Brusc Maxence Master 2 IVI

Vision Artificielle :

TP1 : Éléments de géométrie projective et calibration de caméra

Introduction :

Lors de ce premier TP, nous devions nous familiariser avec les termes de calibration de caméra et d'éléments de géométrie projective.
Nous avons travaillé sur la méthode de calibration décrite par Zhang qui permet de calibrer une caméra grâce à une mire.
Après avoir étudié la publication de Zhang dans le but de nous familiariser avec les notations utilisés dans la méthode, nous avons travaillé sur le cas des caméras sans distorsion.
La première étape travaillée est l'estimation des paramètres intrinsèques de la caméra.
Nous avons ensuite travaillé sur l'estimation des paramètres extrinsèques de la caméra.
Ce TP a pour but de nous faire comprendre les différentes étapes de la calibration ainsi que l'utilité des matrices permettant de calculer la calibration (matrices d'homographie, intrinsèques et extrinsèques). Nous avons abordé le travail sous l'environnement Scilab. Les fichiers contenant le travail sont joints en annexe du document.

I) Étude de la publication de Zhang :

La première étape effectuée durant le TP a été l'étude de la publication de Zhang expliquant la méthode de calibration qu'il a mise au point. L'analyse de ce document nous a permis de nous familiariser avec les différentes notations utilisées afin de déterminer la calibration d'une caméra grâce à une mire.
Nous allons d'abord voir les principales étapes permettant cette calibration.

Dans sa publication, Zhang définit trois étapes pour calibrer une caméra :

- Dans un premier temps, il s'agit d'estimer pour chaque image une matrice d'homographie entre les points de la scène et les points projetés. Chaque homographie impose deux contraintes sur les paramètres intrinsèques de la caméra.

- Dans un second temps, on réalise l'estimation des paramètres intrinsèques de la caméra. La matrice intrinsèque permet de déterminer les données propres aux réglages optiques de la caméra (comme la distance focale par exemple).

- Enfin, il reste à estimer les paramètres extrinsèques pour chaque image. La matrice extrinsèque permet de déterminer le placement et la rotation de la caméra dans l'espace 3D.

Lorsque l'on analyse le code du script *zhang.sce*, qui nous est fourni, on peut constater que ces trois étapes sont bien réalisées.

```
// Boucler pour toutes les images
for i = 1:ni
  // Lire les points de l'image
  m(1:2,:,i) = read('points-'+string(i)+'.txt', -1, 2)';
  m(3,:,i) = ones(1, np);
  // Estimer l'homographie entre la mire et l' image
  H(:,:,i) = ZhangHomography(M(sansZ,:), m(:,:,i));
  // Ajouter deux lignes de contraintes dans V
  V = [V; ZhangConstraints(H(:,:,i))];
end

// Estimation de la matrice intrinseque
A = IntrinsicMatrix(b);
iA = inv(A);

// Estimations des matrices extrinseques
E = zeros(3, 4, ni);
for i = 1:ni
  E(:,:,i) = ExtrinsicMatrix(iA, H(:,:,i));
end
```

II) Estimation des paramètres intrinsèques :

Après avoir étudié cette publication et compris les étapes de la calibration de la caméra, il nous était demandé de compléter certaines fonctions du code qui nous a été fourni.

Nous avons d'abord compléter la fonction *ZhangConstraintTerm* permettant de faire le calcul d'un terme de contrainte à partir d'une homographie.
Nous avons pour ce faire ajouter la formule suivante tirée du rapport de Zhang :

$$\mathbf{v}_{ij} = [h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, \\ h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]^T$$

La seconde fonction que nous avons implémentée est la fonction visant à déterminer les paramètres intrinsèques de la caméra.
Dans le rapport de Zhang sont décrites les différentes formules permettant de déterminer la matrice intrinsèque. Nous avons donc codé les formules suivantes :

$$
\begin{aligned}
v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\
\lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\
\alpha &= \sqrt{\lambda/B_{11}} \\
\beta &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\
\gamma &= -B_{12}\alpha^2\beta/\lambda \\
u_0 &= \gamma v_0/\beta - B_{13}\alpha^2/\lambda \ .
\end{aligned}
$$

Les valeurs de *u0* et *v0* permettent de définir la position du centre de l'image.
Alpha et *Beta* correspondent à la distance focale de la caméra sur chacun des deux axes de l'image. Le facteur *gama* correspond à l'éventuelle non orthogonalité de ces axes.

La fonction *IntrinsicMatrix* obtenue est donc la suivante :

```
function A = IntrinsicMatrix(b)
  vo = (b(2)*b(4)-b(1)*b(5))/(b(1)*b(3)-b(2)*b(2));
  lambda = b(6) - (b(4)*b(4) + vo*(b(2)*b(4) - b(1)*b(5)))/b(1);
  alpha = sqrt(lambda/b(1));
  beta = sqrt((lambda*b(1)) / (b(1)*b(3)-(b(2)*b(2))));
  gama = (-b(2)*alpha*alpha*beta)/lambda;
  uo = (gama*vo)/(beta)-(b(4)*alpha*alpha)/lambda;
  A = [alpha,lambda,uo;0,beta,vo;0,0,1];
endfunction
```

Une fois cette fonction écrite, nous pouvons donc déterminer la matrice intrinsèque grâce au script.
Nous avons obtenu les résultats suivant :

```
-->disp(A)

   3498.2767    0.9869483    336.76583
   0.           3503.8946    220.1142
   0.           0.           1.
```

On peut comparer les résultats obtenus aux résultats théoriques obtenus dans les scripts povray qui sont les suivant :

```
| 3546.099291   0.000000       320.000000|
| 0.000000      3546.099291    240.000000|
| 0.000000      0.000000         1.000000|
```

On remarque que nos valeurs calculées sont très proches. La différence entre les deux est négligeable et existante probablement à cause des différents calculs permettant d'arriver au calcul de la matrice intrinsèque.

III) Estimation des paramètres extrinsèques :

Après avoir déterminé les paramètres intrinsèques de la caméra, nous pouvons maintenant estimer les paramètres extrinsèques.
La matrice extrinsèque est composée de trois vecteurs de rotation de la caméra et d'un de translation.
Elle peut être déterminée par les formules suivantes :

$$\mathbf{r}_1 = \lambda \mathbf{A}^{-1} \mathbf{h}_1$$
$$\mathbf{r}_2 = \lambda \mathbf{A}^{-1} \mathbf{h}_2$$
$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$
$$\mathbf{t} = \lambda \mathbf{A}^{-1} \mathbf{h}_3$$

$$\text{with } \lambda = 1/\|\mathbf{A}^{-1}\mathbf{h}_1\| = 1/\|\mathbf{A}^{-1}\mathbf{h}_2\|$$

A^{-1} est la matrice inverse de la matrice intrinsèque A.

Nous avons donc complété la fonction *ExtrinsicMatrix* de la manière suivante :

```
function E = ExtrinsicMatrix(iA, H)
        lambda = 1/abs(iA*H(:,1));
        lambda = lambda(1);
        r1 = lambda * iA * H(:,1);
        r2 = lambda * iA * H(:,2);
        r3 = CrossProduct(r1,r2);
        t = lambda * iA * H(:,3);
        E  = [r1,r2,r3,t];
endfunction
```

On peut maintenant vérifier le bon fonctionnement de notre fonction en comparant nos résultats aux valeurs théoriques.

Après l'application de notre fonction, nous obtenons les valeurs suivantes ((:,:,1) représentant les valeurs pour l'image 1, les 3x3 premières valeurs étant la rotation et la dernière colonne le vecteur translation) :

```
-->disp (E)

(:,:,1)

    1.          - 0.0002699   - 0.0006686   - 48.876008
    0.0000377     0.9982951     0.0015761     54.733323
    0.0006696   - 0.0015763     0.9982951     9854.3632
(:,:,2)

    1.          - 0.0005644     1.3838643   - 64.810898
  - 0.0055727     1.4050444     0.0014512     61.520185
  - 0.9849222   - 0.0008953     1.4050412     11096.698
(:,:,3)

    1.            0.1760446   - 0.0003857   - 44.536143
  - 0.1760797     0.9982399   - 0.0005101     49.94335
    0.0002875     0.0005607     1.0292378     9005.3261
(:,:,4)

    1.            0.0037072     0.0000411   - 143.91914
    0.0000023     0.7020868     0.7143858     42.078156
  - 0.0000609   - 0.714386      0.7020868     8872.4252
```

Pour pouvoir comparer nos valeurs de rotation aux valeurs théoriques, on doit appliquer une rotation rodrigues à la valeur fournie dans les fichiers "*image-i.txt*".

Pour l'image 1, nous devrions obtenir la matrice identité pour la rotation et un vecteur (0,0,10000) pour la translation. On peut remarquer un écart très faible sur les valeurs (probablement dû au premier écart sur la matrice intrinsèque).

En ajoutant tous les écarts pour la rotation on obtient $4.7982*10^{-3}$. L'écart cumulé sur la translation est lui de 249,246131.
On répète ces calculs pour les images 2, 3 et 4.

image	2	3	4
différence rotation	2,3589285	0,0644356	0,0284101
différence translation	3223,029083	99,805593	213,572096

On remarque ainsi que nos résultats pour les translations sont très similaires. De plus étant donné la "grandeur" des valeurs des translations on peut relativiser les écarts perçus.
On peut donc estimer que la calibration est correcte pour les différentes cibles. On peut supposer qu'un calcul sans erreur (par rapport aux valeurs théoriques) de la matrice intrinsèque pourrait permettre de réduire ces écarts.

Conclusion :

Pour conclure, nous avons lors de ce TP pris en main la méthode de Zhang pour la calibration de caméra grâce à une mire.
Nous avons procédé par étapes afin d'assimiler les différentes notions de calibration.
Nous avons d'abord étudié la publication de Zhang sur sa proposition d'une méthode de calibration afin d'en comprendre le fonctionnement général.
Ensuite, nous avons complété les fonctions du script fourni afin de pouvoir déterminer les valeurs des matrices intrinsèques et extrinsèques.
L'analyse des résultats obtenus a pu nous permettre de mieux comprendre les valeurs des matrices intrinsèques et extrinsèques ainsi que leur impact sur la calibration de la caméra.

Annexe :

Voici le fichier *methodes.sci :*

```
// ---------------------------------------------------------------------
/// \brief Calcule un terme de contrainte a partir d'une homographie.
/// \param H: matrice 3*3 définissant l'homographie.
/// \param i: premiere colonne.
/// \param j: deuxieme colonne.
/// \return vecteur definissant le terme de contrainte.
// ---------------------------------------------------------------------
function v = ZhangConstraintTerm(H, i, j)
  v1 = H(1,i,:)*H(1,j,:);
  v2 = (H(1,i,:)*H(2,j,:))+(H(2,i,:)*H(1,j,:));
  v3 = H(2,i,:)*H(2,j,:);
  v4 = (H(3,i,:)*H(1,j,:))+(H(1,i,:)*H(3,j,:));
  v5 = (H(3,i,:)*H(2,j,:))+(H(2,i,:)*H(3,j,:));
  v6 = H(3,i,:)*H(3,j,:);
  v = [v1,v2,v3,v4,v5,v6];
   // disp(v);
endfunction

// ---------------------------------------------------------------------
/// \brief Calcule deux equations de contrainte a partir d'une homographie
/// \param H: matrice 3*3 définissant l'homographie.
/// \return matrice 2*6 definissant les deux contraintes.
// ---------------------------------------------------------------------
function v = ZhangConstraints(H)
        v = [ZhangConstraintTerm(H, 1, 2); ...
        ZhangConstraintTerm(H, 1, 1) - ZhangConstraintTerm(H, 2, 2)];
endfunction

// ---------------------------------------------------------------------
/// \brief Calcule la matrice des parametres intrinseques.
/// \param b: vecteur resultant de l'optimisation de Zhang.
/// \return matrice 3*3 des parametres intrinseques.
// ---------------------------------------------------------------------
function A = IntrinsicMatrix(b)
  vo = (b(2)*b(4)-b(1)*b(5))/(b(1)*b(3)-b(2)*b(2));
  lambda = b(6) - (b(4)*b(4) + vo*(b(2)*b(4) - b(1)*b(5)))/b(1);
  alpha = sqrt(lambda/b(1));
  beta = sqrt((lambda*b(1)) / (b(1)*b(3)-(b(2)*b(2))));
  gama = (-b(2)*alpha*alpha*beta)/lambda;
  uo = (gama*vo)/(beta)-(b(4)*alpha*alpha)/lambda;
  A = [alpha,lambda,uo;0,beta,vo;0,0,1];
endfunction

// ---------------------------------------------------------------------
/// \brief Calcule la matrice des parametres extrinseques.
/// \param iA: inverse de la matrice intrinseque.
/// \param H: matrice 3*3 definissant l'homographie.
/// \return matrice 3*4 des parametres extrinseques.
// ---------------------------------------------------------------------
function E = ExtrinsicMatrix(iA, H)
        lambda = 1/abs(iA*H(:,1));
        lambda = lambda(1);
```

```
        r1 = lambda * iA * H(:,1);
        r2 = lambda * iA * H(:,2);
        r3 = CrossProduct(r1,r2);
        t = lambda * iA * H(:,3);
        E  = [r1,r2,r3,t];
endfunction
```

Pour réaliser la rotation rodrigues, on utilise le script suivant :

```
function R = RotationRodriguesMatrix(theta, n)
        N = [ 0,   -n(3), n(2);
        n(3), 0,   -n(1);
        -n(2), n(1), 0];

        R = cos(theta)*eye(3, 3) + (1 - cos(theta)) * n*n' + sin(theta) * N;
        return R;
endfunction

R2 = RotationRodriguesMatrix(0.785398, [0; 1; 0]);
```