M3DA - TP 2 : mise en correspondance stéréoscopique

François Lepan

28 septembre 2013

Introduction

Dans ce rapport nous allons voir comment faire pour reconstruire en 3D certains points d'une scène à partir de deux images de cette scène possédant un point de vue différent.

Pour ce faire nous allons passer par plusieurs étapes :
- le calcul de la matrice fondamentale,
- l'extraction des "coins",
- le calcule des distances,
- Et enfin il faut faire la mise en correspondance des points des deux images.

1 Calcul de la matrice fondamentale

La matrice fondamentale est une matrice qui permet de passer d'un point d'une image de la scène à une droite sur l'autre image de cette scène appelé droite épipolaire.

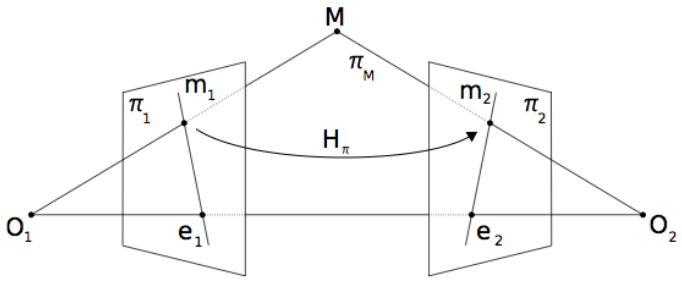

Figure 1 – Schéma matrice fondamentale

Sur la Fig. 1 la matrice fondamentale F est la matrice qui permet de passer du point m_1 à la droite e_2m_2.

Cette matrice nous donnant pour un point de l'image de gauche une droite épipolaire sur l'image de droite (et inversement) il va falloir déterminer quel point sera le meilleur candidat.

Pour ce faire nous allons voir comment tout d'abord trouver des points particulier permettant de facilement les mettre en correspondance sur chaque image.

2 Extraction des coins

Afin de mettre en correspondance pixels d'intérêt dans les deux images nous allons extraire les "coins" grâce à une méthode proposé par *Jianbo Shi et Carlo Tomasi* en 1994. On appel ces pixels d'intérêt "coins" car ils correspondent principalement aux pixels situés à l'intersection de deux lignes de pixels d'une même couleur.

Pour ce faire nous utiliserons une méthode de la librairie openCV appelé *goodFeaturesToTrack* qui s'occupe de trouvé ces coins.

```
void goodFeaturesToTrack(InputArray image,
                         OutputArray corners,
                         int maxCorners,
                         double qualityLevel,
                         double minDistance,
                         InputArray mask=noArray(),
                         int blockSize=3,
                         bool useHarrisDetector=false,
                         double k=0.04 )
```

Cette méthode prend en paramètre une image à analyser, un tableau à remplir, une distance minimum entre chaque points trouvé et d'autres paramètres permettant d'affiné la recherche.

Après utilisation de cette méthode on se retrouve avec nos points d'intérêts pour chaque images. Reste la mise en correspondance entre les points de l'image de gauche et l'image de droite.

3 Calcul des distances

Une première étape de mise en correspondance consiste à calculer les distances :

- pour chaque droites épipolaires issue des points de l'image de gauche les distances entre ces droites et tous les points de l'image de droite que l'on va stocker dans une matrice m1,
- pour chaque droites épipolaires issue des points de l'image de droite les distances entre ces droites et tous les points de l'image de gauche que l'on va stocker dans une matrice m2,
- enfin il ne reste qu'a additionner chaque distance D correspondante : $m = m1 * m2^t$.

Explication pour deux points quelconque :

```
A point image de gauche.
B point image de droite.
Ad droite épipolaire du point A
Bd droite épipolaire du point B

d1 = distance entre A et Bd
d2  = distance entre B et Ad

d = d1 + d2;
```

Maintenant que nous avons les distances il ne reste qu'a trouver les correspondances entre ces points.

4 Mise en correspondance

La mise en correspondance consiste à dire qu'un point A de l'image de gauche à le plus de chance d'être un point B de l'image de droite.

Pour ce faire nous allons procéder de la façon suivante :

```
// mDistances = matrice des distances
// dMaxDistance = distance maximale autorisant une association

// nombre de tour a effectuer
int nbTurn = 0;
int index = 0;

// ces deux listes sont initialisee avec des -1
// liste des correspondants des points gauche
mRightHomologous = initMatrix(mDistances.cols);
// liste des correspondants des points droite
mLeftHomologous = initMatrix(mDistances.rows);

// calcule du nombre d'association a effectuer :
// le plus petit nombre de point trouve pour une image
// car on ne peut avoir qu'une association par point
if (mDistances.rows > mDistances.cols)
        nbTurn = mDistances.cols;
else
        nbTurn = mDistances.rows;

// les indices des points a associer
int x;
int y;

// tant qu'il reste des points a associer
while (index < nbTurn) {
        // si on a trouvé une association
        if (findMinValue(mDistances,x,y,dMaxDistance)) {
        // on associe les points
        mRightHomologous.at<int>(x) = y;
        mLeftHomologous.at<int>(y) = x;
        // on empêche d'autres association pour ces deux points
        removeLigneColFrom(mDistances,x,y);
        }
        index++;
}
```

À la sortie de cette algorithme on obtient deux vecteur contenant pour chaque point son correspondant et si ce point n'a pas de correspondant alors on à la valeur -1.

Example :

```
mRightHomologous = [4; -1; 1; 9; 3; -1; 5; 7; -1; 8]
mLeftHomologous =  [-1; 2; -1; 4; 0; 6; -1; 7; 9; 3]
```

Conclusion

Cette méthode de mise en correspondance stéréoscopique via l'analyse de point d'intérêts des deux images fonctionne correctement. Mais notre approche peut être améliorée car on ne prend que la position géométrique des points. On pourrait prendre la colorimétrie des images afin d'affiner la recherche de correspondant.