# HAE809E

Sujet #36 :

Développement d'un driver Kinect sous ROS2

MARTELET Curtis
(curtis.martelet@etu.umontpellier.fr)

EL-HAGE Andrew
(andrew.el-hage@etu.umontpellier.fr)

*En collaboration avec le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier*

***Abstract* — This document presents the report of the Master's 1 project aimed at developing a Kinect driver. The Kinect camera is an image acquisition device commonly used in robotics. The objective of this project is to develop a Kinect driver in C++ language, which controls a first calibration step and a second step of recording RGBD images at the required frequency, while familiarizing with Kinect usage, the differences between ROS 1 and ROS 2, and understanding the advantages and disadvantages of this type of camera in terms of image processing.**

***Keywords* — Red Green Blue Depth image (RGBD image), Robot Operating System (ROS), Calibration, Image recording**

## I. Introduction

Dans le domaine en pleine expansion de l'interaction homme-robot (H-R), une amélioration de la collaboration entre les humains et les robots est visée, en particulier dans les scènes industrielles.

L'objectif principal de ce projet est le développement d'un driver pour une caméra, qui a pour rôle de manipuler la caméra au service de l'enregistrement des images des mouvements d'une personne pour une analyser. Grâce à cette analyse détaillée, un robot sera en mesure de déterminer la manière la plus efficace d'interagir avec la personne et de lui apporter son assistance.

Pour minimiser toute perturbation ou impact sur le travail de la personne observée, une caméra externe a été choisie pour l'observation. Ce choix a mené à l'adoption de la technologie Kinect 2.

Cette caméra RGBD, développée par Microsoft, a été choisie en raison de sa capacité à fournir des informations 3D précises, essentielles pour l'interaction homme-robot. En effet, le robot doit naviguer et interagir dans un environnement humain, qui est intrinsèquement tridimensionnel.

Ce projet fait une connexion directe avec l'Industrie 4.0. Caractérisée comme la quatrième révolution industrielle, l'Industrie 4.0 est marquée par l'interconnexion des systèmes de production grâce à la numérisation et à l'Internet des Objets (IoT). La procédure de fabrication moderne en industrie 4.0 intègre de plus en plus des scénarios de coopération homme-robot, ceci fait appel à la nécessité d'équiper le robot avec une observabilité en 3d afin de s'intégrer dans la scène de production. Le développement des technologies d'observation en 3d se place directement dans le cadre de l'évolution de l'Industrie 4.0. [1]

Iai_kinect2, une suite d'outils et de bibliothèques programmé avec ROS conçu pour l'interaction avec la Kinect One (Kinect v2), est au cœur de ce projet. Utilisant le pilote Libfreenect2 pour interagir avec la Kinect, il est nécessaire pour construire un pont entre la caméra et notre ordinateur.

Iai_kinect2 comprend un outil de calibration nécessaire pour aligner le capteur IR avec le capteur RGB, et est également doté d'un afficheur qui permet de visualiser l'image en couleur ainsi que la profondeur sous forme de nuages de points. [2]

Un projet notable est un wrapper de iai_kinect2 par Yu Li et ses collaborateurs. Ce dernier propose une intégration de cet outil dans ROS 2.

Notre projet constitue en une migration de l'interface ROS pour Kinect One (Kinect v2) de ROS1 vers ROS2, basée sur la version OpenCV4. [3]

Il est à noter que, malgré l'intérêt pour ROS2 et les avancées réalisées par des projets similaires, notre travail a dû rester sur ROS1 en raison de l'incompatibilité de libfreenect2 avec ROS2. Cette contrainte technique a souligné l'importance de l'adaptabilité et de la résilience dans la recherche et le développement technologique.

## II. Objectifs

Dans le cadre de ce projet, plusieurs objectifs ont été définis pour garantir une interaction efficace avec la Kinect. Ces objectifs sont les suivants :

- Effectuer la calibration de la caméra Kinect
- Procéder à l'enregistrement d'une vidéo à une fréquence spécifique (vidéo RGB et vidéo IR). Enregistrer également un gif si possible.
- Evaluer les performances de l'enregistrement.

## III. L'Etat de l'Art

### A. *L'intérêt du passage de ROS1 à ROS2*

Un des buts principaux de ce projet était d'essayer de passer de ROS1 à ROS2. Les motivations de cette transition sont explorées dans cette partie.

En effet, le contrôle du robot de type manipulateur (robot mobile), spécifiquement le robot de type BAZAR utilisé dans cette étude, est assuré par un ensemble de logiciels développés sous ROS 2. Par conséquent, une meilleure intégration de la Kinect avec le robot est envisagée en passant à ROS 2.

De plus, la manipulation de la caméra Kinect est assurée par iai et libfreenect2, des librairies qui ont été développées sous ROS 1 à une époque où l'utilisation de ROS 2 n'était pas courante. Cette constatation justifie la nécessité de passer à ROS 2 pour une meilleure intégration avec les technologies actuelles.

Pour conclure, il est important de noter que ROS 1 et ROS 2 sont construits sur le même principe de système d'exploitation et partagent une architecture similaire. Cette similitude structurelle facilite la transition de ROS 1 à ROS 2.

Le tableau comparatif qui suit (Tableau 1) permet de comprendre pourquoi ROS 2 se révèle être une option plus avantageuse.

*Tableau 1 : Comparaison entre ROS 1 et ROS 2 [4]*

| Aspect | ROS1 | ROS2 |
|---|---|---|
| Architecture | Architecture monolithique avec un maître ROS central. | Architecture distribuée sans maître ROS central. |
| Langages de programmation | Prend principalement en charge Python et C++. | Prend en charge Python, C++ et d'autres (par exemple, Rust, Java). |
| Capacités en temps réel | Capacités en temps réel limitées. | Capacités en temps réel améliorées. |
| Sécurité | Manque de fonctionnalités de sécurité. | Met l'accent sur la sécurité (par exemple, les plugins de sécurité « Data Distribution Service » - DDS). |

*B. La Kinect 2*

La Kinect 2 est une caméra RGBD développée par Microsoft qui comprend une caméra couleur et une caméra IR. Le capteur IR est composé d'un émetteur qui projette des points invisibles, similaires à des pointeurs laser, dans l'environnement, et d'un capteur qui mesure le temps qu'il faut pour que la lumière émise soit réfléchie.
Cette technologie, nommée « Time of Flight » - ToF, permet à la Kinect 2 de déterminer des données de profondeur et de mouvement, ce qui en fait un outil puissant pour la vision par ordinateur, la détection de mouvement et la cartographie 3D. [5]

Voici les spécifications techniques principales des caméras de la Kinect 2 :

- Caméra de profondeur (IR) :
  - Résolution : 512x424 pixels
  - FPS max : 30 images par seconde
  - Portée : de 0.5 à 4.5 mètres

- Caméra couleur (RGB) :
  - Résolution : 1920x1080 pixels
  - FPS max : 30 images par seconde
  - Portée : de 0.5 à 4.5 mètres [6]

En effet, la différence de résolution entre la caméra de profondeur et la caméra couleur de la Kinect 2 est notable. C'est pourquoi la calibration est essentielle pour aligner correctement les informations provenant de ces deux sources.

*C. Iai_kinect2*

Iai_kinect2 est une collection d'outils et de bibliothèques pour une interface ROS avec la caméra Kinect 2.

Voici la liste des fonctionnalités qu'elle contient :

- **Outil de calibration (kinect2_calibration)**

Permet de calibrer le capteur infrarouge (IR) de la Kinect par rapport au capteur couleur (RGB).

- **Bibliothèque de recalage de profondeur (kinect2_registration)**

Prend en charge le réalignement des données de profondeur (en utilisant OpenCL pour accélérer le traitement).

- **Pont entre libfreenect2 et ROS (kinect2_bridge)**

Permet de commander et de recevoir les données de la caméra dans un format utile via ROS.
Utilise libfreenect2 pour la communication avec la Kinect.

- **Visionneur d'images et du nuage de points (kinect2_viewer)**

Affiche les images et les nuages de points capturés par la Kinect.

Pour faire fonctionner iai_kinect2, les étapes suivantes doivent être suivies :

- S'assurer que ROS 1 est installé sur le système
- Cloner le dépôt iai_kinect2 dans l'espace de travail ROS
- Compiler le package
- Exécuter les nœuds appropriés pour recevoir les données de la Kinect. [2]

*D. Erreurs rencontrées*

Dans cette section, nous allons développer les erreurs et difficultés que nous avons rencontrées tout au long du projet. L'objectif principal est de documenter ces problèmes afin d'éviter que d'autres personnes voulant reproduire ce projet ne perdent du temps à les reproduire. En partageant nos expériences, l'accent est mis sur la facilitation du processus pour les futurs utilisateurs et la contribution à l'amélioration du travail.

- **Problèmes spécifiques à l'installation initiale :**

- Après avoir suivi le tutoriel de Nvidia pour installer CUDA, nous avons découvert qu'un des ordinateurs à un GPU non compatible avec l'accélération matérielle, ce qui a nécessité une réinstallation d'Ubuntu. En raison de cette incompatibilité, nous avons été forcés d'utiliser le CPU, limitant la capture des caméras à 1 FPS.
- L'accès aux ports USB pour Kinect2_bridge nécessitait des permissions d'administration.
- Des problèmes sont survenus lors de la compilation des programmes, mais une réinstallation a résolu ces problèmes après modification de la version de CMake.
- L'installation de Conda s'est avérée nécessaire pour pouvoir utiliser ROS sur un ordinateur équipée d'une version de Ubuntu dépassé. Cependant, cette dernière a échoué et nous avons dû installer.
- La Kinect n'a pas pu établir de connexion avec la machine virtuelle dû à l'échec de communication du port USB sur la passerelle entre la machine-hôte et la VM, malgré l'installation des extensions nécessaires.

- **Libfreenect2**

- Étant incompatible avec ROS2, nous avons été forcés de réaliser notre projet sur ROS1.
- Le tutoriel d'installation de libfreenect2 sur la page GitHub d'IAI est incorrect. Nous avons suivi le guide présent sur le dépôt de Libfreenect2.
- Similaire aux problèmes sur VM, l'installation sur Docker rencontre aussi un problème d'accès restreint aux ports USB de l'ordinateur.

- **IAI**

- Iai a été développé dans la version 2.0 d'OpenCV. Cette dernière n'est plus disponible en téléchargement tandis que les versions récentes d'Ubuntu sont équipées d'OpenCV 4.0.
- La documentation de iai_kinect2 est peu fourni. Nous avons eu du mal à trouver les arguments nécessaires pour exécuter les nodes.

- **Problèmes spécifiques à l'ordinateur utilisé**

- Des paramètres obtenus suite à l'étape de calibration se sont révélés incorrects, ce qui ne fut pas le cas avec un ordinateur plus puissant.
- OpenCV ne disposait pas d'un writer GIF.
- Évitez d'utiliser un ordinateur trop récent, car les ports USB 4.0 ne sont pas compatibles avec la Kinect, qui nécessite un port USB 3.0 ou 3.1.

## IV. Calibration

### A. *Introduction*

À la suite de l'installation de l'environnement de travail, la première étape du projet est la calibration de la caméra.

La Kinect 2, n'ayant pas été conçue pour la recherche scientifique, présente des caractéristiques qui limitent son utilisation dans des environnements où la précision est de rigueur.
L'une de ces caractéristiques est la présence de distorsions optiques dans les images qu'elle produit. Ces distorsions sont principalement dues aux lentilles et à d'autres composants optiques de la caméra. En conséquence, les images capturées par la Kinect peuvent ne pas être géométriquement précises et peuvent présenter des déformations.

Un package est disponible dans iai pour réaliser la calibration des caméras : *kinect2_calibration*. Cette dernière utilise les algorithmes de calibration de caméra de OpenCV pour déterminer les matrices de transformations et les paramètres intrinsèques des caméras, ce qui permet de corriger les distorsions optiques et d'obtenir des images rectifiées et calibrées.

### B. *OpenCV*

OpenCV (pour Open Source Computer Vision Library) est une bibliothèque open source pour le traitement d'images et la vision par ordinateur. Elle comprend une multitude de fonctionnalités et d'algorithmes qui permettent de réaliser des tâches telles que la détection d'objets, la reconnaissance faciale, le suivi d'objets, la calibration de caméra, et bien plus encore.

La calibration de caméra par OpenCV vise à estimer les paramètres intrinsèques et extrinsèques de la caméra, tels que la matrice de caméra, les coefficients de distorsion, la position et l'orientation de la caméra par rapport à la scène.
Ce processus est initié par la capture d'un ensemble d'images d'un motif bien défini, tel qu'une grille de calibration ou un damier, dans différentes positions et orientations. Avec ces images, des techniques de détection de coins sont utilisées pour extraire les points caractéristiques du motif (dans le cas d'un damier, les coins des cases), puis des algorithmes d'optimisation sont appliqués pour estimer les paramètres de la caméra qui minimisent les erreurs de projection entre les points mesurés dans l'image, et les positions réelles du motif dans l'espace 3D.
Une fois la calibration terminée, les paramètres de la caméra peuvent être utilisés pour rectifier les distorsions et pour estimer avec précision les propriétés géométriques des objets dans les images capturées par la caméra.

Pour faire simple, OpenCV va tracer des lignes entre tous les points sur l'axe horizontal et vertical. Si ces lignes ne se superposent pas, alors il y a distorsions et OpenCV va calculer les coefficients. [7][8]

### C. *Méthodologie utilisée lors l'étape de Calibration*

Nous avons suivi la méthode présente sur le dépôt de iai pour réaliser la calibration de la caméra.
Du fait de la présence de deux caméras sur la Kinect, trois étapes de calibration sont nécessaires :

- La calibration de la caméra en couleur ;
- La calibration de la caméra infrarouge ;
- La calibration des deux caméras simultanément.

La calibration avec iai implique la capture d'images à l'aide de la Kinect, où le damier fourni est positionné.
Ce damier a été déplacé à différents endroits dans le cadre de l'image, avec différentes orientations et distances.

La luminosité de la pièce et la réflexion du papier peuvent jouer un rôle dans la reconnaissance du motif du damier par la caméra. C'est pourquoi une pièce sombre a été choisie avec une feuille de papier classique imprimée à l'aide d'une imprimante laser pour réaliser la calibration. [9]

## V. Implémentation du Driver

### A. *Explication de Kinect2_record*

La seconde partie du projet implique l'enregistrement d'images provenant de la Kinect. Pour simplifier, un package a été ajouté au projet iai_kinect2 : ce package doit être en mesure d'enregistrer les flux vidéo Color, Depth et IR pendant une durée et une fréquence spécifiées, puis d'enregistrer ces vidéos à un emplacement et dans un format spécifié lors du lancement du programme développé dans un rosnode.

Cette méthode présente l'avantage que les images capturées par la Kinect sont déjà corrigées et traitées par la node Registration de iai.

L'inconvénient de cette méthode est la nécessité de comprendre les fonctions et spécificités de iai_kinect2, ce qui s'est avéré compliqué car le programme n'est pas commenté et les README ne sont pas très fournis.

Notre contribution à iai se situe dans le fichier kinect2_record.cpp qui contient la node kinect2_record.
La description et l'explication de ses fonctions sont présentées dans les parties ci-dessous.

### B. *La fonction main*

La fonction main, en plus d'initialiser et de lancer la node, récupère les valeurs des paramètres (durée, fréquence, chemin et format de l'enregistrement) définis en argument lors du lancement de la node. Ces paramètres ne sont pas définis comme étant des paramètres ROS, ce qui simplifie l'écriture du programme de la node en évitant l'utilisation d'un fichier launch. Des valeurs par défaut sont définies : une fréquence de 20fps, une durée de 5s, un format .mp4 et le chemin de sortie est le dossier "Pictures" de Linux.

### C. *Initialisation de la Node*

Pour enregistrer les images couleurs, de profondeurs et infrarouges, la node s'abonne à trois topics :

- **/kinect2/hd/image_color_rect** pour les images couleur ;
- **/kinect2/hd/image_depth_rect** pour les relevés de profondeurs ;
- **/kinect2/hd/image_depth_rect** pour une superposition de la couleur et de la profondeur.

Nous avons choisi une taille de file (d'attente) égale à 1 pour que l'on enregistre toujours la dernière image reçue d'un topic.

Cette initialisation vérifie que les arguments déclarés ci-dessus sont possibles : par exemple, la fréquence de capture de iai est limitée à 30fps. De plus, un compteur est initialisé pour être utilisé comme outil de vérification lors de la phase de traitement.
Enfin, un flag est levé pour indiquer que le programme peut lancer l'enregistrement des images.

### D. *La fonction Run*

La fonction run joue le rôle de la fonction main de notre classe.

On enregistre le temps au moment où cette fonction est lancée : nous en aurons besoin pour calculer le temps écoulé.
Avant d'entrer dans la boucle qui sera exécutée pendant toute la durée de l'enregistrement, la fonction *ros Rate* est utilisée pour définir la fréquence d'exécution du programme.
Dans la boucle, le temps écoulé est calculé :
Si le temps n'est pas écoulé, tous les callbacks des *subscribers* sont exécutés. Si le temps est écoulé, le flag d'enregistrement est désactivé et l'on passe au traitement des images reçues.
Une fois le traitement terminé, la mémoire est libérée et la node est fermée.

### *E. Les Callbacks*

Il y a trois fonctions callbacks, une pour chaque topic. Ces fonctions ont pour seul but d'enregistrer les images reçues dans un vecteur d'images de type cv::Mat (le format par défaut de OpenCV).
Les compteurs sont incrémentés lorsqu'un type d'image correspondant au compteur est enregistrée. La seule différence entre ces trois fonctions est l'encodage d'image : les images en couleur utilisent un encodage 8 bits tandis que l'image en profondeur utilise un encodage 16 bits.

### *F. Le Traitement*

La fonction Traitement réalise toutes les étapes de traitement d'images nécessaires :
CorrectChannel pour inverser le canal rouge et bleu des images en couleur, *ArrayToVid* pour convertir le vecteur d'images en une vidéo, et *DumpImage* pour enregistrer chaque image du vecteur dans un dossier.

### *G. CorrectChannel*

Cette fonction effectue une inversion des canaux rouge et bleu des images en couleur.
Une observation a été faite sur les photos prises : les couleurs bleue et rouge étaient inversées. Il est supposé qu'il s'agît d'une erreur dans le code de iai, étant donné que d'autres programmes utilisant la caméra que nous avons testés affichaient correctement les images.

CorrectChannel est une fonction de traitement d'image qui prend un vecteur d'images en entrée et inverse les canaux rouge et bleu à l'aide de la fonction *cvtColor* de la librairie OpenCV. Cette fonction nécessite l'image à traiter, une variable pour stocker l'image traitée et l'opération à réaliser (dans notre cas, COLOR_BGR2RGB (BGR -> RGB)).

### *H. ArrayToVid*

Cette fonction est utilisée pour écrire une vidéo à partir d'un vecteur d'images. Elle permet de changer le format de la vidéo écrite par kinect2_record.

ArrayToVid est une fonction qui prend en entrée un vecteur d'images et un format. Cette fonction utilise *videowriter*, une fonction de OpenCV qui permet de réaliser une vidéo en fournissant un vecteur d'images, un nom de fichier avec son extension correspondant au FOURCC, un code FOURCC, le nombre d'images par seconde de la vidéo et la taille des images. Le code FOURCC est un code de quatre caractères qui identifie le codec vidéo utilisé. Pour ce projet, "**mp4v**" a été utilisée pour réaliser un fichier .mp4 et "**MJPG**" pour .avi. Une structure conditionnelle pour choisir quel code FOURCC utiliser a été implémenté. (À noter qu'on a remarqué que *videowriter* ne prend pas en charge les images 16 bits.)

### *I. DumpImage*

Cette fonction est utilisée pour enregistrer toutes les images reçues des topics dans un dossier.

DumpImage est une fonction qui prend en entrée un vecteur d'images. En utilisant le chemin spécifié, cette fonction crée un dossier correspondant au topic du vecteur d'images (color pour les images couleur, etc.). Une fois le dossier créé (s'il n'existait pas déjà), les images sont enregistrées au format .png à l'aide de la fonction *imwrite* de la librairie OpenCV.

## VI. Résultats

### *A. Résultats de la calibration*

Les résultats de la calibration de la caméra RGB de la Kinect sont présentés ci-dessous, comprenant les paramètres intrinsèques (matrice de la caméra et coefficients de distorsion) ainsi que les paramètres extrinsèques (rotation et projection).

- **Matrice de la caméra**

$$\begin{bmatrix} 3.1e^3 & 0. & 2.1e^3 \\ 0. & 9.9e^2 & 5.4e^2 \\ 0. & 0. & 1. \end{bmatrix}$$

Cette matrice (3x3) représente les paramètres intrinsèques de la caméra, tels que la mise à l'échelle des axes x et y, ainsi que les coordonnées du point principal. Elle est essentielle pour la conversion des coordonnées 3D en coordonnées 2D dans l'image.

- **Coefficients de distorsion**

$$\begin{bmatrix} 1.7e^{-2} & -2.1e^{-1} & -1.4e^{-3} & -4.7e^{-1} & 1.5e^{-1} \end{bmatrix}$$

Ces coefficients (1x5) décrivent les distorsions radiales et tangentes de la lentille de la caméra, permettant de compenser les distorsions géométriques présentes dans les images capturées.

- **Rotation**

$$\begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0. & 0. & 1. \end{bmatrix}$$

Cette matrice (3x3) représente la rotation de la caméra par rapport à son orientation initiale. Elle est importante pour aligner les images avec le repère de référence.

- **Projection**

$$\begin{bmatrix} 3.1e^3 & 0. & 2.13e^3 & 0. \\ 0. & 9.9e^2 & 5.4e^2 & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

Cette matrice (4x4) est utilisée pour projeter les points 3D de l'espace de la scène sur le plan image 2D de la caméra, prenant en compte à la fois les paramètres intrinsèques et extrinsèques.

De même, les résultats de la calibration de la caméra infrarouge de la Kinect ont été calculés.

- **Décalage de profondeur**

$$-6.7e^1$$

Cette valeur calculée par OpenCV représente le décalage de profondeur pour la caméra de profondeur de la Kinect. Elle est utilisée pour ajuster les valeurs de profondeur capturées par la caméra afin de les aligner avec les valeurs de profondeur réelles.

Enfin, les résultats de la calibration sous forme de graphes sont présents ci-dessous :

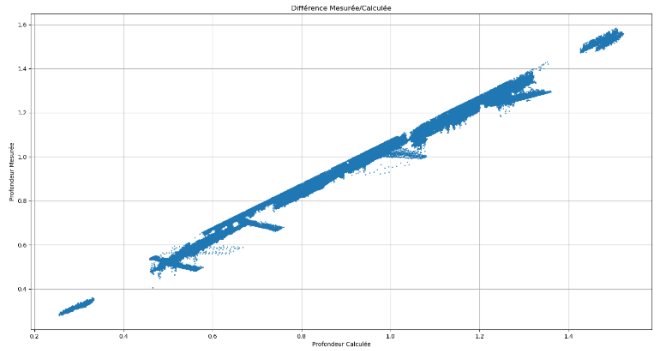


*Figure 1: Différence entre la profondeur mesurée et la profondeur calculée*

Dans cette première figure, un graphique met en relation la profondeur mesurée par la caméra par rapport à la profondeur calculée par iai.
On observe un comportement linéaire : la profondeur calculée et mesurée sont environ identique. On peut tout de fois noter une absence de points pour des mesures entre 0.3 et 0.5m, et entre 1.3 et 1.5m.
Ces résultats sont similaires à d'autres Kinect que nous avons trouvé sur internet.

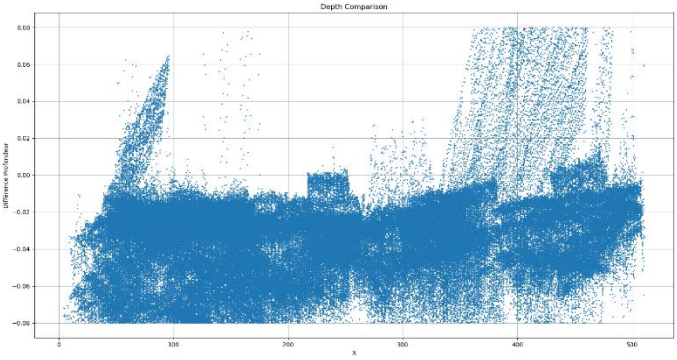


*Figure 2 : Consistance et Variabilité des Mesures de Profondeur par rapport à l'axe des x*

Le graphique présent dans la Figure 2 permet d'observer la précision des mesures de profondeur prises par la caméra Kinect par rapport au placement sur l'axe des abscisses de l'image prise.
La concentration des points entre les valeurs de -0.02 et 0.02 sur l'axe des ordonnées indique que la majorité des mesures sont précises et présentent peu de variation par rapport à une valeur de référence, ce qui est un signe de fiabilité.
Les points en dehors de ces bandes, considérés comme des outliers, pourraient signaler des erreurs ou des cas atypiques.

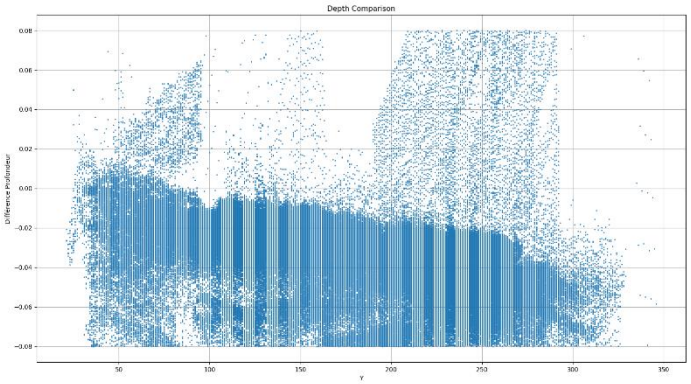


*Figure 3 : Consistance et Variabilité des Mesures de Profondeur – par rapport à l'axe des y*

Le nuage de points de la Figure 3 est comme la Figure 2, mais sur l'axe des ordonnée de l'image.
La précision des mesures de profondeur est comprise entre -0,08 et 0,08, avec la majoritée des points entre -0.08 et 0.02.
La concentration des points est plus dense vers le bas, indiquant une cohérence dans les mesures de profondeur à ces niveaux. À mesure que l'on monte sur l'axe vertical, les points deviennent plus dispersés, ce qui pourrait indiquer une variabilité accrue dans les mesures à ces profondeurs.

### *B. Résultats du Record (Package)*

- **Test de Performance**

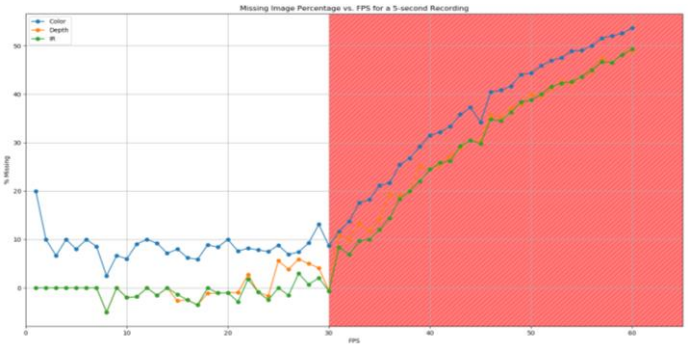


*Figure 4 : Perte d'images par rapport au FPS*

Sur cette première figure, nous pouvons observer la relation entre le pourcentage de perte d'images par rapport à la fréquence de capture d'images. Pour rappel, les objectifs de la Kinect v2 ne peuvent pas dépasser une fréquence de 30fps.
Le pourcentage de perte d'images est le rapport du nombre d'images capturés sur le total théorique (temps multiplié par fréquence).
Nous observons que le pourcentage de perte reste constant entre 2 et 30 fps (autour de 0% pour les relevés de profondeurs et synchronisé et 8% pour les images couleurs). Au-delà de 30fps, iai n'arrive plus à suivre et le pourcentage d'erreur s'emballe.

- **Vidéos**

Vous trouvrez ci-joint les liens des vidéos des topics capturés par la Kinect2.
- Vidéo couleur : https://urlz.fr/qkYf
- Vidéo depth : https://urlz.fr/qkYc
- Vidéo synchronisé : https://urlz.fr/qkYj

Le décallage entre les contours des objets et le nuage de points noir est normal : il n'agit d'une erreur intrasèque aux caméras Kinect d'environ 3cm.

## VII. Conclusion

Nous sommes parvenus à compléter tous les objectifs demandés : la caméra est calibrée et peut enregistrer des vidéos comme notre encadrante le souhaite.
La node dispose de plusieurs arguments qui permettent de personnaliser son output.
Les performances de la capture et de l'enregistrement sont plus que correct.

Malheureusement, des contraintes techniques ont fait que nous n'avons pas pu réaliser ce projet sous ROS2, et des difficultés techniques ont grandement ralenti notre travail.

Ce projet nous permit de travailler notre adaptabilité et notre habilité à résoudre des problèmes.
De plus, nous avons appris comment le calibrage d'une caméra se faisait, et des outils proposés par OpenCV. Ce projet a également été une introduction à la programmation en classe, et aux subtilités de ROS1.

# REFERENCES


[1] IFM 4.0, “ L’Industrie 4.0 c’est quoi ? ”, [En ligne]. Disponible sur : https://www.ifm40.fr/industrie-4-0-cest-quoi/

[2] IAI Kinect2, “iai_kinect2: Tools and libraries for a ROS Interface to the Kinect One (Kinect v2)”, GitHub repository: https://github.com/code-iai/iai_kinect2

[3] Y. Li et al., “kinect2_ros2: Tools and libraries for a ROS Interface to the Kinect One (Kinect v2)”, GitHub repository: https://github.com/YuLiHN/kinect2_ros2

[4] A. J. Sharda, “Navigating the Evolution: ROS1 vs. ROS2 - A Comprehensive Comparison,” Medium, février 2024. [En ligne]. Disponible : https://medium.com/@ashishjsharda/navigating-the-evolution-ros1-vs-ros2-a-comprehensive-comparison-1b3d4d4b57fe

[5] Jeremy STEWARD, Dr. Derek LICHTI, Dr. Jacky CHOW, Dr. Reed FERBER, Sean OSIS, “Performance Assessment and Calibration of the Kinect 2.0 Time-of-Flight”, [En ligne]. Disponible : https://www.fig.net/resources/proceedings/fig_proceedings/fig2015/papers/ts06e/TS06E_steward_lichti_et_al_7692.pdf

[6] Kourosh Khoshelham,Sander Oude Elberink,“Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications”, Sensors, vol. 12, no. 2, pp. 1437-1454, 31 Janvier 2012. [En ligne]. Disponible: https://www.mdpi.com/1424-8220/12/2/1437

[7] OpenCV, “Camera Calibration and 3D Reconstruction”, 27 Septembre 2023. [En ligne]. Disponible : https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

[8] OpenCV, “Camera Calibration With OpenCV”, 12 Avril 2024. [En ligne]. Disponible : : https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html. Publié le.

[9] Fankhauser, Péter; Bloesch, Michael; Rodriguez, Diego; Kaestner, Ralf; Hutter, Marco, Siegwart, Roland, “Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling”, 10 Septembre 2015. [En ligne]. Disponible : https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/104272/eth-48073-01.pdf;jsessionid=584053728D2C43DAEA2B0A0869749117?sequence=1