



Université
de Lille
1 SCIENCES
ET TECHNOLOGIES

Université de Lille

MASTER 2 INFORMATIQUE
Premier semestre



**FACULTÉ
DES SCIENCES ET
TECHNOLOGIES**
Département Informatique

Vision artificielle

TP n°03

Stéréovision dense

BARCHID Sami

Année académique 2019-2020

Introduction

La matière vue dans ce TP traite de la stéréovision dense, et plus précisément, nous verrons la minimisation d'une fonction de dissimilarité calculée sur les images d'un stéréoscope.

En résumé, ce que nous ferons :

- Créer une carte des disparités sur les images émises par chaque caméra au moyen d'une fonction de dissimilarité (une première fois avec l'image de gauche comme référence, puis l'image de droite)
- Vérifier la cohérence des cartes de disparités pour voir quand la disparité a été calculée de manière correcte pour une position donnée.

Stéréovision dense

Dans le rapport précédent, nous avons appréhendé la stéréovision épars, c'est-à-dire la reconstruction de la 3ème dimension de la scène prise par un stéréoscope en utilisant uniquement quelques points caractéristiques. Ce type de stéréovision a l'avantage de réduire le volume de données à traiter tout en ayant des performances liées à celles du détecteur de points d'intérêt utilisé.

Ici, le type de stéréovision est différent : nous utilisons la stéréovision dense qui consiste à produire une « carte de profondeur » capable de donner la 3ème coordonnée de profondeur à tout point. Ce type de stéréovision, quant à lui, possède un coût de calcul élevé du fait que le volume de données à traiter est plus grand.

Comme pour la stéréovision épars, la stéréovision dense comprend deux étapes principales :

1. Récupérer des primitives dans les deux images (celle de la caméra gauche et celle de la caméra droite) et les apparier

2. Reconstruire la 3ème dimension sur base de ces appariements

Dans ce TP, nous verrons uniquement la première étape, à savoir l'extraction des primitives des images du stéréoscope et leur appariement.

Les parties vues dans ce rapport sont les suivantes :

- **Minimisation de la dissimilarité:** résumé du procédé qui est utilisé pour ce TP et description de l'expérience.
- **Similarité par SSD :** explications sur l'implémentation du calcul de la carte de la disparité avec une des deux images comme référence en utilisant le SSD comme fonction de dissimilarité.
- **Vérification gauche-droite :** explications sur l'implémentation de la vérification des cohérences entre les cartes de disparités avec les images de gauche ou de droite comme référence.
- **Calcul efficace du SSD :** exploration d'une méthode récursive inspirée d'un rapport technique de l'INRIA pour optimiser le calcul du SSD.

Minimisation de dissimilarité

La manipulation présentée dans ce rapport est relative à la première grande étape réalisée dans la stéréovision dense : l'extraction de primitive dans les images du stéréoscope et leur appariement. Cette manipulation est nommée « la minimisation de dissimilarité ».

Contraintes

L'utilisation de la minimisation de dissimilarité implique d'abord plusieurs contraintes. Nous ferons référence à ces contraintes dans la suite du TP.

- Le stéréoscope doit être en **configuration canonique**
- **Contrainte d'unicité** : chaque pixel de l'image de référence est associée au maximum à un pixel de l'autre image.
- **Contrainte d'ordre** : deux paires de points (un sur l'image de référence et un sur l'autre image) doivent être présentés dans le même ordre d'abscisse dans les deux images.
 - Exemple : un pixel ne peut pas être situé à gauche de son homologue dans l'image de gauche et à droite de son homologue dans l'autre image. Le pixel doit être aussi situé à gauche dans l'autre image.
- **Contraintes liées à la photométrie** : les intensités des pixels mesurés sur les deux images doit être située dans la même plage de valeur. Les niveaux de gris dans une des images ne peuvent pas être fort différents par rapport aux niveaux de gris des points homologues dans l'autre image.

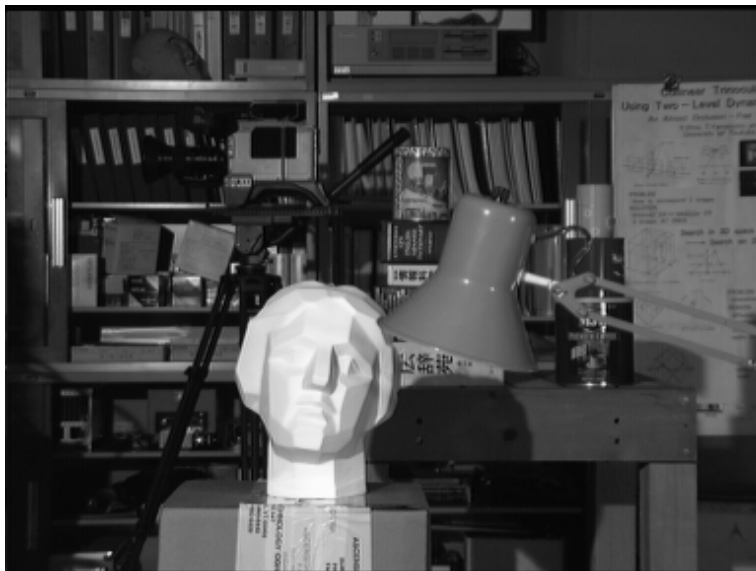
Résumé de la méthode

La méthode peut se résumer grossièrement en plusieurs points. Ces points seront repris dans les prochaines parties du rapport :

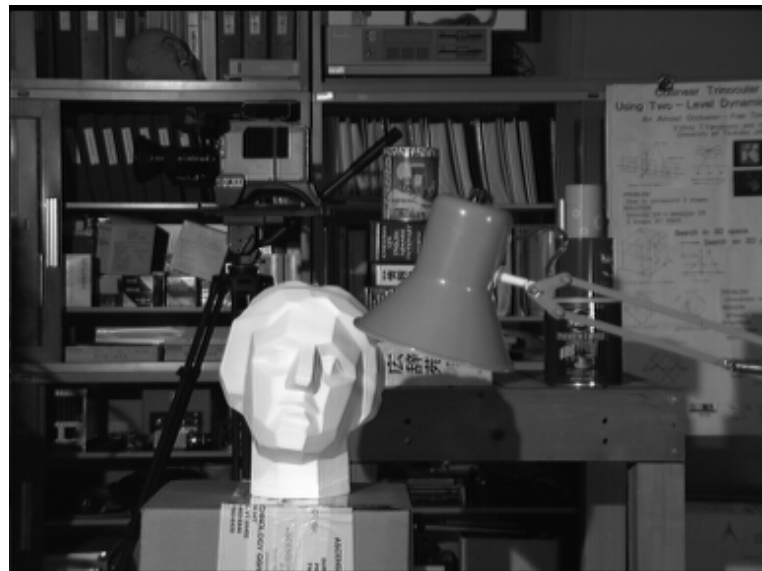
- **Calculer** la carte des disparités en prenant l'image de gauche comme référence.
- **Calculer** la carte des disparités en prenant l'image de droite comme référence.
- **Vérifier** que les paires de pixels homologues estimées sont identiques grâce à la **cohérence gauche-droite**.

Présentation de l'expérience

Afin de tester cette méthode, nous allons tester notre implémentation sur les images suivantes. Ces images sont celles prises de la même scène par un stéréoscope à configuration canonique (et qui respecte toutes les contraintes mentionnées précédemment).



Gauche



Droite

Similarité par SSD

La première étape pour l'implémentation de la méthode est de calculer la carte de disparité. Dans cette partie du TP, c'est l'implémentation de cette première étape qui sera abordée.

La carte de disparité est recherchée par un l'algorithme dont les entrées et sorties sont :

Entrée

- Image **gauche** avec $I_l(x_l, y_l)$ est le niveau de gris du pixel (x_l, y_l)
- Image **droite** avec $I_r(x_r, y_r)$ est le niveau de gris du pixel (x_r, y_r)
- Le choix d'une des deux images que l'on prendra comme « image de référence ». Ici, nous prendrons l'image de gauche
 - La carte de disparité sera alors précisée comme « avec gauche comme référence ».
- w_x et w_y , les tailles de la fenêtre du voisinage qui sera centré sur chaque pixel (cette notion sera éclaircie plus loin).
- s_{max} un décalage maximal possible

Sortie

La carte des disparités avec l'image de gauche comme référence.

La carte de disparité qui sera en sortie de cet algorithme est donc une matrice reprenant, pour chaque pixel, le décalage $s \in [0, s_{max}]$ qui permet d'obtenir un résultat pour la fonction de dissimilarité le plus faible possible.

La fonction de dissimilarité utilisée est le SSD (sum of squared differences) :

$$\text{SSD}(x_l, y, s) = \sum_{i=-w_x}^{w_x} \sum_{j=-w_y}^{w_y} \left(I_l(x_l + i, y + j) - I_r(x_l + i - s, y + j) \right)^2$$

Cette fonction calcule une erreur entre la fenêtre de pixels de l'image gauche (prise comme référence ici) et la fenêtre de pixels de l'image droite pour laquelle un décalage s sur les abscisses a été fait.

Le calcul de la carte de disparité consiste donc, pour chaque décalage $s \in [0, s_{\max}]$, à calculer le SSD pour chaque pixel afin d'obtenir le décalage minimal à la fin.

Nous pouvons, avec ceci, affirmer déjà l'utilité des contraintes que nous avons posées juste avant :

- **Configuration canonique** : les deux pixels homologues dans l'image gauche et droite se retrouve à la même coordonnée y (on n'introduit pas de décalage pour y). La seule chose qui varie est donc l'abscisse, ce qui est logique vis-à-vis de la contrainte épipolaire.
- **Contrainte d'ordre** : la recherche du décalage s minimum montre que l'on recherche, à chaque fois, le SSD avec un pixel de l'image droite en faisant décaler la fenêtre de s dans l'image droite dans les abscisses. Si un point se retrouvait à gauche et non à droite dans l'image droite, nous ne pourrions jamais retrouver un décalage pertinent.
- **Contrainte de photométrie** : de par le calcul du SSD qui vérifie les niveaux de gris des deux images, si nous ne captions pas la même intensité pour les deux images, la dissimilarité sera alors trop grande.

Implémentation

L'implémentation de l'algorithme pour trouver la carte de disparité (ici, avec l'image de gauche comme référence) est implémentée de cette manière :

```

// -----
/// \brief Estime la disparite par minimisation du SSD, image gauchee
/// prise comme reference.
///
/// @param mLeftGray: image gauche
/// @param mRightGray: image droite
/// @param iMaxDisparity: disparite maximale recherchee
/// @param iWindowHalfSize: demi-taille de la fenetre de correlation
/// @return image des disparites
// -----
Mat iviLeftDisparityMap(const Mat& mLeftGray,
                        const Mat& mRightGray,
                        int iMaxDisparity,
                        int iWindowHalfSize) {
    // Applique la formule du SSD

    // Image resultat
    Mat mLeftDisparityMap(mLeftGray.size(), CV_8U);

    // Initialisation de l'image du minimum de SSD a la valeur maximale
    // pouvant etre obtenue sur une fenetre carree
    Mat mMinSSD = Mat::ones(mLeftGray.size(), CV_64F) *
        pow((double)(2 * iWindowHalfSize + 1), 2.0) * pow(255.0, 2.0);

    // Boucler pour tous les decalages possibles
    for (int iShift = 0; iShift < iMaxDisparity; iShift++) {
        // Calculer le cout SSD pour ce decalage
        Mat mSSD = iviComputeLeftSSDCost(mLeftGray, mRightGray,
                                         iShift, iWindowHalfSize);
        // Conserver le decalage sur les pixels ou le SSD est minimum
        mLeftDisparityMap.setTo((unsigned char)iShift, mSSD < mMinSSD);
        // Et mettre a jour l'image du SSD minimum
        mSSD.copyTo(mMinSSD, mSSD < mMinSSD);
    }
    return mLeftDisparityMap;
}

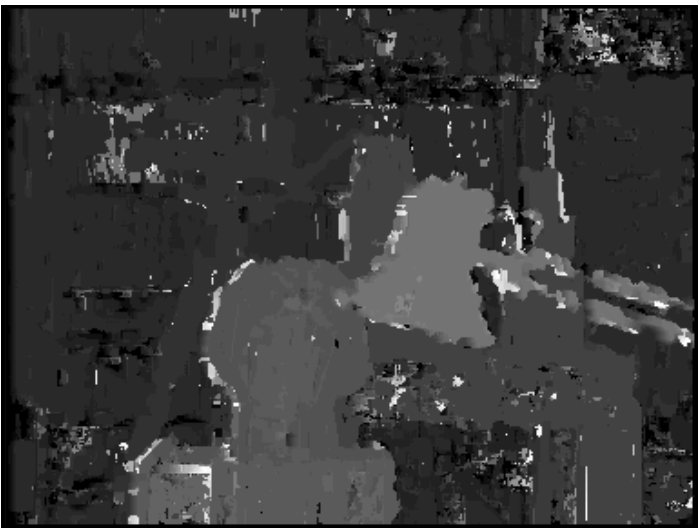
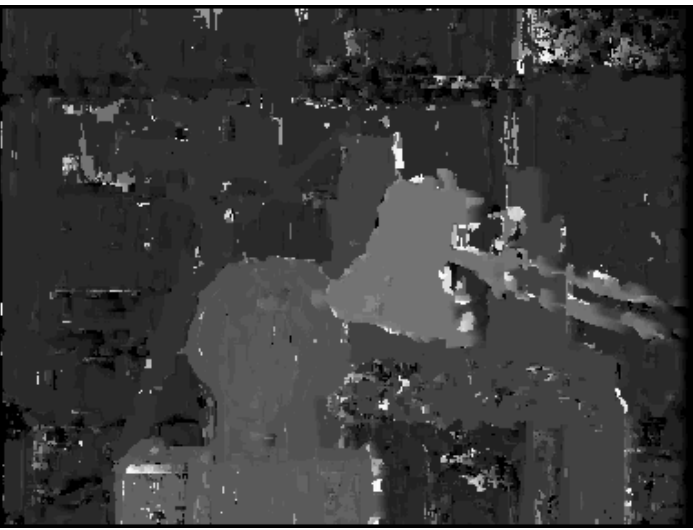
```

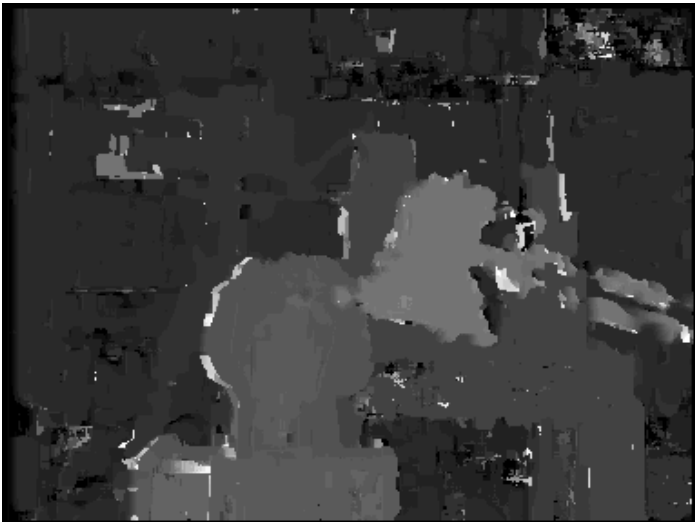

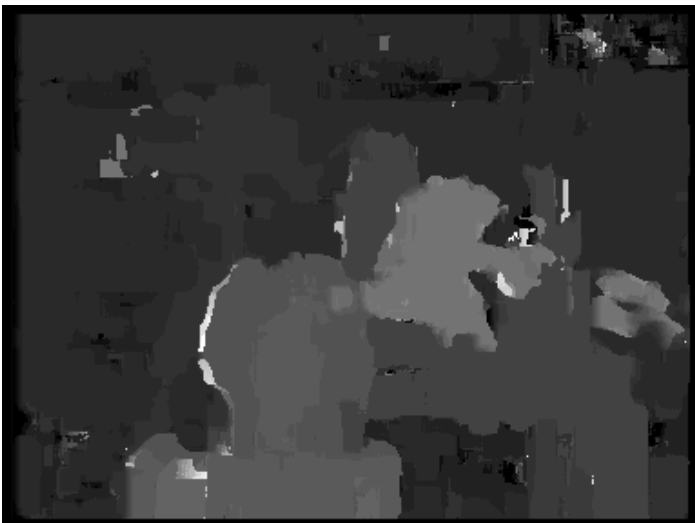

- On boucle sur chaque décalage, de 0 à s_{max} (ici représenté par la variable *iMaxDisparity*).
- La méthode « **iviComputeLeftSSDCost** » permet de calculer le SSD de l'image de gauche en référence en prenant un décalage en paramètre.
- La variable **mSSD** contient, à chaque itération sur le décalage, le SSD calculé par la méthode « **iviComputeLeftSSDCost** ».
- La variable **minSSD** permet de retenir, pour chaque pixel, la valeur de SSD pour un décalage s qui était minimale pour ce pixel.

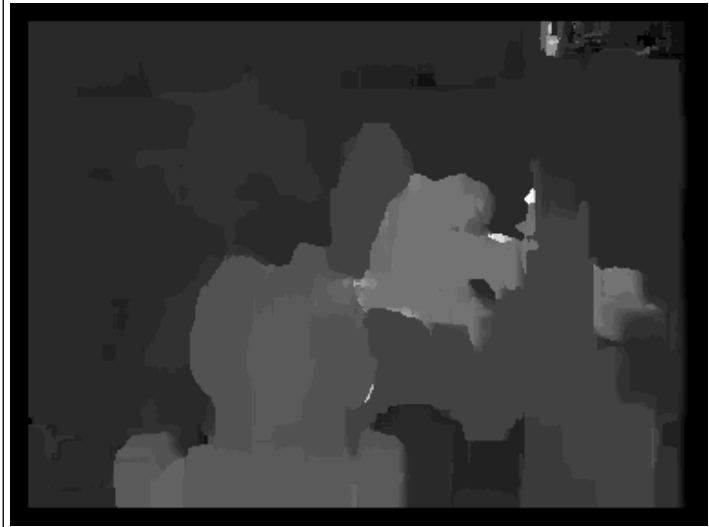
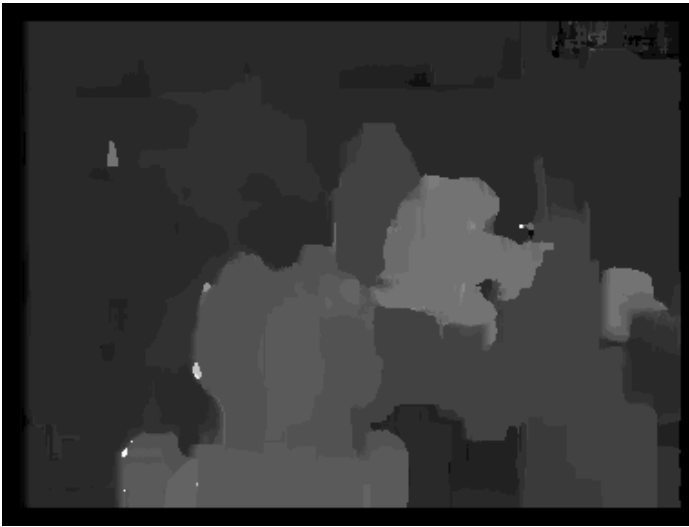
- Cette variable est mise à jour à chaque itération sur le décalage pour mettre à jour toutes les valeurs supérieures au ***mSSD*** trouvé.
- La variable ***mLeftDisparityMap*** est la carte de disparité en sortie et contient, pour chaque pixel de l'image de gauche, le décalage s nécessaire pour obtenir le SSD minimal.
 - Cette variable est mise à jour d'une manière similaire à ***minSSD***

Tests sur l'expérience

Nous pouvons alors tester les cartes de disparités trouvées en faisant varier le décalage maximal autorisé (s_{max}).

s_{max}	Image gauche comme référence	Image droite comme référence
2		

3		
5		



Comment obtenir ces images ?

L'algorithme présenté précédemment permet d'obtenir les cartes de disparités (une matrice composée des shifts qui donnent un SSD minimal). Ces images n'auraient donc pas dû être possibles avec ces données. C'est pour cela que les valeurs de la carte de disparité ont été « normalisées » sur des valeurs entre 0 et 1 suivant la disparité minimal et maximal. Cette normalisation a été faite au moyen de la fonction ***normalize***.

L'effet que cela nous pouvons constater avec cela est le fait que des valeurs de niveaux de gris très claires dans l'image normalisée correspond à une grande disparité et le contraire pour les niveaux de gris faibles.

Nous pouvons interpréter alors que plus un pixel est clair, plus le pixel représente un point de la scène qui est proche de la caméra (étant donné qu'une grande disparité implique un avant-plan).

Analyse des résultats

- On remarque que, même avec un s_{max} réduit, le résultat n'est jamais aux attentes d'un résultat « parfait ».

- Il subsiste un « cadre noir » tout autour de l'image d'une carte de disparité. Ce « cadre » est dû au fait que la fenêtre du voisinage ne pouvait pas calculer le SSD sur les extrémités des images.
- Un grand s_{max} tend à montrer des « formes » moins parfaites au sein des mêmes gammes de disparités. Par exemple, nous voyons que la lampe est « floutée » pour un $s_{max}=10$. Ceci est dû au fait que, le décalage étant grand, le calcul prendra plus de pixels en compte, dont certains qui n'ont rien à voir avec le point de la scène à la base.
- La dernière observation est une observation de temps de calcul : lors de la création des résultats, un temps de latence de quelques secondes a été remarqué afin que le programme ne fournisse de résultat. Cette « latence » démontre bien que cette technique est lourde en calculs.

Vérification gauche – droite

Dans cette partie du TP, nous précisons la deuxième étape dans l'appariement des paires de points homologues dans la stéréovision dense. Ici, nous utiliserons la vérification gauche-droite.

La vérification gauche-droite permet de vérifier les pixels homologues entre les deux images gauche et droite. Cette vérification permet aussi d'éliminer les faux appariements causés par les occultations.

L'algorithme de vérification gauche droite définit les entrées et sorties suivantes :


Entrées



- La carte de disparités avec l'image de gauche pour référence
- La carte de disparités avec l'image de droite pour référence

Sortie

- L'image d'un masque de validité qui consiste en une image qui contient :
 - Des pixels noirs (valeur 0) lorsque la disparité a été calculée correctement pour la position donnée par le pixel
 - Des pixels blancs (valeur 255) lorsque la disparité n'a pas été calculée correctement pour la position donnée par le pixel.

Ainsi, avec la vérification gauche-droite, nous obtenons, pour les différents décalages maximums définis dans la partie précédente :

S_{max}	Masque de validité
2	
3	

5	
10	

On remarque plusieurs choses :

- On remarque que, plus le décalage maximal est réduit, plus la fenêtre d'occultation contient des points de calcul de disparités incorrectes. Cela est dû au fait que, comme on ne décale la fenêtre de voisinage que d'une distance courte, il est possible que les images gauches et

droites sont distantes au point que le décalage maximal choisi ne soit pas adéquat.

- Pour un décalage maximal un peu plus important, on peut commencer à décrire de très grossiers « contours » en tant que point blanc pour mentionner les erreurs (notamment sur des objets en avant-plan comme la lampe). Ceci s'explique par le fait que des points de l'image de gauche ne se retrouvent simplement pas dans l'image de droite (occultation aux contours).
- De grandes valeurs pour un décalage maximal engendre beaucoup moins de pixel marqués comme « erreur ». Cela pourrait venir du fait que d'autres points de l'image droite sont affiliés aux points de l'image gauche de manière erroné à cause de la fenêtre de voisinage du SSD qui finissait par « aller trop loin » à cause de son grand décalage.

Calcul efficace du SSD

Soit l'équation du SSD

$$\text{SSD}(x_l, y, s) = \sum_{i=-w_x}^{w_x} \sum_{j=-w_y}^{w_y} \left(I_l(x_l + i, y + j) - I_r(x_l + i - s, y + j) \right)^2$$

L'intention de la méthode de calcul efficace introduite dans le rapport technique <http://master-ivi.univ-lille1.fr/fichiers/Cours/VisA-semaine-3-techreport-faugeras.pdf> de l'INRIA est de supprimer ces deux « boucles for » (

$\sum_{i=-w_x}^{w_x}$ et $\sum_{j=-w_y}^{w_y}$) et de les remplacer par deux équations de récurrences.

Malheureusement, par manque de temps, il m'a été impossible de compléter les calculs pour définir cette méthode plus efficace.

Conclusion

Ce TP m'a permis de comprendre et de manipuler une partie de l'étape de la stéréovision dense, à savoir la recherche des points homologues grâce à une carte de disparité dense.

Il a de plus été possible de tester et comprendre ce qu'impliquait la variation du paramètre de la distance de décalage maximale dans les algorithmes présentés.

Néanmoins, si on veut voir un défaut dans la méthode, le nombre de contraintes que l'on doit poser sur notre stéréoscope est très élevé, ce qui pourrait entraîner un questionnement sur la faisabilité de cette méthode dans des contextes de prise d'image peu contrôlés.