Projet de reconstruction 3D

1 Description du projet

A partir de plusieurs images d'une même scène, une méthode de reconstruction 3D vise à obtenir un nuage de points 3D de la scène ainsi que la pose (rotation et translation) de chaque caméra (Fig. 1). Une telle méthode comporte en général trois étapes : une étape de détection de points d'intérêt, une étape de mise en correspondance et de création de "chemins" et finalement une étape d'estimation du nuage de points 3D et des poses.

Dans ce projet, nous considérons le cas où les deux premières étapes ont été effectuées. De plus nous supposerons que ces correspondances **ne contiennent pas de correspondances aberrantes**. L'objectif du projet consiste donc à réaliser la 3ème étape, c'est-à-dire à estimer le nuage de points 3D et les poses des caméras.

(a) (b)

Fig. 1: Reconstruction 3D : (1a) Exemples d'images en entrée de l'algorithme de reconstruction, (1b) Nuage de points 3D et poses des caméras estimés

2 Reconstruction 3D incrémentale

Une manière d'obtenir une reconstruction 3D à partir de "chemins" consiste à effectuer une reconstruction pour deux images, puis à ajouter les images les unes après les autres. On parle alors de reconstruction 3D incrémentale. Les étapes d'un tel algorithme sont les suivantes :

1. **INITIALISATION**
   À partir de deux images $\mathtt{I}_1$ et $\mathtt{I}_2$, estimer les poses $\mathtt{R}_{w1}$, $\mathbf{t}_{w1}$, $\mathtt{R}_{w2}$, $\mathbf{t}_{w2}$ ainsi que les coordonnées des points 3D vus dans ces deux images.

2. Choisir une troisième image $\mathtt{I}_3$ (par exemple l'image suivante dans la liste car les images sont issues d'une vidéo, il y a donc du recouvrement entre les images consécutives).

3. **LOCALISATION**
   Estimer $\mathtt{R}_{w3}$, $\mathbf{t}_{w3}$ en minimisant l'erreur de reprojection de cette 3ème caméra par rapport aux points 3D déjà reconstruits (les points 3D sont figés durant cette étape). Il s'agit donc ici d'appliquer un algorithme d'ajustement de faisceaux simplifié car uniquement $\mathtt{R}_{w3}$ et $\mathbf{t}_{w3}$ sont optimisés. Les paramètres $\mathtt{R}_{w3}$ et $\mathbf{t}_{w3}$ seront initialisés avec les valeurs de $\mathtt{R}_{w2}$ et $\mathbf{t}_{w2}$.

4. **TRIANGULATION**
   Trianguler les points 3D qui peuvent l'être grâce à l'ajout de cette troisième image.

5. **RAFFINEMENT**
   Appliquer l'algorithme d'ajustement de faisceaux aux poses des 3 caméras ($\{\mathtt{R}_{wi}, \mathbf{t}_{wi}\}_{i=1\ldots3}$) ainsi qu'aux points 3D.

6. Choisir une quatrième image $\mathtt{I}_4$ (par exemple l'image suivante dans la liste car les images sont issues d'une vidéo, il y a donc du recouvrement entre les images consécutives).

7. **LOCALISATION**
   Estimer $\mathtt{R}_{w4}$, $\mathbf{t}_{w4}$ en minimisant l'erreur de reprojection de cette 4ème caméra par rapport aux points 3D déjà reconstruits (les points 3D sont figés durant cette étape).

8. **TRIANGULATION**
   Trianguler les points 3D qui peuvent l'être grâce à l'ajout de cette quatrième image.

9. **RAFFINEMENT** Appliquer l'algorithme d'ajustement de faisceaux aux poses des 4 caméras ($\{\mathtt{R}_{wi}, \mathbf{t}_{wi}\}_{i=1\ldots4}$) ainsi qu'aux points 3D.

10. Continuer jusqu'à ce qu'il n'y ait plus d'image à ajouter.

3 Travail à effectuer

L'objectif du projet est de réaliser une reconstruction incrémentale en implémentant les étapes décrites précédemment. Un code Matlab vous est fourni. Ce code contient un script principal où l'étape 1 est déjà effectuée. Vous disposez donc en particulier de fonctions correspondant :

- à une implémentation de l'algorithme d'ajustement de faisceaux pour deux images,
- à une implémentation d'un algorithme de triangulation.

Il est possible de réaliser le projet en utilisant la fonction de triangulation fournie **sans la modifier** et en vous inspirant de la fonction d'ajustement de faisceaux de deux images **pour coder une fonction d'ajustement de faisceaux** capable de traiter un nombre arbitraire d'images. **Il sera également indispensable de modifier les fonctions d'affichage fournies et d'en créer de nouvelles afin de vous assurer du bon fonctionnement de votre code.**
Vous trouverez en annexe, la description mathématique de l'algorithme d'ajustement de faisceaux.
Conseil : Dans un premier temps, vous pouvez effectuer une reconstruction 3D incrémentale sans effectuer d'étape de raffinement (étape 5 et étape 9 ci-dessus). Dans un second temps, vous pourrez rajouter ces étapes de raffinement qui permettront d'obtenir une reconstruction 3D de bien meilleure qualité.

A Algorithme d'ajustement de faisceaux

A l'issue de la mise en correspondances, l'information disponible est représentée sous la forme de "chemins" ("tracks" en anglais). Un "chemin" est propre à un point 3D et indique pour chaque image (si le point 3D est vu dans cette image) l'indice du point 2D auquel il correspond (parmi tous les points détectés dans l'image). Pour chaque image $\mathtt{I}_j$, si C_j points 3D sont vus dans cette image, nous disposons de deux vecteurs p2DId et p3DId de longueur C_j. L'élément p2DId (c) indique l'indice du point 2D parmi tous les points ayant été détectés dans $\mathtt{I}_j$, alors que p3DId (c) indique l'indice du point 3D.

Ainsi l'algorithme d'ajustement de faisceaux consiste à minimiser la fonction de coût suivante :

$$\min_{\substack{\{\mathtt{R}_{wj},\mathbf{t}_{wj}\}_{j=1,M} \\ \{\mathbf{u}_i^w\}_{i=1\ldots N}}} \sum_{j=1}^{M}\sum_{c=1}^{C_j} \left\|\mathbf{p}_{j,\mathrm{p2DId}(c)} - \mathtt{K}\pi\left(\mathtt{R}_{wj}^\top\left(\mathbf{u}^w_{\mathrm{p3DId}(c)} - \mathbf{t}_{wj}\right)\right)\right\|_2^2 \tag{1}$$

où C_j est le nombre de points 3D vus dans l'image $\mathtt{I}_j$, $\pi(\cdot)$ est la fonction de projection, $\mathtt{K}$ est la matrice de calibration linéaire de la forme $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{bmatrix}$ et $\mathbf{p}_{j,\mathrm{p2DId}(c)}$ est un point 2D (censé correspondre à la reprojection du point 3D $\mathbf{u}^w_{\mathrm{p3DId}(c)}$) dans l'image j.

B Calcul des matrices jacobiennes

Afin d'implémenter un algorithme de Levenberg-Marquardt, il est nécessaire de connaître l'expression de la dérivée de $\mathtt{K}\pi\left(\mathtt{R}_{wj}^\top(\mathbf{u}_i^w - \mathbf{t}_{wj})\right)$ par rapport aux paramètres $\{\mathtt{R}_{wj},\mathbf{t}_{wj}\}_{j=1,M}$ et $\{\mathbf{u}_i^w\}_{i=1\ldots N}$. Utilisons les notations suivantes pour définir les incréments estimés à chaque itération du Levenberg-Marquardt : $\mathtt{R}_{wj}^{(l+1)} = \mathtt{R}_{wj}^{(l)}\mathrm{expm}\left(\left[\boldsymbol{\delta}_{\mathtt{R}_{wj}}\right]^\wedge\right)$, $\mathbf{t}_{wj}^{(l+1)} = \mathbf{t}_{wj}^{(l)} + \boldsymbol{\delta}_{\mathbf{t}_{wj}}$ et $\mathbf{u}_i^{w,(l+1)} = \mathbf{u}_i^{w,(l)} + \boldsymbol{\delta}_{\mathbf{u}_i^w}$.

Remarque : Une matrice de rotation n'étant pas un élément d'un espace euclidien mais d'un groupe de Lie, l'incrément $\boldsymbol{\delta}_{\mathtt{R}_{wj}}$ ne peut pas être additif et doit être adapté à la géométrie de ce groupe de Lie. Ainsi $\boldsymbol{\delta}_{\mathtt{R}_{wj}}$ est un vecteur de taille 3 (car une matrice de rotation a 3 degrés de liberté) et se convertit en une matrice de rotation grâce à la fonction exponentielle de matrice (fonction expm en Matlab).

La linéarisation de l'erreur de reprojection s'écrit alors (en omettant les indices l et $l+1$) :

$$\mathbf{p}_{j,i} - \mathtt{K}\pi\left(\left(\mathtt{R}_{wj}\mathrm{expm}\left(\left[\boldsymbol{\delta}_{\mathtt{R}_{wj}}\right]^\wedge\right)\right)^\top\left(\mathbf{u}_i^w + \boldsymbol{\delta}_{\mathbf{u}_i^w} - \mathbf{t}_{wj} - \boldsymbol{\delta}_{\mathbf{t}_{wj}}\right)\right) \approx \mathbf{r}_{j,i} - \mathtt{J}_{j,i}\boldsymbol{\delta} \tag{2}$$

où nous avons utilisé les notations suivantes :

$$\mathbf{r}_{j,i} = \mathbf{p}_{j,i} - \mathtt{K}\pi\left(\mathtt{R}_{wj}^\top(\mathbf{u}_i^w - \mathbf{t}_{wj})\right), \tag{3}$$

$$\boldsymbol{\delta} = \begin{bmatrix} \boldsymbol{\delta}_{\mathtt{R}_{w1}} & \boldsymbol{\delta}_{\mathbf{t}_{w1}} & \boldsymbol{\delta}_{\mathtt{R}_{w2}} & \boldsymbol{\delta}_{\mathbf{t}_{w2}} & \cdots & \boldsymbol{\delta}_{\mathtt{R}_{wM}} & \boldsymbol{\delta}_{\mathbf{t}_{wM}} & \boldsymbol{\delta}_{\mathbf{u}_1^w} & \boldsymbol{\delta}_{\mathbf{u}_2^w} & \cdots & \boldsymbol{\delta}_{\mathbf{u}_N^w} \end{bmatrix}^\top$$

$$\mathtt{J}_{j,i} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix}\mathtt{J}_\pi\begin{bmatrix} \mathtt{J}_{\mathtt{R}_{w1}} & \mathtt{J}_{\mathbf{t}_{w1}} & \mathtt{J}_{\mathtt{R}_{w2}} & \mathtt{J}_{\mathbf{t}_{w2}} & \cdots & \mathtt{J}_{\mathtt{R}_{wM}} & \mathtt{J}_{\mathbf{t}_{wM}} & \mathtt{J}_{\mathbf{u}_1^w} & \mathtt{J}_{\mathbf{u}_2^w} & \cdots & \mathtt{J}_{\mathbf{u}_N^w} \end{bmatrix}, \tag{4}$$

$$\mathtt{J}_{\pi} = \frac{\partial}{\partial \boldsymbol{\delta}_{\mathbf{u}_i^j}} \pi\left(\mathbf{u}_i^j + \boldsymbol{\delta}_{\mathbf{u}_i^j}\right)\Big|_{\boldsymbol{\delta}_{\mathbf{u}_i^j}=\mathbf{0}} = \begin{bmatrix} \frac{1}{\mathbf{u}_i^j(z)} & 0 & -\frac{\mathbf{u}_i^j(x)}{\mathbf{u}_i^j(z)^2} \\ 0 & \frac{1}{\mathbf{u}_i^j(z)} & -\frac{\mathbf{u}_i^j(y)}{\mathbf{u}_i^j(z)^2} \end{bmatrix}, \tag{5}$$

$$\begin{aligned} \mathtt{J}_{\mathtt{R}_{wk}} &= \frac{\partial}{\partial \boldsymbol{\delta}_{\mathtt{R}_{wk}}} \left(\mathtt{R}_{wj} \mathrm{expm}\left(\left[\boldsymbol{\delta}_{\mathtt{R}_{wj}}\right]^{\wedge}\right)\right)^{\top} \left(\mathbf{u}_i^w - \mathbf{t}_{wj}\right)\Big|_{\delta_{\mathtt{R}_{wk}}=\mathbf{0}} \\ &= \begin{cases} \mathbf{0} & \text{si } k \neq j \\ \begin{bmatrix} \mathtt{G}_x^{\top}\mathbf{u}_i^j & \mathtt{G}_y^{\top}\mathbf{u}_i^j & \mathtt{G}_z^{\top}\mathbf{u}_i^j \end{bmatrix} & \text{si } k = j \end{cases}, \end{aligned} \tag{6}$$

$$\mathtt{G}_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \mathtt{G}_y = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \mathtt{G}_z = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{7}$$

$$\begin{aligned} \mathtt{J}_{\mathbf{t}_{wk}} &= \frac{\partial}{\partial \boldsymbol{\delta}_{\mathbf{t}_{wk}}} \mathtt{R}_{wj}^{\top} \left(\mathbf{u}_i^w - \mathbf{t}_{wj} - \boldsymbol{\delta}_{\mathbf{t}_{wj}}\right)\Big|_{\boldsymbol{\delta}_{\mathbf{t}_{wk}}=\mathbf{0}} \\ &= \begin{cases} \mathbf{0} & \text{si } k \neq j \\ -\mathtt{R}_{wj}^{\top} & \text{si } k = j \end{cases}, \end{aligned} \tag{8}$$

$$\begin{aligned} \mathtt{J}_{\mathbf{u}_k^w} &= \frac{\partial}{\partial \boldsymbol{\delta}_{\mathbf{u}_k^w}} \mathtt{R}_{wj}^{\top} \left(\mathbf{u}_i^w + \boldsymbol{\delta}_{\mathbf{u}_i^w} - \mathbf{t}_{wj}\right)\Big|_{\boldsymbol{\delta}_{\mathbf{t}_{wk}}=\mathbf{0}} \\ &= \begin{cases} \mathbf{0} & \text{si } k \neq i \\ \mathtt{R}_{wj}^{\top} & \text{si } k = i \end{cases}. \end{aligned} \tag{9}$$

C Fonction "sparse" de Matlab

Chaque itération de l'algorithme de Levenberg-Marquardt consiste à résoudre un problème de la forme

$$\min_{\boldsymbol{\delta}} \|\mathbf{r} - \mathtt{J}\boldsymbol{\delta}\|_2^2 + \lambda \|\boldsymbol{\delta}\|_2^2 \tag{10}$$

ce qui équivaut à résoudre le système linéaire suivant :

$$\left(\mathtt{J}^{\top}\mathtt{J} + \lambda \mathtt{Id}\right)\boldsymbol{\delta} = \mathtt{J}^{\top}\mathbf{r}. \tag{11}$$

Cependant lorsque le nombre de caméras et de points 3D devient grand, la matrice J devient elle-même très grande. Il devient alors difficile de la stocker en mémoire et les calculs où elle intervient deviennent très lents.

En pratique, pour être capable d'obtenir une reconstruction pour un grand nombre de caméras, il faut supprimer la création de la matrice J dense et la remplacer par la création de la matrice J "sparse".

La fonction "sparse" permet de prendre en compte le fait que la matrice `J` contient beaucoup de zéros qui n'ont pas besoin d'être stockés. **Attention, il ne faut pas utiliser la fonction "sparse" en lui fournissant la matrice `J` dense. En effet, cela nécessiterait de créer la matrice `J` dense ce qu'on souhaite justement éviter.** Il faut procéder en trois étapes :

1. Créer les vecteurs $\mathbf{i}, \mathbf{j}, \mathbf{v}$ qui correspondent aux coordonnées et aux valeurs des éléments non-nuls de `J`.
2. Remplir les vecteurs $\mathbf{i}, \mathbf{j}, \mathbf{v}$.
3. Utiliser les vecteurs $\mathbf{i}, \mathbf{j}, \mathbf{v}$ pour créer la matrice `J` "sparse" : J = sparse(i,j,v).