

PNP_Backend - Manuel

Introduction

Ce projet contient un solveur pour le problème "[Perspective-n-Points](#)" [↗](#) (PnP). Il s'agit, étant donnée une photo, d'estimer les caractéristiques et la position de l'appareil de prise de vue.

Cette estimation se fait à partir de **n** points dont on connaît à la fois les coordonnées réelles (en 3 dimensions) et les coordonnées dans la photo (en 2 dimensions). L'estimation repose sur une modélisation du comportement des appareils photo basée sur la projection perspective. D'où le nom de "Perspective-n-Points".

Cette page indique comment installer, démarrer, et utiliser le solveur.

Pour visualiser un exemple d'utilisation ou en savoir plus sur les bases théoriques de ce projet, consultez [notre wiki](#) [↗](#).

Implémentation

Le **solveur** en lui-même est implémenté en C++, avec l'aide de la librairie [openCV](#) [↗](#).

Un **serveur web** vient compléter l'implémentation : il permet de rendre le solveur accessible depuis une machine distante, et facilite la communication avec le solveur grâce à l'utilisation du format *JSON*. Ce serveur est codé en *nodeJS*.

Enfin, l'application (solveur + serveur) est encapsulée dans un **conteneur** [docker](#) [↗](#). Cela permet de l'installer sur n'importe quelle type de machine (windows, mac, linux), avec pour seule dépendance docker lui-même.

Installation

- Assurez-vous d'avoir **docker** installé sur votre machine. Sinon, consultez la [page de téléchargement](#) [↗](#) et installez la version adaptée à votre machine.
- Téléchargez ce repo.
- Optionnel : téléchargez l'application [Postman](#) [↗](#) pour effectuer plus facilement des requêtes vers le serveur.

Usage

Démarrage du serveur

Pour démarrer le serveur, lancez la commande : `docker-compose up`.

Le premier démarrage peut être long (quelques minutes), car les dépendances sont téléchargées et compilées dans le conteneur. Cette étape ne sera pas exécutée aux démarrages suivants, les rendant bien plus rapides (de l'ordre de quelques secondes). Le serveur est prêt à recevoir des requêtes dès qu'il affiche le message *"PnP solver running on port 3000. Waiting for requests..."*.

Pour vérifier votre configuration, entrez l'adresse *"localhost:3000"* dans un navigateur. Il devrait afficher la phrase *"Hello from PnP solver"*.

Effectuer un calcul

Pour lancer un calcul, on envoie une requête **POST** sur la route **localhost:3000/solve**. Le corps de la requête doit être au format JSON, et respecter le format suivant :

```
1 | {
2 |     // tableau contenant les correspondances 2D-3D.
3 |     // Doit contenir au minimum 6 correspondances
4 |     bindings: [
5 |         {
6 |             // coordonnées d'un point réel
7 |             point3D: { x: Number, y: Number, z: Number },
8 |             // coordonnées du point correspondant sur la photo
9 |             point2D: { x: Number, y: Number }
10 |        }
11 |        //...
12 |    ],
13 |    // dimensions de la photo en pixels
14 |    imageDimensions: {
15 |        width: Number, // largeur de l'image en pixels
16 |        height: Number // hauteur de l'image en pixels
17 |    }
18 | }
```

Attention, **les coordonnées 2D doivent être comprises entre $-\frac{1}{2}$ et $\frac{1}{2}$** (le coin inférieur gauche de la photo ayant pour coordonnées $[-\frac{1}{2}, -\frac{1}{2}]$).

Interpréter le résultat


La réponse du serveur contient une estimation des paramètres de l'appareil ayant pris la photo. Elle contient également une estimation de l'erreur globale, et de l'erreur par point :

```
1 | {
2 |     camera: {
3 |         // la matrice de la caméra,
4 |         // indiquant le positionnement de l'appareil en 3d
5 |         matrix: [Number(16)],
```

```

6      // angle de vue vertical (en degrés)
7      vFOV: Number,
8      // déviation horizontale du centre optique de l'appareil
9      cx: Number,
10     // déviation verticale du centre optique de l'appareil
11     cy: Number,
12     // largeur de la photo en pixels
13     width: Number,
14     // hauteur de la photo en pixels
15     height: Number,
16   },
17   errorEstimation: {
18     // tableau contenant les erreurs par point
19     bindingsError: [Number(n)],
20     // estimation de l'erreur globale
21     globalError: Number
22   }
23 }

```

Attention : le champ **matrix** correspond aux éléments de la matrice selon le format adopté par *THREEjs* (cf <https://threejs.org/docs/#api/en/math/Matrix4> )

L'appareil peut ainsi être simulé en *THREEjs* de la façon suivante :

```

1  function simulateCamera(camera) {
2      const { matrix, cx, cy, vFOV, width, height } = camera;
3      const fakeCamera3D = new THREE.PerspectiveCamera(vFOV, width / height, 1,
4      const m = new THREE.Matrix4().set(...matrix).transpose();
5      const position = new THREE.Vector3();
6      const quaternion = new THREE.Quaternion();
7      const scale = new THREE.Vector3();
8      m.decompose(position, quaternion, scale);
9      fakeCamera3D.position.copy(position);
10     fakeCamera3D.quaternion.copy(quaternion);
11     fakeCamera3D.setViewOffset(width, height, width * cx, height * cy, width,
12
13     fakeCamera3D.updateMatrix();
14     fakeCamera3D.updateMatrixWorld();
15     fakeCamera3D.updateProjectionMatrix();
16 }

```

Développement

Solveur

Le code du solveur se trouve dans le fichier *src/solver/camCalibNode.cpp*. En cas de mise à jour, il doit être recompilé. Pour cela, stoppez le conteneur docker (Ctrl-C) et relancez-en un nouveau avec l'option `--build`, pour forcer la compilation :

```
docker-compose up --build
```

Serveur

Le code du serveur se trouve dans le dossier *src/server*. Il n'est pas nécessaire de relancer le conteneur docker lorsque vous mettez à jour ce code : la mise à jour est faite automatiquement.