Introduction à la vision

Rémy Malgouyres
LAIC, IUT, département info
B.P. 86
63172 AUBIERE cedex
e-mail : Remy.Malgouyres@laic.u-clermont1.fr
http ://laic.u-clermont1.fr/~mr

Table des matières

Chapitre 1

Méthode des moindres carrés

1.1 Valeurs propres et vecteurs propres

Définition 1.1.1 Soit A une matrice carrée $n \times n$. Soit λ un nombre réel et x un vecteur non nul à n coordonnées. On dit que x est un *vecteur propre de* A de valeur propre λ si

$$A.x = \lambda x$$

Autrement dit un vecteur propre x de A est tel que $A.x$ soit colinéaire à x.

1.2 Le problème des moindres carrés : cas linéaire homogène

1.2.1 Exemple

Donnons nous un système de trois équations à 2 inconnues x_1 et x_2.

$$\left\{ \begin{array}{rcrcl} 2x_1 & + & x_2 & = & 0 \\ x_1 & + & x_2 & = & 0 \\ 3x_1 & + & 2x_2 & = & 0 \end{array} \right.$$

On veut trouver une solution autre que la solution triviale $x_1 = 0$ et $x_2 = 0$. Il y a plus d'équations que d'inconnues. Le système d'équations n'a pas de solution. On cherche alors une **solution approchée**, c'est à dire une vecteur (x_1, x_2) tels que

$$\left\{ \begin{array}{rcrl} 2x_1 & + & x_2 & \text{soit petit} \\ x_1 & + & x_2 & \text{soit petit} \\ 3x_1 & + & 2x_2 & \text{soit petit} \end{array} \right.$$

On veut aussi que x_1 et x_2 ne soit pas trop près de l'origine $(0, 0)$. On peut par exemple demander que le point (x_1, x_2) soit sur le cercle de centre $(0, 0)$ et de rayon 1. Le point (x_1, x_2) doit être à distance 1 de $(0, 0)$:

$$\sqrt{x_1^2 + x_2^2} = 1$$

Le système d'équations se réécrit sous forme matricielle :

$$\mathcal{U}.x = 0$$

avec $\mathcal{U}$ la matrice des coefficients et x le vecteur des inconnues :

$$\mathcal{U} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 3 & 2 \end{bmatrix} \text{ et } x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

On cherche x tel que $\mathcal{U}.x$ soit petit. La méthode des mondres carrés consiste à chercher x tel que $\mathcal{U}.x$ soit le plus près possible de l'origine. On cherche donc x tel que la distance $||\mathcal{U}.x||$ à l'origine soit la plus petite possible. Ceci est équivalent à charcher c tel que $||\mathcal{U}.x||^2$ soit le plus petit possible.

1.2.2 Cas général

Supposons que nous avons un système de p équations à q inconnues :

$$\begin{cases} u_{1,1}x_1 + u_{1,2}x_2 + \cdots + u_{1,q}x_q = 0 \\ u_{2,1}x_1 + u_{2,2}x_2 + \cdots + u_{2,q}x_q = 0 \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \\ u_{p,1}x_1 + u_{p,2}x_2 + \cdots + u_{p,q}x_q = 0 \end{cases}$$

Les x_i sont les inconues et les $u_{i,j}$ sont les coefficients des équations. On cherche $(x_1, x_2, \ldots, x_q)$ pas trop proche de l'origine $(0, 0, \ldots, 0)$. On impose donc que

$$\sqrt{x_1^2 + x_2^2 + \cdots + x_q^2} = 1.$$

Supposons que le nombre d'équations p est plus grand que le nombre d'inconnues q. En général, il n'y a pas de solutions au système d'équations car y y a trop d'équations par rapport au nombre de variables. Les variables ne peuvent pas satisfaire toutes ces équations.

On réécrit l'équation sous forme matricielle :

$$\mathcal{U}.x = 0$$

avec

$$\mathcal{U} = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,q} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,q} \\ \vdots & \vdots & & \vdots \\ u_{p,1} & u_{p,2} & \cdots & p_{p,q} \end{bmatrix} \text{ et } x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_q \end{pmatrix}$$

Et on cherche x tel que $||x|| = 1$ et $||\mathcal{U}.x||^2$ soit minimum. La solution x est alors appelée *solution des moindres carrés*.

1.3 Méthode des moindres carrés

La solution du problème des moindres carrés est la suivante :

1. On calcule la matrice produit $A = \mathcal{U}^T.\mathcal{U}$, où $\mathcal{U}^T$ est la transposée de $\mathcal{U}$. La matrice A est une matrice $q \times q$. La matrice A est symétrique, c'est à dire qu'elle est égale à sa transposée.
2. On calcule le vecteur propre x correspondant à la plus petite des valeurs propres de A. (il y a des méthodes d'algèbre linéaire pour calculer les valeurs propres et les vecteurs propres d'une matrice symétrique). Le vecteur x est notre solution.

1.4 Exercices

Exercice 1.1 $(*)$ Soit A la matrice :

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Donner trois vecteurs propres de A avec les trois valeurs propres correspondantes.

Exercice 1.2 $(*)$ Soit A la matrice :

$$\begin{bmatrix} -1 & -2 & -2 \\ 3 & 4 & 2 \\ -3 & -3 & -1 \end{bmatrix}$$

Montrer que les vecteurs suivants sont vecteurs propres et calculer les valeurs propres correspondantes :

$$v_1 = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} v_2 = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} v_3 = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

Exercice 1.3 $(**)$ Soit le système d'équations :

$$\begin{cases} 2x_1 + x_2 = 0 \\ x_1 + x_2 = 0 \\ 3x_1 + 2x_2 = 0 \\ 4x_1 + 3x_2 = 0 \\ 4x_1 + 2x_2 = 0 \end{cases}$$

En utilisant Maple, donnez la meilleure solution par la méthode des moindres carrés.

Exercice 1.4 $(**)$ Soit le système d'équations :

$$\begin{cases} 2x_1 + x_2 - x_3 = 0 \\ x_1 + x_2 - x_3 = 0 \\ 3x_1 + 2x_2 - 2x_3 = 0 \\ 4x_1 + 3x_2 - 3x_3 = 0 \\ 4x_1 + 2x_2 - 3x_3 = 0 \end{cases}$$

En utilisant Maple, donnez la meilleure solution par la méthode des moindres carrés.

Chapitre 2

Un modèle de caméra pour la vision

Un modèle de caméra est donné par tous les paramètres dont dépend la matrice de projection qui à un point $3D$ (x, y, z) du réel associe les coordonnées du pixel (x_p, y_p) qui lui correspond sur l'image prise par la caméra. Ce modèle de caméra dépendra donc de :

- La distance focale d des lentilles de la caméra ;
- La taille des pixels ;
- La position du point de l'image correspondant à l'axe de visée ;
- La position et l'orientation de la caméra.

On distingue les paramètres intrinsèques qui ne dépendent que de la caméra (distance focale, taille des pixels, position de l'axe de visée dans l'image), et les paramètres extrinsèques tels que la position et l'orientation de la caméra dans l'espace.

2.1 Paramètres intrinsèques

Pour modéliser la caméra réelle, on introduit une caméra virtuelle avec un écran virtuel, parallèle à la rétine de la caméra réelle. Le projeté d'un point réel (X, Y, Z) sur le plan virtuel a des coordonnées notées (x_1, y_1, z_1). Les formules de projection donnant x_1 et y_1 en fonction de X, Y et Z sont calculées en fonction de la distance focale d (voir la figure 2.1).

La troisième coordonnée z_1 du projeté sera évidemment égale à d de part la définition du plan de projection.
D'après le théorème de Thalès (voir figure 2.2), on a :

$$\frac{x_1}{d} = \frac{X}{Z} \quad \text{et} \quad \frac{y_1}{d} = \frac{Y}{Z}$$

En multipliant ces égalités par d, on obtient :

$$x_1 = d\,\frac{X}{Z} \quad \text{et} \quad y_1 = d\,\frac{X}{Z}$$

Soit $(C, \overrightarrow{\imath}_1, \overrightarrow{\jmath}_1)$ le repère orthonormé dans le plan de projection d'équation $Z = d$. La projection P de (X, Y, Z) a pour coordonnées (x_1, y_1) dans ce repère, et on a donc :

$$P = C + x_1 \overrightarrow{\imath}_1 + y_1 \overrightarrow{\jmath}_1$$

En raison des imperfection, les axes d'une caméra réelle ne sont pas tout à fait orthogonaux. De même les pixels ne sont pas tout à fait rectangulaires ; ce sont de parallélogrammes. Soit $(C, \overrightarrow{\imath}_2, \overrightarrow{\jmath}_2)$ le repère de la caméra réelle sur lequel sont alignés les pixels. On introduit l'angle θ entre $\overrightarrow{\imath}_2$ et $\overrightarrow{\jmath}_2$. L'angle θ est proche de $90°$ mais pas nécessairement égal à $90°$. On peut supposer que $\overrightarrow{\imath}_2 = \overrightarrow{\imath}_1$ mais $\overrightarrow{\jmath}_2 = \overrightarrow{\imath}_1 \cos\theta + \overrightarrow{\jmath}_1 \sin\theta$ (voir la figure 2.3)

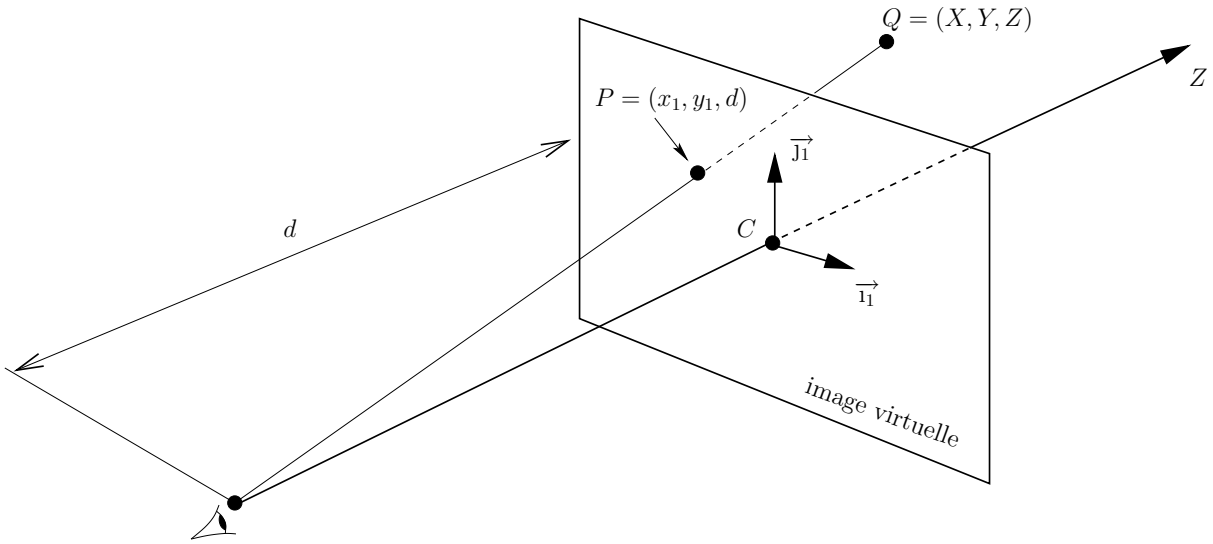

FIG. 2.1: Le modèle de caméra *pinhole* virtuel

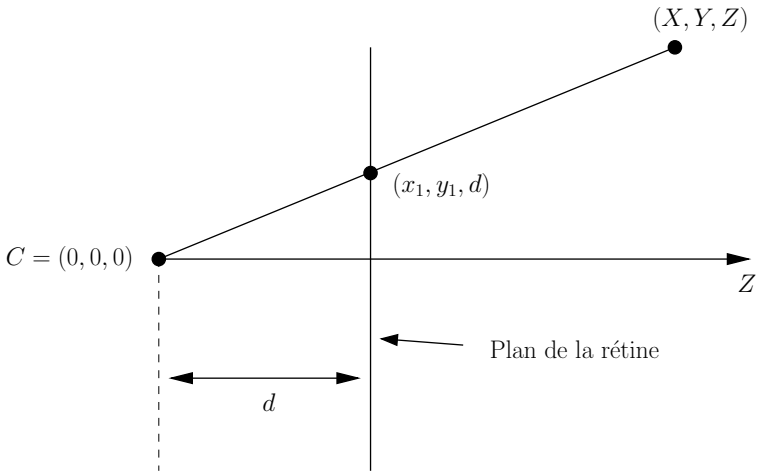

FIG. 2.2: Calcul des coordonnées du projeté.

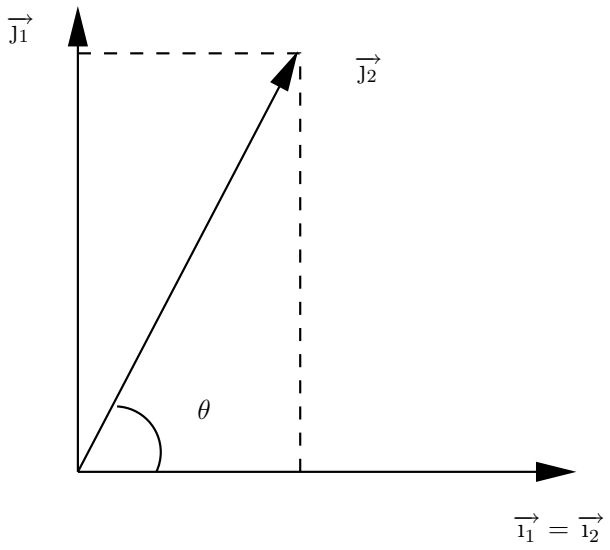

FIG. 2.3: Les vecteurs du repère font un angle θ proche de 90°

On a :

$$\overrightarrow{\jmath}_1 = \overrightarrow{\jmath}_2 \frac{1}{\sin\theta} - \overrightarrow{\imath}_1 \frac{\cos\theta}{\sin\theta} = \overrightarrow{\jmath}_2 \frac{1}{\sin\theta} - \overrightarrow{\imath}_1 \cot\theta$$

$$\begin{aligned} P &= C + x_1 \overrightarrow{\imath}_1 + y_1 \overrightarrow{\jmath}_1 \\ &= C + x_1 \overrightarrow{\imath}_2 + y_1 (\overrightarrow{\jmath}_2 \tfrac{1}{\sin\theta} - \overrightarrow{\imath}_2 \cot\theta) \\ &= C + (x_1 - y_1 \cot\theta) \overrightarrow{\imath}_2 + y_1 \tfrac{1}{\sin\theta} \overrightarrow{\jmath}_2 \end{aligned}$$

d'où si (x_2, y_2) représentent les coordonnées du projeté P dans le repère $(C, \overrightarrow{\imath}_2, \overrightarrow{\jmath}_2)$ de l'image de la caméra réelle, on a :

$$\left\{ \begin{array}{rcl} x_2 &=& x_1 - y_1 \cot\theta \\ y_2 &=& y_1 \frac{1}{\sin\theta} \end{array} \right.$$

soit en remplaçant x_1 et y_1 par leur valeur :

$$\left\{ \begin{array}{rcl} x_2 &=& d\frac{X}{Z} - d\frac{Y}{Z}\cot\theta \\ y_2 &=& \frac{d}{\sin\theta}\frac{Y}{Z} \end{array} \right.$$

Dans l'image réelle donnée par la caméra réelle, les coordonnées sont des nombres de pixels. Supposons que la distance d est en mètres et les dimensions du pixel sont l et h en mètres. De plus, l'axe des Z ne passera pas forcément par le pixel $(0,0)$, mais passe par un pixel C_0 de coordonnées (x_0, y_0). On introduit les coordonnée (x_p, y_p) de la projection en tenant compte du facteur d'échelle dû aux dimensions du pixel :

$$x_p = \frac{x_2}{l} + x_0 \text{ et } \frac{y_2}{h} + y_0$$

D'où en posant $\alpha = \frac{d}{l}$ et $\beta = \frac{d}{h}$,

$$\left\{ \begin{array}{rcl} x_p &=& \alpha\frac{X}{Z} - \alpha\frac{Y}{Z}\cot\theta + x_0 \\ y_p &=& \frac{\beta}{\sin\theta}\frac{Y}{Z} + y_0 \end{array} \right. .$$

En coordonnées homogènes, ces formules s'écrivent :

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = \frac{1}{Z}\mathcal{M}. \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

avec

$$\mathcal{M} = \begin{bmatrix} \alpha & -\alpha\cot\theta & x_0 & 0 \\ 0 & \frac{\beta}{\sin\theta} & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

2.2 Paramètres extrinsèques

Jusqu'à maintenant, nous avons fait les calculs dans le repère de la caméra, avec les coordonnées (X, Y, Z) dans ce repère. Introduisons maintenant les coordonnées (x, y, z) dans le repère du monde.

On a le changement de repère affine entre les deux systèmes de coordonnées :

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathcal{R}. \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

où $\mathcal{R}$ est une matrice de changement de repère en coordonnées homogènes.

On a :

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = \frac{1}{Z}\mathcal{M} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

d'où finalement :

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = \frac{1}{Z}\mathcal{M}.\mathcal{R} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Notons

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Notons m_1, m_2 et m_3 les lignes de la matrice $\mathcal{M}.\mathcal{R}$. On a :

$$\begin{cases} x_p &= \frac{1}{Z}m_1.P \\ y_p &= \frac{1}{Z}m_1.P \\ 1 &= \frac{1}{Z}m_3.P \end{cases}$$

d'où

$$Z = m_3.P$$
$$\begin{cases} x_p &= \frac{m_1.P}{m_3.P} \\ y_p &= \frac{m_2.P}{m_3.P} \end{cases}.$$

Chapitre 3

Calibration de caméra

3.1 Deux équations linéaires en chaque point

Les coordonnées x_p, y_p du projeté d'un point $3D$ (x, y, z) sur l'image sont données par :

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = \frac{1}{Z}\mathcal{M}.\mathcal{R}.P \text{ avec } P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Dans la pratique, lorsqu'on achète une caméra, les paramètres de cette caméra ne sont pas connus, et on ne connaît donc pas les matrices $\mathcal{M}$ et $\mathcal{R}$. La première chose à faire pour faire de la vision est donc de déterminer ces matrices, ou au moins déterminer le produit $\mathcal{M}.\mathcal{R}$. Ce problème s'appelle la *calibration de caméra*.

Comme nous l'avons vu, en notant m_1, m_2 et m_3 les lignes de la matrice $\mathcal{M}.\mathcal{R}$. On a :

$$\begin{cases} x_p &= \frac{1}{Z}m_1.P \\ y_p &= \frac{1}{Z}m_2.P \\ 1 &= \frac{1}{Z}m_3.P \end{cases}$$

d'où

$$Z = m_3.P$$
$$\begin{cases} x_p &= \frac{m_1.P}{m_3.P} \\ y_p &= \frac{m_2.P}{m_3.P} \end{cases}.$$

Nous en déduisons les équations linéaires suivantes vérifiées en tout point :

$$\begin{cases} m_1.P - x_p.m_3.P &= 0 \\ m_2.P - y_p m_3.P &= 0 \end{cases}.$$

3.2 Méthode de calibrage

Nous allons calculer une appoximation des vecteurs m_1, m_2 et m_3, ce qui nous donnera la matrice $\mathcal{M}$.

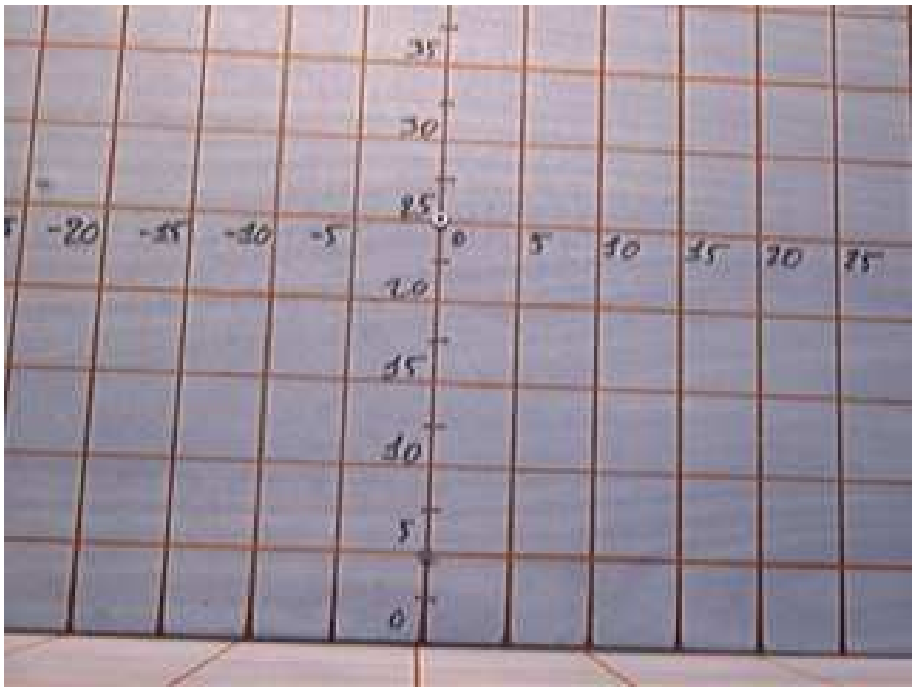

FIG. 3.1: Exemple d'image de mire. Les coordonnées $3D$ des points de la grille sont connues.

Pour cela, nous allons montrer une mire à la caméra, ce qui nous permettra d'observer l'image de points $3D$ dont la position en $3D$ est connue dans le repère du monde (voir la figure 3.1).

Nous allons relever, à la main ou automatiquement, les coordonnées $3D$ de n points $P_1, \ldots, P_n$, ainsi que les pixels correspondants dans l'image. Par exemple, nous cliquerons sur des pixels (x_p^i, y_p^i) de l'image de la mire, et nous saisirons au clavier les coordonnées $3D$ (x_i, y_i, z_i) des points correspondant. Nous obtiendrons les données suivantes :

coordonnées 3D			pixels	
x_1	y_1	z_1	x_p^1	y_p^1
x_2	y_2	z_2	x_p^2	y_p^2
x_3	y_3	z_3	x_p^3	y_p^3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_n	y_n	z_n	x_p^n	y_p^n

Notons les lignes

$$\begin{cases} m_1 &= (m_{1,1}, m_{1,2}, m_{1,3}, m_{1,4}) \\ m_2 &= (m_{2,1}, m_{2,2}, m_{2,3}, m_{2,4}) \\ m_3 &= (m_{3,1}, m_{3,2}, m_{3,3}, m_{3,4}) \end{cases}$$

Avec les coordonnées $m_{i,j}$ chechées, nous construisons le vecteur collonne de 12 coordonnées suivant :

$$\mu = (m_{1,1}, m_{1,2}, m_{1,3}, m_{1,4}, m_{2,1}, m_{2,2}, m_{2,3}, m_{2,4}, m_{3,1}, m_{3,2}, m_{3,3}, m_{3,4})^T$$

En écrivant les équations $m_1.P - x_p.m_3.P = 0$ et $m_2.P - y_p m_3.P = 0$ ci-dessus pour chacun des points P_i relevés, nous obtenons le système d'équations suivant :

$$\mathcal{P}.\mu = 0$$

avec

$$\mathcal{P} = \begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -x_p^1 x_1 & -x_p^1 y_1 & -x_p^1 z_1 & -x_p^1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -y_p^1 x_1 & -y_p^1 y_1 & -y_p^1 z_1 & -y_p^1 \\ x_2 & y_2 & z_2 & 1 & 0 & 0 & 0 & 0 & -x_p^2 x_2 & -x_p^2 y_2 & -x_p^2 z_2 & -x_p^2 \\ 0 & 0 & 0 & 0 & x_2 & y_2 & z_2 & 1 & -y_p^2 x_2 & -y_p^2 y_2 & -y_p^2 z_2 & -y_p^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 & 0 & 0 & 0 & 0 & -x_p^n x_n & -x_p^n y_n & -x_p^n z_n & -x_p^n \\ 0 & 0 & 0 & 0 & x_n & y_n & z_n & 1 & -y_p^n x_n & -y_p^n y_n & -y_p^n z_n & -y_p^n \end{bmatrix}$$

La solution approchée de ce système linéaire par la méthode des moindres carrés donnera les coefficients $m_{i,j}$ cherchés **à un facteur multiplicatif près**.

Pour récupérer la matrice $\mathcal{M}.\mathcal{R}$ recherchée, on la reconstitue à partir de ses lignes m_1, m_2 et m_3, puis on évalue en moyenne le rapport des valeurs x (ou y) des vecteurs $\mathcal{M}.\mathcal{R}.P_i$ avec les x_p^i observés (ou y_p^i). Il suffit ensuite de diviser les trois premières lignes m_1, m_2 et m_3 de la matrice $\mathcal{M}.\mathcal{R}$ par le rapport moyen obtenu. Cela résoud le problème du facteur multiplicatif.

Chapitre 4

Stéréovision

4.1 Trois équations linéaires en chaque point

Les coordonnées x_p, y_p du projeté d'un point $3D$ (x, y, z) sur l'image sont données par :

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = \frac{1}{Z}\mathcal{M}.\mathcal{R}.P \text{ avec } P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

La matrice $\mathcal{M}.\mathcal{R}$ peut être déterminée par calibrage de la caméra. Comme nous l'avons vu, en notant m_1, m_2 et m_3 les lignes de la matrice $\mathcal{M}.\mathcal{R}$. On a :

$$\begin{cases} x_p &= \frac{1}{Z}m_1.P \\ y_p &= \frac{1}{Z}m_2.P \\ 1 &= \frac{1}{Z}m_3.P \end{cases}$$

d'où

$$Z = m_3.P$$

$$\begin{cases} x_p &= \frac{m_1.P}{m_3.P} \\ y_p &= \frac{m_2.P}{m_3.P} \end{cases}.$$

Nous en déduisons les trois équations linéaires suivantes vérifiées en tout point :

$$\begin{cases} m_1.P - x_p.m_3.P &= 0 \\ m_2.P - y_p m_3.P &= 0 \\ y_p m_1.P - x_p m_2.P &= 0 \end{cases}.$$

4.2 Calculer la position d'un point par séréovision

Supposons que deux caméras observent une même scène, donnant deux images. Un même point P apparaît sur les deux images. Dans cette partie, nous exposons une méthode pour déterminer la position de P dans l'espace, connaissant les coordonnées (x_p^1, y_p^1) et (x_p^2, y_p^2) des pixels de l'image de P pour la caméra 1 et la céméra 2 (voir la figure 4.1).

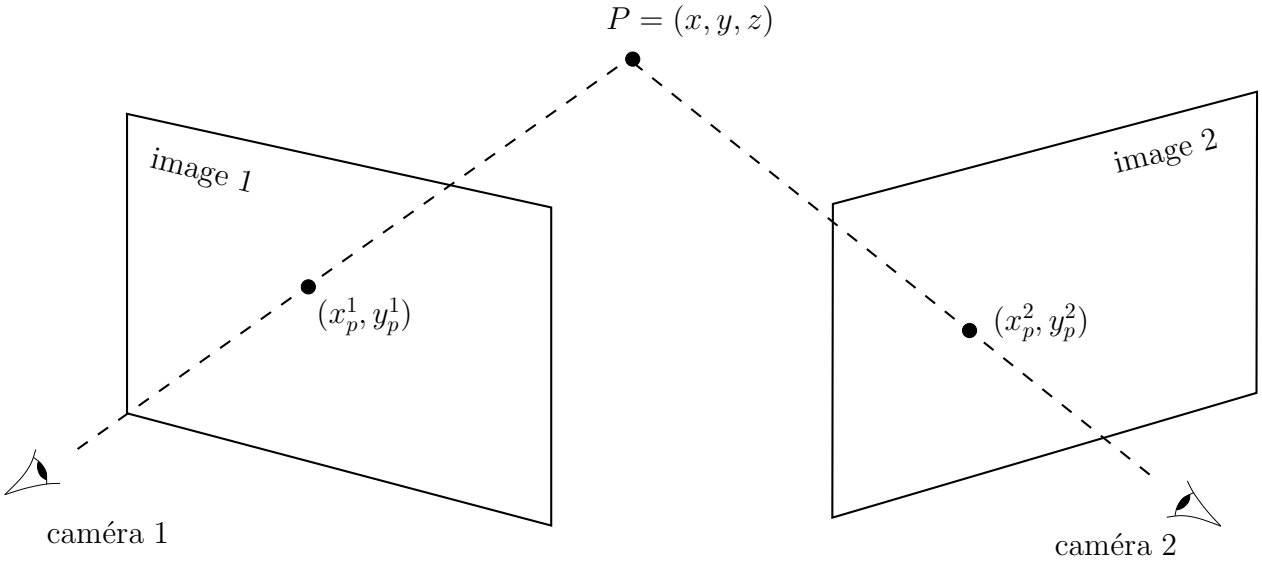

FIG. 4.1: Détermination de la position d'un point par stéréovision

On suppose que l'on a calibré la caméra 1 et la caméra 2, et que nous connaissons donc les deux matrices de projection $\mathcal{M}_1.\mathcal{R}_1$ et $\mathcal{M}_2.\mathcal{M}_2$ coorespondantes.

Notons m_1^1, m_2^1 et m_3^1 les lignes de la matrice $\mathcal{M}_1.\mathcal{R}_1$ pour la caméra 1, et notons de même m_1^2, m_2^2 et m_3^2 les lignes de la matrice $\mathcal{M}_2.\mathcal{R}_2$ pour la caméra 2.

Notons les lignes

$$\left\{\begin{array}{rcl} m_1^1 & = & (m_{1,1}^1, m_{1,2}^1, m_{1,3}^1, m_{1,4}^1) \\ m_2^1 & = & (m_{2,1}^1, m_{2,2}^1, m_{2,3}^1, m_{2,4}^1) \\ m_3^1 & = & (m_{3,1}^1, m_{3,2}^1, m_{3,3}^1, m_{3,4}^1) \end{array}\right.$$

et

$$\left\{\begin{array}{rcl} m_1^2 & = & (m_{1,1}^2, m_{1,2}^2, m_{1,3}^2, m_{1,4}^2) \\ m_2^2 & = & (m_{2,1}^2, m_{2,2}^2, m_{2,3}^2, m_{2,4}^2) \\ m_3^2 & = & (m_{3,1}^2, m_{3,2}^2, m_{3,3}^2, m_{3,4}^2) \end{array}\right.$$

Écrivons les trois équations linéaires vérifiées par le point P pour la caméra 1.

$$\left\{\begin{array}{rcl} m_1^1.P - x_p^1.m_3^1.P & = & 0 \\ m_2^1.P - y_p^1 m_3^1.P & = & 0 \\ y_p^1 m_1^1.P - x_p^1 m_2^1.P & = & 0 \end{array}\right. .$$

En détaillant les produits, on obtient les trois équations suivantes :

$$\begin{array}{ccccccccccc} (m_{1,1}^1 - x_p^1 m_{3,1}^1)x & + & (m_{1,2}^1 - x_p^1 m_{3,2}^1)y & + & (m_{1,3}^1 - x_p^1 m_{3,3}^1)z & + & (m_{1,4}^1 - x_p^1 m_{3,4}^1) & = & 0 \\ (m_{2,1}^1 - y_p^1 m_{3,1}^1)x & + & (m_{2,2}^1 - y_p^1 m_{3,2}^1)y & + & (m_{2,3}^1 - y_p^1 m_{3,3}^1)z & + & (m_{2,4}^1 - y_p^1 m_{3,4}^1) & = & 0 \\ (y_p^1 m_{1,1}^1 - x_p^1 m_{2,1}^1)x & + & (y_p^1 m_{1,2}^1 - x_p^1 m_{2,2}^1)y & + & (y_p^1 m_{1,3}^1 - x_p^1 m_{2,3}^1)z & + & (y_p^1 m_{1,4}^1 - x_p^1 m_{2,4}^1) & = & 0 \end{array}$$

De même, en écrivant les mêmes équations pour la caméra 2, on obtient les équations :

$$\begin{array}{ccccccccccc} (m_{1,1}^2 - x_p^2 m_{3,1}^2)x & + & (m_{1,2}^2 - x_p^2 m_{3,2}^2)y & + & (m_{1,3}^2 - x_p^2 m_{3,3}^2)z & + & (m_{1,4}^2 - x_p^2 m_{3,4}^2) & = & 0 \\ (m_{2,1}^2 - y_p^2 m_{3,1}^2)x & + & (m_{2,2}^2 - y_p^2 m_{3,2}^2)y & + & (m_{2,3}^2 - y_p^2 m_{3,3}^2)z & + & (m_{2,4}^2 - y_p^2 m_{3,4}^2) & = & 0 \\ (y_p^2 m_{1,1}^2 - x_p^2 m_{2,1}^2)x & + & (y_p^2 m_{1,2}^2 - x_p^2 m_{2,2}^2)y & + & (y_p^2 m_{1,3}^2 - x_p^2 m_{2,3}^2)z & + & (y_p^2 m_{1,4}^2 - x_p^2 m_{2,4}^2) & = & 0 \end{array}$$

D'où finalement le système de 6 équations à 6 inconnues :

$$\mathcal{U}.P = 0$$

avec

$$\mathcal{U} = \begin{bmatrix} (m^1_{1,1} - x^1_p m^1_{3,1}) & (m^1_{1,2} - x^1_p m^1_{3,2}) & (m^1_{1,3} - x^1_p m^1_{3,3}) & (m^1_{1,4} - x^1_p m^1_{3,4}) \\ (m^1_{2,1} - y^1_p m^1_{3,1}) & (m^1_{2,2} - y^1_p m^1_{3,2}) & (m^1_{2,3} - y^1_p m^1_{3,3}) & (m^1_{2,4} - y^1_p m^1_{3,4}) \\ (y^1_p m^1_{1,1} - x^1_p m^1_{2,1}) & (y^1_p m^1_{1,2} - x^1_p m^1_{2,2}) & (y^1_p m^1_{1,3} - x^1_p m^1_{2,3}) & (y^1_p m^1_{1,4} - x^1_p m^1_{2,4}) \\ (m^2_{1,1} - x^2_p m^2_{3,1}) & (m^2_{1,2} - x^2_p m^2_{3,2}) & (m^2_{1,3} - x^2_p m^2_{3,3}) & (m^2_{1,4} - x^2_p m^2_{3,4}) \\ (m^2_{2,1} - y^2_p m^2_{3,1}) & (m^2_{2,2} - y^2_p m^2_{3,2}) & (m^2_{2,3} - y^2_p m^2_{3,3}) & (m^2_{2,4} - y^2_p m^2_{3,4}) \\ (y^2_p m^2_{1,1} - x^2_p m^2_{2,1}) & (y^2_p m^2_{1,2} - x^2_p m^2_{2,2}) & (y^2_p m^2_{1,3} - x^2_p m^2_{2,3}) & (y^2_p m^2_{1,4} - x^2_p m^2_{2,4}) \end{bmatrix}$$

Ce système de 6 équations linéaires homogènes nous permet de déterminer les coordonnées (homogènes) inconnues (x, y, z, w) de P par la méthode des mondres carrés.

Annexe A

La librairie scientifique GSL

La librairie *GSL* ou *GNU scientific library* est disponible gratuitement et permet de faire des calculs d'algèbre linéaire (multiplications et inversion de matrices, calcul de vecteurs propres, etc...). Dans cette partie, nous voyons brievement les types et fonctions de la *GSL* qui nous permetront de calibrer une caméra par la méthode des moindres carrés comme expliqué ci-dessus.

Pour utiliser la librairie *GSL* dans un programme C ou $C++$, il faut inclure le fichier $<$ `gsl/gsl_linalg.h` $>$.

A.1 Les vecteurs et le type `gsl_vector`

Un vecteur est donné par un pointeur sur `gsl_vector`. Avant d'utiliser un vecteur, il faut l'allouer avec `gsl_vector_alloc` qui prend en paramètre le nombre de coordonnées du vecteur. On peut affecter les coordonnées d'un vecteur avec `gsl_vector_set`, et on peut accéder aux coordonnées d'un vecteurs par `gsl_vector_get`. Le vecteur doit être libéré avec `gsl_vector_free`.

```
1  /* la fonction retourne le vecteur et fait un passage par adresse */
2  gsl_vector* SaisieVecteur(int *n)
3  {
4    gsl_vector *v; /* déclaration d'un vecteur */
5    int i;
6    double valeur;
7
8    puts("Veuilez entrer le nombre de coordonnées du vecteur");
9    scanf("%d", n);
10
11   v = gsl_vector_alloc(*n);   /* allocation du vecteur */
12   for (i=0 ; i<*n ; i++)
13      {
14         scanf("%lf", &valeur);
15         /* initialisation de la coordonnée v[i] : */
16         gsl_vector_set(v, i, valeur);
17      }
18   return v;
19 }
```

```
1  void AfficheVecteur(gsl_vector* v, int n)
2  {
3     int i;
4     double valeur;
5     for (i=0 ; i<n ; i++)
6        {
7            /* récupération de la coordonnée v[i] : */
8            valeur = gsl_vector_get(v, i);
9            printf("%.3f\n", valeur);
10       }
11 }
```

```
1  int main(void)
2  {
3    gsl_vector *v;  /* déclaration d'un vecteur */
4    int n;   /* nombre de coordonnées du vecteur */
5    v = SaisieVecteur(&n)
6    AfficheVecteur(v, n);
7    gsl_vector_free(v);  /* libération du vecteur */
8    return 0;
9  }
```

A.2 Les matrices et le type `gsl_matrix`

Une matrice est donné par un pointeur sur `gsl_matrix`. Avant d'utiliser une matrice, il faut l'allouer avec `gsl_matrix_alloc` qui prend en paramètre le nombre de lignes et le nombre de

colonnes de la matrice. On peut affecter les coefficients d'une matrice avec `gsl_matrix_set`, et on peut accéder aux coefficients d'une matrice par `gsl_matrix_get`. La matrice doit être libéré avec `gsl_matrix_free`.

```
1  /* la fonction retourne la matrice et fait des passages par adresse */
2  gsl_matrix* SaisieMatrice(int *n, inte *m)
3  {
4    gsl_matrix *A; /* déclaration d'une matrice */
5    int i, j;
6    double valeur;
7
8    puts("Veuilez entrer le nombre de lignes et de colonnes");
9    scanf("%d %d", n, m);
10   A = gsl_matrix_alloc(*n, *m);   /* allocation du vecteur */
11   for (i=0 ; i<*n ; i++)
12      for (j=0 ; j<*m ; j++)
13         {
14            scanf("%lf", &valeur);
15            /* initialisation du coefficient A[i,j] : */
16            gsl_matrix_set(A, i, j, valeur);
17         }
18   return A;
19 }
```

```
1  void AfficheMatrice(gsl_matrix* A, int n, int m)
2  {
3     int i, j;
4     double valeur;
5     for (i=0 ; i<n ; i++)
6        {
7           for (j=0 ; j<m ; j++)
8              {
9                 /* récupération du coefficient A[i,j] : */
10                valeur = gsl_matrix_get(A, i, j);
11                printf("%.3f ", valeur);
12             }
13          printf("\n");
14       }
15 }
```

```
1  int main(void)
2  {
3    gsl_matrix *A;  /* déclaration d'une matrice */
4    int n, m;   /* nombre de lignes et de colonnes */
5    A = SaisieMatrice(&n, &m)
6    AfficheMatrice(A, n, m);
7    gsl_matrix_free(A);  /* libération de la matrice */
8    return 0;
9  }
```

A.3 Opérations sur les vecteurs et les matrices

Pour utiliser les opérations sur les matrices de la librairie *cblas* de la *GSL* dans un programme C ou $C++$, il faut inclure le fichier < gsl/gsl_blas.h >.

A.3.1 Multiplication de deux matrices

Pour multiplier une matrice A par une matrice B, un utilise la fonction gsl_blas_dgemm.

```
1  void MutliplieMatriceMatrice(void)
2  {
3     gsl_matrix *A, *B, *C;
4     int na, ma, nb, mb;
5
6     A = SaisieMatrice(&na, &ma);
7     B = SaisieMatrice(&nb, &mb);
8
9     if (ma != nb)
10       {
11          puts("Erreur, dimensions incompatibles");
12          return;
13       }
14
15    /* calcul de C = A*B : */
16    C = gsl_matrix_alloc(na, mb);  /* allocation aux bonnes dimensions */
17    gsl_blas_dgemm(CblasNoTrans, CblasNoTrans, 1.0, A, B, 0.0, C);
18
19    AfficheMatrice(C);  /* affichage du résultat */
20
21    gsl_matrix_free(A); /* libération des matrices */
22    gsl_matrix_free(B);
23    gsl_matrix_free(C);
   }
```

A.3.2 Multiplication d'une matrice par la transposée d'une matrice

Pour multiplier la transposée A^T d'une matrice A par une matrice B, un utilise aussi la fonction `gsl_blas_dgemm`. La différence est dans l'option `CblasTrans`.

```
1  void MutliplieTranspose(void)
2  {
3     gsl_matrix *A, *B, *C;
4     int na, ma, nb, mb;
5
6     A = SaisieMatrice(&na, &ma);
7     B = SaisieMatrice(&nb, &mb);
8
9     if (na != nb)
10       {
11          puts("Erreur, dimensions incompatibles");
12          return;
13       }
14
15    /* calcul de C = A^T * B : */
16    C = gsl_matrix_alloc(ma, mb);  /* allocation aux bonnes dimensions */
17    gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, A, B, 0.0, C);
18
19    AfficheMatrice(C);  /* affichage du résultat */
20
21    gsl_matrix_free(A); /* libération des matrices */
22    gsl_matrix_free(B);
23    gsl_matrix_free(C);
24 }
```

A.3.3 Multiplication d'une matrice par un vecteur

Pour multiplier une matrice par un vecteur, on utilise la fonction `gsl_blas_dgemv`.

```
1   void MultiplieMatriceVecteur(void)
2   {
3      gsl_vector *v, *res;
4      gsl_matrix *A;
5      int na, ma, nv;
6
7      v = SaisieVecteur(&nv);
8      A = SaisieMatrice(&na, &ma);
9      if (nv != ma)
10        {
11           puts ("Erreur, dimensions incomatibles");
12           return;
13        }
14
15     /* calcul de res = A*v : */
16     res = gsl_vector_alloc(nv),
17     gsl_blas_dgemv(CblasNoTrans, 1.0, A, v, 0.0, res);
18
19     AfficheVecteur(res);
20
21     gsl_matrix_free(A); /* libération des matrices */
22     gsl_vector_free(v); /* libération des vecteurs */
23     gsl_vector_free(res);
   }
```

A.3.4 Calcul de vecteurs propres d'une matrice symétrique

Pour calculer des valeurs propres et vecteurs propre de la *GSL* dans un programme *C* ou *C++*, il faut inclure le fichier <gsl/gsl_eigen.h>.

Pour calculer les vecteurs propres d'une matrice symétrique, on utilise la fonction `gsl_eigen_symmv`. Il faut pour cela allouer un espace de travail avec la fonction `gsl_eigen_symmv_alloc`. On récupère les valeures propres dans un vecteur `eval`, et les vecteurs propres dans les colonnes d'une matrice `evec`.

```
1   void CalculVecteursProprесSymetrique(void)
2   {
3     gsl_matrix *A;
4     int i, j, n, m;
5
6     /* déclaration des varaiables nécessaires */
7     /* au calcul des vecteurs propres : */
8     gsl_matrix *evec;
9     gsl_vector *eval;
10    gsl_eigen_symmv_workspace *w;
11
12    puts("Veuillez saisir une matrice symétrique :");
13    A = SaisieMatrice(&n,&m);
14
15    /* calcul des vecteurs propres de A : */
16    w = gsl_eigen_symmv_alloc(n);
17    evec = gsl_matrix_alloc(n,n);
18    eval = gsl_vector_alloc(n);
19    gsl_eigen_symmv(A, eval, evec, w);
20
21    /* affichage des résultats : */
22    for (j=0 ; j<m ; j++)
23      {
24         printf("La valeur propre numéro %d est %f\n",
25                j, gsl_vector_get(eval, j));
26         puts("et le vecteur propre correspondant est :");
27         for (i=0 ; i<n ; i++)
28            printf("%f ", gsl_matrix_get(evec, i, j));
29      }
30    gsl_matrix_free(A);  /* Libération de mémoire */
31    gsl_matrix_free(evec);
32    gsl_vector_free(eval);
      gsl_eigen_symmv_free(w);
    }
```