Brusc Maxence Master 2 IVI

Vision Artificielle :

TP3 : Stéréovision dense

Introduction :

Nous avons effectué lors du TP précédent un travail sur la mise en correspondance stéréoscopique. Il s'agit de chercher la correspondance de certains points (comme les coins par exemple) de l'image droite à ceux présent dans l'image gauche. Par cette méthode, on peut donc mettre en correspondance les pixels des deux images.
Nous avons vu au cours de cette troisième séance comment déterminer la similarité entre les deux images.
Afin de calculer les similarités entre les images, on doit appliquer un décalage pour que l'on est dans les deux images la même partie visible. On peut ensuite calculer la différence entre les deux images.

Nous allons donc d'abord voir, dans ce rapport, la similarité par SSD. Ensuite, nous ferons une vérification gauche-droite pour trouver la disparité entre les deux images.

Les bouts de code modifiés durant notre travail seront joints dans leurs parties correspondantes.

I) Similarité par SSD :

Nous avons donc d'abord travaillé sur la similarité par SSD.

Après avoir analysé le code de la fonction *iviLeftDisparityMap* nous avons pu déterminer les informations contenues dans les images mSSD et mMinSSD.
-mSSD contient le coût SSD minimal pour un certain décalage.
-mMinSSD contient elle les valeurs des coût SSD minimaux pour les décalages dans une zone autour du pixel courant.

Les pointeurs sont calculés en parcourant les SSD pour tous les décalages possibles de la taille de la fenêtre de corrélation. Le plus petit est stocké en mémoire et on parcourt les décalages du plus important (taille de la fenêtre de corrélation) au décalage de zéro.

Nous avons ensuite compléter la fonction *iviComputeLeftSSDCost* afin de calculer l'indice de similarité constitué par la somme des différences au carré. Cette fonction parcourt les deux images et calcule la somme des différences.
Elle utilise la formule suivante :

$$\mathrm{SSD}(x_l, y, s) = \sum_{i=-w_x}^{w_x} \sum_{j=-w_y}^{w_y} \left(\mathbf{I}_l(x_l + i, y + j) - \mathbf{I}_r(x_l + i - s, y + j) \right)^2$$

Nous avons donc complété la fonction comme suit :

```
Mat iviComputeLeftSSDCost(const Mat& mLeftGray,
                    const Mat& mRightGray,
                    int iShift,
                    int iWindowHalfSize) {
          Mat mLeftSSDCost(mLeftGray.size(), CV_64F);

          for (int x = iWindowHalfSize ; x < mLeftGray.size().height-iWindowHalfSize; x++){
          for (int y = iWindowHalfSize; y < mLeftGray.size().width-iWindowHalfSize; y++) {
          for (int i = -iWindowHalfSize; i <= iWindowHalfSize; i++)
          for (int j = -iWindowHalfSize; j <= iWindowHalfSize; j++)
                    mLeftSSDCost.at<double>(x,y) += pow(mLeftGray.row(x+i).at<uchar>(y+j)-
mRightGray.row(x+i).at<uchar>(y+j-iShift), 2);
          }
          }
          return mLeftSSDCost;
}
```

L'image de référence est ici l'image gauche.
Pour chaque décalage possible dans notre fenêtre, on cale l'image droite avec ce décalage. On fait ensuite la différence entre la valeur du pixel de gauche par celle du pixel de droite. Cette différence est mise au carré. On répète le calcul pour chacun des pixels de la fenêtre afin d'obtenir la somme. Cette dernière est l'indice de similarité.

On obtient lors de l'exécution du code l'image suivante dans Left Disparity :

On peut reconnaître sur cette image les formes de l'image de base (lampe, statue). On peut donc déduire de cette image qu'il y a des différences entre les images gauches et droites même après l'application du décalage.
On peut remarquer sur l'image que les objets proches montrent plus de disparités :
Ceci peut être expliqué par le fait que plus un objet et proche plus l'angle de vision aura un effet sur ces objets. Ils seront donc démarquer du fond qui diffèrent moins.

Cette image a été rendu visible par les fonctions minMaxLoc et normalize qui vont normaliser l'image suivant les valeurs extrêmes de l'image (minimales et maximales). minMaxLoc permet d'extraire ces valeurs qui seront ensuite appelées dans la fonction normalize qui normalise l'image.

II) Vérification gauche-droite :

Dans cette seconde partie, il nous est d'abord demandé de compléter le code des fonctions *iviRightDisparityMap* et *iviComputeRightSSDCost*. Ces fonctions sont similaires aux fonctions déjà écrites lorsque l'image gauche est la référence.
On obtient donc dans ces fonctions le code suivant :

Pour *iviRightDisparityMap* :

```
Mat iviRightDisparityMap(const Mat& mLeftGray,
                         const Mat& mRightGray,
                         int iMaxDisparity,
                         int iWindowHalfSize) {
Mat mRightDisparityMap(mRightGray.size(), CV_8U);

// Images pour les resultats intermediaires
Mat mSSD(mRightGray.size(), CV_64F);
Mat mMinSSD(mRightGray.size(), CV_64F);
double dMinSSD, *pdPtr1, *pdPtr2;
unsigned char *pucDisparity;
int iShift, iRow, iCol;
        // Initialisation de l'image du minimum de SSD
        dMinSSD = pow((double)(2 * iWindowHalfSize + 1), 2.0) * 255* 255;
        for (iRow = iWindowHalfSize;
        iRow < mMinSSD.size().height - iWindowHalfSize;
        iRow++) {
        // Pointeur sur le debut de la ligne
        pdPtr1 = mMinSSD.ptr<double>(iRow);
        // Sauter la demi fenetre non utilisee
        pdPtr1 += iWindowHalfSize;
        // Remplir le reste de la ligne
        for (iCol = iWindowHalfSize;
        iCol < mMinSSD.size().width - iWindowHalfSize;
        iCol++)
        *pdPtr1++ = dMinSSD;
        }
        // Boucler pour tous les decalages possibles
```

```
        for (iShift = 0; iShift < iMaxDisparity; iShift++) {
        // Calculer le cout SSD pour ce decalage
        mSSD = iviComputeRightSSDCost(mLeftGray,mRightGray,  iShift, iWindowHalfSize);
        // Mettre a jour les valeurs minimales
        for (iRow = iWindowHalfSize;
        iRow < mMinSSD.size().height - iWindowHalfSize;
        iRow++) {
        // Pointeurs vers les debuts des lignes
        pdPtr1 = mMinSSD.ptr<double>(iRow);
        pdPtr2 = mSSD.ptr<double>(iRow);
        pucDisparity = mRightDisparityMap.ptr<unsigned char>(iRow);
        // Sauter la demi fenetre non utilisee
        pdPtr1 += iWindowHalfSize;
        pdPtr2 += iWindowHalfSize;
        pucDisparity += iWindowHalfSize;
        // Comparer sur le reste de la ligne
        for (iCol = iWindowHalfSize;
        iCol < mMinSSD.size().width - iWindowHalfSize;
        iCol++) {
        // SSD plus faible que le minimum precedent
        if (*pdPtr1 > *pdPtr2) {
                *pucDisparity = (unsigned char)iShift;
                *pdPtr1 = *pdPtr2;
        }
        // Pixels suivants
        pdPtr1++; pdPtr2++; pucDisparity++;
        }
        }
        }
        return mRightDisparityMap;
}
```

et pour *iviComputeRightSSDCost* :

```
Mat iviComputeRightSSDCost(const Mat& mLeftGray,
                const Mat& mRightGray,
                int iShift,
                int iWindowHalfSize) {
        Mat mRightSSDCost(mRightGray.size(), CV_64F);

        for (int x = iWindowHalfSize ; x < mRightGray.size().height-iWindowHalfSize; x++){

        for (int y = iWindowHalfSize; y < mRightGray.size().width-iWindowHalfSize; y++) {
        for (int i = -iWindowHalfSize; i <= iWindowHalfSize; i++){
        for (int j = -iWindowHalfSize; j <= iWindowHalfSize; j++) {
                mRightSSDCost.at<double>(x,y) += pow(mRightGray.row(x+i).at<uchar>(y+j)-
mLeftGray.row(x+i).at<uchar>(y+j-iShift), 2);
        }
        }
        }
        }
        return mRightSSDCost;
}
```

En exécutant ce code on obtient l'image suivante (à gauche) comme "disparité droite" que l'on peut comparer à la "disparité gauche" (à droite) :

Afin de valider le calcul des disparités, et de marquer les pixels n'ayant aucun homologue, on effectue une vérification gauche-droite en comparant les deux disparités.
Pour réaliser ceci, nous avons complété la fonction *iviLeftRightConsistency*.
Pour pouvoir compléter notre masque de validité ainsi que la carte des disparités (basé sur l'image gauche en référence), on compare les disparités droites et gauches.

Afin d'éliminer les faux appariements causés par les occultations, on va vérifier les disparités estimées.
Ces dernières doivent donc vérifier les conditions suivante :

$$\hat{d}_r(x_r, y) = \hat{d}_l(x_r + \hat{d}_r(x_r, y), y)$$
$$\hat{d}_l(x_l, y) = \hat{d}_r(x_l - \hat{d}_l(x_l, y), y)$$

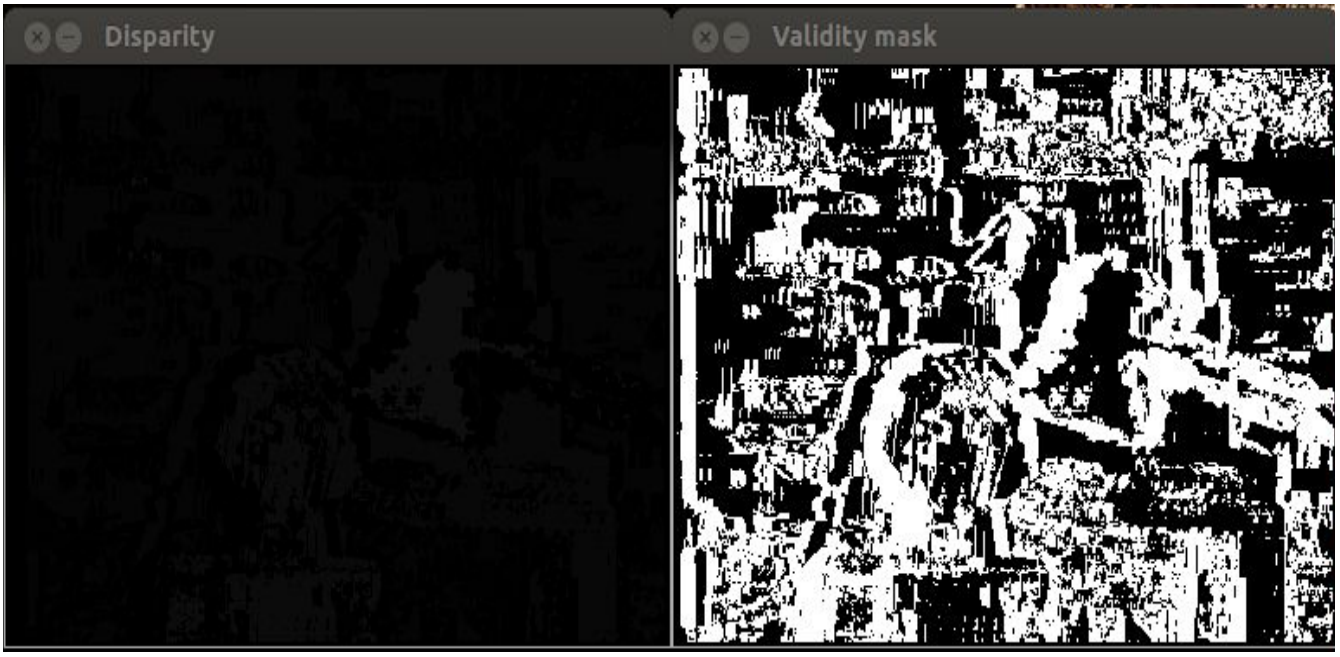
On se base ici sur la disparité gauche puisque l'image gauche reste notre référence.
Si la disparité n'a pas été calculé de façon correcte (suivant la deuxième formule) le masque prend la valeur 255 pour le pixel de coordonnées (xl,y).
Sinon le masque est mis à zéro et la disparité fusionnée prend la valeur de la disparité gauche.

On obtient alors le code suivant pour la fonction *iviLeftRightConsistency* :

```
Mat iviLeftRightConsistency(const Mat& mLeftDisparity,
                            const Mat& mRightDisparity,
                            Mat& mValidityMask) {
          Mat mDisparity(mLeftDisparity.size(), CV_8U);
           for (int i = 0 ; i < mLeftDisparity.size().height; i++)
           for (int j = 0; j < mLeftDisparity.size().width; j++) {
           if ((double)mLeftDisparity.at<uchar>(i,j) !=
(double)mRightDisparity.at<uchar>(i,j-(double)mLeftDisparity.at<uchar>(i,j)))
           mValidityMask.at<uchar>(i,j) = 255;
```

```
        else {
        mDisparity.at<uchar>(i,j) = (double)mLeftDisparity.at<uchar>(i,j);
        mValidityMask.at<uchar>(i,j) = 0;
        }
        }
        return mDisparity;
}
```

Suite à l'application de ce code, on obtient les résultats suivant :



On voit sur l'image de gauche la disparité fusionnée des disparités gauches et droites.
L'image de droite présente quand à elle le résultat du masque.
On peut remarquer grâce au masque quelles parties de l'image sont occultées par l'angle de vue différent. On voit que les objets proches présentent de nombreuses différences et sont plus impactés par le changement d'angle.

Conclusion :

Pour conclure, nous avons, durant ce tp, vu une deuxième manière de mettre en correspondance deux images ayant un point de vue différent. Lors de la précédente séance, nous nous intéressions aux points caractéristiques de l'image (les coins).
Ici, on essaye de mettre en correspondance tous les pixels de l'image en utilisant des décalages.
On peut déterminer par cette technique les points occultés entre l'image de référence et la seconde image ayant un point de vue différent.
Avec l'utilisation du SSD, on peut donc reconstituer les images avec la disparité en sachant quelles parties d'images sont différentes entre les deux angles de vue.