



VPOA - Computer Vision

Rapport travaux pratiques

Nicolas Blin
Ilan Guenet

December 9, 2021

1 Perception 3D

1.1 Vision monoculaire

La perception 3D est le fait de visualiser le monde 3D sur un plan 2D, c'est-à-dire une image 2D. Dans le cas de la vision monoculaire avec une seule caméra, pour transformer des points 3D monde vers des points 2D sur l'image, il suffit de trouver une matrice **caméra** notée C de dimension (3, 4) tel que :

$$P_{2d} = [x \ y \ 1]^T \approx C.P_w = C. [X \ Y \ Z \ 1]^T$$

La matrice caméra est le produit de deux matrices tel que
 $C = K. [R \ | \ t]$ avec :

- $[R \ | \ t]$ la matrice extrinsèque de dimension (3, 4) qui permet de transformer un point 3D dans l'espace monde en un point 3D dans l'espace 3D de la caméra. Cette matrice est composé de deux autres matrices :

- R la matrice rotation
- t un vecteur de translation

- K la matrice intrinsèque de dimension (3, 3) qui permet de transformer un point 3D dans l'espace 3D de la caméra en un point 2D sur un plan.

Les coefficients de K sont les suivants : $K = \begin{bmatrix} \alpha_x & 0 & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ avec

$$\begin{cases} \alpha_x = \frac{f_x}{d_x} \\ \alpha_y = \frac{f_y}{d_y} \\ [c_x, c_y]^T \text{ le centre de l'image} \end{cases}$$

Enfin, le modèle complet utilise des coefficients de distorsions pour justement prendre en compte la distorsion. Un modèle de distorsion utilisé est celui de *Brown-Conrady* tel que

$$P_{2d} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x.x_{distorted} + c_x \\ \alpha_y.y_{distorted} + c_y \\ 1 \end{bmatrix} \approx K. [R \ | \ t]. P_w$$

avec $\begin{cases} x_{distorted} = x. \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + 2p_1xy + p_2(r^2 + 2x^2) \\ y_{distorted} = y. \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + p_1(r^2 + 2y^2) + 2p_2xy \end{cases}$

Attention, les équations ne sont plus linéaires.

1.1.1 Calibration

La calibration d'une caméra monoculaire permet de calculer la matrice caméra C à partir de points de références et de points d'intérêts associés à ces points de références. On peut trouver ces points d'intérêts dans une photo. Avec

l'association des points théoriques et les points calculés, il suffit de résoudre des équations pour récupérer la matrice caméra C de la webcam. Pour ce faire, il faut utiliser une **mire**. Dans notre cas, nous allons utiliser une mire de type *chessboard*. Il en existe d'autres : *cercles symétriques*, *cercles asymétriques*, etc.

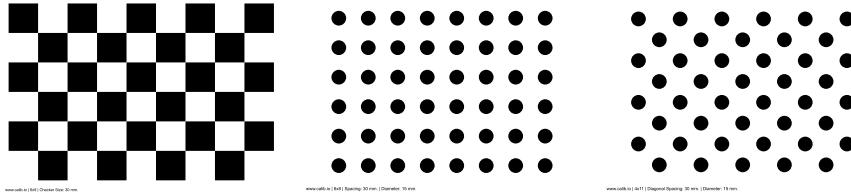


Figure 1: 3 mires (de gauche à droite) : mire chessboard, mire cercles asymétriques, mire cercles symétriques

Pour une mire de type *chessboard*, les points d'intérêts dont nous connaissons leurs positions théoriques sont les intersections (coins) entre les carrés de différentes couleurs et qui sont tous séparé par la même distance.

Afin de détecter les coins de la mire sur une acquisition, on va utiliser la fonction d'openCV `findChessboardCorner`. La fonction essaye de déterminer si l'image contient un pattern de *chessboard*. Si elle peut localiser les coins internes du *chessboard* elle renvoie les coins en les réordonnant ligne par ligne de gauche à droite. Définissons :

- *objectPoints* correspond aux points de référence, c'est-à-dire les coins (i, j) décalés de *patternSize*
- *imgPoints* correspond aux coins récupérés par la détection, donc ce qui a été concrètement détecté lors de la capture.

De plus, afin d'obtenir un meilleur calibrage, il est mieux d'acquérir plusieurs photos de *chessboard* avec des positions différentes sur les images. La feuille de papier doit tout même rester droite (pas d'inclinaison) afin de supposer la coordonnée Z à 0. Plus il y a de points détectés associés à des points de références, plus le calcul de la matrice sera précis et générique sur l'ensemble de l'image. Ci-dessous, toutes les photos de *chessboard* acquises sur lesquels les points d'intérêts ont été calculés et qui vont permettre de calculer la matrice caméra C .

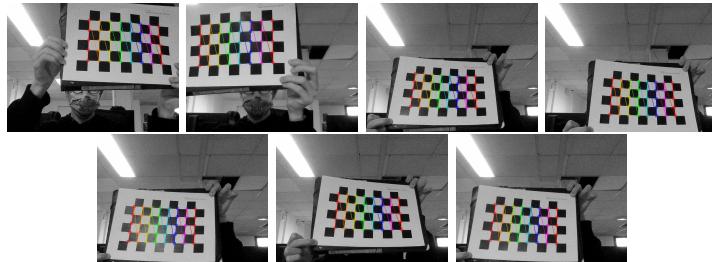


Figure 2: Résultat de *findChessboard* sur différentes images acquises

Le calibrage de la caméra peut être effectué en calculant la matrice caméra à l'aide de la fonction d'openCV `calibrateCameraExtended`. Cette fonction va calculer la matrice caméra ainsi qu'extraire la matrice intrinsèque, extrinsèque (rotation et translation) et les coefficients de distorsion. On peut estimer la qualité de la calibration de la caméra en calculant la Root Mean Square (RMS) erreur. Une fois la matrice caméra calculée, les points *objPoints* sont reprojetées avec cette nouvelle matrice caméra. Ensuite l'erreur est calculée entre les points reprojetés et les *imgPoints* précédemment calculés. Ensuite, une moyenne est calculée pour chaque image.

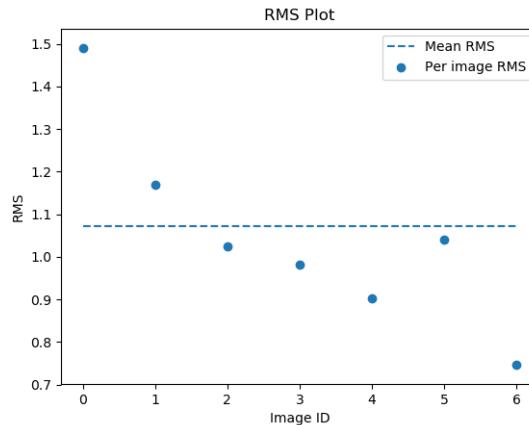


Figure 3: Résultat RMS de calibrage pour chaque image

La RMS de notre calibrage semble assez correcte. La RMS se situe autour de 1 pour chaque image. Elle n'est cependant pas idéale. La RMS idéal pour une bonne calibration doit être comprise entre 0.1 et 0.25. Ci-dessous, les résultats de calibrations :

- Matrice caméra :
$$\begin{bmatrix} 856.35521123 & 0. & 487.12291102 \\ 0. & 804.97433103 & 85.0992183 \\ 0. & 0. & 1. \end{bmatrix}$$

- Coefficient de distorsions :

$$[-0.5162738 \quad 0.71400404 \quad 0.05614805 \quad -0.00922509 \quad -0.77096983]$$

1.1.2 rectification

La rectification est un procédé qui permet d'enlever les effets de distorsions sur les caméras. La rectification se fait après la calibration de la caméra.

Pour commencer, il faut calculer une nouvelle matrice caméra à l'aide d'une fonction opencv getOptimalNewCameraMatrix. Cette fonction calcul une nouvelle matrice intrinsèque basé sur un paramètre alpha. Si alpha est égal à 0, la fonction renvoie une matrice qui permet de calculer une image sans distorsion en enlevant tous les pixels non voulus. Au contraire, si alpha est égal à 1, tous les pixels sont retenus et il y aura des bornes noires. Ces bandes noires permettront de voir les différents types de distorsion que la caméra provoque.

Une fois cette nouvelle matrice intrinsèque est calculée, la fonction initUndistortRectifyMap va calculer deux fonctions de mapping à partir de la matrice intrinsèque originelle, la matrice intrinsèque calculée pour limiter la distorsion et les coefficients de distorsion calculés pendant le calibrage de la caméra. Une fonction de mapping permet de décaler les pixels dans l'image plutôt que de d'avoir à résoudre l'équation contenant les coefficients de correction à chaque fois. Il y a une fonction de mapping pour les translations sur l'axe x et une autre pour les translations sur l'axe y. Enfin, il ne manque plus qu'à remapper tous les pixels de l'image d'entrées avec les deux fonctions de mapping.

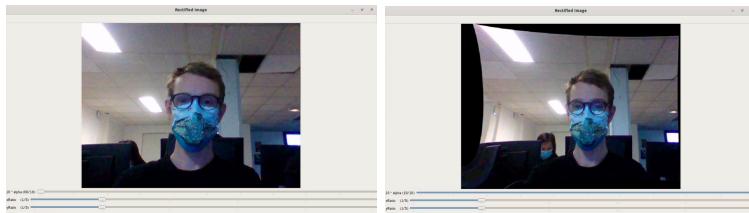


Figure 4: Rectification de l'image. À gauche alpha = 0, l'image est rectifiée en enlevant les bandes noires. À droite alpha = 1, l'image est rectifiée en gardant les bandes noires.

On remarque que la caméra présente une distorsion en coussin assez élevé. On le constate aussi lorsque l'on regarde l'image non corrigée, les traits du plafond sont bombés. On constate aussi une distorsion tangentielle (coin en haut à gauche). Ce dernier n'est pas positionné correctement, de même avec le coin en haut à droite.

1.2 Stéréovision

Nous voulons maintenant calibrer et rectifier une caméra en stéréovision. Le procédé sera le même, seul les calculs vont changer pour s'adapter au cas de la stéréovision.

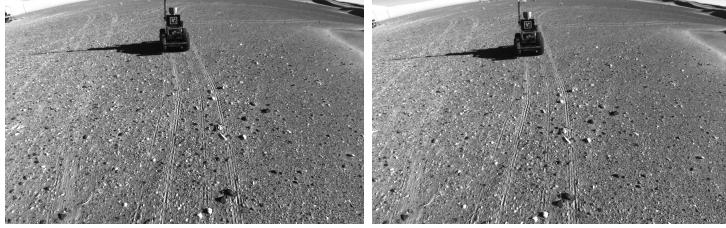


Figure 5: Exemple d'image en stérovision

1.2.1 Calibration

Cette fois-ci au lieu d'avoir des couples de (`objPoints`, `imgPoints`), on a des couples (`objPoints`, `imgPoints` image de gauche, `imgPoints` image de droite). La fonction d'openCV `stereoCalibrateExtended` va calculer comme `cameraCalibrateExtended` la matrice intrinsèque, la matrice extrinsèque et les coefficients de distorsions.

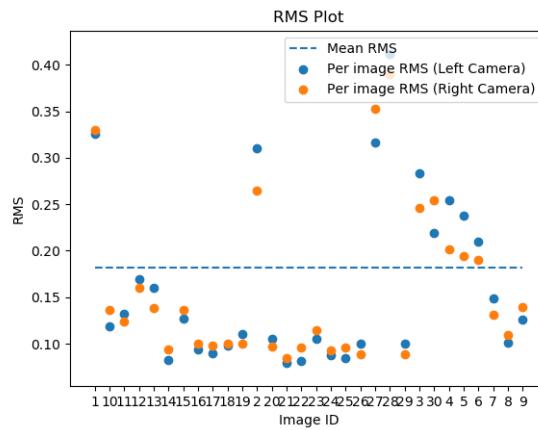


Figure 6: La RMS de calibration sur des images de chessboard en stéréovisions

La RMS semble correcte une nouvelle fois (pas plus de 0.4 pour chaque image). De plus, on trouve les coefficients suivants :

- Matrice caméra gauche :
$$\begin{bmatrix} 1.05636408e^{03} & 0.00000000e^{00} & 9.62640002e^{02} \\ 0.00000000e^{00} & 1.05667848e^{03} & 6.12172662e^{02} \\ 0.00000000e^{00} & 0.00000000e^{00} & 1.00000000e^{00} \end{bmatrix}$$
- Coefficient de distorsion gauche :
$$[-0.26237687 \quad 0.13443635 \quad 0.00044231 \quad 0.00031455 \quad -0.04058719]$$

- Matrice caméra droite : $\begin{bmatrix} 1.05323706e^{03} & 0.00000000e^{00} & 9.54063330e^{02} \\ 0.00000000e^{00} & 1.05345348e^{03} & 6.08116097e^{02} \\ 0.00000000e^{00} & 0.00000000e^{00} & 1.00000000e^{00} \end{bmatrix}$
- Coefficient de distorsion droite : $[-0.26106344 \quad 0.13154969 \quad 0.00043333 \quad 0.00062924 \quad -0.03734474]$
- Matrice de rotation : $\begin{bmatrix} 9.99930963e^{-01} & 1.17330085e^{-02} & -6.37169933e^{-04} \\ -1.17167208e^{-02} & 9.99703091e^{-01} & 2.13646620e^{-02} \\ 8.87652512e^{-04} & -2.13557215e^{-02} & 9.99771547e^{-01} \end{bmatrix}$
- Vecteur de translation : $[-0.27060747 \quad 0.00258373 \quad 0.00029731]$

1.2.2 Rectification

Pour la rectification stéréovision, il suffirait d'appeler des fonctions similaires que rectification monovision. Il faut que ces fonctions changent un peu leurs calculs pour s'adapter à la stereovision. De plus, il faut calculer 4 fonctions de mapping, 2 fonctions de mapping par image. Bien évidemment, il faudra remapper les pixels de l'image d'entrées de gauche avec les fonctions de mapping associés à la partie gauche. Et il faudra faire de même avec les pixels d'entrées de droite avec les fonctions de mapping associés à la partie droite.

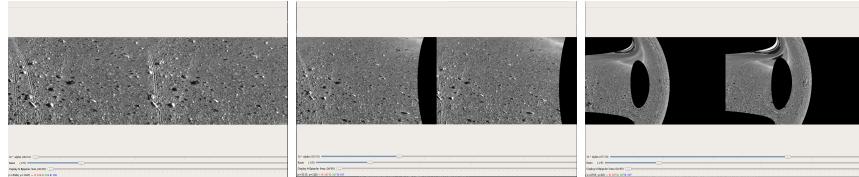


Figure 7: Rectification sur stéréovision avec différents alpha. De gauche à droite, alpha=0, alpha=3 et alpha=7

Quand alpha est grand, on peut remarquer qu'il y a beaucoup de distorsion. Il y a des distorsions radiales (positive sur le bord droit et négative sur le bord haut et bas). De plus les quatre coins sont largement décalés à cause d'une distorsion tangentielle. Il est donc réellement nécessaire de rectifier cette caméra afin d'acquérir des images correctes. Avec alpha égal à 0, l'image est correctement rectifiée et les bandes noires enlevées.

2 Segmentation & Registration

2.1 Segmentation 3D

La **segmentation 3D** est le procédé qui permet de récupérer un objet 3D dans une scène 3D. La segmentation peut se faire à la main avec la fonction `draw_geometries_with_editing` d'open3D.

Cependant, le fait que la capture ne soit pas parfaite et seulement faite par sélection à la main génère forcément des outliers. Afin de supprimer les points aberrant nous utilisons la fonction `statistical_outlier_removal` d'open3D dont le fonctionnement est similaire à **DBSCAN**. Deux paramètres peuvent être modifiés :

- `nb_neighbors`, qui représente le nombre de voisins à prendre en compte pour calculer la distance moyenne.
- `std_ratio`, représente le seuil d'écart-type pour sélectionner ou non une distance moyenne de points. Cela permet de filtrer plus ou moins les ensembles.

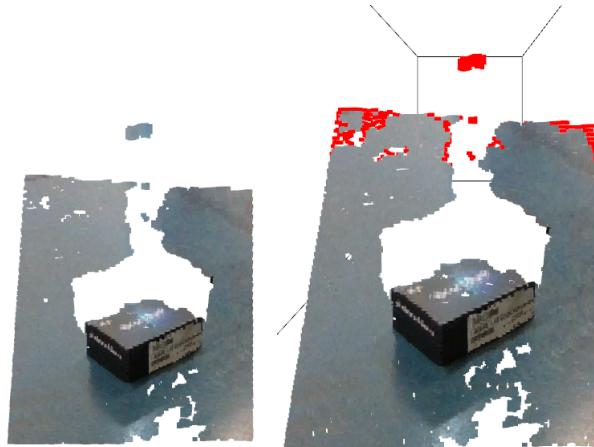


Figure 8: Visualisation de l'objet segmenté (à gauche) et des outliers (en rouge) sur cet objet (à droite)

Il reste un problème. Nous voulons segmenter l'objet et seulement l'objet. La segmentation à la main implique que l'on récupère le sol. Il existe un moyen de supprimer le sol. La fonction `estimate_normals` pour calculer les normales. C'est grâce à une analyse de la matrice de covariance en regardant l'axe principal des points adjacents (PCA) que la fonction trouve le plan et ainsi la normale. L'étape d'alignement sert à faire en sorte que les normales soient toutes aligner dans la même direction. En effet, pour un plan donné il existe une infinité de normales au plan. Les normales peuvent être placées de part et d'autre du plan et peuvent avoir des normes différentes. L'étape de normalisation des normales nous assure qu'elles ont toutes la même norme.

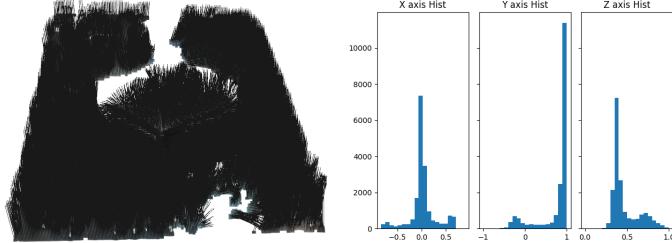


Figure 9: Visualisation des normales (à gauche) et de la distribution des normales sur les trois axes (à droite)

Les normales correspondant au sol sont en majorité. On vient donc sélectionner dans l'histogramme les normales dont les caractéristiques correspondent aux normales majoritaires ce qui permet de ne sélectionner que celle du sol. Les valeurs de ce vecteur sont les suivantes :

- x : 0.026771903723373125
- y : 0.9983725134009729
- z : 0.4020599587684169

L'objectif est de trouver un algorithme qui, à partir de 2 vecteurs (celui du plan horizontal et celui ci-dessus) nous donne une matrice de rotation permettant de passer de l'un à l'autre. On a la formule suivante pour la matrice de

$$\text{rotation : } R = I + [v]_{\times} + [v]_{\times}^2 \frac{1-c}{s^2} \text{ avec } \left\{ \begin{array}{l} I \text{ matrice identité} \\ a \text{ et } b \text{ nos 2 vecteurs} \\ v = ab \\ s = \|v\| \\ c = a \cdot b \\ [v]_{\times} \stackrel{\text{def}}{=} \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \end{array} \right.$$



Figure 10: Objet aligné sur le plan horizontal

Il suffit pour supprimer le sol de faire l'inverse que ce que l'on a effectué pour estimer le sol. On regarde notre histogramme de vecteur et au lieu de conserver les vecteurs majoritaires qui correspondent au sol on ne conserve que ceux qui ne sont pas majoritaires, il ne reste donc que l'objet.

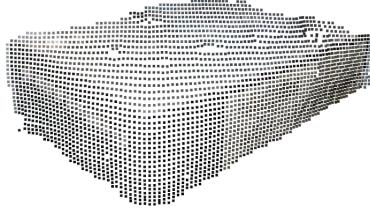


Figure 11: Boite segmentée sans le sol

2.2 Contrôle qualité

Après segmentation de l'objet de référence, la détection des potentiels défauts peut être effectuée. La finalité est d'automatiser le processus de tri entre les pièces correctes et les pièces défectueuses. Le processus est appelé **recalage** et fonctionne de la manière suivante : aligner de jeux de points (2D ou 3D) et rechercher une transformation qui permet de projeter les points p'_i du nuage P' sur les points p_i du nuage P .

Nous utilisons deux algorithmes pour obtenir des résultats précis assez rapidement. Le premier algorithme **RANSAC** permet d'estimer une transformation initiale. Le deuxième algorithme **ICP** permet de raffiner l'estimation.

L'ICP (Iterative Closest Points) est un algorithme itératif. On cherche une erreur minimale entre les points p_i et les points p'_i projetés. À chaque itération, une nouvelle matrice de rotation et un vecteur de translation qui réduisent l'erreur sont calculés. Il existe deux méthodes de résolution.

La première méthode est une résolution par la méthode des moindres carrés.

$$\arg \min_{R,t} f(R,t) = \arg \min_{R,t} \sum_{i=1}^n \|p_i - (R.p'_i + t)\|^2$$

Au minimum de f , s'il existe, on a: $\nabla f = 0 \Rightarrow \begin{cases} \frac{\partial f}{\partial R} = 0 \\ \frac{\partial f}{\partial t} = 0 \end{cases}$

La deuxième méthode de résolution est une résolution par SVD.

Soit (P, P') les jeux deux points d'entrées.

1. Déterminer les barycentres $p_m = \frac{1}{n} \sum_{i=1}^n p_i$ et $p'_m = \frac{1}{n} \sum_{i=1}^n p'_i$
2. Calculer la matrice $H = \sum_{i=1}^n q'_i \cdot q_i^T$ avec $\forall i \in [1, n] \begin{cases} q_i = p_i - p_m \\ q'_i = p'_i - p'_m \end{cases}$

3. Décomposer H en valeurs singulières : $\exists(U, V, \Sigma) \in M_3(\mathbb{R})^3$ $tqH = U.\Sigma.V^T$
4. Calculer $R = V.U^T$ et $t = p_m - R.p'_m$

On peut trouver des exemples d'implémentation sur ce lien

En terme d'implémentation, la fonction `registration_ransac_based_on`

`feature_matching` d'open3D effectue le premier recalage global avec l'algorithme RANSAC sur le nuage de point. Les données utilisées pour ce recalage sont sous-échantillonnées pour des questions de performances. Dans un second temps, c'est la fonction `register_icp` d'open3D qui effectue l'ICP à partir des résultats du premier recalage. La méthode de résolution est la méthode des moindres carrés point à point comme décrit plus haut.

Ces deux dernières fonctions renvoient les résultats sur le recalage. Définissons ce qu'est un **inlier**. Les points inliers sont les points qui ne sont pas des **outliers**. Les points outliers sont des points considérés comme de bruit. Il y a la mesure **fitness** qui mesure l'air de chevauchement, c'est-à-dire $\frac{\# \text{ de correspondances inlier}}{\# \text{ de points}}$. Il y a aussi la mesure **inlier_rmse** qui mesure la Root Mean Square Error (RMSE) de toutes les correspondances inliers. Ansi, plus le inlier_rmse est petit, mieux le recalage est de bonne qualité. Enfin, il y a **correspondance_set_size** qui correspond à la taille de points correspondants, associés.

Visualisons quelques résultats de recalage avec la boîte.

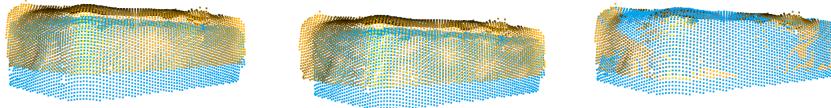


Figure 12: Recalage d'une boîte sans défaut. De gauche à droite : 2 nuages de points initiaux, après recalage global avec RANSAC (fitness=1.0, inlier_rmse= $6.948525e^{-03}$, and correspondence_set size of 2955), après recalage ICP (fitness=1.0, inlier_rmse= $1.363070e^{-03}$, and correspondence_set size of 2955)

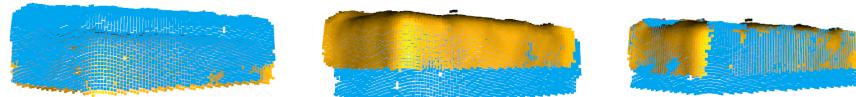


Figure 13: Recalage d'une autre boîte sans défaut. De gauche à droite : 2 nuages de points initiaux, après recalage global avec RANSAC (fitness=1.0, inlier_rmse= $8.616345e^{-03}$, and correspondence_set size of 2724), après recalage ICP (fitness=1.0, inlier_rmse= $8.616345e^{-03}$, and correspondence_set size of 2724)

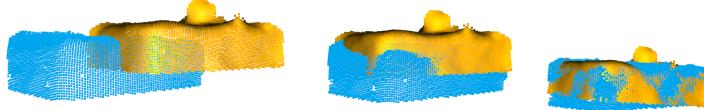


Figure 14: Recalage d'une autre boîte qui présente un défaut. De gauche à droite : 2 nuages de points initiaux, après recalage global avec RANSAC ($\text{fitness}=9.751693e^{-01}$, $\text{inlier_rmse}=7.543891e^{-03}$, and $\text{correspondence_set size of } 3024$), après recalage ICP ($\text{fitness}=1.0$, $\text{inlier_rmse}=3.233094e^{-03}$, and $\text{correspondence_set size of } 3101$)

Lorsque l'objet présente des défauts, on remarque bien que le recalage présente plus d'erreur (inlier_rmse plus élevée). On remarque avec la mesure fitness que les objets ne présentent pas d'outliers sauf lorsque l'objet présente des défauts.

3 Localisation

La localisation en computer vision permet détecter un objet en 2D sur une image donc une vidéo. Puis, l'objectif va être de localiser cet objet dans l'espace 3D. Toute cette partie-là se fait à l'aide d'un système monocaméra. La caméra est supposée étalonnée auparavant. Trois étapes sont nécessaires :

- Création de la référence
- Tracking de l'objet 2D à partir de la référence
- Détection de pose de l'objet en 3D à partir du tracking 2D

Les étapes sont liées entre elles et doivent se faire dans cet ordre.

3.1 Crédation de la référence

Dans cette partie, le but est de créer une image référence de notre objet référence. Pour atteindre ce but, 3 étapes sont nécessaires. Premièrement, il faut acquérir une photo de notre objet de référence. Pour faciliter les calculs, il est supposé que l'objet est plat sans relief. La photo acquise n'a pas besoin d'être bien cadrer. C'est justement les prochaines étape qui vont le faire. En deuxième étape, il faut, à la main, détecter les coins de notre objet.

Pour la troisième et dernière étapes, le but est de trouver une transformation tel que notre objet défini par ces quatre coins soit l'image de référence entière. On définit une dimension d'image attendue pour notre objet de référence. Les dimensions de l'objet sont connues. Il faut choisir un rapport largeur/hauteur qui correspond aux dimensions de l'objet afin de ne pas le déformer. D'un côté, il y a les coins de l'objet détecter à la main. De l'autre, il y a les coins de l'image définis par sa dimension tel que $\begin{bmatrix} (0, 0) & (largeur, 0) \\ (0, hauteur) & (largeur, hauteur) \end{bmatrix}$.

Ces points sont appelés points cibles. Notons que la coordonnée de la largeur est en première car openCV fonctionne de cette manière et soyons consistent avec openCV.

Maintenant, il ne manque plus qu'à trouver une transformation de perspective entre les points détectés et les points cibles. Cela va se faire en deux temps. Premièrement, trouver une homographie, entre les points détectés et les points cibles. Une homographie est une transformation (matrice (3, 3)) qui map tous les points d'une image vers des points correspondants sur une autre image.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ avec } H \text{ une homographie.}$$

Le calcul d'une homographie peut se faire avec la fonction `findHomography` d'openCV. Une fois cette transformation trouvée, il faut l'appliquer sur toutes l'images d'entrée. Cela peut se faire avec la fonction `warpPerspective` d'openCV qui prend en paramètre l'image à transformer et l'homographie.

À la fin de toutes ces étapes, une image seulement avec notre objet de référence est bien obtenue. La prochaine étape va être de tracker cet objet dans une vidéo.



Figure 15: Création de l'image référence. 1. Acquérir l'objet de référence. 2. Déetecter les coins de l'objet. 3. Trouver une homographie pour recadrer l'objet référence.

3.2 Tracking

Le tracking d'un objet est le fait de trouver cet objet dit de référence dans une image. Si cet objet est trouvé dans plusieurs images consécutives, l'objet est alors suivi (tracker) au cours du temps.

Pour trouver notre objet de référence dans la scène, il va falloir trouver des points de correspondances entre l'objet de référence et des points dans la scène. Si suffisamment de points de correspondance sont trouvés on peut alors trouver la position de l'objet dans l'image 2D. Pour détecter des points de correspondances aussi appelés **feature 2D**, un matcher de feature 2D va être nécessaire. Il existe ORB ou AKAZE. En premier, les points clés aussi appelés keypoints sont calculés sur l'image. Ces keypoints sont des points qui ont plus

d'information que les autres par exemple grâce à du contour dans son voisinage. Puis, il faut calculer des descripteurs pour chacun de ses points. Ces descripteurs permettent de décrire ces points clés. Ainsi, avec une description ces points pourrait être retrouvé. Ces descripteurs vont surtout être utilisés pour matcher des points clés entre deux images.

Dans notre cas, l'implémentation ORB d'openCV est utilisé. La fonction `detectAndCompute` permet de détecter les points clés puis calculés leurs descripteurs. Cette fonction est appliquée sur l'image de référence de l'objet et sur l'image de la scène acquise. Les points clés et descripteurs sont récupérés pour chaque image. Il faut maintenant associer les points clés des deux images entre eux. On utilise le matcher plus proche voisins `knnMatch`. Ce matcher renvoie les N points clés les plus proches des points clés de références. On prend N = 2 pour obtenir les deux plus proches points clés. Ensuite ces associations de points vont être filtrées avec le ratio de Lowe. La notion de proche est défini par le ratio de Lowe tel que :

$$\begin{cases} \text{OK si } \text{distance}(\text{match1}) / \text{distance}(\text{match2}) < \text{matchingRatio} \\ \text{KO si } \text{distance}(\text{match1}) / \text{distance}(\text{match2}) \geq \text{matchingRatio} \end{cases}$$

Si la distances entre les deux points est trop proche, une décision ne peut pas être prise entre quels des deux points doit être associé au point clé de référence. Ces associations sont alors enlevées de la liste d'association.



Figure 16: Résultat de matching de points clés avec différent ratio de Lowe

Plus le ratio de Lowe est petit, moins de points clés sont gardés. Par contre, plus le ratio de Lowe est élevés plus il y a de matching qui sont incorrects. Il faut trouver un juste milieu.

Pour avoir un effet de tracking, il faut afficher à l'écran les contours de l'objet sur l'image acquise. Les contours de l'objet de références vont être utilisé. Une homographie entre les associations de points clés est calculés avec `findHomography` d'openCV. Ainsi, une transformation entre le monde de l'objet de référence et la scène est trouvée. Il suffit de transformer les contours de l'objet de référence avec cette homographie. Les contours sont déterminés par les quatre coins de l'objet plat.

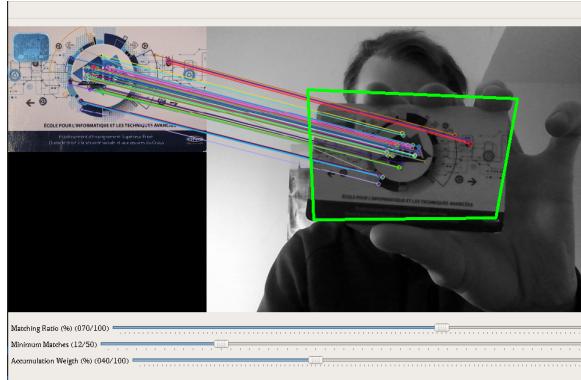


Figure 17: Résultat du tracking de l'objet de référence dans la scène 2D.

On peut utiliser une somme accumulée avec des poids `accumulateWeighted` entre l'homographie de l'image précédente et l'homographie de l'image courante afin que le mouvement du carré de tracking soit plus smooth. Mathématiquement, la fonction fonctionne comme suit :

$$H = (1 - weight).H_{previous} + weight.H$$

où H est l'homographie courante et $H_{previous}$ est l'homographie de la détection précédente.

Finalement, notons qu'il faut redimensionner l'image de référence à la taille de l'objet dans la scène, car sinon l'homographie a du mal à être calculé.

3.3 Détection de pose

La détection de la position l'objet dans l'espace 2D a été effectué dans la partie précédente. On veut maintenant détecter la pose 3D de l'objet à partir de l'image 2D.

Le procédé est identique au tracking avec une étape de plus à la fin. Reprenons après le calcul de l'homographie entre les points clés de références et les points clés de l'image acquise. Les coins de référence sont projetés grâce à l'homographie précédemment calculée. Il faut ensuite résoudre un **Perspective N-points Problem (PnP)**. Cela consiste à trouver les paramètres extrinsèques (rotation et translation) à partir de correspondances 3D et 2D. Le PnP est résolu par la fonction `solvePnP`. Les correspondances sont les coins 3D de la face de l'objet de références et les coins 2D de l'objet de références précédemment projetés.

Enfin, il ne manque plus qu'à projeter tous les coins 3D de références de l'objet à partir la matrice caméra pré calibrée, les coefficients de distorsion pré calculés et les paramètre extrinsèque calculé à l'instant. Puis, les contours sont dessinés à partir des coins 3D projetés.

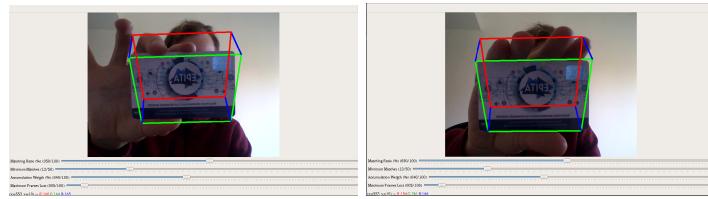


Figure 18: Résultat de la détection de pose d'une carte

Visuellement, la détection ne semble pas très bien fonctionné avec une carte. Cela est probablement dû au fait que la carte n'a pas de volume notamment sur l'axe Z. Il faudrait essayer avec des objets plats qui ont plus de volume tel qu'un livre.