Brusc Maxence Master 2 IVI

Vision Artificielle :

TP2 : Mise en correspondance stéréoscopique

Introduction :

Lors de ce TP, nous avons travaillé sur une méthode de mise en correspondance de points d'intérêts sur des images stéréoscopiques.
L'implémentation de cette méthode a été réalisée en plusieurs étapes :
Il faut d'abord calculer une matrice fondamentale qui permet de déterminer les droites épipolaires.
Ensuite il s'agit de trouver les points d'intérêts de l'image (comme les points contours par exemple), pour enfin calculer la distance de ces points par rapport aux droites épipolaires.
Ceci nous permettra de réaliser la mise en correspondance stéréoscopique.
Ce TP a pour but de nous faire comprendre cette méthode de mise en correspondance à travers la réalisation de ces différentes étapes.
Nous allons d'abord rappeler le fonctionnement de cette méthode puis nous verrons comment nous l'avons implémentée dans notre TP.
Le développement du travail a été effectué en C++ en utilisant la bibliothèque OpenCV. Les fonctions complétées sont jointes dans le document.

I) Mise en correspondance stéréoscopique :

Nous allons d'abord commencer par des rappels sur la mise en correspondance stéréoscopique. L'objectif de cette méthode est de reconstituer la troisième dimension qui a été perdu lors de la projection sur le plan image.
La reconstruction stéréoscopique est un procédé qui permet d'estimer par triangulation la troisième dimension. Ceci peut être fait en déterminant des paires de points homologues entre deux images. La reconstruction se fait en trois principales étapes :
-l'identification de points ou de primitives d'intérêt.
-la recherche de correspondance entre les primitives.
-la reconstruction 3D par triangulation.
La reconstruction peut être dense, si l'on cherche à déterminer toutes les paires de points et identifier les zones occultées, ou éparse si l'on s'intéresse seulement à quelques points.
Nous avons travaillé lors du tp sur une reconstruction éparse.

Nous allons donc maintenant nous intéresser aux étapes principales de cette méthode de mise en correspondance.

II) Calcul de la matrice fondamentale :

La première étape que l'on a abordé est le calcul de la matrice fondamentale. Il s'agit d'une matrice 3x3 qui permet de relier les points dans une image stéréoscopique.
Cette matrice va permettre par la suite de déterminer la droite épipolaire d'un point dans un plan opposé.
Une droite épipolaire est elle une droite placée sur la deuxième image qui contient notre point d'intérêt.

La première étape pour obtenir la matrice fondamentale était de calculer une matrice de transformation linéaire équivalente à un produit vectoriel.
Pour ce faire nous avons complété la fonction *iviVectorProductMatrix* comme suit :

```
Mat iviVectorProductMatrix(const Mat& v) {
        Mat mVectorProduct = Mat::eye(3, 3, CV_64F);
        mVectorProduct.at<double>(0,0) = 0;
        mVectorProduct.at<double>(0,1) = -v.at<double>(2);
        mVectorProduct.at<double>(0,2) = v.at<double>(1);
        mVectorProduct.at<double>(1,0) = v.at<double>(2);
        mVectorProduct.at<double>(1,1) = 0;
        mVectorProduct.at<double>(1,2) = -v.at<double>(0);
        mVectorProduct.at<double>(2,0) = -v.at<double>(1);
        mVectorProduct.at<double>(2,1) = v.at<double>(0);
        mVectorProduct.at<double>(2,2) = 0;
        return mVectorProduct;
}
```

Cette fonction a été complété suivant cette formule vue en cours :

$$\mathbf{p} \times \mathbf{q} = \mathbf{p}^{\times} \cdot \mathbf{q}, \text{ avec :}$$

$$\mathbf{p}^{\times} = \begin{pmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{pmatrix}$$

Afin de calculer la matrice fondamentale, on utilise la formule suivante :

$$\mathbf{F} = (\mathbf{P_2 O_1})^{\times} \mathbf{P_2 P_1^{+}}$$

(P2*O1) est la projection de O1 sur pi2, P2 et P1 sont elles les matrices de projections des plans droit et gauche. $P1^{+}$ désigne la pseudo inverse de la matrice P1.
Nous avons donc obtenu le code suivant pour la fonction *iviFundamentalMatrix* :

```
Mat iviFundamentalMatrix(const Mat& mLeftIntrinsic,
                         const Mat& mLeftExtrinsic,
                         const Mat& mRightIntrinsic,
                         const Mat& mRightExtrinsic) {
        Mat m = (Mat_<double>(3,4) <<
        1, 0, 0, 0,
```

```
        0, 1, 0, 0,
        0, 0, 1, 0
        );
        Mat p1 = mLeftIntrinsic * m * mLeftExtrinsic;
        Mat p2 = mRightIntrinsic * m * mRightExtrinsic;

        Mat o1 = mLeftExtrinsic.inv().col(3);
        Mat o2 = mRightExtrinsic.inv().col(3);

        Mat mFundamental = iviVectorProductMatrix(p2*o1) * p2 * p1.inv(DECOMP_SVD);
        return mFundamental;
}
```

La matrice fondamentale une fois calculée par cette fonction permet de déterminer les droites épipolaires utiles pour la mise en correspondance des points d'intérêts entre les images gauche et droite.
En testant notre programme on trouve les valeurs suivantes pour la matrice fondamentale :

```
-4.80201e-16, -1.87385e+01,  4.49723e+03
-1.87384e+01,  1.91536e-14,  3.05726e+05
 4.49722e+03, -2.93733e+05, -2.87822e+06
```

L'équation de la droite épipolaire de l'image droite associée au centre de l'image de gauche sera donc :
d2 = F * m1, avec F la matrice fondamentale et m1 la projection de M sur pi1 (m1 est au centre de l'image).

Pour la droite épipolaire de l'image gauche associée au centre de l'image de droite on aura donc :
d1 = F^+ * m2, avec F la matrice fondamentale et m1 la projection de M sur pi1 (m2 est au centre de l'image).

Enfin pour la droite épipolaire de l'image droite associée au point situé au centre du côté haut de l'image gauche on obtiendra :
d2 = F * m1, avec F la matrice fondamentale et m1 la projection de M sur pi1 (m1 est au centre du côté haut de l'image).

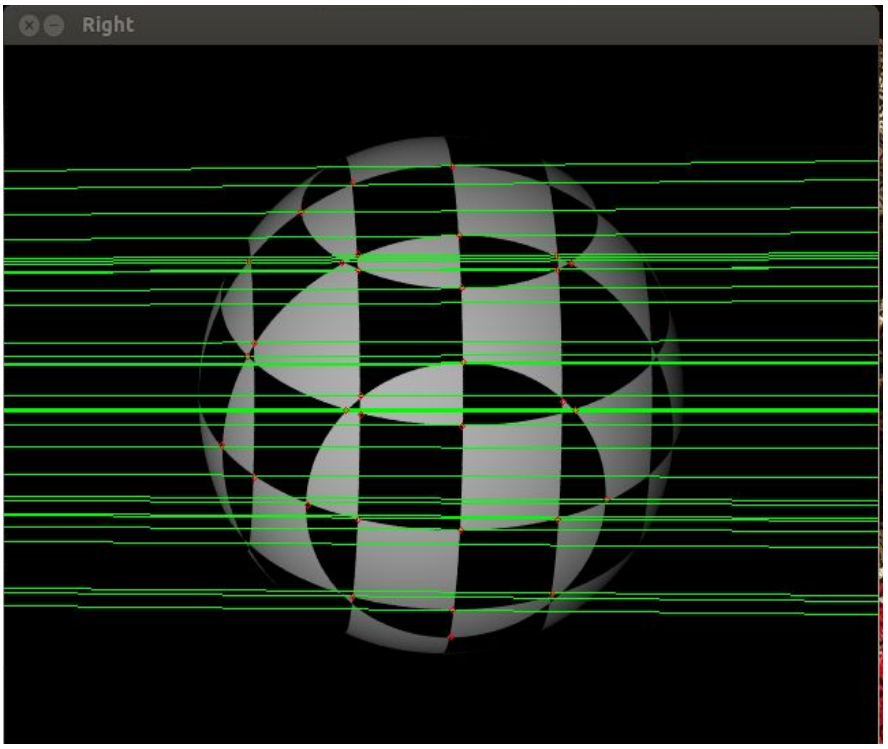
Le calcul de la matrice fondamentale, dont nous avons vu les étapes précédemment peut nous permettre de déterminer les droites épipolaires associées aux points des images.
Nous allons maintenant voir comment sélectionner des points intéressant pour faire la mise en correspondance stéréoscopique de notre image.

III) Extraction des coins :

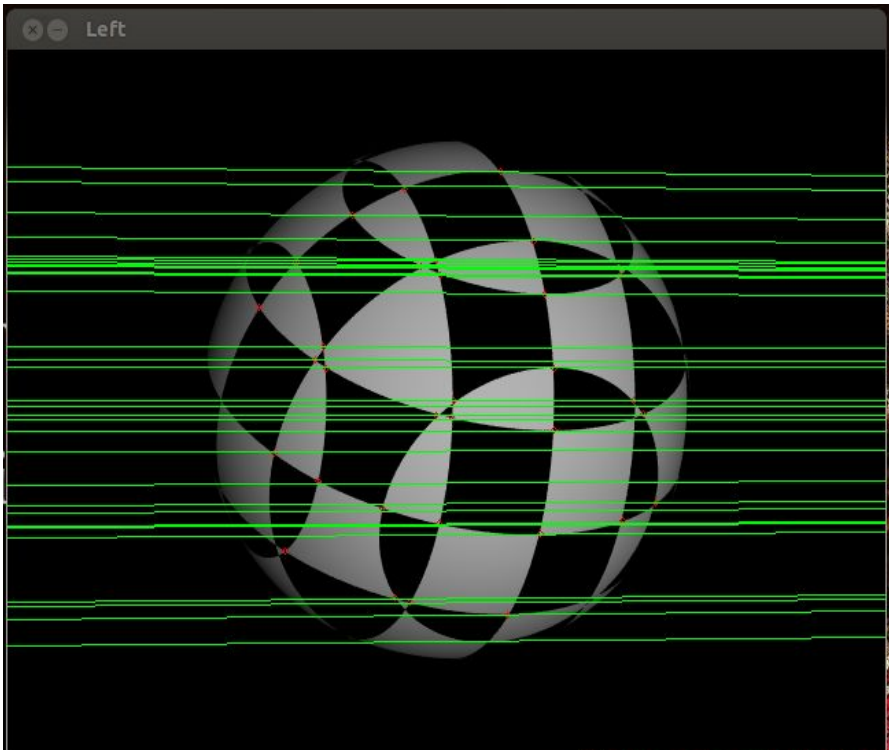
Nous allons mettre en évidence les coins de l'image qui sont des pixels d'intérêt entre les deux images. On utilise pour cela l'opérateur proposé par Shi et Tomasi implanté dans la méthode goodFeaturesToTrack.
Nous avons donc modifié la fonction *iviDetectCorners* pour effectuer la détection des points. En voici le code :

```
Mat iviDetectCorners(const Mat& mImage,
                     int iMaxCorners) {

          vector<Point2f> vCorners;
          double qualityLevel = 0.01;
          double minDistance = 10;
          int blockSize = 3;
          bool useHarrisDetector = false;
          double k = 0.04;

        // Apply corner detection
        goodFeaturesToTrack( mImage,vCorners,iMaxCorners,qualityLevel,minDistance,
                             Mat(),blockSize,useHarrisDetector,k );

          Mat mCorners(3,vCorners.size(),CV_64F);
          for(int i=0;i<vCorners.size();i++){
              mCorners.at<double>(0,i)=(double)vCorners[i].x;
              mCorners.at<double>(1,i)=(double)vCorners[i].y;
              mCorners.at<double>(2,i)=1;
          }
          return mCorners;
}
```

Ce code nous permet d'obtenir la détection des points. On peut en effet le constater sur les images ci-dessous.



Détection des coins dans l'image droite



Détection des coins dans l'image gauche

On peut remarquer que les points repérés sur les deux images sont les mêmes.

IV) Calcul des distances :

Après avoir extrait les points d'intérêts de nos deux images, il faut maintenant réaliser le calcul des distances entre les différentes paires de points. On fait ceci afin de trouver la meilleur correspondance possible entre toutes les paires. On dispose alors de l'ensemble des points et de leurs droites épipolaires.

Nous avons donc compléter le code de la fonction *iviDistancesMatrix* :

```
Mat iviDistancesMatrix(const Mat& m2DLeftCorners,
                       const Mat& m2DRightCorners,
                       const Mat& mFundamental) {

          int widthL = m2DLeftCorners.size().width;
          int widthR = m2DRightCorners.size().width;
          Mat mDistances(widthL, widthR,CV_64F);

          for(int i=0;i<widthL;i++){
            Mat pL = m2DLeftCorners.col(i);
            Mat epiL = mFundamental * pL;
          for(int j = 0; j < widthR; j++) {
           Mat pR = m2DRightCorners.col(j);
          Mat epiR = mFundamental.t() * pR;
         mDistances.at<double>(i,j) = ((abs(epiL.at<double>(0) * pR.at<double>(0) +
epiL.at<double>(1) * pR.at<double>(1) + epiL.at<double>(2)))
                                       /(sqrt(pow(epiL.at<double>(0),2) + pow(epiL.at<double>(1),2)))) +

((abs(epiR.at<double>(0)*pL.at<double>(0)+epiR.at<double>(1)*pL.at<double>(1)+epiR.at<double
>(2))) / (sqrt(pow(epiR.at<double>(0),2) + pow(epiR.at<double>(1),2))));
          }
          }
          // Retour de la matrice des distances entre points des paires
          return mDistances;
}
```

L'idée de notre fonction de calcul des distances est de calculer les distances entre un point avec l'ensemble des droites épipolaires dans l'image opposée. En cherchant la distance minimale, on pourra alors déterminer les correspondances entre les points.

Pour calculer la distance entre un point et sa droite, on utilise la formule suivante :

$$\frac{|ax_A + by_A + c|}{\sqrt{a^2 + b^2}}$$

où a, b et c correspondent aux x,y et z de la droite épipolaire et xA et yA correspondent aux coordonnées de notre point d'intérêt.
Une fois que cette distance est calculée, il nous reste à faire la somme des deux distances euclidiennes; la première étant celle entre le point de l'image gauche et la droite épipolaire de l'image gauche associée au point de l'image droite et la seconde correspondant à celle

entre le point de l'image droite et la droite épipolaire de l'image droite associée au point de l'image gauche.

V) Mise en correspondance :

Après avoir effectué le calcul de la matrice des distances, on doit maintenant travailler sur la mise en correspondance des points afin de définir les paires de points homologues entre les deux images stéréoscopiques.

Nous avons complété la fonction *iviMarkAssociations* afin de définir les paires de points homologues. Voici le code de cette dernière ci-dessous. Nous expliquerons ensuite le raisonnement adopté pour réaliser ces associations.

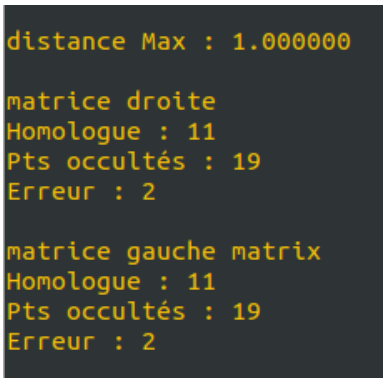
```
void iviMarkAssociations(const Mat& mDistances,
                         double dMaxDistance,
                         Mat& mRHomologous,
                         Mat& mLHomologous) {
        int width =mDistances.size().width, height =mDistances.size().height;

        int homologue = 0, occulte= 0;
        mRHomologous = Mat::eye(width, 1, CV_64F);
        mLHomologous = Mat::eye(height, 1, CV_64F);

        for(int i= 0; i< width ; i++) {
        int min = dMaxDistance;
        int minIndex = -1;
        for(int j= 0; j< height; j++) {
        if(min>mDistances.at<double>(i,j) && mDistances.at<double>(i,j)<dMaxDistance) {
        min = mDistances.at<double>(i,j);
        minIndex = j;
        }
        }
        mRHomologous.at<double>(i,0) = minIndex;
        }

        for(int i= 0; i< height ; i++) {
        int min = dMaxDistance;
        int minIndex = -1;
        for(int j= 0; j< width; j++) {
        if(min>mDistances.at<double>(j,i) && mDistances.at<double>(j,i)<dMaxDistance) {
        min = mDistances.at<double>(j,i);
        minIndex = j;
        }
        }
        mLHomologous.at<double>(i,0) = minIndex;
        }
        printf("\n\ndistance Max : %f\n",dMaxDistance);

        printf("\nmatrice droite \n");
        for(int i = 0; i<mRHomologous.size().height; i++) {
        if(mLHomologous.at<double>(mRHomologous.at<double>(i,0),0)==i)
```

```
        homologue++;
        else if(mRHomologous.at<double>(i,0)==-1)
        occulte++;
        //printf("%d : %f \n",i,mRHomologous.at<double>(i,0));
        }
        printf("Homologue : %d\n",homologue);
        printf("Pts occultés : %d\n",occulte);
        printf("Erreur : %d\n",mRHomologous.size().height-(homologue+occulte));

        int homologue2 = 0;
        int occulte2 = 0;
        printf("\nmatrice gauche matrix \n");
        for(int i = 0; i<mLHomologous.size().height; i++) {
        if(mRHomologous.at<double>(mLHomologous.at<double>(i,0),0)==i)
        homologue2++;
        else if (mLHomologous.at<double>(i,0)==-1)
        occulte2++;
        //printf("%d : %f \n",i,mLHomologous.at<double>(i,0));
        }
        printf("Homologue : %d\n",homologue2);
        printf("Pts occultés : %d\n",occulte2);
        printf("Erreur : %d\n\n",mRHomologous.size().height-(homologue2+occulte2));
}
```

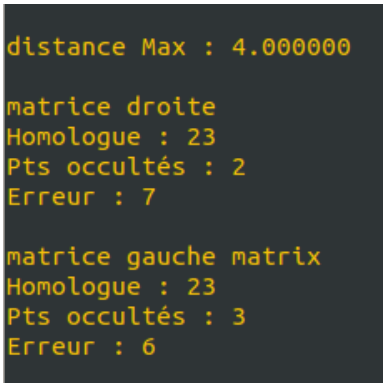
Nous allons donc décrire le raisonnement entrepris pour obtenir cette fonction de mise en correspondance de nos points d'intérêt.
Il faut d'abord parcourir la matrice des distances calculées dans la partie précédente. On récupère la valeur minimale, on garde en mémoire l'index pour lequel la distance est minimale (dans le tableau mRHomologous lors du travail sur la partie droite). On répète ensuite l'opération pour la partie gauche (le résultat sera stocké dans le tableau mLHomologous).

On dispose désormais d'une estimation de la correspondance entre les points des images gauche et droite.
Certains points peuvent n'avoir trouvé aucune correspondance. Ceci peut être dû au fait que ces points là sont occultés dans la deuxième image ou alors à un seuil trop faible pour la distance maximale.
Cependant, régler une distance maximale plus importante peut augmenter les mauvaises correspondances pour les points relativement proches.

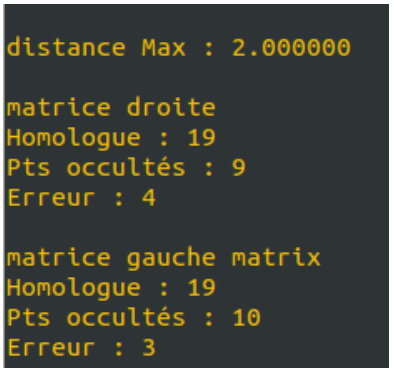
On peut constater ces erreurs avec les exemples ci-dessous.



Résultat pour une distance maximale de 1 (peu de correspondance)



Résultat pour une distance maximale de 4 (beaucoup d'erreur de correspondance)



Résultat pour une distance maximale de 2

Pour une distance maximale de 2, on peut retrouver 19 points homologues dans nos deux images, 9 n'ont pas trouvé de correspondances (ils sont occultés) et 4 montrent des incohérences (correspondance de plusieurs points de la première image sur un même point de la seconde image).
Si on abaisse ce seuil, on aura trop peu de points homologues mais les erreurs seront aussi diminuées. Au contraire, l'augmentation de cette distance maximale va permettre d'avoir plus de correspondance mais avec un risque d'erreur plus important.
Le seuil de 2 semble donc offrir un bon compromis entre le nombre d'erreur et le nombre de points homologues.

Conclusion :

Pour conclure, le travail effectué au cours de ce TP m'a permis de comprendre les différentes étapes nécessaire à la mise en correspondance stéréoscopique.
Dans un premier temps, on a pu découvrir comment déterminer une matrice fondamentale qui permet, une fois l'obtention des différents points (ici les coins), de déterminer les droites épipolaires associées à ceux ci.
Ces droites nous ont ensuite permis de trouver la correspondance entre les points des deux images après avoir effectué une mesure des distances entre ces points et les droites épipolaires.
Ce TP a pu me faire prendre conscience de la complexité à réaliser une bonne correspondance entre deux images stéréoscopiques.
À travers la réalisation de ce tp, j'ai aussi pu mieux comprendre les notions abordées dans le travail de mise en correspondance stéréoscopique.