

# Rapport: Stéréovision

Travail d'automne

Sol Rosca (INF3b)

Automne 2019

# Resumé

Ce rapport décrit le développement du projet Stéréovision. Composé de deux parties distinctes, ses objectifs sont d'offrir d'une part un cours théorique sur le sujet de la vision stéréoscopique et d'autre part un logiciel compagnon capable d'illustrer les différents points théoriques du cours théorique.

---

This report describes the development processes of Stéréovision, a project with two goals. The first one is to write a syllabus that gives the theoretical basis needed for understanding the stereoscopic vision and could be used as support for a lecture. The second one is to deliver a software that is able to demonstrate the theoretical concepts of the first part.

# Table des matières

<b>1. Introduction</b>	<b>1</b>
<b>2. Cours théorique</b>	<b>3</b>
2.1. Cahier des charges	3
2.2. Analyse	3
2.2.1. Problématique	3
2.3. Conception	4
2.3.1. Structure	4
2.3.1.1. I. Modélisation de caméra	4
2.3.1.2. II. Distorsion	4
2.3.1.3. III. Préliminaires à la vision stéréoscopique	4
2.3.1.4. IV. Géométrie épipolaire	4
2.3.1.5. V. Vision stéréoscopique	5
2.3.2. Illustrations	5

<b>3. Logiciel</b>	<b>6</b>
3.1. Cahier des charges	6
3.2. Analyse	7
3.2.1. Besoins de l'application	7
3.2.2. Spécifications	8
3.2.2.1. Modes	8
3.2.2.2. Options	8
3.2.2.3. Interface utilisateur	9
3.2.3. Technologies	9
3.2.3.1. Langage	9
3.2.3.2. Librairies	9
3.2.3.3. Conventions et style	10
3.3. Réalisation	11
3.3.1. Architecture	12
3.3.2. Packages	13
3.3.2.1. camera_system	13
3.3.2.2. strategies	16
3.3.2.3. controllers	17
3.3.3. Modules	21
3.3.3.1. store.py	21
3.3.3.2. settings.py	21
3.3.4. Clients	22
3.3.5. GUI	22
3.4. Matériel	23
3.5. Utilisation	25
3.5.1. Dépendances	25
3.5.2. Installation	25
3.5.3. Configuration	25
3.5.4. CLI	26
3.5.5. GUI	31
3.6. Expérimentation	35
<b>4. Conclusion</b>	<b>37</b>
<b>5. Références</b>	<b>38</b>
5.1. Outils	38
5.2. Librairies python	38
5.3. Papier	39
5.4. Articles, Cours & Tutos	39
5.5. Cours vidéo	39
5.6. Interactif	39
<b>6. Annexes</b>	<b>40</b>

# 1. Introduction

Le sens de la vue est probablement l'un des plus importants dans notre société. Cela peut sembler banal mais l'être humain se sert quotidiennement de son système de vision pour de nombreuses tâche: déplacement, estimation des distances, identification des objets, ... Et tout cela sans même y préter attention ou considérer la complexité des mécanismes en oeuvre.

La vision stéréoscopique est une spécialisation de la vision par ordinateur, discipline à cheval sur entre autres, les mathématiques, l'informatique, la physique, la biologie et les sciences du cerveau, dont le but est de comprendre la vision humaine pour la simuler et en doter des machines. Ce domaine de recherche ne cesse de se développer et se trouve au coeur de nombreux secteurs d'activité.

Globalement, l'objectif principal de la vision par ordinateur est de déduire, à partir d'images capturées par un ou plusieurs capteurs, des informations concernant les mouvements, les formes et les distances entre les éléments d'une scène. La déduction des distances concerne l'automatisation de l'extraction d'informations concernant la structure dans l'espace d'une scène à partir d'une ou *plusieurs images*<sup>0</sup>.

La vision stéréoscopique et plus particulièrement la vision stéréoscopique binoculaire se base sur la déduction d'informations 3D d'une scène, en utilisant deux images prises à partir de points de vue différents.

C'est dans ce contexte que se trouve le projet "stéréovision" proposé par le professeur Tièche dans le cadre du travail d'automne. Initialement, le cahier des charges de ce projet visait une potentielle application industrielle de la vision stéréoscopique binoculaire mais, dans les premières semaines du travail, un changement d'orientation a été décidé en accord avec le professeur.

Il a été décidé que le travail prendrait une dimension pédagogique et serait préliminaire à un éventuel cours de deux périodes, à donner au second semestre aux étudiants en dernière année de bachelor de la filière développement logiciel et multimédia. Aussi, ce travail doit pouvoir être une base permettant à un élève futur de rapidement comprendre les concepts liés à la vision stéréoscopique, pour pouvoir les adapter à une application concrète.

C'est donc dans ce nouveau contexte que se trouve la justification des deux parties distinctes de ce rapport:

- Une première partie axée sur l'élaboration d'un cours théorique sur la vision stéréoscopique binoculaire.
- Une seconde partie axée sur le logiciel en charge d'illustrer les concepts de la première partie.

Ensemble, ces deux parties forment le rapport du travail d'automne 2019 "Stéréovision" offert par le professeur Tièche et réalisé par Sol Rosca, étudiant de 3ème année de la filière DLM.

## 2. Cours théorique

Cette première partie s'articule autour de la réflexion portée sur la mise en place et la réalisation d'un support de cours introductif à la vision stéréoscopique, destiné à des élèves de 3ème année dans la filière DLM.

### 2.1. Cahier des charges

Cette partie du projet a pour objectif de:

- Comprendre le sujet de la stéréovision
- Développer une vision globale de la matière
- Déterminer les objectifs à atteindre à l'issue d'un cours introductif sur cette matière
- Rédiger un cours qui va dans le sens du point précédent

### 2.2. Analyse

Ce chapitre décrit le problème et identifie les axes principaux qu'il sera nécessaire de développer. À partir de maintenant, les termes "vision stéréoscopique" et "stéréovision" désigneront tous deux la vision stéréoscopique binoculaire.

#### 2.2.1. Problématique

La stéréovision se base sur plusieurs branches de la géométrie et nécessite, entre autres, une certaine aisance avec les concepts de l'optique, pour pouvoir saisir la pertinence ainsi que l'élégance des solutions apportées. De plus la stéréovision étant dans le prolongement de l'acquisition d'images et donc des techniques qui permettent la projection d'un objet du monde 3D sur un plan 2D, en effectuant un certain nombre de transformations et de rotations sur ce dernier, nécessite de bonnes notions d'algèbre linéaire et plus particulièrement d'être familier avec les matrices, ce qu'elles représentent et des opérations avec et sur ces dernières.

Le cours devra donc nécessairement passer par une forme de rappel des notions précédemment citées. Il ne s'agit pas de faire un cours sur ces points mais de rester dans une démarche pédagogique, en n'hésitant pas à faire certains rappels et en tentant de développer le plus clairement possible les démonstrations mathématiques.

La complexité du sujet traité est compensé par son caractère très visuel. En effet, l'utilisation d'infographies pertinentes peut grandement faciliter l'assimilation des concepts. De ce fait, le choix des outils pour la création de ces illustrations est un point qui ne doit pas être négligé.

## 2.3. Conception

### 2.3.1. Structure

Après une étude sommaire du sujet, les grands points suivants se dégagent comme étant des candidats pour les chapitres du cours:

#### 2.3.1.1. I. Modélisation de caméra

Ce chapitre se veut introductif, il apporte les notions de base et ne prend en compte qu'une seule caméra. Il comporte les sous chapitres suivants:

- Les modèles géométriques de base ainsi que leurs équivalents algébriques
- Les différents référentiels impliqués
- Les notions de paramètres intrinsèques et extrinsèques

#### 2.3.1.2. II. Distortion

Ce chapitre sera un rappel sur les différents types de distorsion et leur modélisation algébrique. Ce chapitre doit être bref pour ne pas disperser le lecteur dans de nombreuses problématiques pas forcément liées au sujet principal.

#### 2.3.1.3. III. Préliminaires à la vision stéréoscopique

Introduction d'éléments essentiels à la compréhension de la géométrie épipolaire:

- Notion de disparité
- Triangulation

#### 2.3.1.4. IV. Géométrie épipolaire

Ce chapitre est clé et devra, dans un premier temps, poser les concepts de base pour sensibiliser le lecteur à l'élégance de la solution apportée, avant de se concentrer sur la partie plus formelle.

- Modèle géométrique de deux caméras observant la même scène
- Épipôles
- Contrainte épipolaire
- Matrice essentielle
- Matrice fondamentale

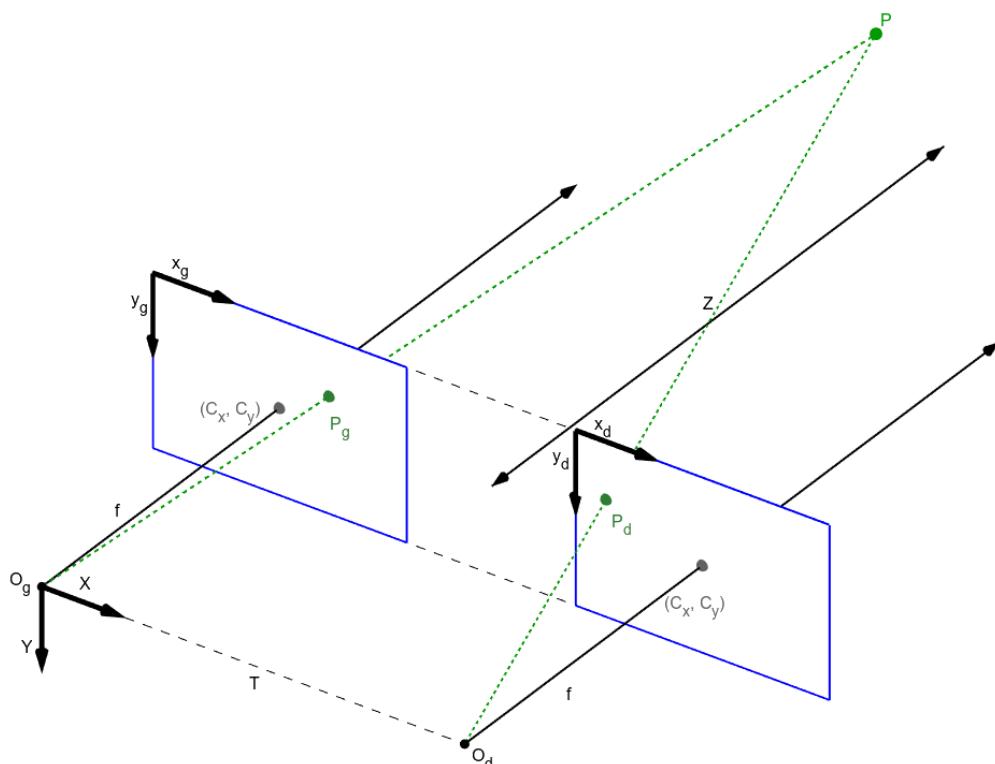
### 2.3.1.5. V. Vision stéréoscopique

Ce chapitre regroupe ce qui a été vu dans les précédents et couvre les 4 étapes nécessaires à la mise en place d'un système de vision stéréoscopique:

- Calibrage stéréo
- Rectification
- Recherche de correspondance
- Reconstruction 3D

### 2.3.2. Illustrations

Pour illustrer les concepts géométriques 3D, le logiciel [geogebra](#) est utilisé. En plus de générer des infographies pertinentes, les différents projets dont sont issus ces dernières sont accessibles en ligne ce qui permet aux lecteurs du cours de regarder une scène sous tous les angles pour faciliter la compréhension.



**Figure 1** <https://www.geogebra.org/m/arzudnnb> Exemple d'utilisation de géogebra

## 3. Logiciel

Cette seconde partie du rapport décrit les différentes phases du développement d'un logiciel dont le but est d'illustrer les concepts théoriques du cours détaillé dans la première partie du rapport.

### 3.1. Cahier des charges

Le logiciel devra posséder les caractéristiques suivantes:

- Illustrer les concepts clé de la vision stéréoscopique binoculaire:
  - Rectification d'images
  - Correction de la distorsion
  - Affichage d'une carte de disparité
  - Affichage d'une carte de profondeur
- Opérer sur un flux d'images (vidéo)
- CLI ou GUI
- Être maintenable pour permettre son extension

## 3.2. Analyse

Ce chapitre décrit l'analyse effectuée suite à l'étude du sujet de la stéréovision. Plus spécifiquement il traite des besoins ainsi que des spécifications du logiciel à produire.

### 3.2.1. Besoins de l'application

Avant de pouvoir afficher des cartes de profondeur, un système de vision stéréoscopique passe par plusieurs étapes:

1. Calibrage
2. Déduction des matrices essentielle et fondamentale
3. Rectification et correction de la distorsion
4. Génération de la carte de disparité
5. Traitement de la carte de disparité
6. Affichage de la carte de profondeur

En plus de ces étapes, une étape de pré-calibrage permettant à l'utilisateur de manuellement optimiser l'alignement vertical des caméras, en lui offrant des repères visuels, est souhaitable pour améliorer la qualité finale.

Aussi, il n'est pas nécessaire de passer par toutes les étapes à chaque utilisation du programme, ce dernier aura donc un comportement non déterministe qui se base sur l'existence de fichiers ou sur les choix de l'utilisateur.

### 3.2.2. Spécifications

Cette partie décrit les spécificités techniques du logiciel sans rentrer dans les détails d'implémentation.

- **Plateforme:** Desktop
- **OS:** Windows et Linux

#### 3.2.2.1. Modes

L'état du programme est géré par une machine d'état. Une interface permet à l'utilisateur de transiter entre ces états. Chaque état correspond à une des étapes du système de vision stéréoscopique:

- **Pré-calibrage**
- **Calibrage**
- **Affichage de la rectification/distorsion**
- **Carte de profondeur**

Dans la suite de ce rapport, ces états seront appelés "modes de l'application".

#### 3.2.2.2. Options

Chacun des modes de l'application affiche soit une *paire d'images*<sup>1</sup> issues des caméras du systèmes, soit *une image*<sup>2</sup> résultant du traitement fait sur les deux captures. Chaque mode permet d'activer ou de désactiver des options qui lui sont spécifiques:

- **Pré-calibrage:** Affichage de lignes horizontales pour aider à la calibration. Ces lignes sont appelées "lignes horizontales"
- **Calibrage:** Ce mode est seulement utilisé lorsque l'état physique du système de caméras est modifié. Il permet dans un premier temps de capturer un certain nombre d'images avec une mire de calibrage visible et, ensuite, à partir de ces images, il lance le processus de calibration qui génère un certain nombre de fichiers contenant les données issues de cette calibration.
- **Affichage de la rectification/distorsion:** Ce mode permet d'afficher les mêmes lignes horizontales que le mode "pré-calibrage" mais offre, en plus, la possibilité d'appliquer ou non aux deux images une transformation issue du résultat du calibrage.
- **Carte de profondeur:** Ce mode affiche une seule image qui est le produit du traitement des deux captures auquelles sont appliquées la rectification et la correction de la distorsion ainsi que le traitement nécessaire à calculer une carte de disparité. Ce mode offre la possibilité de changer plusieurs paramètres qui influencent l'image produite.

---

1. Pré-calibrage, calibrage, rectification

2. Carte de profondeur

### 3.2.2.3. Interface utilisateur

Pour permettre l'interaction avec le logiciel, ce dernier propose une interface à l'utilisateur. Cette interface est en charge de la transition entre les différents modes ainsi que de l'activation des différentes options possibles dans le mode courant. De plus, l'interface est en charge de relayer à l'utilisateur les éventuelles exceptions levées par le programme.

## 3.2.3. Technologies

Cette partie décrit les choix concernant les technologies employées.

### 3.2.3.1. Langage

Pour une application qui a pour but d'illustrer des concepts théoriques et sans avoir de contraintes particulières au niveau performances, Python est un sérieux candidat. Les autres avantages de Python sont:

- L'existence d'un wrapper OpenCV avec une communauté très active
- Facilité de *mise place d'un environement*<sup>1</sup>
- Aisance d'utilisation
- Popularité du langage
- Portabilité du langage

### 3.2.3.2. Librairies

- **OpenCV:** Écrit en C++, c'est probablement la librairie de vision par ordinateur la plus populaire. Son wrapper python est très bien documentée et de nombreux exemples sont trouvables sur le net. Après recherche, elle possède toutes les abstractions nécessaires pour réaliser un programme autour de la vision stéréoscopique.
- **Numpy:** Dépendance du wrapper python d'OpenCV.
- **Stereovision:** Disponible sur PyPI et sur [github](#): Offre une abstraction pour l'utilisation de systèmes stéréoscopiques. Malheureusement, elle n'est plus maintenue et est douteuse dans certains de ses résultats. Elle n'est finalement utilisée que pour abstraire la partie calibration stéréoscopique et est entièrement incluse aux sources du projet. Ce dernier point est nécessaire pour permettre sa modification afin de l'adapter aux besoins de l'application.
- **Eel:** Librairie lightweight disponible sur PyPI et sur [github](#): Permet de faire le pont entre les technologies du web et python pour offrir une interface utilisateur moderne sans s'encombrer de technologies comme PyQt. Cette librairie s'apparente à ce qu'offre le plugin javascript Electron mais pour Python.

---

1. Du moins sur Linux

### 3.2.3.3. Conventions et style

Globalement, le style suit les conventions de *PEP8*<sup>1</sup> et le code tente d'être le plus idiomatique possible.

#### Annotations de type

Le projet utilise presque systématiquement les [annotations de type](#). Cette pratique optionnelle permet de spécifier des informations concernant le type, directement dans le code:

```
1 from typing import Dict, List
2
3 variable_classique = 5
4 variable_hint: int = 5
5 dictionnaire_hint: Dict[str, int] = {}
6
7 def fonction_classique(a, b):
8     pass
9
10 def fonction_avec_hint(a: str, b: List[int]) -> str:
11     pass
```

Python

La ligne 4 n'apporte pas grand chose. Il est facile de visuellement inférer le type de la variable. La ligne 5 démontre bien mieux le potentiel de ces annotations. Grâce à elles, en un coup d'oeil, le lecteur sait qu'il s'agit d'un mapping d'un entier sur une chaîne de caractères. De la même façon, la ligne 10 démontre le gain de lisibilité.

Cette pratique ne change rien au caractère dynamique du typage de python. Il apporte des éléments de typage statique *mais ce n'est que visuel*<sup>2</sup>.

#### Classes de données

La version 3.7 de Python apporte les [dataclass](#), une nouvelle façon de construire des classes n'ayant pour but que de contenir des données à l'aide de la librairie `dataclasses`. Ces classes sont des classes 100% comme les autres après définition et leur seul avantage est de proposer une syntaxe de définition plus légère:

```
1 from dataclasses import dataclass
2
3 @dataclass
4 class Poule: name: str; age: int
```

Python

- 
1. Standard concernant l'écriture et le formatage du code Python
  2. Il est possible, en utilisant des librairies dédiés, de faire un check statique d'un code qui utilise ces annotations.

### 3.3. Réalisation

Cette partie décrit le fonctionnement interne du logiciel. Dans les pages qui suivent, les termes modules, package et librairie seront utilisés. De nombreux abus de langages circulent concernant ces termes. Pour lever tout doute, voici ce qu'ils veulent dire dans ce rapport conformément aux [spécifications du langage Python](#):

- **Module:** Tout fichier source Python avec une extension en `.py`
- **Package:** Dossier contenant une collection de modules et un fichier `__init__.py`
- **Librairie:** Représente une collection de packages

De manière générale, un module du logiciel comporte soit une unique classe, soit des fonctions globales à un package.

Avant de rentrer dans le vif, la **figure 2** présente une vue de la hiérarchie du projet. Pour plus de lisibilité, ce diagramme ne contient pas tous les fichiers. Notamment, tous les fichiers `__init__.py` en sont exclus.

- `external/` contient une librairie externe au projet qu'il était nécessaire de modifier comme expliqué dans le chapitre 3.2.3.2.
- `generated/` n'existe pas initialement et est créé lors de la première calibration.
- `public/` contient les points d'entrée du programme ainsi qu'un fichier de configuration.
- `sources/` contient les sources du projet.
- `sources/backend` contient la logique fonctionnelle.
- `sources/camera_system` contient la logique métier.
- `sources/gui` contient le code lié à l'interface utilisateur graphique.

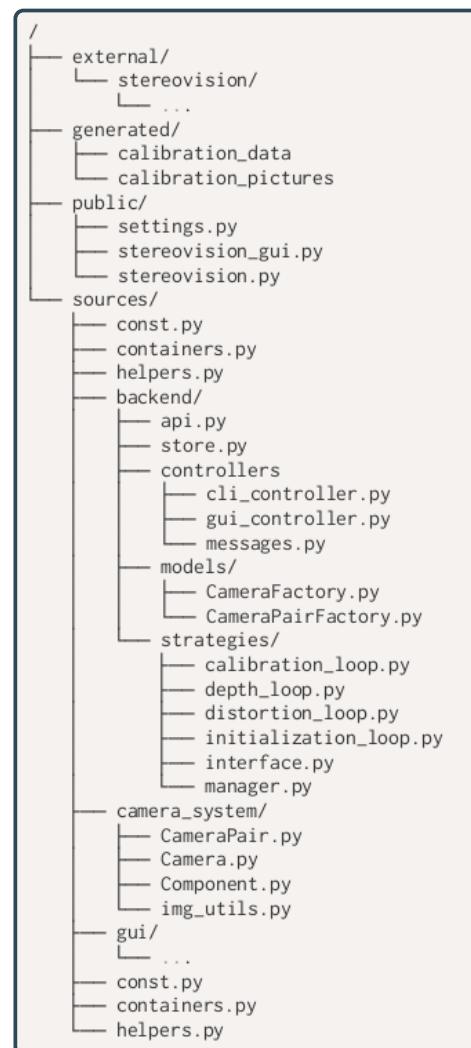
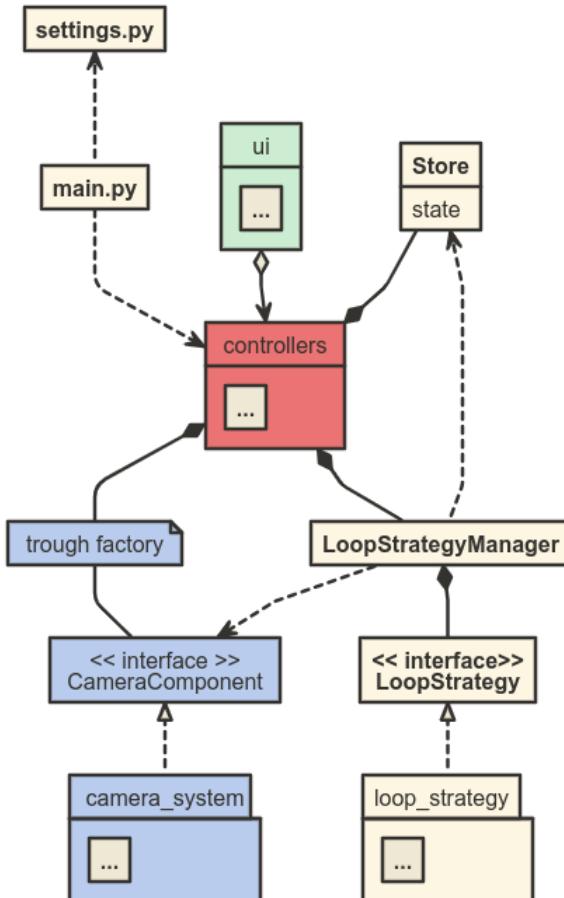


Figure 2 Hierarchy

### 3.3.1. Architecture

Le pattern architectural employé est MVC et suit la convention du fat model skinny controller.



La logique métier se trouve dans les modèles et la seule logique que les contrôleurs implémentent est fonctionnelle. La différence principale vis-à-vis d'un MVC *traditionnel*<sup>1</sup> est que la source de données n'est pas une API ou une base de données mais le flux des caméras.

L'avantage que propose cette architecture est le fait de renforcer le principe de responsabilité unique ainsi que la réduction du couplage entre composants, ce qui rend le logiciel plus maintenable.

- Models
- Présentation
- contrôleurs
- logique fonctionnelle

Figure 3 Vue haut niveau de l'architecture logicielle

Ces qualités tendent également à favoriser les contraintes pédagogique du projet, en proposant un code métier ségrégué et donc plus facile à comprendre.

À un niveau d'abstraction plus bas, les composants de l'application tendent à favoriser la composition sur l'héritage. Cette façon de faire est une autre façon de réduire le couplage entre les composants et d'ainsi favoriser la maintenabilité du logiciel.

La **figure 3** présente une vue haut niveau des composants qui seront décrits dans les prochains sous-chapitres.

1. Traditionnel car MVC est principalement un pattern architecturale employé dans le domaine du web.

### 3.3.2. Packages

Ce chapitre décrit le fonctionnement des différents composants ainsi que leurs liens.

Ce point décrit les différents packages du logiciel.

#### 3.3.2.1. camera\_system

Ce package est totalement indépendant du reste de l'application et peut être considéré comme une librairie utilisable dans d'autres contextes, stéréovision ou pas.

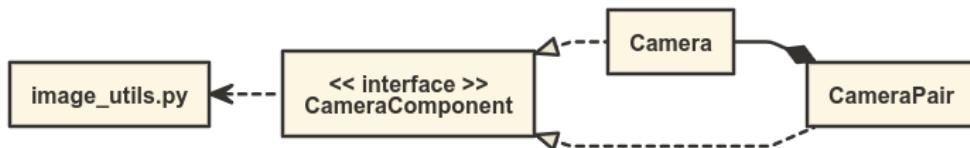


Figure 4 Détail du package camera\_system

L'implémentation de *camera\_system* suit le design pattern **composite**. En effet, `CameraPair` est une composition de `Camera` et il est intéressant de les modéliser de façon à mettre en évidence leurs points communs, tout en tirant profit de la hiérarchie "naturelle" qui les lie en les faisant implémenter la même interface. Cette façon de procéder présente plusieurs avantages:

- Utilisation polymorphique de `Camera` et `CameraPair` à travers l'interface `CameraComponent`, ce qui permet d'une part de généraliser le code et d'autre part de renforcer le principe *ouvert fermé*<sup>1</sup>.
- Structure le code, en donnant une ligne directrice quant aux responsabilités de chaque classe. `CameraPair` est la partie *composite* et donc en charge de déléguer à ses `Camera`, qui elles sont les *component* et donc en charge du "réel" travail, sans se préoccuper de qui le leur demande.

```

1  class CameraPair(CameraComponent):
2      def __init__(self, ...):
3          self._children: List[CameraComponent] = []
4          self._frames: List[np.ndarray] = []
5          # ...
6      # ...
7      def frame(self) -> List[np.ndarray]:
8          if not self._frames: self._frames = [c.frame() for c in self._children]
9          return self._frames
10     # ...
11     def show_frame(self) -> None:
12         cv2.imshow(str(self.id), self.frame())
13     # ...
  
```

Python

1. Principes SOLID, open/closed: Ouvert à l'extension, fermé à la modification.

```

1 class Camera(CameraComponent):
2     def __init__(self, ...):
3         self._frame: np.ndarray = None
4         # ...
5     # ...
6     def frame(self) -> np.ndarray:
7         if self._frame is None: self._frame = self.video.read()[1]
8         return self._frame
9     # ...
10    def show_frame(self) -> None:
11        cv2.imshow(str(self.id), self.frame())
12    # ...

```

Python

Plusieurs choses à observer dans ces échantillons:

- L'implémentation de la méthode `show_frame()` n'est pas la même dans les deux classes mais, malgré ça, un client qui utiliserait un objet ou l'autre aura un résultat cohérent. Sur une instance de `CameraPair`, l'appel de `show_frame()` la fait itérer sur son aggrégation de `Camera` et les fait appeler leur propre `show_frame()`. Indépendamment, les instances de `Camera` s'exécutent et produisent chacune une frame.
- Les implémentations des méthodes `frame()` utilisent le design pattern **Lazy loading**. Ce pattern permet de charger une ressource en mémoire une unique fois, seulement si elle est nécessaire et au moment où elle est nécessaire. Un peu à la façon d'un Singleton dans son implémentation, mais pour un attribut d'une classe, *Lazy loading* permet de ne pas refaire un calcul couteux, s'il a déjà été fait et que le résultat ne changera pas, soit car la ressource ne change pas dans le temps, soit car elle change dans le temps mais que plusieurs accès peuvent être nécessaires entre deux changements. Dans le cas de `Camera` et `CameraPair` il s'agit du second cas.

Sans l'utilisation du **Lazy loading**, ce package serait inutile car il détruirait les performances de l'application. En effet, sans, `Camera` ferait un appel à `self.video.read()` (coûteux en ressources) à chaque appel de méthode appliquant un traitement sur une frame. La **figure 5** présente ces méthodes (en faisant abstraction des méthodes spécifiques au pattern **composite**, pour plus de lisibilité).

<b>&lt;&lt; interface &gt;&gt;</b>
<b>CameraComponent</b>
+ clear_frames(): None
+ frame(): any
+ frame_lines(): any
+ frame_gray(): any
+ frame_corrected(): any
+ frame_corrected_lines(): any
+ show_frame(): None
+ show_lines(): None
+ show_gray(): None
+ show_corrected(): None
+ show_corrected_lines(): None
+ jpg_frame(): Tuple
+ jpg_lines(): Tuple
+ jpg_gray(): Tuple
+ jpg_corrected(): Tuple
+ jpg_corrected_lines(): Tuple

Figure 5 CameraComponent

La classe `CameraPair` implémente une série de méthodes et d'attributs qui lui sont propres. Il s'agit de méthodes spécifiques au calcul et à l'affichage d'images composites issues d'un traitement sur les deux flux ainsi que les attributs dédiés à la gestion du **lazy loading** de ces images. Ces traitements nécessitent l'utilisation de *blockmatchers*<sup>1</sup> représentés par des membres de type `cv2.StereoMatcher`.

La **figure 6** montre le diagramme complet de la hiérarchie, à l'exception des méthodes propres au pattern **composite**.

Sur la **figure 3** apparaît un dernier élément qui n'a pas encore été décrit. Il s'agit du module `img_utils.py`. Ce module est global au package `camera_system` et concentre toutes les méthodes qui effectuent des traitements sur les images. Autrement dit, tout le code métier lié à OpenCV. Cela veut dire que même la logique de traitement des images est découpée des classes qui l'utilisent. Les raisons de cette nouvelle segmentation sont les mêmes que les précédentes: maintenabilité et lisibilité du code.

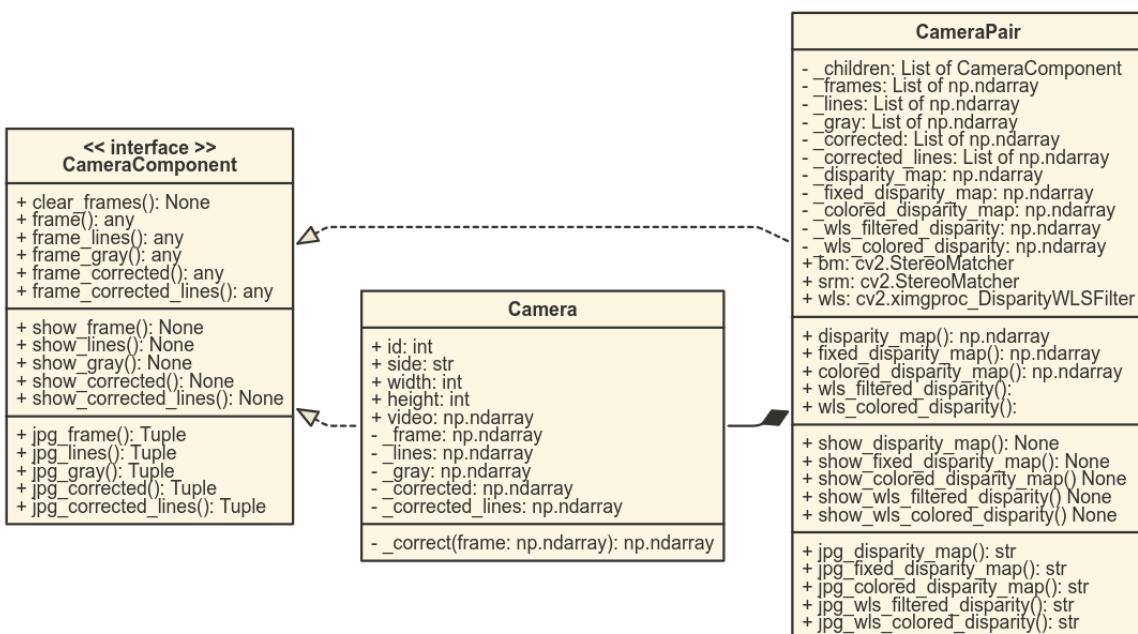


Figure 6 CameraComponent ainsi que ses implémentations Camera et CameraPair

1. Blockmatcher est l'outil d'OpenCV en charge de la recherche de correspondance d'un pixel de la première caméra sur la droite épipolaire correspondante de l'autre caméra.

### 3.3.2.2. strategies

Ce package fait partie de la librairie `sources/backend/` et c'est une implémentation du design pattern **strategy**. Ces stratégies représentent les modes de l'application tel que l'illustre la **figure 7**.

Ici, la classe `LoopStrategyManager` est le contexte d'exécution des stratégies. Il contient un membre de type `LoopStrategy` qui représente une stratégie assignée par le contrôleur (qui sera détaillé par la suite). C'est également le contrôleur qui, dans la boucle principale du programme, fera appel à la méthode `run_loop` du contexte, en lui passant les arguments nécessaires.

Chaque stratégie contient le code spécifique à un mode de l'application et, encore une fois, c'est le contrôleur qui est en charge du changement de stratégie et c'est par le biais du *setter*<sup>1</sup> `strategy` (`strategy: LoopStrategy`) qu'il le fait.

Cette façon de faire n'est autre qu'une adaptation du design pattern **State** dans lequel chaque *état*<sup>2</sup> est représenté par une stratégie.

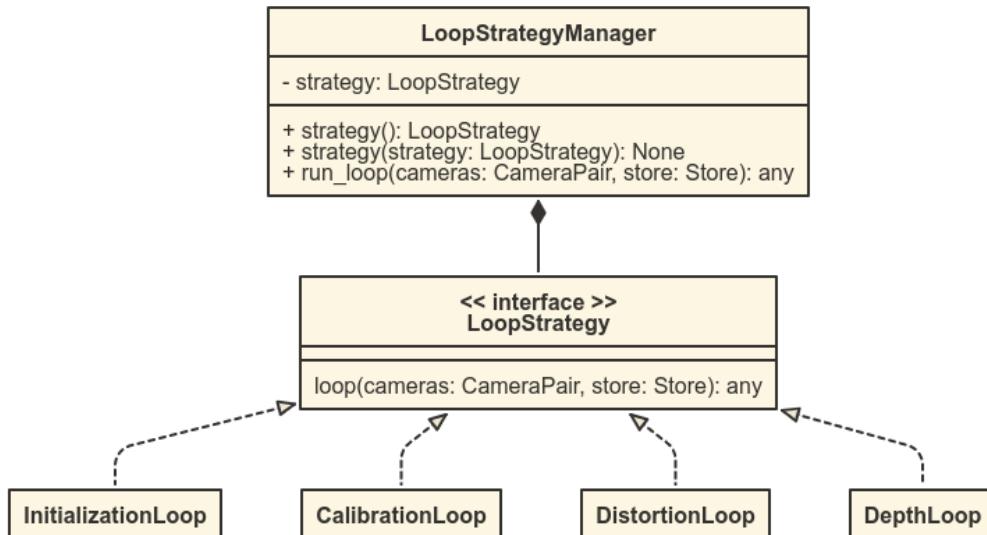


Figure 7 strategy package

1. via le décorateur `@method\_name.setter`

2. Un état représente un mode de l'application et un état est représenté par une stratégie.

Le premier paramètre de la méthode `run_loop` est de type `CameraPair`. C'est le choix des méthodes ainsi que l'ordre d'appel des méthodes de cet objet qui caractérisent une stratégie. Le second paramètre est de type `Store`, un conteneur qui contient l'état de l'application et qui sera détaillé dans la suite du rapport. Un élément qu'il est important de comprendre c'est que même si `LoopStrategyManager` n'est pas en charge de la logique de changement d'état, les stratégies, elles, sont en charge de la vérification de l'état **des options** qui concernent leur mode, en testant les attributs de l'objet de type `Store`, comme le montre l'échantillon suivant:

```

1 class InitializationLoopStrategy(LoopStrategy):
2     def loop(self, cameras: CameraPair, store: Store) -> any:
3
4         if store.state.lines:
5             return cameras.show_lines()
6             return cameras.show_frame()

```

Python

Cet échantillon est une version légèrement modifiée de la définition originale alégrée de la complexité liée à la gestion du type de contrôleur utilisant la stratégie.

### 3.3.2.3. controllers

Fait partie de la librairie

`sources/backend/`. Les classes de ce package sont en charge du changement d'*état*<sup>1</sup>, pendant l'exécution de l'application. Ces classes sont spécialisées en fonction du type d'interface utilisateur qu'elles représentent. Pour le moment il en existe deux: `GUIController` et `CLIController` et elles héritent, toutes les deux, de la classe abstraite `BaseController`. Cette classe abstraite offre d'une part une spécification à d'autres éventuels contrôleurs et d'autre part, une implémentation par défaut aux méthodes communes partagés par tous les contrôleurs. Ces méthodes communes forment l'API exposée au client utilisant le contrôleur.

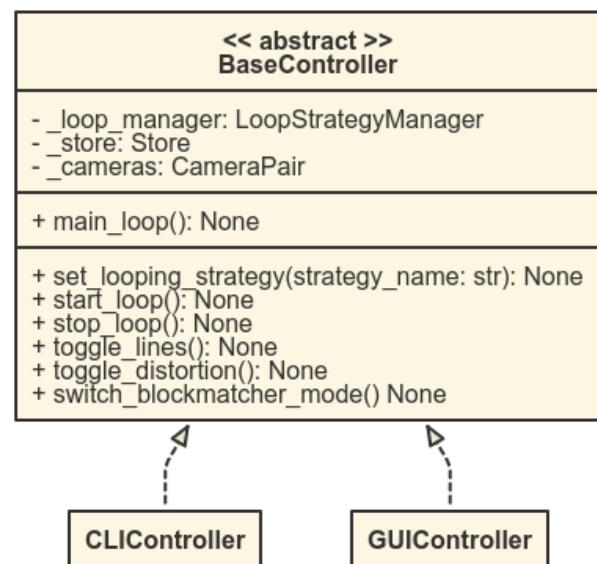


Figure 8 package: controller

1. Représenté par une stratégie

Les contrôleurs sont en charge de l'orchestration de l'application. C'est eux qui distribuent et altèrent les informations concernant l'état de l'application. Un coup d'oeil aux attributs d'un contrôleur donne une idée de ses responsabilités. L'attribut `_loop_manager` contient une instance de `LoopStrategyManager` et c'est lui qui permet au contrôleur d'itérer sur la stratégie courante mais aussi de changer de stratégie en fonction de l'état actuel définit dans un membre de l'instance de `Store` représentée par l'attribut `_store`. C'est également le contrôleur qui est en charge de la gestion des caméras, représentées par l'instance de `CameraPair` contenue dans l'attribut `_cameras`.

L'utilisation du design pattern **factory** pour l'instanciation de `CameraPair` vise encore une fois à réduire le couplage. Le package `sources/backend/models/` contient deux classes qui ne servent qu'à la construction d'objets de type `CameraComponent`. Cette façon de faire permet de ne pas utiliser directement les variables globales du module `public/settings.py`, dans les modules des packages `sources/camera_system` et `sources/backend/controller`. À la place, si des variables de `public/settings.py` doivent arriver dans les caméras ou dans un contrôleur, leur passage se fait lors de leurs construction dans la factory correspondante.

Le cœur de la logique fonctionnelle de l'application se trouve dans la méthode `main_loop()` et l'échantillon suivant en montre l'implémentation dans la classe `CLIController`:

```

1 def main_loop(self) -> None:
2     self.state.streaming = True
3     self.cameras = CameraPairFactory.create_camera_pair()
4
5     while True:
6         self.loop_manager.run_loop(self.cameras, self.store)
7         self._keyboard_handler()
8         self.cameras.clear_frames()

```

Python

La méthode `main_loop` est appellée, une première fois, au démarage de l'application et, ensuite, à chaque changement d'état déclenché par un input utilisateur. Il est intéressant de noter que tous les composants précédemment décrits sont orchestrés lors de l'exécution de cette méthode:

- **Ligne 2:** Set du flag d'état `state.streaming`. Sa remise à zéro se fera lors d'un appel du client à la méthode `stop_loop`.
- **Ligne 3:** Chaque appel de `main_loop` réinitialise le contenu de la variable `_cameras`, via appel à la méthode statique `create_camera_pair` de `CameraPairFactory`.

- **Ligne 6:** Exécution de la stratégie courante, via un appel à la méthode `run_loop` en lui passant en argument les caméras ainsi que l'état courant.
- **Ligne 7:** `_keyboard_handler` est une méthode spécifique à ce contrôleur qui est en charge de la gestion des inputs clavier de l'utilisateur.
- **Ligne 8:** finalement, à la fin de chaque itération, après que tous les traitements nécessaires soient faits sur une paire de frames, l'appel à `clear_frames` de la classe `CameraPair` permet de vider les variables dédiées à la gestion du `Lazy loading`.

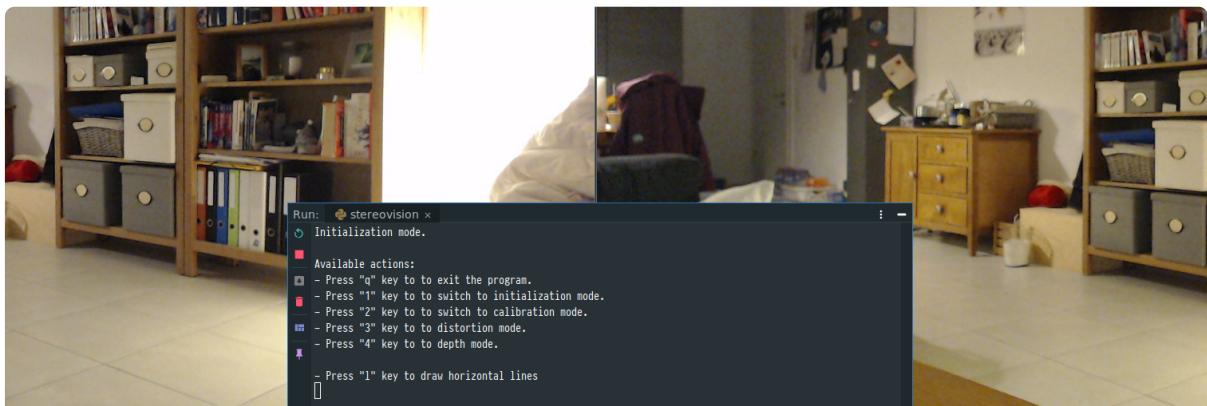


Figure 9 Lancement de l'application avec utilisation du CLICController

La **figure 9** montre ce qu'affiche le terminal suite à l'exécution de l'application en utilisant le `CLICController`. Le contenu des textes ainsi que la logique qui gère leur affichage se trouve dans les méthodes statiques de la classe `CLIMessages`, contenue dans le module `sources/backend/controllers/messages.py`.

L'implémentation de `main_loop` dans `GUIController` est légèrement différente. La logique de la GUI étant gérée par du javascript, il est avant tout nécessaire de mettre en place un pont entre les deux langages. Pour ce faire, la librairie `Eel`<sup>1</sup> est utilisée. Dans le module `sources/backend/api.py` une API est déclarée sous formes de fonctions décorées par des décorateurs mis à disposition par `Eel`. Cette API n'est qu'un simple wrapper autour des méthodes de l'API de `GUIController`. Encore une fois, cette ségmentation du code permet de gagner en lisibilité et réduit le couplage interne, en ségrégant les composants qui font appel à `Eel` dans un module spécifique. Malheureusement, cette ségrégation n'est pas complète, par manque de temps pour trouver une solution élégante à un problème de dépendance circulaire, ce qui a pour conséquence la nécessité d'utiliser des méthodes de `Eel` directement dans `GUIController`.

---

1. dépendance externe du projet

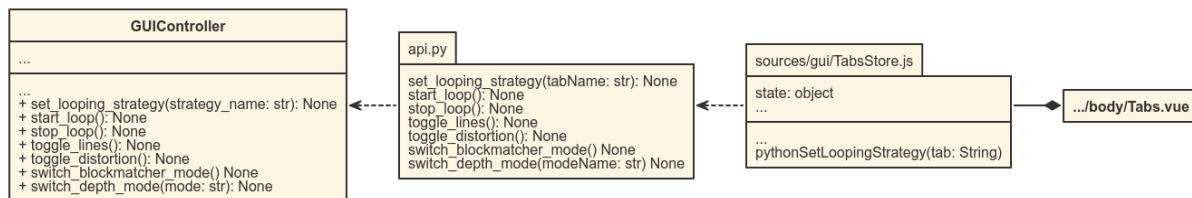


Figure 10 Communication avec la GUI javascript

La figure 9 illustre comment le code javascript peut appeler des méthodes du code python. Les méthodes accessibles par le code javascript altèrent le contenu du `Store` contenu dans `GUIController`, ce qui a des répercussions sur le comportement des méthodes des classes du package `strategy`.

L'échantillon qui suit décortique l'implémentation de `main_loop` dans `GUIController`:

```

1 def main_loop(self) -> None:
2     self.state.streaming = True
3     self.cameras = CameraPairFactory.create_camera_pair()
4
5     while self.state.streaming:
6         jpgs = self.loop_manager.run_loop(self.cameras, self.store)
7         self._update_frontend_images(jpgs)
8         self.cameras.clear_frames()
9
10    self.cameras.__del__()
  
```

Python

Les lignes 2 et 3 sont identiques à celles de la première implémentation. La **ligne 6** est beaucoup plus intéressante car cette fois, `run_loop` retourne une valeur. En effet, si utilisé par `GUIController`, les stratégies du packages

`sources/backend:strategies` appellent les méthodes `jpg*` de la classe `CamerasPair` et non pas les méthodes `show*`. Ces méthodes convertissent la paire de frames courantes en jpg, puis les encodent au format base64, avant de les retournent dans un `tuple`. En **ligne 7** un appel à une méthode privée de `GUIController` dont le job est d'injecter chaque jpg contenu dans `jpgs`, dans le code exotique de la GUI. La **ligne 8** est pareille que celle de la première implémentation. Finalement, la **ligne 10**, bien qu'intriguante, n'est qu'une façon de contourner un problème de `del` qui empêche la destruction des cameras à la sortie de la boucle.

Échantillon contenant le code responsable de l'encodage de la frame:

```

1 # sources/camera_system/img_utils.py
2 def frame_to_jpg(frame: np.ndarray) -> str:
3     jpg = cv2.imencode('.jpg', frame)[1]
4     return base64.b64encode(jpg).decode('utf-8')
  
```

Python

### 3.3.3. Modules

Ici sont décrits les différents modules ne faisant pas partie d'un package et n'ayant pas été expliqués dans le sous-chapitre précédent.

#### 3.3.3.1. `store.py`

Fait partie de la librairie `sources/backend/`. Ce module contient deux *dataclass*<sup>1</sup> `State` et `Store` qui sont décrits par la **figure 11**.

Son utilisation est inspiré de patterns architecturaux spécifiques au frontends web et plus spécifiquement à **Flux**. Il n'est pas forcément utile, de par les spécifications de python et du fait qu'il n'y ait pas d'actions asynchrones explicites pendant l'exécution. L'utilisation d'un store est, malgré ça, une façon commode de gérer l'état général d'une application *event driven*<sup>2</sup>. À la place d'avoir de multiples variables contenant des portions de l'état, parsemées dans tous les composants de l'application, l'état est centralisé dans un objet de type `Store` qui (dans cette implémentation) n'est qu'un conteneur pour un objet de type `State` qui contient tout l'état de l'application.

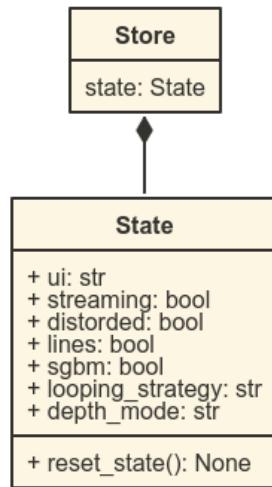


Figure 11 module `store.py`

Dans cette application, l'objet contenant l'état (donc une instance de `Store`) n'est pas global, mais instancié par un contrôleur et passé, en tant qu'argument, lors de certains appels de méthodes.

#### 3.3.3.2. `settings.py`

Se trouve dans le dossier `public/`. Simple fichier de configuration utilisé à des endroits clés de l'application. Il contient également quelques vérifications concernant les chemins spécifiés. Des valeurs par défaut sont pré-assignées mais ne garantissent pas les mêmes résultats que sur les captures d'écran du chapitre suivant. Pour avoir un résultat similaire il sera probablement nécessaire d'adapter les valeurs contenues dans le dictionnaire `DEPTH_MAP_DEFAULTS` au système stéréoscopique employé.

1. Voir explication dans le chapitre 3.2.3.3

2. L'application attend une action de l'utilisateur pour continuer son exécution.

### 3.3.4. Clients

- public/

Cette partie décrit les clients qui utilisent l'application. Ces clients sont les points d'entrée de l'utilisateur et sont au nombre de deux:

- public/stereovision.py
- public/stereovision\_gui.py

Ils sont exécutables à la ligne de commande à travers l'interpréteur Python3:

```
1 | $ python3 public/stereovision.py
2 | $ python3 public/stereovision_gui.py
```

Bash

Mais peuvent être rendu exécutables si un `chmod` est effectué au préalable:

```
1 | $ sudo chmod a+x public/stereovision.py
2 | $ public/stereovision
```

Bash

### 3.3.5. GUI

Le dossier `/sources/gui/` contient le code de l'interface graphique. Initialement son but était de permettre une personnalisation des paramètres de disparité plus accessible que par des modifications manuelles dans le fichier de configuration. Comme expliqué dans le point 3.3.2.3 qui décrit les contrôleur, cette interface est écrite avec des *technologies du web*<sup>1</sup> et le pont vers Python est fait avec la librairie python `Eel`. Par manque de temps, il a été décidé de ne pas décrire les spécifications de cette partie qui est aussi vaste que le reste de l'application et qui s'éloigne du thème principal du rapport.

Cette partie du projet, bien que tout à fait fonctionnelle et agréable à utiliser, n'est pas finie et n'atteint pas son objectif principal, c'est à dire proposer une personnalisation des paramètres de disparité sur le flux d'images.

---

1. Html, css, js et VueJS

### 3.4. Matériel

Ce chapitre décrit le matériel utilisé. Il n'est pas nécessaire d'avoir le même, pour avoir un système de vision stéréoscopique fonctionnel ou pour utiliser le logiciel.



Figure 12 Matériel utilisé

- 2 x Logitech c920 HD Pro
- Dell XPS 15 7590 (i7-9750H @ max 4.5ghz)
- Coffee @100% arabica

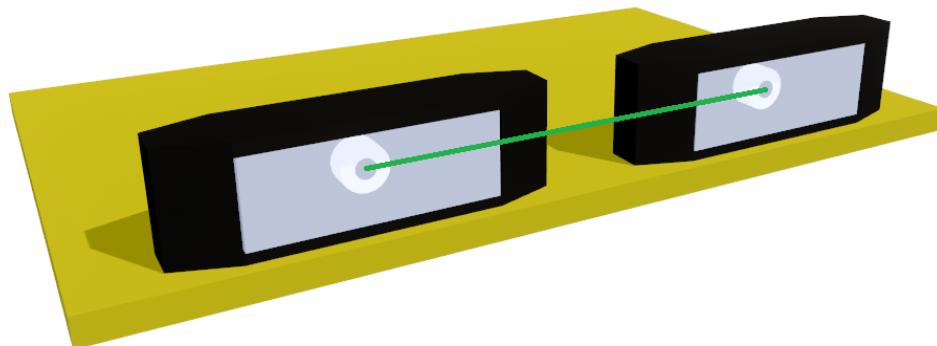
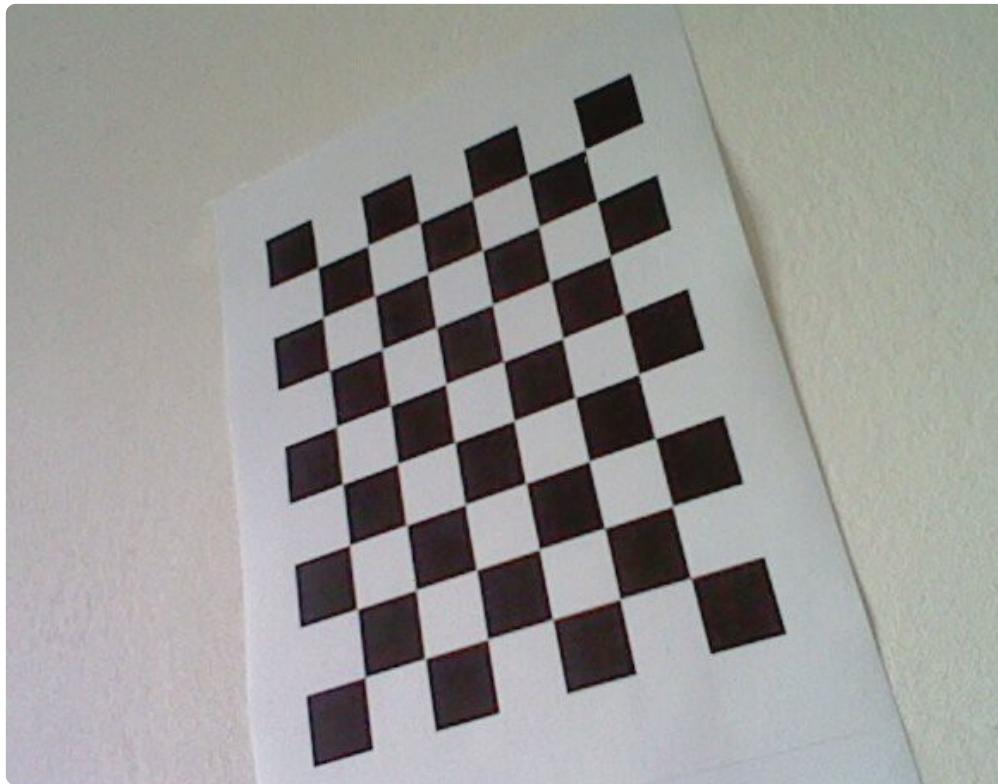


Figure 13 Écart entre les deux capteurs

L'écart de capteur à capteur est de 110 mm (trait vert sur la figure précédente). Cette distance est arbitraire mais les valeurs par défaut de certains paramètres accessibles dans le fichier `public/settings.py` sont calibrées sur cette base.

Une mire de calibration est nécessaire pour calibrer le système:



**Figure 14** Mire de calibration

Pour s'en procurer une, en l'absence d'un jeu d'échec, il est possible de simplement imprimer ce [pdf](#). Il est également conseillé de rigidifier cette mire en la collant sur une surface rigide.

## 3.5. Utilisation

Il est possible d'utiliser le logiciel avec deux interfaces différentes:

- **CLI** qui contient toutes les fonctionnalités du logiciel.
- **GUI** plus facile d'utilisation mais ne permet pas de calibrer les caméras. Elle nécessite une calibration préliminaire à l'aide du CLI.

### 3.5.1. Dépendances

- Python3.7

### 3.5.2. Installation

**!** Le logiciel n'a pas été testé sous Windows et MacOS, aucun système de ce type n'étant disponible sur la machine de l'étudiant et le temps ne permettant pas de faire des tests extensifs sur différents systèmes. Néanmoins, il n'y a pas de raison particulière que ça ne fonctionne pas sur ces OS. Les précautions de base de portabilité ont été prises et généralement elles suffisent à assurer la portabilité du code Python.

Il est vivement conseillé de faire usage d'un environnement virtuel de type [virtualenvwrapper](#) ou [pyenv](#) pour ségréguer les dépendances du projet des dépendances de l'installation système de Python.

1. Téléchargez ou clonez les sources du projet sur [github](#).
2. Après décompression de l'archive, ouvrez un terminal à la racine du projet.
3. (optionnel) Créez un nouvel environnement virtuel et activez le.
4. Installation des dépendances avec `$ pip install -e .`

### 3.5.3. Configuration

Le fichier `public/settings.py` contient les configurations du logiciel. La majorité des valeurs par défaut sont valables mais il est nécessaire d'en adapter certaines:

- **DEVICE** : Les valeurs des champs `left` et `right` correspondent à l'id usb qu'occupent vos caméras.
- **CHESSBOARD** :
  - `rows` correspond au nombre de lignes -1 *nombre de lignes -1*<sup>1</sup>
  - `columns` correspond au nombre de colonnes - 1.
  - `square_size` représente la taille d'un carré en *cm*.

1. en réalité OpenCV compte les intersections et non pas les lignes ou colonnes

### 3.5.4. CLI

Pour lancer le CLI, À la racine du projet:

```
1 | $ python public/stereovision.py
```

Bash

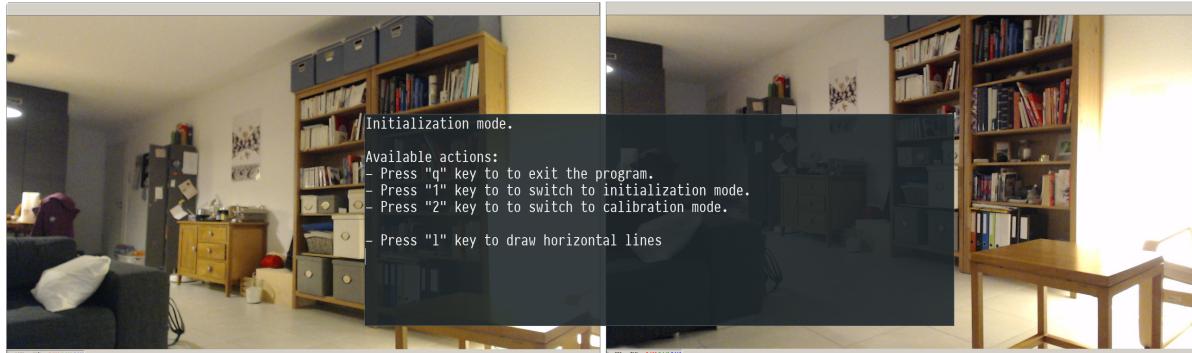


Figure 15 mode: initialization

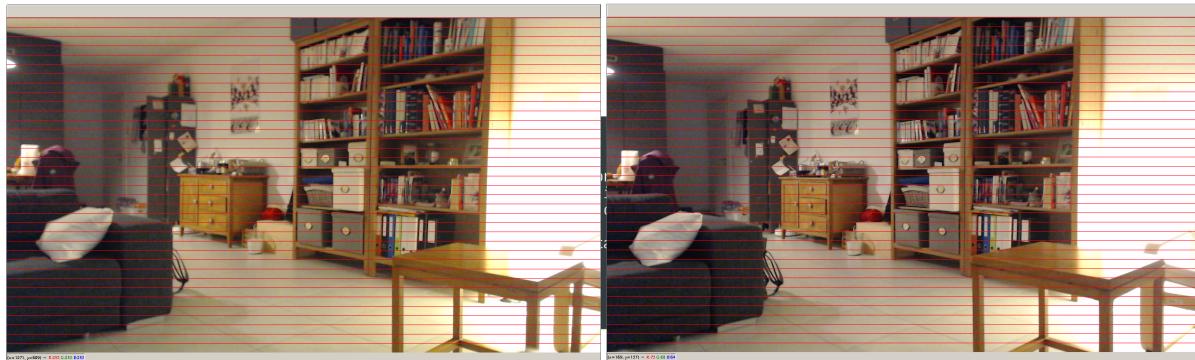
Les 4 actions possibles sont affichés. Les autres modes ne sont pas visibles car l'application ne détecte pas les fichiers contenant les données de calibration. Avant de lancer la procédure de calibration, il est utile de faire un pré-réglage manuellement des caméras. En pressant la touche "l" (L minuscule) du clavier, des lignes apparaissent:



Figure 16 Pré-réglage

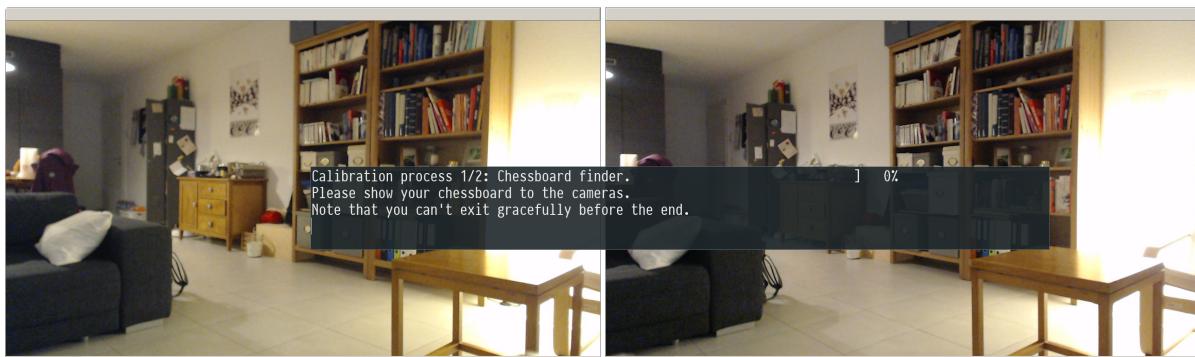
Ces lignes servent à régler grossièrement la position verticale des caméras. Sur la figure précédente, la caméra de gauche devrait être pivotée vers le bas, manuellement.

La figure suivante montre le résultat, après une correction grossière:



**Figure 17** Correction manuelle

Le système est maintenant prêt à être calibré. Pour ce faire, il est nécessaire de changer de mode pour le mode 2, qui est dédié à la calibration, en pressant sur la touche "2" du clavier:



**Figure 18** mode: calibration

Le mode calibration est un mode en deux procédures:

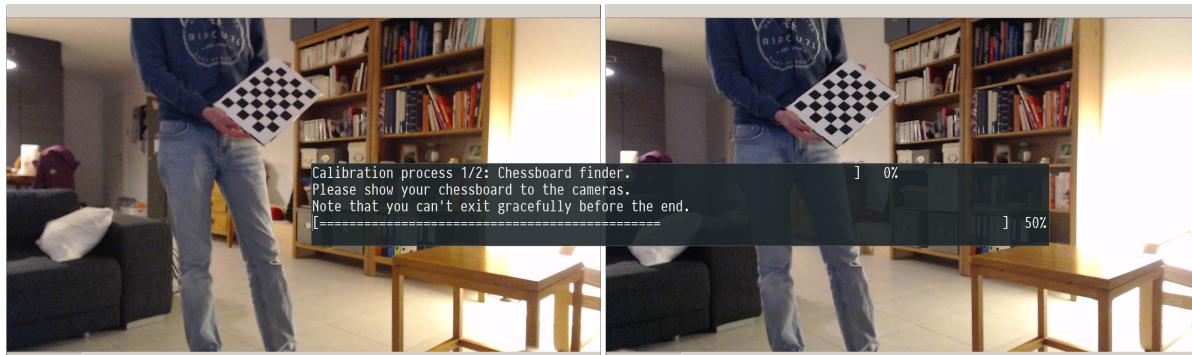
1. Le logiciel va capturer un *certain nombre*<sup>1</sup> de paires d'images.
2. Le logiciel utilise les images pour déterminer un certain nombre de paramètres, notamment les paramètres intrinsèques, les données de distorsion, les matrices fondamentale et essentielle.

Pour effectuer les captures, il est nécessaire de se munir de la mire de calibration et de se balader avec, devant les caméras. Les captures se font automatiquement lors de la détection de la mire dans les deux images. Après une capture, le logiciel laisse 2 secondes à l'utilisateur pour changer de position, avant de recommencer à chercher la mire.

---

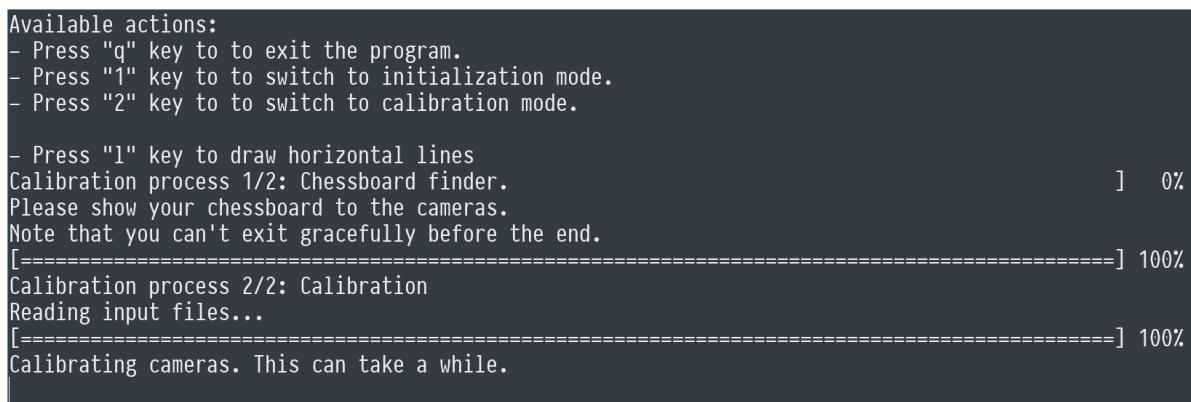
1. définit dans le module public/settings.py

La barre de progression permet de connaître l'état d'avancement de la procédure:



**Figure 19** Capture de la mire

La transition vers la seconde procédure se fait automatiquement lorsque la barre de progression est complète. La seconde procédure ne demande *aucune action de l'utilisateur*<sup>1</sup> et prend un *certain temps*<sup>2</sup> pour faire ses calculs:



**Figure 20** Calcul des données de calibration

Si la première procédure échoue pour une raison ou pour une autre ou si l'utilisateur souhaite faire une nouvelle calibration, il lui suffit de relancer ce mode.

- 
1. Sauf si l'option CALIBRATION.show\_chessboard est à True dans le module public/settings.py
  2. Dépend de la résolution et de la puissance de la machine

Une fois les calculs effectués, l'application change automatiquement de mode pour aller dans le numéro 3, c'est à dire le mode distorsion:



Figure 21 Mode: distorsion

Dans ce mode il est possible d'afficher les lignes d'aide (touche "l") qui permettent de visualiser la qualité de la rectification ainsi que de la suppression de la distorsion. Pour alterner entre images normales et images réctifiées, il est possible de presser la touche "d".

La touche "4" active le mode depth:

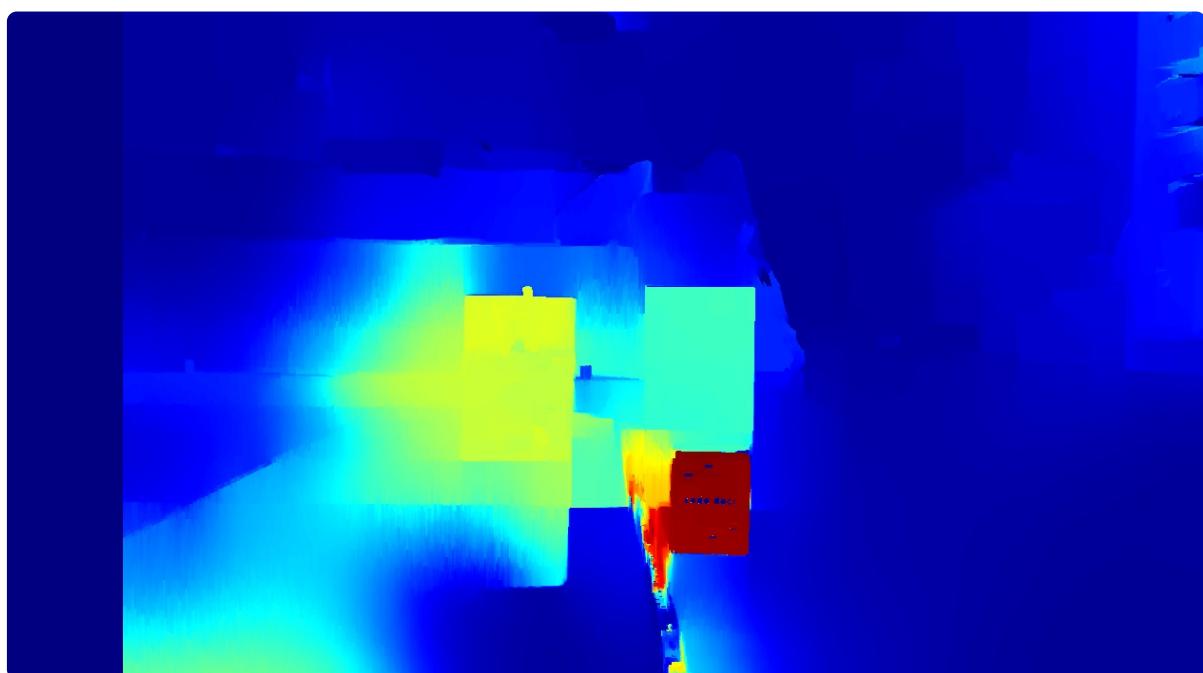


Figure 22 Mode: depth

Dans ce mode, plus la couleur tend vers le rouge, plus l'objet est proche. Et dans la précédente capture (qui correspond à la scène de la **figure 21**), il est facile de voir que l'objet au premier plan est rouge vif, celui au second jaune et celui dans le 3e bleu clair.

Ce mode offre de nombreuses options qui seront détaillées dans le prochain sous-chapitre:

```
Depth mode.

Available actions:
- Press "q" key to exit the program.
- Press "1" key to switch to initialization mode.
- Press "2" key to switch to calibration mode.
- Press "3" key to distortion mode.
- Press "4" key to depth mode

- Press "b" key to change between blockmatcher mode
- Press "d" key to show raw disparity map
- Press "c" key to show fixed and colored disparity map
- Press "w" key to show a WLS filtered map
In this mode you can double click to get a conversion
from disparity to distance. (Needs to be setup)
|
```

Figure 23 Depth: options

Les dernières lignes de la figure précédente laissent sous entendre qu'il est possible d'obtenir la distance en double cliquant sur un point de l'image. Cette fonctionnalité est expérimentale et est décrite dans le chapitre suivant.

### 3.5.5. GUI

Lancer le programme en mode GUI donne accès à une interface graphique. Pour lancer ce mode, à la racine du projet:

```
1 | $ python public/stereovision_gui.py
```

Bash

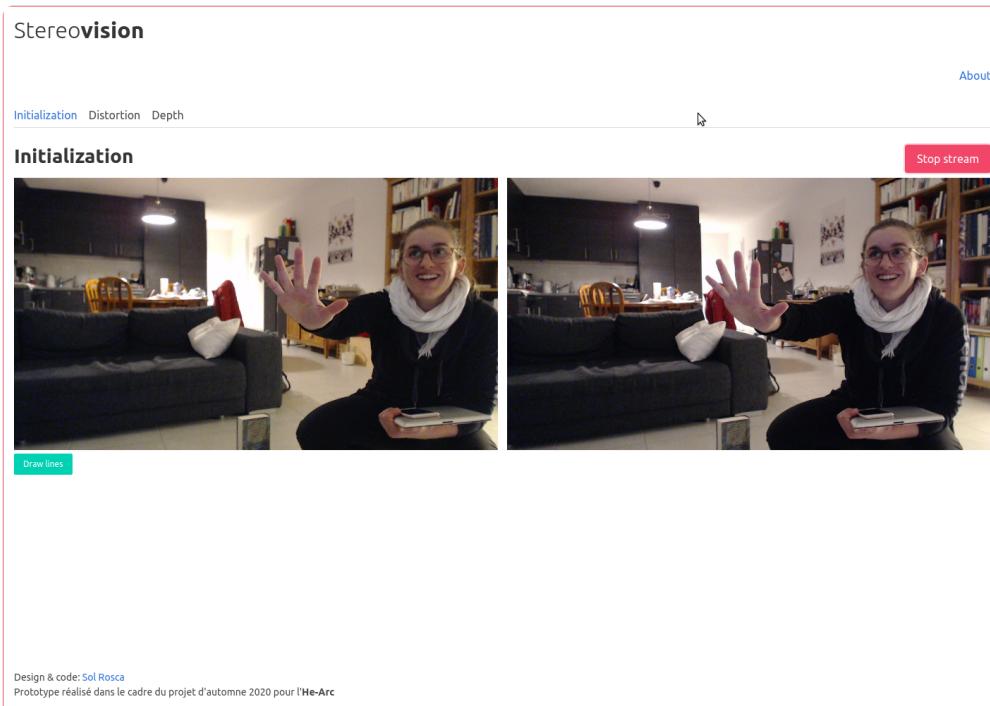


Figure 24 GUI

Dans la GUI, on retrouve les modes de l'application sous forme d'onglets, en haut à gauche. Il y en a un de moins qu'en mode CLI car il n'est pas possible pour le moment de calibrer dans ce mode. Dans chaque onglet, en haut à droite se trouve un bouton pour lancer / stopper la capture. Si une capture est en cours et que l'onglet est changé, la capture cesse automatiquement et doit être relancée dans le nouveau mode. Et finalement, en bas à gauche des images, il y a les différentes options du mode courant, qui sont les mêmes que celles du CLI. Sur la précédente figure, on peut voir un bouton pour activer l'affichage des lignes horizontales d'aide.

Dans l'onglet "Depth", on retrouve l'image résultante du traitement sur les deux caméras qui affiche une carte de profondeur. Par défaut c'est le mode WLS qui est activé et qui propose un filtre particulier qui permet de repliquer le résultat de la carte de profondeur sur une image filtrée:



Figure 25 Carte de profondeur avec filtre WLS

Si on le retire en cliquant sur la radio "Colored", l'image résultante n'est plus traitée avec le filtre WLS et affiche juste une carte de profondeur colorée:

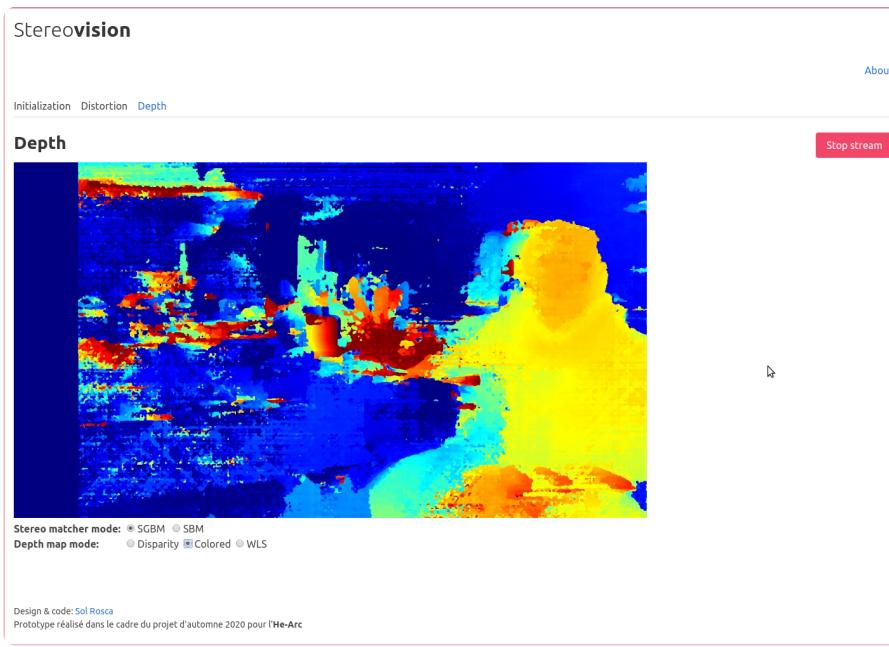
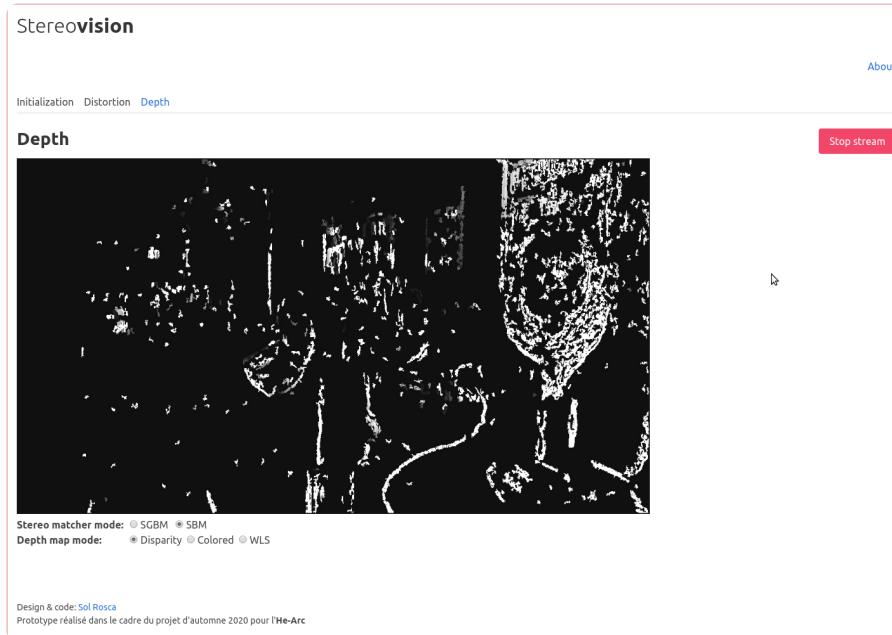


Figure 26 Carte de profondeur colorée

Et si on retire la couleur, en cliquant sur la radio "Disparity", il ne reste que la carte de disparité sans aucun traitement autre qu'une normalisation ramenant la disparité maximum à 0.



**Figure 27** Carte de disparité avec blockmatcher simple

Sur la figure précédente, le mode du *blockmatcher*<sup>1</sup> a également été changé. Il existe deux modes:

- **SBM** (StereoBlockMatcher): Un mode qui crée des cartes moins riches mais permet d'avoir une image fluide.
- **SGBM**: Un mode qui crée des cartes disparité riches mais très coûteux en puissance de calcul. Entraine une réduction conséquente du nombre d'images par secondes.

Il est possible de changer les paramètres du blockmatcher dans le module `public/settings.py` dans la partie `DEPTH_MAP_DEFAULTS`.

---

1. Algorithme de recherche de correspondance entre les deux images



Figure 28 SBM + color map

La capture précédente montre une carte de profondeur colorée (plus c'est rouge, plus c'est proche) mais avec le blockmatcher **SBM**. Dans ce mode, pour une résolution de capture HD (1280 x 920), le nombre d'images par seconde est proche de 60 alors qu'avec le mode **SGBM** et le filtre **WLS** il oscille entre 2 et 3 images par secondes.

## 3.6. Expérimentation

Comment transformer les données de la carte de disparité en distances ?



**Figure 29** Banc d'expérimentation

En utilisant la fonction `average_disparity_at_position` définie dans le module `sources/camera_system/img_utils.py`:

```

1 def average_disparity_at_position(x, y, disparity_map) -> float:
2     average = 0
3     for j in range(-1, 2):
4         for i in range(-1, 2):
5             average += disparity_map[y + j, x + i]
6
7     return average

```

Python

Couplée au système de capture des coordonnées du click de OpenCV, il est possible d'afficher la valeur moyenne de la disparité à l'endroit d'un double clique sur l'image. La moyenne permet de légèrement lisser cette valeur et de la rendre moins sensible à un changement brusque de la disparité induit par une erreur de calcul sur une frame.

Comme expliqué dans le cours annexe de ce projet, nous savons que la distance est inversément proportionnelle à la disparité. Si on accumule suffisamment de paires de données (distance, disparité) il est envisageable de trouver une approximation de la fonction de la distance, en appliquant une régression polynomiale aux données relevées.

La démarche est la suivante:

Déplacer l'objet visible sur la **figure 29**, petit à petit, et reporter dans un tableau, à chaque étape, la distance en cm ainsi que la valeur de disparité obtenue grâce à un double clique sur l'objet.



**Les données relevées dépendent de la distance entre les capteurs des caméras du système qui pour rappel vaut 11 cm.**

En utilisant un tableur, il est possible de ploter ces points et de demander au tableur d'appliquer une régression, sur base d'une fonction hypothèse  $h$  (dont la courbe ressemble à la répartition de nos données) pour qu'il en retourne la valeur des coefficients. Après quelques tentatives, les meilleurs résultats sont obtenus avec un polynôme de degré 3:  $h(x) = ax^3 + bx^2 + cx + d$  dont voici la forme pour les données relevées:

Données relevées

Approximation par régression polynomiale

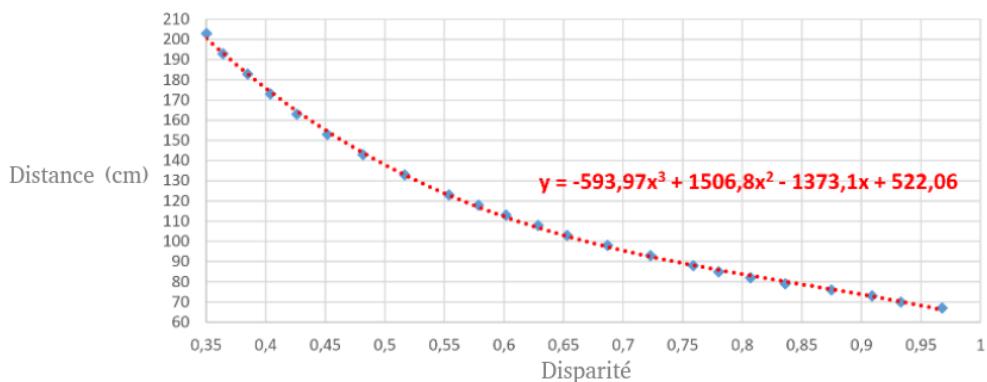


Figure 30 Régression polynomiale des données relevées

En substituant les coefficients dans  $h$ , on se retrouve avec une fonction qui prend une valeur de disparité et retourne une approximation de la distance correspondante:  
 $f(x) = -593.97x^3 + 1506.8x^2 - 1373.1x + 522.06$

En encodant cette fonction dans le fichier

`sources/camera_system/img_utils.py`, et en y injectant la valeur de la disparité lors d'un clique sur l'image, nous avons maintenant une fonction qui permet de calculer la distance d'un point de la scène. Après de nombreux tests, les distances retournées ont un taux d'erreur +- 3 cm entre 61 cm et 477 cm. En dehors de cet intervalle, l'erreur croît dramatiquement, bien avant que la disparité soit nulle. Ce qui correspond aux conclusions tirées dans le cours annexe au projet: "Les systèmes d'imagerie stéréo ont une haute précision de la profondeur, uniquement pour les objets relativement proches des caméras".

## 4. Conclusion

---

Le projet est arrivé à son terme et les objectifs du cahier des charges pour la partie "cours théorique" sont partiellement atteints, tandis que les objectifs du cahier des charges pour la partie "logiciel" sont intégralement atteints.

L'écriture d'un cours théorique fut une bonne façon de faire pour comprendre le problème dans sa globalité, mais la durée de travail nécessaire pour comprendre tous ses aspects de façon à pouvoir les expliquer lors d'une leçon a posé de nombreux problèmes de gestion du temps. Des questions comme "À quel point est-il nécessaire de granulariser un sujet ?" sont restées sans réponse et même si le résultat actuel est détaillé, relativement clair et agréable à lire pour les aspects qu'il couvre, il ne les couvre malheureusement pas tous et en l'état, il ne permet pas de dispenser un cours complet sur le sujet. Par manque de temps, il a été décidé de le laisser dans son état actuel. Si une couverture plus superficielle de certains aspects avait été faite, il aurait très probablement été possible d'offrir quelque chose de mieux construit et qui couvre tous les aspects essentiels.

La partie logiciel du projet, quant à elle, respecte tous les objectifs fixés. L'étude de nombreux détails de par l'écriture du cours a permis d'avoir des objectifs clairs pour chaque étape de son développement. L'analyse a su mettre le doigt sur les points importants à garder en tête lors de la conception, ce qui a mené à des spécifications suffisamment détaillées pour permettre la mise en place d'une architecture robuste et maintenable, avec le potentiel de respecter les objectifs fixés. Grâce à cette base solide, la réalisation a été l'occasion de se concentrer sur la qualité du code, sans faire des allés-retours entre l'écriture et l'architecture. Le produit final livre un logiciel simple à utiliser dont les multiples interfaces témoignent de sa maintenabilité et de sa flexibilité. Ces deux qualités font de lui un outil pédagogique qui remplit les objectifs, en offrant un code où la partie métier est suffisamment ségrégée pour permettre une explication simple des concepts liés à la vision stéréoscopique binoculaire, sans s'encombrer des détails d'implémentation.

Le chapitre "Expérimentation" de ce rapport a été l'occasion de croiser les deux parties de ce projet. Ainsi, une affirmation de la partie théorique du cours a pu être démontrée de façon expérimentale en créant et ajoutant les outils nécessaires dans le module adéquat, sans batailler avec la structure ou le langage.

## 5. Références

---

### 5.1. Outils

- [VueWrite](#): (Pas encore publié. Outils maison de rédaction de documents)
- [Nomnoml](#): Crédit de diagrammes uml de façon programatique.
- [Geogebra](#): Géométrie interactive en 2D et 3D.
- [unity 3D](#): Moteur 3D.

### 5.2. Librairies python

- [eel](#): Clone d'Electron pour Python.
- [opencv-contrib-python](#): OpenCV + extras.
- [numpy](#): Calculs scientifiques.

## 5.3. Papier

- G. Bradski et A. Kaehler, Learning OpenCV, O'Reilly, 2008

## 5.4. Articles, Cours & Tutos

- Coordonnées Homogènes
- Vision par ordinateur (inria.fr)
- Modélisation à partir d'images (inria.fr)
- Vision stéréoscopique (Marie-Odile Berger)
- Stereo vision using one camera
- Tutorial made by the dev of Py stereovision module

## 5.5. Cours vidéo

- Disparité binoculaire
- Triangulation
- Tutoriel Vision stéréoscopique (matlab)
- William Hoff Lecture 21-1 Stereo Vision 1
- William Hoff Lecture 21-2 Stereo Vision 2
- William Hoff Lecture 21-3 Stereo Vision 3
- William Hoff Lecture 23-1 Epipolar and Essential
- William Hoff Lecture 23-2 Epipolar and Essential 2

## 5.6. Interactif

- Pinhole (stenopé) (Geogebra)
- Disparité (Geogebra)
- Disparité II (Geogebra)
- Géometrie épipolaire (Geogebra)

## 6. Annexes

---

- cf.cours\_Stéréovision.pdf
- cf.pv\_reunions.md