus récemment, une étape d'agrégation basée sur l'optimisation linéaire obtient des résultats similaires à ceux des algorithmes à poids adaptatifs en temps constant par rapport à la taille de la fenêtre[1].

Du côté des approches globales, une amélioration du temps de calcul a été obtenue par programmation dynamique[2], et une approche mixte a été développée pour améliorer les résultats d'une recherche locale[5].



(a) référence = gauche



(b) référence = droite

FIGURE 2 – Cartes de disparités pour *motorcycle* calculées par *block matching* + ZNSSD

3 Optimisation Locale

Nous avons implémenté différents algorithmes de mise en correspondance locale afin de les comparer entre eux. Nous sommes partis du block matching classique, que nous avons décliné et amélioré en suivant les propositions de précédents travaux mais aussi en apportant nos propres idées.

3.1 Block matching classique

La première méthode que nous avons implémentée est la méthode de mise en correspondance par blocs, le *block matching*. Il s'agit en effet de la méthode la plus "simple" à implémenter. Cette méthode consiste à considérer une fenêtre de taille arbitraire ($N = 5$ dans les exemples en Figure 2) autour du pixel dont on cherche à trouver la disparité. On déplace cette fenêtre sur l'axe horizontal, dans le sens du déplacement, et on calcule pour chaque déplacement d l'agrégation des "coûts", c'est à dire des distances entre chaque pixel p et son pixel correspondant déplacé $p-d$ dans la fenêtre.

Pour une résistance accrue aux changements de luminosité, on utilise l'agrégateur ZNSSD 1, qui compare non pas directement les valeurs d'intensité, mais une version normalisée et centrée sur la fenêtre de ces dernières.

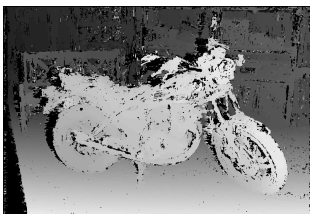
$$\mathrm{ZNSSD}(\Omega_1, \Omega_2) = \sum \left(\frac{\Omega_1 - \widehat{\Omega_1}}{\|\Omega_1 - \widehat{\Omega_1}\|} - \frac{\Omega_2 - \widehat{\Omega_2}}{\|\Omega_2 - \widehat{\Omega_2}\|} \right)^2 \tag{1}$$

Comme vu sur la figure 2, les résultats semblent satisfaisants, particulièrement au niveau du sol, mais l'algorithme échoue sur les zones d'occultation.

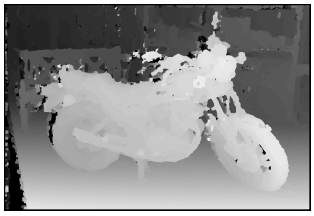
Pour remédier au problème des occultations, nous proposons une étape de fusion des deux cartes de disparités obtenues en prenant les deux images en référence chacune leur tour.

Pour chaque pixel (x, y), ayant pour valeur p sur la carte de disparités prenant en référence l'image de gauche, on trouve q : la valeur de $(x - p, y)$ sur la carte de disparités prenant en référence l'image de droite. Si le point n'est pas occulté, les valeurs de p et q doivent être proches. Si p et q sont trop différents, on remplace p par 0 sur la carte de disparité référence pour signaler une occultation. Cette étape nous permet d'estimer les occultations sur l'image, en noir sur la Figure 4.

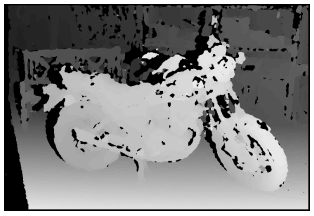
Après cette étape d'estimation des occultations, on réduit le bruit sur la carte de disparités (voir Figure 3a) en appliquant un filtre de mode, soit en excluant les occultations soit en les prenant en compte. Les résultats de cette dernière étape de raffinement sont présentés sur respectivement les Figure 3b et Figure 3c.



(a) Estimation des occultations par fusion des disparités



(b) Lissage par filtre de mode en excluant les occultations



(c) Lissage par filtre de mode en incluant les occultations

FIGURE 3 – Cartes de disparités pour *motorcycle* aux différentes étapes de raffinement

3.2 Poids adaptatifs

En dehors de l'algorithme "naïf" qu'est le block matching implémenté en TD, le premier algorithme que nous avons essayé d'implémenter est le block matching à poids adaptatifs que nous avons découvert dans [4]. Dans une fenêtre de block matching, tous les pixels ne sont pas forcément pertinents pour déterminer la disparité du pixel central : seuls les pixels provenant du même objet auront la même disparité. Ceci nous force à faire l'hypothèse que dans une fenêtre de block matching, tous les pixels appartiennent au même objet. Cette hypothèse est souvent vraie si la fenêtre est relativement petite, mais elle commence à ne plus être pertinente autour de $N = 9$, ce qui limite la quantité d'information que nous pouvons utiliser pour optimiser la disparité.

Dans [4], les chercheurs proposent une méthode permettant d'augmenter considérablement la taille de la fenêtre ($N = 35$ dans leurs exemples), en pondérant les pixels de cette dernière de sorte à prendre en compte surtout les pixels qui ont une forte probabilité d'appartenir au même objet que le pixel central.

La fenêtre est ainsi pondérée en fonction de la différence de couleur et la distance par rapport au pixel central.

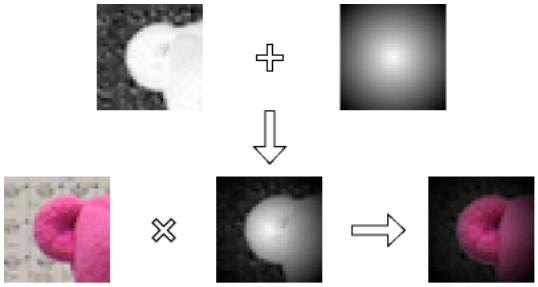
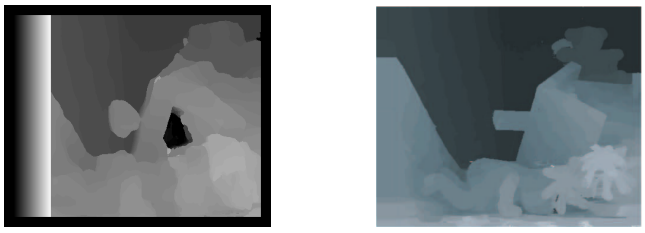


FIGURE 4 – Processus de pondération de la fenêtre d'aggrégation

Nos résultats avec cette technique sont bien moins bons que ceux des auteurs originaux, comme on peut le voir sur la Figure 5. Cette différence pourrait être dûe à la simplicité de notre implémentation par rapport à la proposition de [4], qui pourrait traduire une certaine naïveté de notre part sur certains aspects du problème.

Nous avons également essayé d'implémenter l'algorithme de [4] directement, mais nous avons eu du mal en raison de sa complexité. Nous sommes convaincus que l'algorithme de recherche par fenêtre d'agrégation à poids adaptatifs est une idée intéressante qui donne de bons résultats, et que la seule raison de notre échec est notre piètre capacité à l'implémenter correctement : nous nous sommes donc concentrés par la suite sur des méthodes plus simples à implémenter.



(a) Notre résultat

(b) Résultat des auteurs originaux

FIGURE 5 – Comparaison des résultats de notre implémentation de l'algorithme à poids adaptatifs avec l'implémentation de [4] sur *teddy*

3.3 Optimisation des paramètres

Notre méthode locale la plus efficace est donc le block matching classique. Il reste 3 paramètres à choisir : la taille de la fenêtre d'agrégation, la taille du filtre de mode, et le type de filtre de mode : avec ou sans les occultations.

Pour optimiser ces paramètres, nous avons effectué un quadrillage des possibilités, en considérant les deux jeux d'images de test : *cones* et *teddy*.

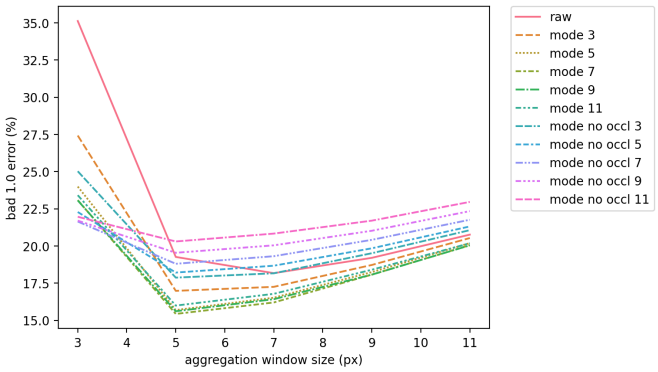


FIGURE 6 – Erreur 1px moyenne sur *teddy* et *cones* en fonction des différents paramètres du block matching

Les meilleurs paramètres sur notre jeu de données sont une fenêtre d'agrégation de taille $N = 5$ et un filtre de mode de taille $M = 7$, prenant en compte les occultations. Avec ces paramètres, notre algorithme obtient une erreur à 1px de 13,28% sur *cones* et 16,99% sur *teddy*, pour une moyenne de 15,43%. Si le filtre de mode ne prenant pas en compte les occultations semble donner de plus belles cartes de disparités à l'oeil humain

(voir Figures 3b et 3c), il donne en réalité systématiquement une moins bonne précision par rapport au filtre de mode normal.

Notre algorithme s'exécute en $\approx 17s$ sur des images de taille 450×375 sur notre CPU.

4 Optimisation Globale

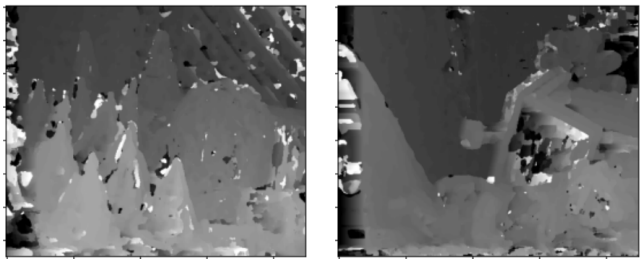
Nous avons implémenté deux différents algorithmes de mise en correspondance globale afin de les comparer entre eux : Semi-global Block Matching et Programmation dynamique.

4.1 Semi-global Block Matching

La première méthode que nous avons implémentée est la méthode de *Semi-global Block Matching* .

La méthode Semi-Global Block Matching (SGBM) est une technique de mise en correspondance stéréoscopique utilisée pour estimer la disparité entre les images stéréo. Elle consiste à comparer les blocs de pixels entre les images gauche et droite, en utilisant des mesures de similarité telles que la somme des différences absolues (SAD) ou la somme des différences au carré (SSD), afin de trouver les correspondances les plus probables. Cette méthode prend en compte les contraintes globales pour améliorer la précision de la correspondance.

Cette méthode consiste à considérer une fenêtre de taille arbitraire ($N = 8$ dans les exemples en Figure 2) autour du pixel dont on cherche à trouver la disparité.



(a) Cones (b) Teddy

FIGURE 7 – cartes de disparité SGBM

Dans la figure Teddy nous voyons clairement qu'il y a une partie bizzare dans la zone du toit à cause du changement de luminosité. Pour une résistance accrue aux changements de luminosité, il nous faut utiliser l'agrégateur ZNSSD, qui compare non pas directement les valeurs d'intensité, mais une version normalisée et centrée sur la fenêtre de ces dernières.

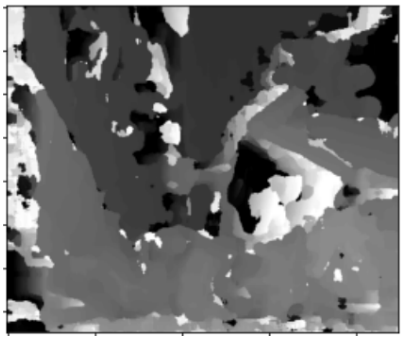


FIGURE 8 – SGBM Teddy avec ZNSSD

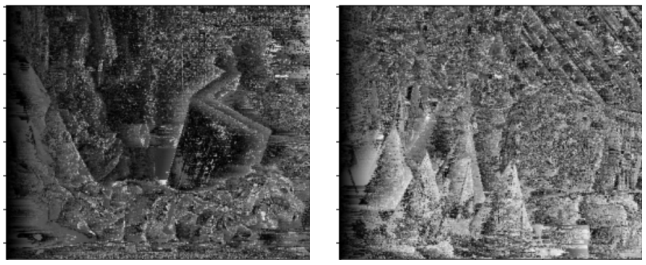
Cependant, on comprends pas pourquoi en utilisant ZNSSD ce que l'on obtient est pier qu'avant.

4.2 Programmation dynamique

La deuxième méthode que nous avons essayé d'implémenter est la méthode de *Programmation dynamique* (window_size = 1).

La mise en correspondance globale utilisant la programmation dynamique est une technique de stéréovision pour estimer la disparité entre une paire d'images stéréo. Elle consiste à calculer une carte de disparité complète en utilisant la programmation dynamique pour trouver la meilleure correspondance pixel par pixel. Le principe consiste à minimiser une fonction de coût globale qui mesure la dissimilarité entre les pixels correspondants. Cela se fait en utilisant une relation de récurrence pour calculer le coût cumulatif à partir des pixels voisins, ce qui permet de prendre en compte le contexte local. La mise en correspondance globale par programmation dynamique offre une précision élevée mais est computationnellement intensive.

Malheureusement, on n'a pas réussi à le bien interpréter à cause de temps limit :



(a) Teddy (b) Cones

FIGURE 9 – cartes de disparité DP

5 Conclusion

Références

[1] L. De-Maeztu, S. Mattoccia, A. Villanueva, and R. Cabeza. Linear stereo matching. In *2011 International Conference on Computer Vision*, pages 1708–1715, Barcelona, Spain, Nov. 2011. IEEE.

[2] S. Forstmann, Y. Kanou, Jun Ohya, S. Thuering, and A. Schmitt. Real-Time Stereo by using Dynamic Programming. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pages 29–29, Washington, DC, USA, 2004. IEEE.

[3] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann. Local stereo matching using geodesic support weights. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 2093–2096, Cairo, Egypt, Nov. 2009. IEEE.

[4] Kuk-Jin Yoon and In So Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4) :650–656, Apr. 2006.

[5] M. Lhuillier and Long Quan. Robust dense matching using local and global geometric constraints. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 1, pages 968–972, Barcelona, Spain, 2000. IEEE Comput. Soc.

[6] J. Li, P. Wang, P. Xiong, T. Cai, Z. Yan, L. Yang, J. Liu, H. Fan, and S. Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16263–16272, 2022.

[7] L. Lipson, Z. Teed, and J. Deng. Raft-stereo : Multilevel recurrent field transforms for stereo matching. In *2021 International Conference on 3D Vision (3DV)*, pages 218–227. IEEE, 2021.

[8] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards robust monocular depth estimation : Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3), 2022.

[9] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47 :7–42, 2002. Publisher : Springer.

[10] R. Szeliski. *Computer vision : algorithms and applications*. Springer Nature, 2022.

A Annexe

Après avoir remarqué que la quasi-totalité des algorithmes classés à l'évaluation Middleburry étaient basé sur du deep learning, nous avons voulu essayer par curiosité quelques-uns de ces modèles.

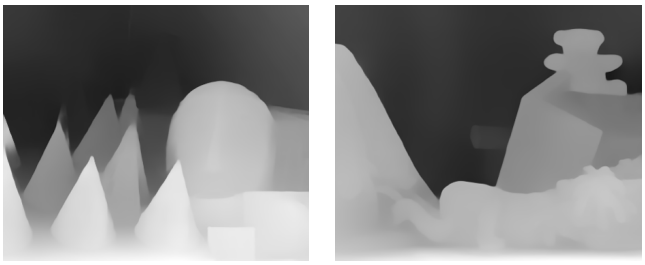
Les modèles que nous avons essayé sont à la fois plus rapides et beaucoup plus précis que les algorithmes plus classiques n'utilisant pas de deep learning : cette technologie semble très intéressante pour l'estimation de profondeur, que ce soit pour avoir une précision quasi parfaite avec une paire d'images stéréo ou pour estimer une carte de disparité à partir d'une seule image.

(a) CREStereo[6] (b) RAFT-Stereo[7]

FIGURE 10 – carte de disparités pour *cones* calculée via deep learning

(a) CREStereo[6] (b) RAFT-Stereo[7]

FIGURE 11 – carte de disparités pour *teddy* calculée via deep learning



(a) *cones* (b) *teddy*

FIGURE 12 – carte de disparités estimées à partir d'une seule image par [8]

Afin de tester rapidement, nous avons utilisé des poids pré-entraînés des modèles précédents. Comme nos images de test (*cones* et *teddy*) sont dans un jeu de données classique de stéréo-vision, il est fort probable qu'elles aient servi à l'entraînement : pour interpréter les résultats, prudence est de mise.