Travaux pratiques de Perception

-

Calibration

Mise en place de l'environnement

Ce TP nécessite l'utilisation du langage de programmation Python. Pour cela, nous allons créer un environnement virtual à l'aide de l'outil Anaconda.

- Lancer *Anaconda Prompt* sous Windows;
- Créer un nouvel environnement avec la commande `conda create -n Calibration python=3.7`
- Activer l'environement créé avec `conda activate Calibration`
- Installer les dépendances nécessaires
  - `conda install scikit-learn numpy matplotlib`
  - `conda install -c conda-forge opencv`
- Lancer le script test.py pour s'assurer que tout fonctionne comme il faut.
  `python test.py`

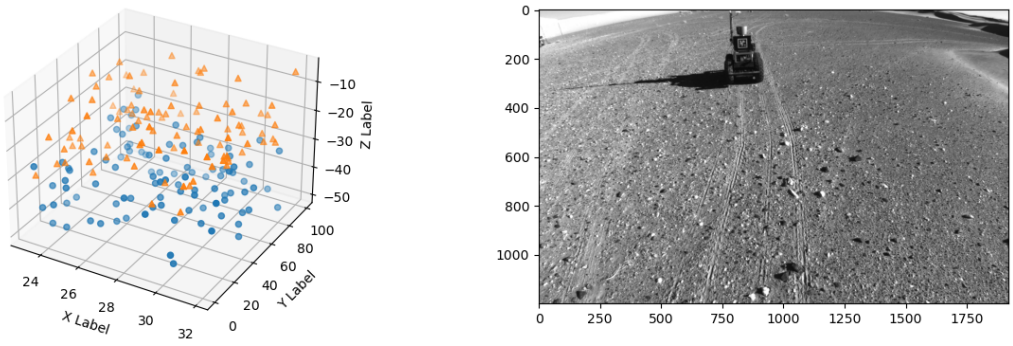

Figure 1: Résultats du test

1 Vision monoculaire

1.1 Calibration Monoculaire

Cette partie a pour objectif de mettre en place un programme permettant la calibration intrinsèque d'une seule caméra. La calibration intrinsèque d'une caméra vise à déterminer les param
'etres tels que la focale (f_x, f_y) et la position du centre optique dans l'image (c_x, c_y) ainsi que les paramètres de distortion de la lentille (distortion radiale et tangentielle principalement).

En pratique, après avoir chargé les images des mires, il est nécessaire de détecter les coins de la mire dans chacune des images, de faire correspondre ces points image avec leur position réelle et de réaliser la calibration.

1.1.1 Détection de la mire

1. Ouvrir une image de calibration et identifier le type de mire et ses caractéristiques (nombre de lignes et de colonnes, taille).

2. Dans le fichier `CalibrationTools.py`, compléter la fonction `detectPattern` afin de détecter la mire dans chacune des images. On utilisera pour cela les fonctions `findChessboardCorners` ou `findCircleGrid` de la bibliothèque openCV.

3. Lancer le programme `CalibrationMono.py` pour tester la détection.

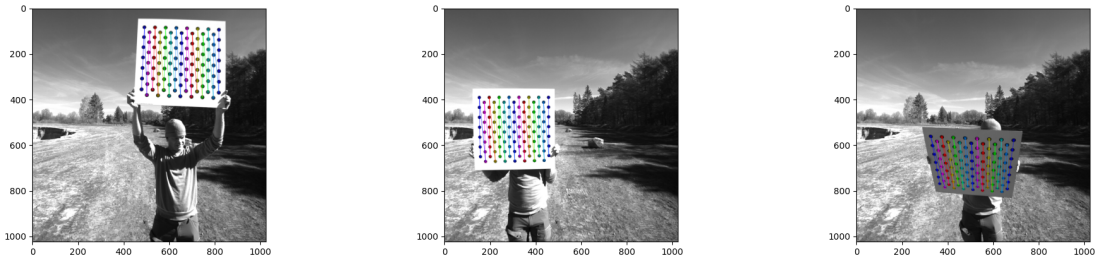

Figure 2: Résultats du test

1.1.2 Calibration

1. Compléter la fonction `asymmetricWorldPoints` afin de créer la liste des points 3D correspondant aux différents coins d'une mire.

2. Compléter la fonction `calibrateMono` afin d'obtenir les paramètres de calibrations de la caméra. On utilisera la fonction `calibrateCameraExtended` de la bibliothèque openCV.

3. A quoi correspondent les variables *mtx* et *dist* retournées par la fonction de calibration? Leurs valeurs semblent-elles correctes?

4. Que représentent les variables *rvecs* et *tvecs*? Pour cela, observer le résultat de l'appel à la fonction `visualizeBoards`

1.1.3 Rectification

Dans cette partie, nous allons procéder à la rectification des images d'une caméra à l'aide de la bibliothèque OpenCV.

1. Pour cela, openCV utilise des *cartes de correction*. Implémenter la fonction `initUndistortRectifyMap` de OpenCV dans la fonction `computeCorrectionMapsMono`. A quoi correspondent *mapx* et *mapy*

2. A l'aide de la fonction `remap` de openCV, compléter la fonction `rectify` du fichier `CalibTools.py`

3. Lancer le programme `CalibrationMono.py` pour afficher les résultats. Que constatez vous sur les images résultat? A quoi cela est-il du?

1.2 Analyse

Nous allons analyser les résultats de la calibration et, si besoin, tenter d'en améliorer les résultats.

1. Observer le résultat de l'appel à la fonction `plotRMS`. Que repésente la figure obtenue? Que peut-on en déduire sur la qualité de la calibration? Quels sont les facteurs importants pour réaliser un étalonnage précis?

2. Dans notre cas, que pourrait-on faire pour améliorer les résultats obtenus?

3. Quelles sont les valeurs intrinsèques, les coefficients de distorsion ainsi que le RMS obtenus en fin de calibration ?

2 Stéréovision

Dans cette partie, nous allons calibrer un banc stéréo. Pour cela, il est nécessaire d'avoir une bonne connaissance des paramètres intrinsèques de chacune des caméras composant le banc stéréo. La calibration du banc consiste à determiner la transformation entre les 2 caméras (paramètres extrinsèque) afin de rectifier chacune des images pour se retrouver dans une configuration où chaque ligne d'une image correspond à la même ligne sur l'autre image (respect de la contrainte épipolaire).

2.1 Calibration Séréo

1. Dans la fonction `CalibrateStereo` du fichier `CalibTools.py`, utiliser la fonction `calibrateCameraExtended` vue au §1.1.2 pour effectuer la calibration intrinsèque de chacune des caméras droite et gauche, puis réaliser une calibration extrinsèque du banc à l'aide de la fonction `stereoCalibrateExtended` d'openCV.

2. A quoi correspondent les variables R et T retournées par la fonction `stereoCalibrateExtended`.

3. Lancer le programme `CalibrationStereo.py` pour afficher les résultats de la calibration stéréo et afficher le graphe du RMS. Quelles sont les valeurs des paramètres intrinsèques, extrinsèques, des coefficients de distorsion et du RMS ? Les valeurs semblent elles cohérentes ?

2.2 Rectification

Une fois l'étalonnage effectué nous allons procéder à la stéréo-rectification afin que notre paire d'image respecte la contrainte épipolaire.

1. Dans la fonction `computeCorrectionMapsStereo` du fichier `CalibTools.py`, utiliser la fonction `stereoRectify` d'openCV afin de déterminer les paramètres de rectification des images gauche et droite.

2. Dans cette même fonction, utiliser la fonction `initUndistortRectifyMap` vue au §1.1.3 afin de créer les cartes de correction pour chacune des images gauche et droite.

3. Lancer le programme `CalibrationStereo.py` pour afficher les résultats de la rectification des images. Que peut-on observer? Que représentent les lignes affichées?

2.3 Reconstruction 3D

La dernière étape consiste à reconstruire l'environnement en 3D à partir d'une paire d'images.

1. Relancer une dernière fois le programme `CalibrationStereo.py` et observer les résultats obtenus sur les images de test.

2. A quoi correspond l'image *Disparity map*? Que représentent les valeurs de cette image?

3. Comment peut-on obtenir une carte de profondeur à partir d'une image de disparité ?

4. Comment reprojeter une carte de profondeur 2D en un nuage de points 3D ?