# Énoncé de la compétition – Stéréovision

Nous y voilà ! Le robot Pioneer p3dx est à votre disposition. Il est équipé de deux caméras qui constituent son système de stéréovision. Chargez la scène qui se trouve dans ***(Tools/ Parcours_robot_avec_simulateur / cuisine_Explore.ttt )*** afin de découvrir votre robot ainsi que la maison de Romain.

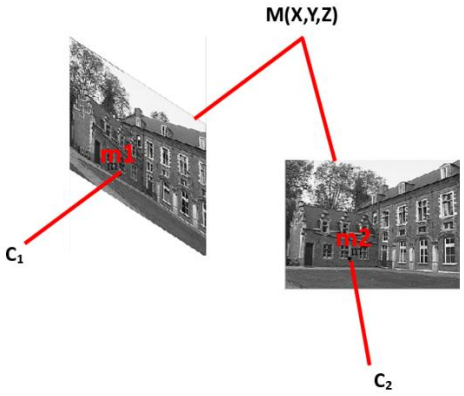


**Figure 01:** Principe de stéréovision. Le point 3D M(X, Y, Z) de la scène a deux projections 2D : $m_1$, $m_2$ sur les images gauche et droite respectivement.

**ETAPE 1 – CALIBRATION (1 point)**

Pour permettre la reconstruction 3D, il nous faut calibrer notre système de stéréovision. La calibration consistera à calculer les paramètres intrinsèques de chaque camera, ainsi que la position relative de la caméra droite par rapport à la camera gauche (paramètres extrinsèques). Nous allons mettre ça en équations !

$M_G$ = matrice de calibration de la caméra gauche.

$M_D$ = matrice de calibration de la caméra droite.

$$M_G = K_G\,[R_G, t_G]$$

$$M_G = \begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Intrinsèques Extrinsèques

$$M_D = K_D\ [R_D, t_D]$$

$$M_D = \underbrace{\begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Intrinsèques}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Extrinsèques}}$$

**Note :** Vous remarquerez que les paramètres extrinsèques de la camera gauche sont : R = Matrice identité, et t = $[0, 0, 0]^t$. Ceci est expliqué par le fait que la caméra gauche est prise comme centre du repère du monde pour la reconstruction 3D. Les coordonnées encadrées par le rectangle rouge sont les coordonnées homogènes, utilisées pour les calculs matricielles. Vous n'avez pas besoin de les créer, les fonctions d'opencv les incluront automatiquement par la suite.

**Votre objectif :** créer les matrices de calibration dans votre code afin de pouvoir les utiliser par la suite. Les matrices à former sont : **$K_D$, $K_G$, $[R_D, t_D]$.** Des indices sont cachés dans la maison de Romain pour récupérer les valeurs des matrices. Vous pouvez soit manipuler la scène avec votre souris, et chercher leur emplacement. Sinon, ce que nous vous conseillons, c'est d'utiliser votre robot qui sait où les retrouver.

Vous avez deux simulations : cuisine_Explore.ttt et salon_Explore.ttt. Lancez chaque simulation, puis surveillez ce qui s'affiche sur les deux fenêtres qui représentent les yeux du robot (ses deux caméras). N'hésitez pas à faire pause dans la simulation, puis de zoomer sur ce qui vous semble un indice.

**ETAPE 2 – Reconstruction 3D**

Maintenant, nous passons à l'étape la plus intéressante. Nous allons implémenter une méthode de reconstruction dense qui permet de retrouver le modèle 3D d'une scène à partir de seulement deux images de cette scène. Voici les étapes de reconstruction 3D :

- Rectification des deux images stéréo afin de rendre les droites épipolaires parallèles.
- Mise en correspondance des points, qui consiste à retrouver dans les images gauche et droite les points homologues, c'est-à-dire, les points qui sont les projections de la même entité de la scène.
- Triangulation des points, qui est le calcul des coordonnées 3D de chaque point à partir des coordonnées 2D de ses deux projections sur l'image gauche et droite.

## 1. Rectification des images (1 point)

La rectification d'une paire d'images stéréoscopiques permet de se ramener à une géométrie épipolaire simple où les droites épipolaires sont parallèles aux lignes des images. L'objectif derrière cette modification est de faciliter la mise en correspondance. En effet, chaque point appartenant à l'image de droite trouvera son homologue dans l'image de gauche sur la même ligne de pixels. Nous utiliserons cette contrainte afin de réduire drastiquement le temps de calcul de la mise en correspondance.

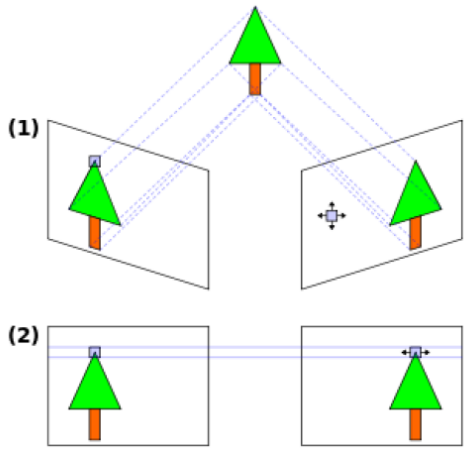


**Figure 02:** Illustration d'images stéréoscopiques : **(1)** avant rectification **(2)** après rectification.

Vous avez à votre disposition un code qui permet de lire deux images stéréo, et de les afficher. **Votre objectif est** de rectifier ces images à l'aide des fonctions fournit par opencv : stereoRectify(), initUndistortRectifyMap(), remap(). Puis d'afficher les images rectifiées.

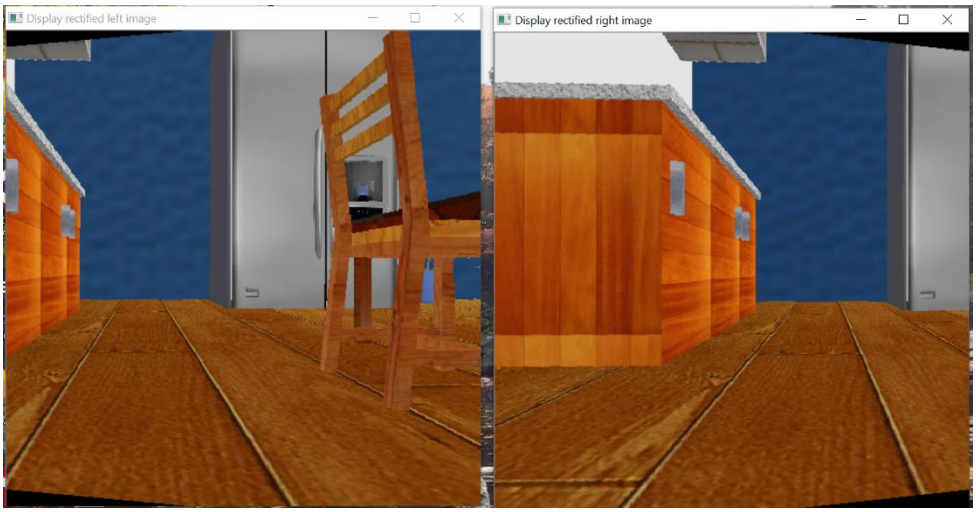


**Figure 03:** Résultat attendu après rectification.

## 2. Mise en correspondance (2 points)

La mise en correspondance est bien l'étape la plus critique de notre approche. En effet, nous devons trouver pour chaque pixel de l'image gauche son homologue dans l'image de droite. Sachant que certains pixels ont des intensités très similaires, sans parler du nombre de pixel que peut avoir une image (exemple 1024x1024). Il est certain que des contraintes doivent être utilisées afin de réduire le champ de recherche. La méthode basique que nous proposons et la simple corrélation.

- ***Fonctionnement de la corrélation***

On utilise une fenêtre fixe de corrélation F1 dans l'image gauche centrée sur le point à apparier, ainsi qu'une fenêtre glissante F2 dans l'image droite qui va parcourir la zone de recherche. Pour chaque position de F2 un score de corrélation $C(x, y)$ est calculé entre F1 et F2. Exemples de mesure de corrélation: la somme des écarts quadratiques SSD, la somme des écarts absolus SAD, la corrélation croisée centrée et normalisée ZNCC, etc. Après avoir calculé pour chaque couple de pixel son score de ressemblance. On réduit le nombre de candidats grâce à un seuil. Puis, on sélectionne les meilleurs.

**Votre objectif** est d'implémenter la méthode de mise en correspondance par corrélation en utilisant la fonction d'opencv : matchTemplate(). Mais attention, n'oubliez pas d'introduire la contrainte épipolaire ou votre code mettra plus de 20 minutes pour s'exécuter ! Autre point important, la fonction matchTemplate() s'applique sur des images en niveaux de gris, n'oubliez pas de transformer vos images couleur en image en niveaux de gris.

## 3. Carte de disparité (2 points)

La carte de disparité permet d'afficher le déplacement que va avoir un même pixel de l'image gauche vers l'image droite en supposant que les deux images sont superposées. On aura besoin des paires de pixels appariés obtenues à l'étape de corrélation. Étant donné que les pixels homologues se retrouvent sur une même ligne grâce à la rectification des images. La disparité se calcule comme suit :

$$\text{disparité}(x, y) = y\_left - y\_right$$

Afin de rehausser le contraste de l'image de disparité, voici la formule qui permet d'uniformiser la distribution des intensités dans la plage de de niveaux de gris [0 - 255] :

$$\text{ImgOut(x,y)} = 255 \left( \frac{\text{ImgIn(x,y)} - \text{c}}{d - c} \right)$$

Avec : c et d = valeurs min et max de l'image.

**Votre objectif** est de générer la carte de disparité et de l'afficher sur écran. Cette carte permet déjà d'observer la profondeur de la scène. En effet plus l'objet est proche de la caméra plus son intensité sera claire, et inversement, les objets au fond de la scène paraitront plus sombre.

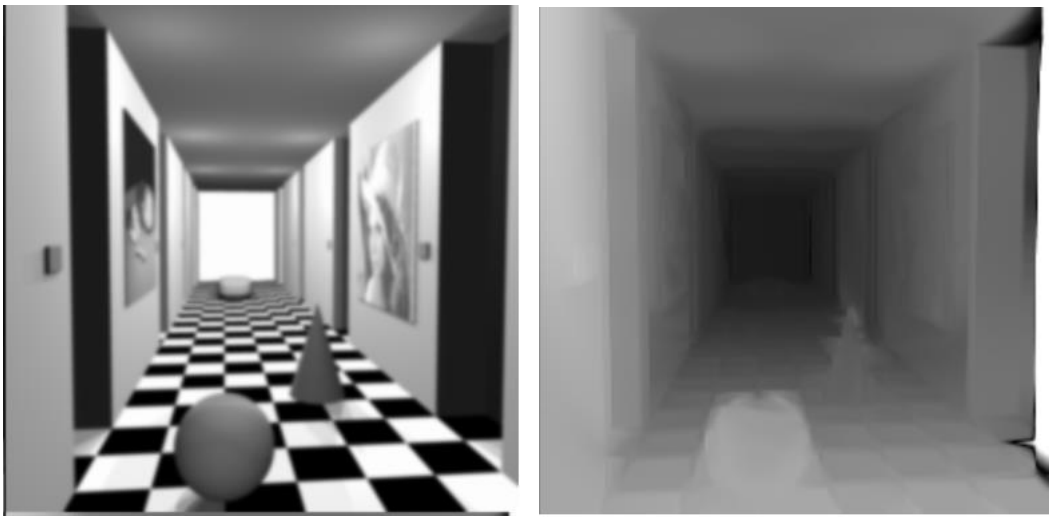

**Figure 04 :** à gauche image originale, à droite carte de disparité.

### 4. Triangulation (1 point)

La triangulation permet de calculer les coordonnées 3D de chaque pixel. Utilisez la fonction : triangulatePoints(). La fonction prend comme entrées : les deux matrices de projection rectifiées ainsi que les matrices des points mis en correspondance.