



---

# RAPPORT DE PROJET INTEG OPTION ROBOTIQUE

Robinson BUCHE, Benjamin JOUVELOT

année 2025

---

---

L'intégralité du code est disponible sur le Github suivant: <https://github.com/ASTARDANGER/INTEG.git>

## Sommaire

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Acquisition des images</b>	<b>3</b>
2.1	Récupération des photos et des données . . . . .	3
2.2	Acquisition des données . . . . .	4
<b>3</b>	<b>Étalonnage de la caméra</b>	<b>4</b>
3.1	Résultat . . . . .	4
3.2	Vérification des résultats . . . . .	5
3.3	Angles d'ouverture de la caméra . . . . .	5
<b>4</b>	<b>Pose de la caméra par rapport à l'outil</b>	<b>6</b>
4.1	Transformation homogène . . . . .	7
4.2	Vérification des résultats . . . . .	7
4.3	Résultat . . . . .	8
<b>5</b>	<b>Reconnaissances des trous dans l'image.</b>	<b>8</b>
5.1	Fonctionnement de l'algorithme . . . . .	9
<b>6</b>	<b>Commande pour réaliser l'insertion.</b>	<b>10</b>
6.1	Loi de commande hybride vision/force . . . . .	10
6.2	Démarche d'insertion . . . . .	10
6.3	Résultat . . . . .	11
6.3.1	En simulation . . . . .	11
6.3.2	En réel . . . . .	11
<b>7</b>	<b>Conclusion</b>	<b>12</b>

## 1 Introduction

L'objectif de ce projet est de réaliser une tâche d'insertion (peg-in-hole), en utilisant une loi de commande hybride (vision/force). Nous avons travaillé sur le robot Franka Emika FR3, équipé d'une caméra Intel Realsense D435 sur son effecteur. Le robot est aussi équipé de capteurs de couple sur chacune de ses articulations, capteurs permettant de reconstruire le torseur dynamique (trois composants de force et trois composants de couple) à son outil. La tâche à réaliser est illustrée en Fig. 1a, et les repères utilisés dans le projet sont détaillés en Fig. 1b.

Le projet s'est décomposé en plusieurs étapes :

1. récupération des données de positionnement du robot ainsi que les images de la caméra sous ROS 1 [2.1]
2. acquisition d'une base de photos de la mire d'étaffonnage et du trou d'insertion par comanipulation du robot [2.2]
3. étaffonnage de la caméra afin d'obtenir ses paramètres intrinsèques et extrinsèques, grâce aux images récupérées en 2.
4. estimation de la pose relative entre la caméra et l'outil
5. développement d'un algorithme de traitement d'images permettant de reconnaître le trou d'insertion dans les images acquises par la caméra
6. développement d'une loi de commande hybride vision/force afin de réaliser la tâche d'insertion

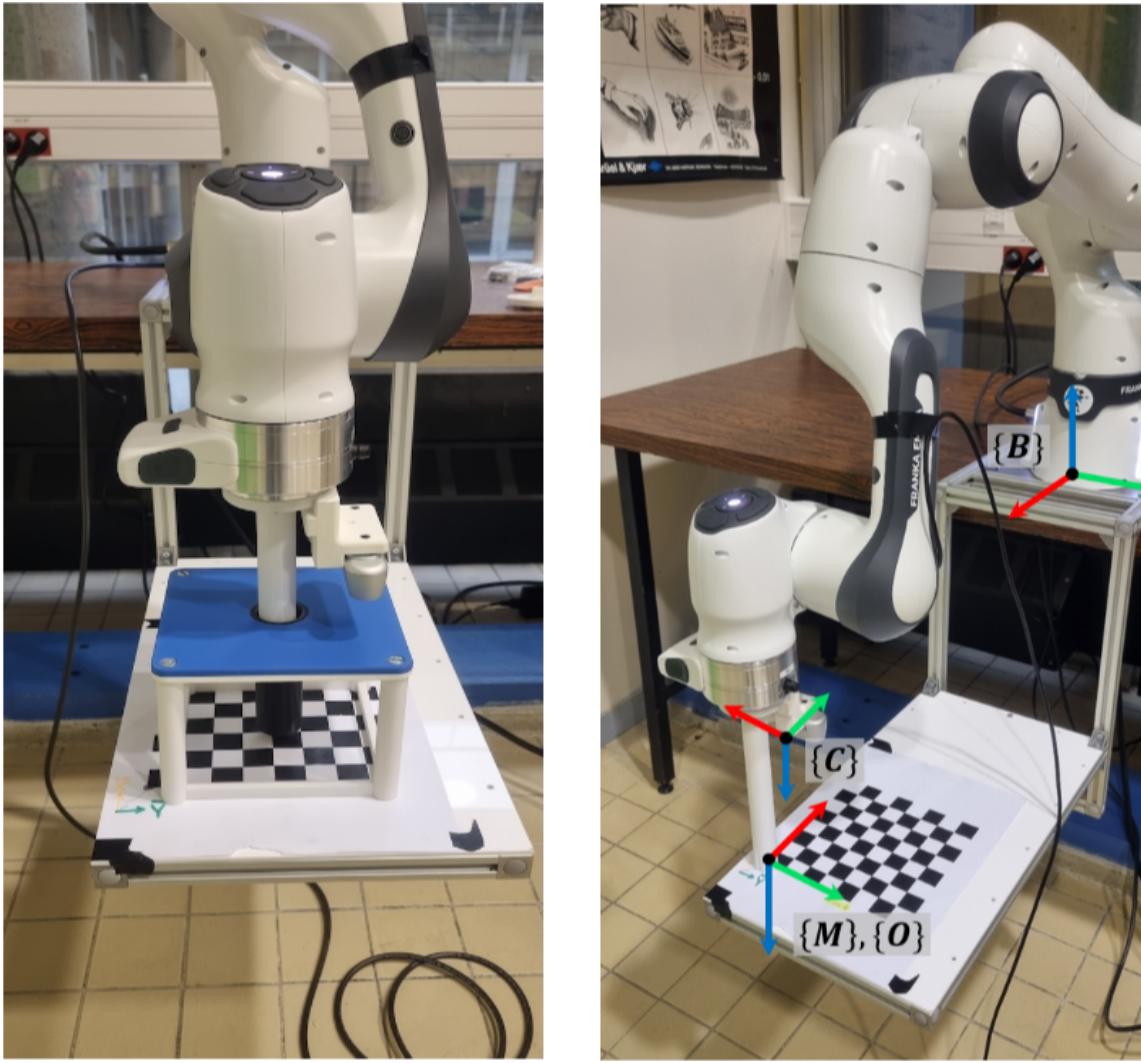


Figure 1: (a) à gauche et (b) à droite

## 2 Acquisition des images

Dans un premier temps, notre objectif est d'acquérir une série de photos depuis la caméra installée à l'extrémité du robot, ainsi que les poses cartésiennes du robot (représentées par les matrices de transformation  ${}^B T_O$ ) associées à chaque photo. Ces données nous permettront d'identifier la pose de la caméra dans le repère de l'outil, notée  ${}^O T_C$ , dans la section 3. Ensuite, nous devons acquérir un flux d'images capturées par la caméra, qui servira au développement d'un algorithme de détection de trous. La communication avec le robot s'effectue via le middleware ROS 1.

### 2.1 Récupération des photos et des données

L'ensemble des fonctionnalités nécessaires pour atteindre nos objectifs a été intégré dans le nœud ROS1 `integ_vision_node`. Ce nœud souscrit aux topics `/franka_state_controller/franka_states/O_T_EE` et `/camera/color/image_raw` afin de récupérer les images de la caméra.

- En appuyant sur la touche p, le programme capture une photo au format .jpg et enregistre sa pose cartésienne  $T_0^B$  dans le fichier `cart_poses.txt`.
- La touche v permet de démarrer et d'arrêter la prise de photos en continu.
- L'interaction avec le clavier est rendue possible grâce à la bibliothèque Python `pynput`.

## 2.2 Acquisition des données

En raison du nombre insuffisant d'ordinateurs dans la salle, nous avons utilisé une machine virtuelle pour programmer notre noeud. Cependant, l'utilisation d'une machine virtuelle a rendu l'acquisition des données particulièrement complexe. Pour contourner cette difficulté, nous avons été autorisés à utiliser les données de nos collègues après les avoir assistés dans leurs manipulations du robot. Certaines photos étant inutilisables, nous disposons finalement de 12 photos pour la calibration de la caméra et de 35 images issues du flux continu pour l'algorithme de détection de trou.

## 3 Étalonnage de la caméra

Pour contrôler le robot à l'aide de la vision, il est nécessaire d'étailler le système robot/caméra. L'algorithme charge une série d'images d'un damier de calibration et détecte automatiquement les coins des carreaux noirs et blancs à l'aide de `cv2.findChessboardCorners`. Une correction est appliquée (`cv2.cornerSubPix`) pour affiner leur position, et les coins sont triés (`sort_corners`) pour correspondre à un ordre cohérent.

Une fois les points image (2D) et leurs correspondants dans l'espace réel (3D) collectés, la calibration est réalisée avec `cv2.calibrateCamera`. Cette étape permet d'estimer :

- La **matrice intrinsèque** de la caméra (`mtx`), qui définit ses caractéristiques optiques.
- Les **coefficients de distorsion** (`dist`), qui corrigent les déformations dues à l'objectif.
- Les **vecteurs de rotation et de translation** (`rvecs`, `tvecs`), qui définissent la pose de la caméra par rapport au damier.

Les résultats sont ensuite enregistrés dans un fichier JSON pour une utilisation ultérieure.

Notre algorithme fournit en sortie :

- Les paramètres intrinsèques de la caméra :  $f_x, f_y, c_x, c_y$ .
- Les paramètres extrinsèques `tvecs` et `rvecs`, ainsi que la transformation homogène  ${}^m\mathbf{T}_c$  caractérisant la pose de la mire dans le repère caméra pour chaque photo.

L'algorithme repose en partie sur la bibliothèque Python OpenCV et le code de calibration développé par [nikatsanka](#).

### 3.1 Résultat

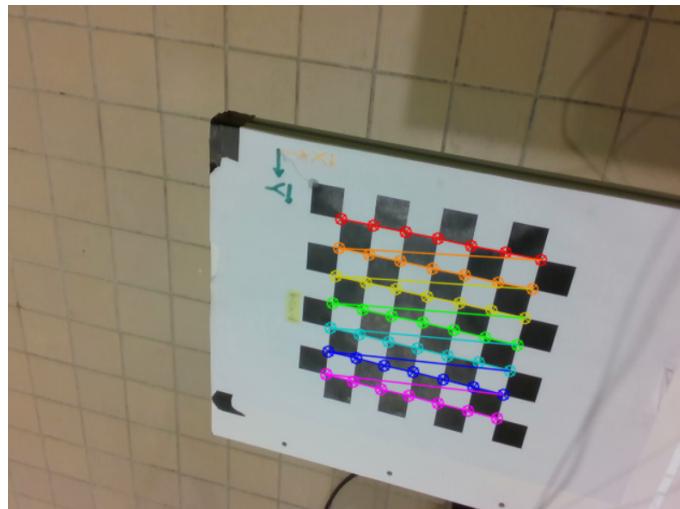


Figure 2: Exécution de l'algorithme

Les paramètres intrinsèques obtenus sont :

$$f_x = 601.0, \quad f_y = 601.1, \quad c_x = 323.6, \quad c_y = 246.5 \quad (1)$$

Ils sont cohérents avec ceux définis par le constructeur ( $f_x = 607.9, \quad f_y = 607.8, \quad c_x = 324.9, \quad c_y = 245.8$ ). On trouve un écart relatif de l'ordre de 1% à chaque fois. C'est satisfaisant compte tenu du jeu de données.

Pour la première image, nous obtenons les paramètres extrinsèques suivants :

Matrice de rotation  $R$  :

$$R = \begin{bmatrix} 0.9618 & -0.1013 & 0.2542 \\ 0.2099 & 0.8690 & -0.4480 \\ -0.1755 & 0.4843 & 0.8571 \end{bmatrix} \quad (2)$$

Vecteur de translation  $t$  :

$$t = \begin{bmatrix} -0.0247 \\ -0.0527 \\ 0.3882 \end{bmatrix} \quad (3)$$

L'analyse de la matrice indique que la caméra est principalement inclinée vers le bas avec une légère rotation autour des axes  $X$  et  $Y$ . Le vecteur montre que la caméra est située à environ 38 cm de l'origine, ce qui est cohérent avec la première image.

### 3.2 Vérification des résultats

Nous projetons le premier point calibré de la mire sur l'image pour évaluer la précision de la calibration :

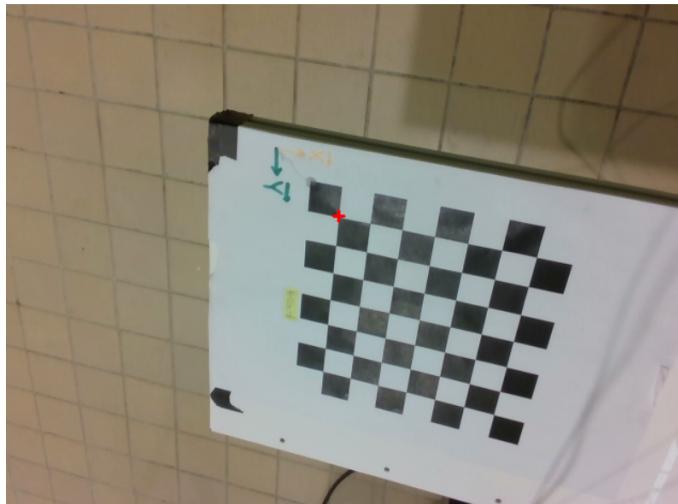


Figure 3: Projection de la mire (point rouge) sur l'image

Le point est correctement placé sur le damier et on obtient une erreur de reprojection de 0.31 pixels. L'erreur étant faible, la calibration est jugée satisfaisante.

### 3.3 Angles d'ouverture de la caméra

En utilisant la résolution de la caméra ( $640 \times 480$ ), nous calculons les angles d'ouverture horizontaux et verticaux :

$$a_x = 2 \times \tan^{-1} \left( \frac{640}{2f_x} \right) \quad (4)$$

$$a_y = 2 \times \tan^{-1} \left( \frac{480}{2f_y} \right) \quad (5)$$

Nous obtenons :

$$a_x = 56.07^\circ, \quad a_y = 43.53^\circ \quad (6)$$

Les valeurs sont conformes aux spécifications du constructeur ( $a_x = 55.52^\circ, \quad a_y = 43.09^\circ$ ). On trouve un écart relatif encore une fois de l'ordre de 1%, ce qui est satisfaisant.

## 4 Pose de la caméra par rapport à l'outil

L'objectif est d'identifier la transformation homogène caractérisant la pose de la caméra dans le repère de l'outil. Pour cela, nous utilisons la relation suivante :

$${}^oT_c = {}^oT_b \cdot {}^bT_m \cdot {}^mT_c \quad (7)$$

où :

- ${}^bT_o$  est la pose de l'outil dans le repère de la base.
- ${}^bT_m$  est la pose constante de la mire dans le repère de la base.
- ${}^mT_c$  est la pose de la caméra dans le repère de la mire, estimée par l'algorithme d'étalonnage.

Les composantes de translation et de rotation sont notées :

- $t_{x,y,z} = t_{x,y,z}({}^1T_2)$  (composantes de translation correspondantes à la transformation  ${}^1T_2$ )
- $\theta u_{x,y,z} = \theta u_{x,y,z}({}^1T_2)$  (composantes de rotation correspondantes à  ${}^1T_2$  en utilisant la représentation angle/axe)

On peut alors réécrire (1) sous la forme d'un système de n équations à 1 inconnue (les inconnues étant:  ${}^o t_{x,C}, {}^o t_{y,C}, \dots, {}^o \theta u_{z,C}$ ):

$$\begin{cases} {}^o t_{x,C} = t_x ({}^{O,1}T_B {}^B T_M {}^M T_{C,1}) \\ \vdots \\ {}^o t_{x,C} = t_x ({}^{O,n}T_B {}^B T_M {}^M T_{C,n}) \end{cases} \quad (8)$$

$$\begin{cases} {}^o t_{y,C} = t_y ({}^{O,1}T_B {}^B T_M {}^M T_{C,1}) \\ \vdots \\ {}^o t_{y,C} = t_y ({}^{O,n}T_B {}^B T_M {}^M T_{C,n}) \end{cases} \quad (9)$$

$$\vdots \quad (10)$$

$$\begin{cases} {}^o \theta u_{z,C} = \theta u_z ({}^{O,1}T_B {}^B T_M {}^M T_{C,1}) \\ \vdots \\ {}^o \theta u_{z,C} = \theta u_z ({}^{O,n}T_B {}^B T_M {}^M T_{C,n}) \end{cases} \quad (11)$$

En posant  $A = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$  et  $b = \begin{bmatrix} t_x({}^{O,1}T_B {}^B T_M {}^M T_{C,1}) \\ \vdots \\ t_x({}^{O,n}T_B {}^B T_M {}^M T_{C,1}) \end{bmatrix}$ , on a:  $A {}^o t_{x,C} = b$  Donc  $A^T A {}^o t_{x,C} = A^T b$

Or:  $A^T A = n$

D'où:  ${}^o t_{x,C} = (A^T A)^{-1} A^T b = \frac{1}{n} A^T b = \frac{1}{n} \sum_{i=1}^n t_x ({}^{O,i}T_B {}^B T_M {}^M T_{C,i})$

En procédant de la même manière pour les autres équations, on montre finalement que leur solution aux moindres carrés n'est rien d'autre qu'une moyenne :

$$\begin{aligned}
 {}^o t_{x,C} &= \sum_{i=1}^n t_x (O_{,B}^i \mathbf{T}_B \mathbf{T}_M^B \mathbf{T}_{C,i}^M) / n, \\
 {}^o t_{y,C} &= \sum_{i=1}^n t_y (O_{,B}^i \mathbf{T}_B \mathbf{T}_M^B \mathbf{T}_{C,i}^M) / n, \\
 {}^o t_{z,C} &= \sum_{i=1}^n t_z (O_{,B}^i \mathbf{T}_B \mathbf{T}_M^B \mathbf{T}_{C,i}^M) / n, \\
 {}^o \theta_{u_x,C} &= \sum_{i=1}^n \theta_{u_x} (O_{,B}^i \mathbf{T}_B \mathbf{T}_M^B \mathbf{T}_{C,i}^M) / n, \\
 {}^o \theta_{u_y,C} &= \sum_{i=1}^n \theta_{u_y} (O_{,B}^i \mathbf{T}_B \mathbf{T}_M^B \mathbf{T}_{C,i}^M) / n, \\
 {}^o \theta_{u_z,C} &= \sum_{i=1}^n \theta_{u_z} (O_{,B}^i \mathbf{T}_B \mathbf{T}_M^B \mathbf{T}_{C,i}^M) / n.
 \end{aligned} \tag{12}$$

#### 4.1 Transformation homogène

La matrice obtenue est la suivante :

$${}^o T_c = \begin{bmatrix} 0.04343948 & 0.9813241 & 0.04829632 & 0.0447773 \\ -0.99696272 & 0.04472324 & -0.04441432 & 0.0644107 \\ -0.04440434 & -0.04664689 & 0.98079758 & -0.13349765 \\ 0. & 0. & 0. & 1. \end{bmatrix} \tag{13}$$

Le déterminant de  ${}^o T_c$  est :

$$\det({}^o T_c) = 0.9656497348257136 \tag{14}$$

La matrice d'orthogonalité obtenue est :

$$\text{ortho} = \begin{bmatrix} 0.96922152 & 0.00131954 & -0.00631337 & 0.0447773 \\ 0.00131954 & 1.00205621 & -0.00997687 & 0.0644107 \\ -0.00631337 & -0.00997687 & 0.98393319 & -0.13349765 \\ 0.0447773 & 0.0644107 & -0.13349765 & 1. \end{bmatrix} \tag{15}$$

L'analyse des résultats montre que :

- La caméra est alignée en Z avec le repère de l'outil. (le bloc de translation de  ${}^o T_C$  correspond aux valeurs mesurées expérimentalement à la règle)
- Elle présente une rotation de 90° selon l'axe z par rapport à la pose de l'outil. (axe bleu sur la figure 1)
- Le déterminant de la matrice est proche de 1, garantissant que  $R$  est bien orthogonale.

#### 4.2 Vérification des résultats

Nous projetons les points de la mire sur l'image en passant par la chaîne robotique et obtenons les erreurs de reprojection suivantes :

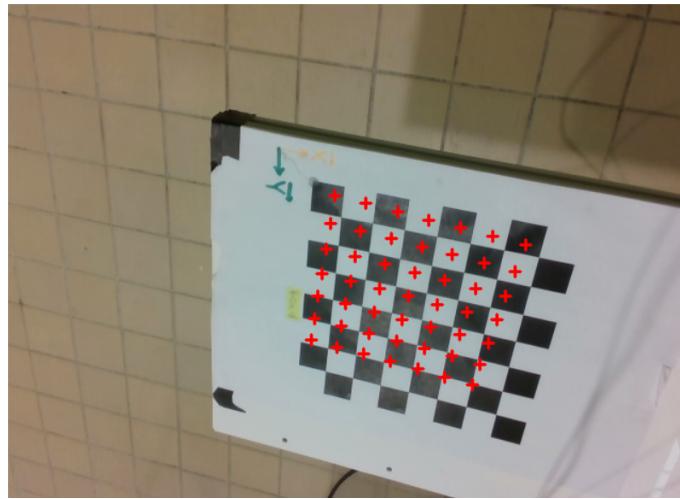


Figure 4: Reprojection des points de la mire

### 4.3 Résultat

L'erreur de reprojection est la suivante :

- Erreur de reprojection moyenne : 25.42 pixels
- Erreur maximale : 35.39 pixels
- Erreur minimale : 16.41 pixels

L'écart-type des erreurs est donné par la matrice :

$$\begin{bmatrix} 0.02460344 & 0.05352457 & 0.17121121 & 0.04918119 \\ 0.0013215 & 0.02299361 & 0.03952314 & 0.0151585 \\ 0.03999403 & 0.17156143 & 0.05343762 & 0.0457826 \\ 0. & 0. & 0. & 0. \end{bmatrix} \quad (16)$$

En utilisant la matrice de transformation  ${}^oT_c$  globale pour la projection, on obtient une erreur de reprojection moyenne relativement élevée. Cela peut s'expliquer par notre petit nombre d'échantillons d'image qui a pu augmenter les incertitudes sur les valeurs de  ${}^oT_c$ .

## 5 Reconnaissances des trous dans l'image.

L'objectif de cette partie est de développer un algorithme de traitement d'images permettant de localiser la position du trou et d'estimer sa surface dans l'image. Nous avons testé notre programme python avec le flux continu d'images acquis en section 2.

À chaque itération, l'algorithme a en entrée une photo et renvoie les coordonnées images du barycentre du trou ainsi que sa surface :  $u$ ,  $v$ ,  $a$  (trois scalaires, exprimés en pixels, voir figure 5).

L'algorithme continue de tourner jusqu'à ce qu'il estime que l'organe terminal est suffisamment proche du trou. En effet, à partir de ce moment-là, on bascule sur un simple contrôle en boucle fermée et on se passe de la vision.

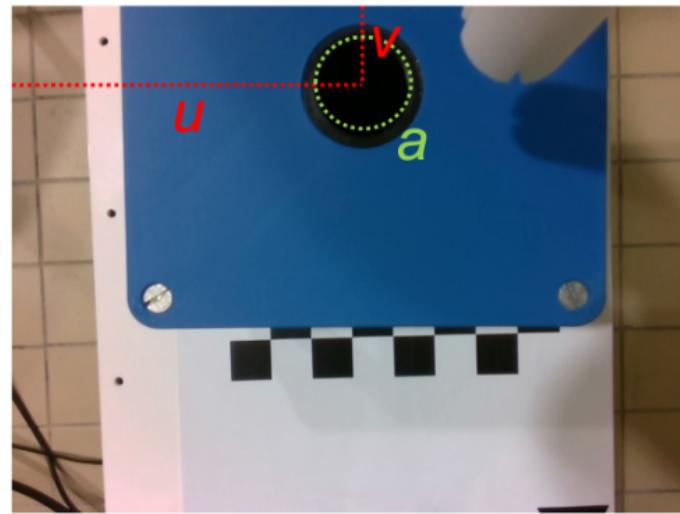


Figure 5: Image du trou acquise par la caméra - (u, v): coordonnées images du barycentre du trou — a: surface

### 5.1 Fonctionnement de l'algorithme

Nous avons considéré le trou comme un blob noir entouré d'un blob bleu. On commence par convertir l'image du format RGB vers le format HSV, puis on définit un intervalle de confiance pour détecter le blob bleu. On rogne ensuite l'image selon ce blob bleu afin de limiter l'analyse à la zone d'intérêt.

Sur cette image rognée, nous détectons ensuite le blob noir en appliquant un seuillage en niveaux de gris, suivi d'une recherche des contours. N.B. La tolérance pour le seuillage a été déterminée expérimentalement.

Parmi les contours détectés (a priori il n'y en a qu'un), nous sélectionnons le plus grand, qui correspond au trou. Nous en extrayons alors le centroïde par calcul des moments qui renvoie le barycentre du contour détecté. Cependant, le contour n'étant pas vraiment circulaire, nous traçons un cercle épousant au mieux le contour en utilisant la méthode des moindres carrés pour obtenir un ajustement optimal. On peut alors comparer les deux centres obtenus. Par la suite, nous n'utiliserons malgré tout que le centre du cercle, plus fidèle à la réalité. On veille ensuite à re-projecter les coordonnées détectées dans l'image rognée vers l'image d'origine.

Enfin, nous avons calculé l'aire du trou sur l'image où l'organe terminal est sur le point de chevaucher le trou. Elle nous sert de référence car au-delà, on passe en contrôle sans vision. Alors, dès que l'aire calculée pour l'image en cours de traitement est suffisamment proche de l'aire de référence, on arrête le processus.

Pour réaliser ce programme, nous nous sommes inspirés de ce site :

<https://learnopencv.com/find-center-of/blob-centroid-using-opencv-cpp-python/>

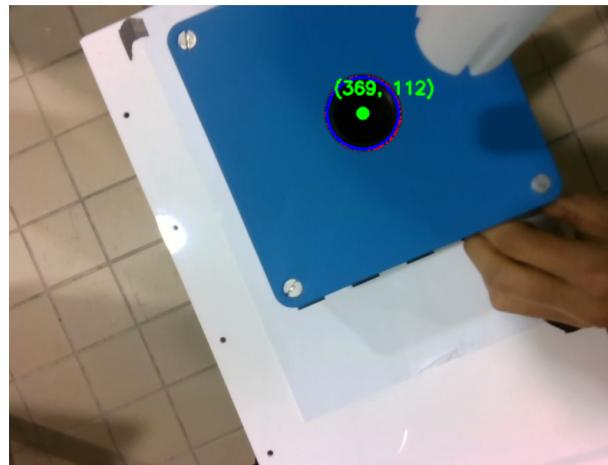


Figure 6: Image du trou après traitement d'image. Contour détecté (en rouge), coordonnées du centre (en vert), cercle optimisé (en bleu). L'aire de référence obtenue est  $a = 4300 \text{ px}^2$ . Ces coordonnées sont: (u,v) = (369,112)

## 6 Commande pour réaliser l'insertion.

### 6.1 Loi de commande hybride vision/force

L'objectif de cette partie est de développer une loi de commande permettant au robot d'insérer l'outil dans le trou. Cette loi de commande a été validée en deux étapes:

1. en simulation (aussi bien du robot que des images) dans Gazebo,
2. puis sur le robot réel

Dans l'hypothèse – réaliste – que la plaque du trou est horizontale (parallèle au sol), et que l'outil démarre toujours de la position verticale illustrée en figure 1) on viendra contrôler uniquement les translations.

La loi d'impédance utilisée sur le robot est décrite par la consigne en couple suivante:

$$\tau_d = \mathbf{J}^T \left[ \mathbf{K}^B \mathbf{V}_O - \mathbf{D} \dot{\mathbf{X}} \right] + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) \quad (17)$$

où  ${}^B v_O$  est la consigne, en vitesse, du déplacement du robot. Cette variable est utilisée pour déplacer le robot en fonction de la détection du trou.

Pour communiquer la consigne  ${}^B v_O$  au robot (en simulation et aussi en réel), nous avons utilisé le topic `/cart_vel_des`. On souhaite:

$$\dot{s} = \mathbf{L}_s^C \mathbf{v}_C = -\Lambda e \quad (18)$$

Avec uniquement les trois vitesses de translation. Donc:

$${}^C \mathbf{v}_C = -\mathbf{L}_s^{-1} \Lambda e \quad (19)$$

qui peut se réécrire dans le repère base comme:

$${}^B \mathbf{v}_C = {}^B \mathbf{R}_C {}^C \mathbf{v}_C \quad (20)$$

La vitesse correspondante de l'outil sera:

$${}^B \mathbf{v}_O = {}^B \mathbf{v}_C \quad (21)$$

Note: L'expression de  $\mathbf{L}_s$  est:

$$\mathbf{L}_s = \begin{bmatrix} -1/Z & 0 & x/Z \\ 0 & -1/Z & y/Z \end{bmatrix} \quad \text{avec} \quad x = (u - u_0)/f_x, \quad y = (v - v_0)/f_y, \quad Z = 1 \quad (22)$$

### 6.2 Démarche d'insertion

L'insertion dans le trou s'effectue en plusieurs étapes. Nous utilisons le noeud `integ_visual_servoing_node.py` du package `franka_integ_node` pour cette dernière étape. Dans ce noeud, nous reprenons l'algorithme développé en section 5 et publions les consignes de  ${}^B \mathbf{v}_O$  comme décrites en section 6.1.

1. Tout d'abord, on déplace le robot dans le plan horizontal (seulement en XY) pour qu'il suive les mouvements du trou, jusqu'à ce qu'il soit aligné avec celui-ci. Cette phase s'appuie grandement sur notre algorithme de vision décrit en section 5.
2. Une fois le premier alignement effectué, on ajoute un déplacement vertical pour que l'air détecté du trou atteigne une valeur prédéterminé, tout en continuant de bouger dans le plan XY pour rester aligné sur le centre du trou.
3. Une fois l'aire prédéterminée atteinte, le mouvement du robot se fait uniquement selon l'axe Z (vertical) pour insérer l'outil dans le trou. On n'utilise plus la vision à cette étape là.
4. Enfin, on mesure la force de contact pendant la phase de descente et on arrête le mouvement d'approche du bras dès que celle-ci dépasse 5 N.m selon l'axe z de l'outil. Cette force peut être mesurée à travers le topic `/franka_states_controllers/F_ext`.

## 6.3 Résultat

### 6.3.1 En simulation

Nous avons testé notre algorithme sur le node `integ_visual_servoing.py` du package `franka_integ_node`. Il a fallu légèrement adapter notre programme de vision car le blob bleu entourant le trou n'avait pas le même bleu que sur les photos. Nous avons donc élargi les bornes de détection de bleu. De plus, l'effecteur apparaissant noir dans la simulation, nous avons rajouté une source de lumière pour éviter toute perturbation avec l'algorithme de vision.

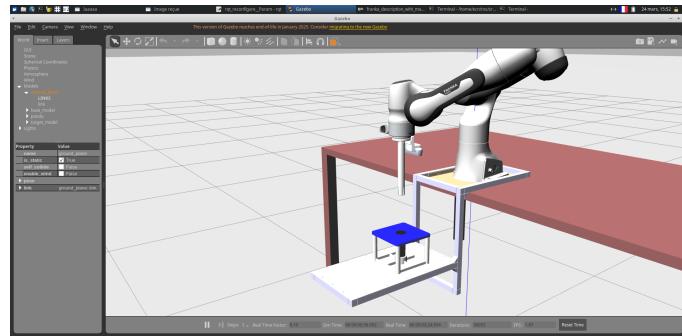


Figure 7: Simulation sur Gazebo: le robot en phase d'insertion

### 6.3.2 En réel

Nous avons ensuite testé notre programme en condition réelle. Pour des raisons qu'on ignore, l'exécution du programme ne fonctionnait qu'au bout de plusieurs essais infructueux sans pour autant avoir apporté la moindre modification à notre programme. Lorsque le programme voulait bien se lancer, on pouvait voir le robot se déplacer et s'aligner avec le trou. Cependant, les conditions en simulation n'étant pas les mêmes, nous avons dû ajuster les raideurs du robot et les vitesses de déplacement. Il était également surprenant de voir que ces ajustements n'était pas les mêmes d'une machine virtuelle à l'autre.

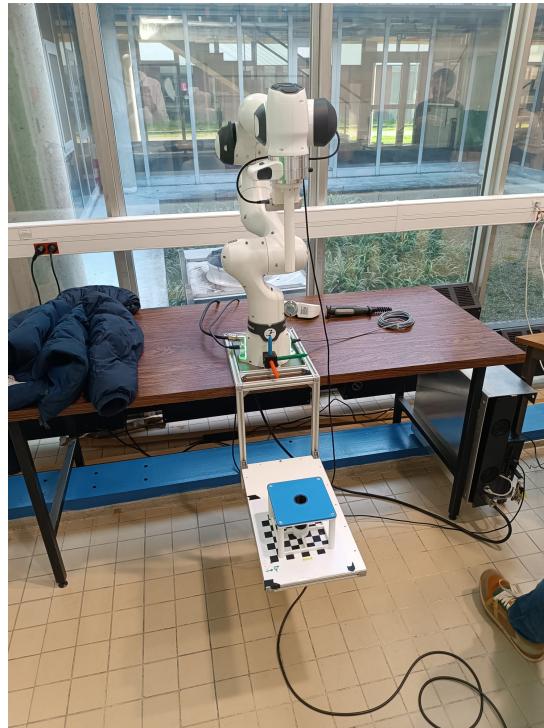


Figure 8: Le robot en train de s'insérer

## 7 Conclusion

Ce projet nous a permis de mettre en œuvre une loi de commande hybride vision/force afin de réaliser une tâche d'insertion (peg-in-hole) avec le robot Franka Emika FR3. Nous avons abordé différentes étapes essentielles, de l'acquisition et l'étalonnage des images à l'identification de la pose de la caméra et au développement d'un algorithme de reconnaissance du trou d'insertion.

L'étalonnage de la caméra a fourni des résultats cohérents avec les spécifications constructeur, validant ainsi la précision des mesures. L'identification de la pose de la caméra dans le repère de l'outil a nécessité l'exploitation de plusieurs transformations homogènes, démontrant l'importance d'une bonne modélisation géométrique du système robot/caméra. Enfin, la mise en œuvre d'une loi de commande hybride a permis d'exploiter conjointement les informations de vision et de force pour améliorer la précision et la robustesse de la tâche d'insertion.

Malgré certaines contraintes, notamment liées à l'acquisition des données et aux ressources matérielles disponibles, les résultats obtenus sont satisfaisants et valident la méthodologie adoptée.