TP2 : STEREOVISION PAR CALIBRATION

Le but de ce TP est d'implémenter une méthode de calibration d'un appareil de vision (webcams ou autres caméras) en utilisant **au choix** un des deux outils permettant de calibrer simplement des caméras à l'aide de mires planes de type damier :

- Camera Calibration Toolbox Matlab : http://www.vision.caltech.edu/bouguetj/calib_doc/index.html
- Intel OpenCV : http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html

Puis d'établir la position d'un point dans l'espace à partir de deux vues par stéréovision.

Préparation :

- Pour ceux qui utiliseront OpenCv, lire la documentation disponible sur le campus : « Learning OpenCv ». Ce document devrait suffire pour la prise en main de la bibliothèque OpenCv, néanmoins vous pouvez également faire d'autres recherches sur Internet. Il y a énormément d'information sur le sujet, à vous d'extraire les bonnes informations, utiles pour programmer une méthode de stéréovision. Pensez à installer la bibliothèque.
- Pour les autres, télécharger et installer la toolbox Matlab à partir du lien ci-dessus. Vous pouvez commencer à lire le tutorial sur la calibration de caméra.
- Quelques références :
  - ✓ http://en.wikipedia.org/wiki/Pinhole_camera_model
  - ✓ http://fr.wikipedia.org/wiki/Calibration_de_caméra

I. Calibration

I.1. Génération de données pour la calibration.

Vous trouverez sur le Campus un répertoire contenant une série d'images tests. Il s'agit d'acquisition d'une mire de calibration type damier de 30 mm de côté. Ces images ont étés prises pour diverses positions de la mire par rapport à un système stéréoscopique constitué de deux caméras que l'on nomme caméra droite et caméra gauche. Normalement il s'agit de deux mêmes caméras mais les paramètres peuvent être légèrement différents.

I.2. MATLAB : Utilisation des fonctions de calibrage

Suivre le tutorial Matlab pour retrouver les paramètres intrinsèques de chaque caméra en utilisant les images à votre disposition :

Pour cela
- Déplacer toutes les images tests de calibration dans le répertoire Toolbox de Calibration de Matlab.
- Lancer *calib_gui* depuis Matlab et suivre le mode standard
- Suivre le premier exemple (http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html) en utilisant les images de chaque caméra jusqu'à obtenir les bons paramètres de calibration.

(Read images – Extract grid corners puis Calibration)
Attention à l'ordre du choix des 4 extrémités du damier.
La taille des carrés est de 30mm, ça pourra vous aider.
Cet outils vous permet de trouver les positions subpixelliques (précision plus fine que le pixel) des points d'étalonnage de la mire. Le détecteur utilisé est supposé avoir une précision de l'ordre de 0.1 pixel.

Normalement, à la fin de cette étape, vous obtenez les différents paramètres de la matrice intrinsèque de chaque caméra.

Calibration results (with uncertainties):
Focal Length: *fc = [xxxx xxxx] ± [0.00000 0.00000]*
Principal point: *cc = [xxxx xxxx] ± [0.00000 0.00000]*
Skew: *alpha_c = [xxxx] ± [0.00000]* *=> angle of pixel axes = xxxx ± 0.00000 degrees*
Distortion: *kc = [xxxx xxxx xxxx xxxx xxxx] ± [0.00000 0.00000 0.00000 0.00000 0.00000]*
Pixel error: *err = [xxxx xxxx]*
Note: The numerical errors are approximately three times the standard deviations (for reference).
Peut on encore améliorer les résultats ?

- ✓ Vous pouvez afficher la projection pour chaque position de la mire au monde réel en cliquant sur le bouton « *Show Extrinsic* ».

- ✓ Le bouton « *Analyse error* » permet d'afficher une fenêtre représentant la distribution des erreurs de reprojection. Elle permet de se rendre compte d'éventuels biais ou groupe de points posant problème.

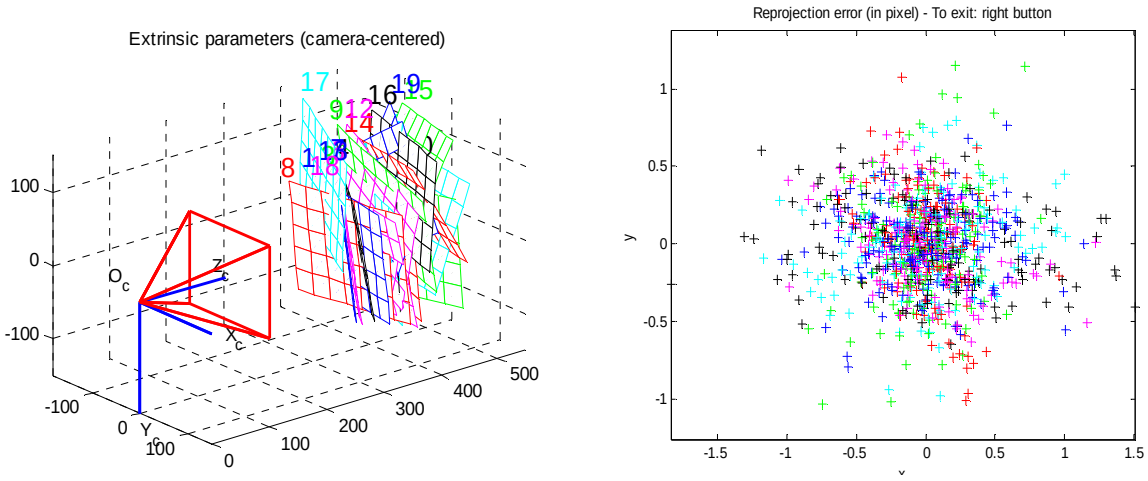

Figure 1 : Affichage de quelques fonctions.
gauche : visu des paramètres extrinsèques ; droite : erreur de reprojection

Le bouton « *Reproject on images* » vous permet de comparer le modèle obtenu par le calibrage avec la caméra réelle, en affichant 4 informations sur chaque image :

- le repère de la mire
- les points extraits de l'image représentés par des croix
- les points projetés grâce au modèle représentés par des ronds
- des flèches qui représentent l'erreur pour chaque points, en amplitude et direction.

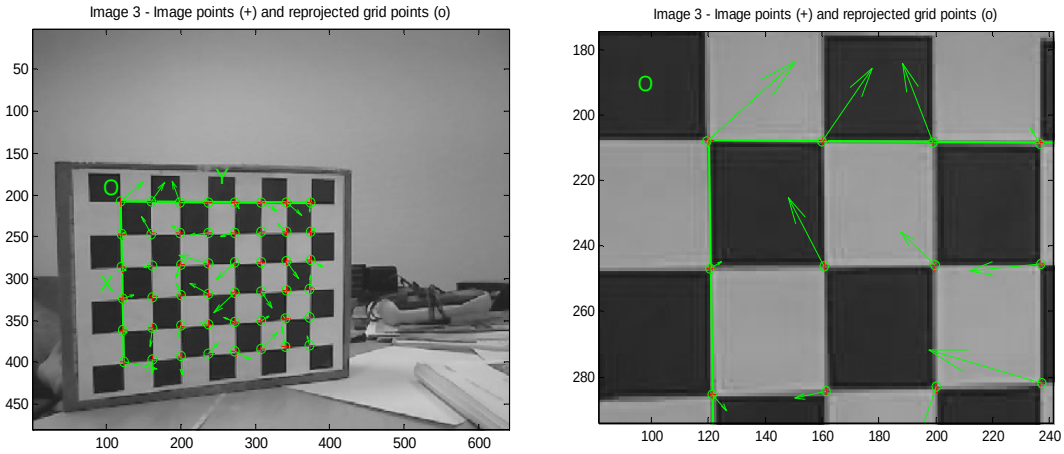

Figure 2 : Projections des points sur l'image.
gauche : Projections de la grille de points ; droite : zoom de la figure gauche

En cliquant sur un point de la figure d'erreur de reprojection, on voit afficher les informations correspondantes :

Selected image: 4
Selected point index: 37
Pattern coordinates (in units of (dX,dY)): (X,Y)=(0,1)
Image coordinates (in pixel): (127.96,212.34)
Pixel error = (-1.18650,0.59937)
Window size: (wintx,winty) = (5,5)
done

Si l'on détecte qu'une image a beaucoup de points présentant des erreurs importantes, cela signifie probablement que cette image n'a pas été bien traitée (flou, extraction de points, …). Il peut donc etre judicieux de supprimer ces images pour qu'elles ne perturbent pas l'estimation des paramètres réels de la caméra. Le bouton « *Add/Supress images* » permet de supprimer les images perturbatrices.

Relancer ensuite le calibrage par le bouton « *Calibration* ».

Le bouton « *Save* » vous permet de sauvegarder les paramètres. Ce qui peut être utile si on ne veut pas reprendre du début en cas de plantage de Matlab …

Le bouton « *Load* » permet de récupérer les données sauvegardées.

I.3. OPENCV : Utilisation des fonctions de calibrage

Etudier l'annexe dans un premier temps, avant de poursuivre les étapes suivantes (faite le avant d'arriver en TP !). La différence entre Matlab et cette bibliothèque est qu'il faut tout programmer, mais l'avantage est que le code réalisé peut être implémenter quelque soit la plateforme. Il n'y a pas de Toolbox graphique associée à la bibliothèque. Néanmoins, le livre « Learning OpenCv » possède tous les codes nécessaires pour calibrer une caméra. Pour ceci, orientez vous vers le chap.11. Après une brève introduction et une présentation de quelques fonctions utiles pour vérifier visuellement les résultats (*cvFindChessboardCorners*, *cvDrawChessboardCorners*, *cvFindCornerSubPix*, …), vous implémenterez un code qui retourne et qui sauvegarde les paramètres intrinsèques des caméras en utilisant la fonction *cvCalibrateCamera2* (aidez vous de l'exemple 11-1 p.398 du livre). N'oublier pas de commenter votre code de manière « intelligente ».
Il n'y a pas de fonctions permettant d'analyser les erreurs de calculs comme dans Matlab. Je laisse ce soin pour les plus perspicaces.

I.4. Vérification des paramètres

Prendre les images *SIER_balle_dr.jpg* et *SIER_balle_ga.jpg*. Ces deux images nous servirons à tester si les distances focales des deux caméras trouvées par les étapes précédentes sont correctes.

Figure 3 : Image test pour la calibration.
gauche : image de la caméra gauche ; droite : image de la caméra droite

Les balles (Bleue – Rouge – Verte) sont placées à des positions déterminées. Le but est de retrouver les distances *Z* séparant les balles des caméras connaissant le diamètre des balles (Ø = 60mm).
En prenant le schéma de la modélisation des paramètres intrinsèques d'une caméra, on peut déduire la relation :

$\frac{D}{d} = \frac{Z}{f}$, avec *D*=diamètre de la balle en mm, *d*=diamètre de la balle en pixel et *f* est la distance focale trouvée précédemment. *Z* sera exprimé en mm.

Pour le test, le diamètre en pixel *d* des balles peut être trouvé soit automatiquement par Matlab ou avec OpenCv (filtrage couleur, segmentation, …), soit vous pouvez vous aider d'un logiciel de dessin (type Gimp). Dans ce cas, faire une moyenne sur plusieurs mesures.
Faire un tableau qui informe sur l'image (gauche ou droite), sur l'outil utilisé (Matlab ou OpenCv), le diamètre des différentes balles mesurées en pixels, les distances focales et enfin sur la distance Z.
Comparer les résultats entre groupe « Matlab » et « OpenCv ».

II. Stéréovision

II.1. Stéréovision sous Matlab

- Lancer *stereo_gui.m* depuis Matlab
- Suivre le cinquième exemple en utilisant les images de chaque caméra jusqu'à obtenir les bons paramètres de calibration (http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example5.html).

Après avoir calibré les caméras par cet outil, le panneau « Stereo Toolbox » vous semblera bien intuitif.
Normalement, à la fin de cette étape, vous obtenez les différents paramètres de la stéréo calibration. N'oubliez pas de sauvegarder les paramètres …

Stereo calibration parameters:

Intrinsic parameters of left camera:
Focal Length: *fc_left = [xxx xxx] ± [000 000]*
Principal point: *cc_left = [xxx xxx] ± [000 000]*
Skew: *alpha_c_left = [x.xxx] ± [0.00000]*
Distortion: *kc_left = [xxx xxx xxx xxx xxx] ± [000 000 000 000 000]*

Intrinsic parameters of right camera:
Focal Length: *fc_right = [xxx xxx] ± [000 000]*
Principal point: *cc_right = [xxx xxx] ± [000 000]*
Skew: *alpha_c_right = [x.xxx] ± [0.00000*
Distortion: *kc_right = [xxx xxx xxx xxx xxx] ± [000 000 000 000 000]*

Extrinsic parameters (position of right camera wrt left camera):
Rotation vector: *om = [xxx xxx xxx] ± [000 000 000]*

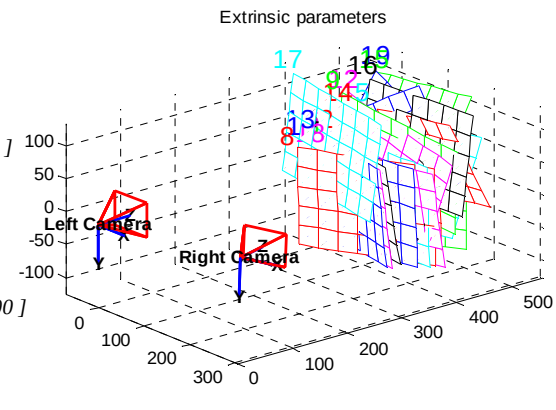

Figure 4 : stéréovision.
gauche : paramètres de la stéréovision ; droite : visu des paramètres extrinsèques

II.2. Stéréovision sous OpenCv

La bibliothèque propose une fonction qui permet de retrouver les relations géométriques entre les deux caméras (chap. 12 p. 427 « *Stereo Calibration* »). Il s'agit de la fonction *cvStereoCalibrate* qui retourne les vecteurs rotation et translation qui lient les caméras entre elles. En vous aidant de la première partie de l'exemple de 12-3 p.446 du livre, implémenter un code qui retourne ces deux vecteurs. Faites afficher la détection des coins de la mire pour la paire d'images caméra gauche et caméra droite.

N’oubliez pas de commenter le code et de sauvegarder les résultats.

II.3. Vérification des paramètres

Prendre les images *SIER_droite.jpg* et *SIER_gauche.jpg*. Ces deux images nous servirons à tester si les vecteurs rotation et translation entre les des deux caméras trouvées par les étapes précédentes sont correctes.

Figure 5 : Image test pour la stéréo calibration.
gauche : image de la caméra gauche ; droite : image de la caméra droite

Le but est de retrouver la distance *Z* qui sépare le point lumineux des caméras par triangulation.

La toolbox Matlab fournit une fonction qui permet de retrouver la coordonnée 3D d’une paire de points homologues (*stereo_triangulation.m*). Par contre, sous OpenCv cette fonction n’existe pas. Mais vous trouverez dans le campus la fonction équivalente traduite en langage C (*cvStereoTriangulation.c*). Vous téléchargerez cette fonction et l’adapterez pour l’utiliser dans votre TP.
Analyser et commenter le code.

Faire un tableau qui informe sur l’outil utilisé (Matlab ou OpenCv), sur l’image (gauche ou droite), sur la position du point lumineux (en pixels), les distances focales de chaque caméra, des vecteurs translation et rotation d’une caméra par rapport à l’autre et enfin sur la coordonnée 3D du point lumineux.

Normalement à la fin de ce TP, vous pourrez retrouver les coordonnées 3D de paire de points homologues provenant du système de stéréovision. Le problème maintenant est de détecter de manière automatique la paire de points homologues. Ce qui est le but du prochain TP …

ANNEXE

Introduction

Opencv est une bibliothèque de traitement d'images et de vision par ordinateur en langage C/C++, optimisée proposée par Intel pour Windows et Linux. Elle comprend un très grand nombre d'opérateurs “classiques”. Parmi lesquels :

- Création/libération d'images, macros d'accès rapide aux pixels
- Opérateurs standard :
  - morphologie
  - filtres dérivatifs, filtres de contours
  - suppression de fond
  - recherche de coins
- Recherche, manipulation, traitement de contours
- Création et utilisation d'histogrammes
- Changements d'espaces de couleurs (RGB, HSV,L*a*b* et YCrCb)
- Interface Utilisateur
  - Lecture/écriture d'images (JPEG, PPM, …)
  - Affichage à l'écran
  - Gestion des signaux sur clic de fenêtre, fermeture, . . .

OpenCV implémente aussi certains opérateurs plus complexes:

- Mean Shift
- Analyse du mouvement : Flot optique, MHI
- Calibrage de caméras (possible à partir d'un échiquier)
- Suivi d'objets 3D avec plusieurs caméras
- Mise en correspondance entre deux images
- lecture d'images à la volée directement depuis une vidéo AVI ou une caméra.

Opencv implémente aussi un certain nombre de structures de donnée particulièrement pratiques:

- Tableaux, matrices et matrices creuses + opérateurs associés
- Opérateurs “classiques” de matrices (inversion, déterminant, transposée, valeurs/vecteurs propres, distance de Mahalanobis, . . .)

Manipulations de base

Remarques Importantes :

- une image OpenCV est stockée dans une structure nommée IplImage ;
- dans une image RVB définie avec OpenCV, les composantes sont inversées (dans l'ordre : Bleu, Vert, Rouge) ;
- l'origine d'une image est par convention le coin en haut à gauche.

Création d'images

Fonctions et structures utilisées :

- Déclaration : IplImage
- Création : cvCreateImage
- Libération de l'espace mémoire : cvReleaseImage
- Dimensions d'une image : cvGetSize

```
// Déclaration et création
#include <cv.h>
IplImage *im; /* Image 512 x 256 de unsigned char avec 3 canaux */
im = cvCreateImage(cvSize(512,256), IPL_DEPTH_8U, 3);
```

```
// Remplissage : (codage par défaut=BGR)
pteur = (unsigned char*)im->imageData;
fin = im->width*im->height;
for(i=0; i<fin; i++)
{
  *(pteur++) = 255; /* Bleu */
  *(pteur++) = 0; /* Vert */
  *(pteur++) = 0; /* Rouge */
}

// Libération de l'espace mémoire
cvReleaseImage(&im);

// Créer une image cpy aux mêmes dimensions qu'une image src
IplImage* cpy = cvCreateImage(cvGetSize(src), src->depth, src->nChannels);
```

Lire, afficher et sauvegarder une image

Fonctions et structures utilisées :

- Lire une image : cvLoadImage
- Sauvegarder une image : cvSaveImage
- Ouvrir une fenêtre graphique : cvNamedWindow
- Afficher une image : cvShowImage
- Interactions clavier : cvWaitKey

```
// Lire une image
#include <highgui.h>
IplImage *im;
im = cvLoadImage("mon_image.jpg", 1); /* (1) */ /* 1 => 3 canaux (0 => 1 seul, -1
=> automatique) */
im = cvLoadImage("mon_image.jpg", 0); /* (2) */

// Ouvrir une fenêtre graphique
cvNamedWindow("Ma fenetre", CV_WINDOW_AUTOSIZE);

// Afficher une image
cvShowImage("Ma fenetre", im);
cvWaitKey(0); /* attend qu'une touche soit pressee */

// Interactions clavier
int key = cvWaitKey(0); /* attend qu'une touche soit pressee */

// Sauvegarder une image
cvSaveImage("mon_image.bmp",im);
```

Détection de contours

On trouve les filtres de Canny, Sobel, Laplacien, détection de coins…

Exemple du Laplacien : cvLaplace

```
im = cvLoadImage(argv[1], 0);
im2 = cvCreateImage(cvGetSize(im), IPL_DEPTH_16S, 1);
cvLaplace(im, im2);
```

Pour afficher le résultat du laplacien, il faut convertir l'image du laplacien en une image affichage par OpenCV : cvConvertScale

```
aff = cvCreateImage(cvGetSize(im), IPL_DEPTH_8U, 1);
cvConvertScale(im2,aff);
```

Compiler un code OpenCV sous linux

Un makefile d'exemple ainsi qu'un début de code est disponible dans le campus.

Préparez vous à utiliser cette bibliothèque avant d'arriver en TP, vous pouvez d'ailleurs suivre le tutoriel « Premiers pas avec OpenCV » à l'adresse suivante :
http://www.siteduzero.com/tutoriel-3-8565-premiers-pas-avec-opencv.html