Projet de conception d'une canne intelligente

Lefebvre Florian

January 22, 2025

Titre : Projet de conception d'une canne intelligente

Résumé : Ce projet vise à développer une canne intelligente pour personnes aveugles en intégrant une caméra et un système de vision artificielle capable de détecter et identifier les objets environnants. Grâce à des algorithmes de traitement d'image et d'intelligence artificielle, la caméra analyse en temps réel l'environnement et détecte les obstacles d'intérêts. Ces informations sont ensuite transmises à l'utilisateur via des signaux audio pour faciliter la navigation en toute sécurité. L'objectif est d'améliorer l'autonomie et la mobilité des personnes malvoyantes dans leur quotidien.

Contents

1 Introduction

1.1 Objectifs

L'objectif est de développer une canne intelligente capable d'aider les personnes aveugles ou malvoyantes à naviguer de manière plus autonome grâce à la vision artificielle et l'intelligence artificielle. La canne devrait pouvoir détecter les obstacles, reconnaître les objets environnants et fournir un retour audio ou haptique à l'utilisateur.

Il est prévu d'utiliser un ou plusieurs capteurs à ultrasons pour détecter les obstacles devant l'utilisateur. En cas de détection d'un obstacle à proximité (moins d'un mètre, par exemple), la canne peut envoyer un signal haptique à l'utilisateur via un moteur de vibration ou le prévenir avec un retour audio.

Avec la caméra et les algorithmes de vision artificielle (par exemple YOLO), la canne pourrait détecter et reconnaître des objets spécifiques (comme les passages piétons, les panneaux de signalisation, les voitures, etc.). Cette information serait communiquée à l'utilisateur via un retour audio.

Il est aussi prévu d'implémenter un agent conversationnel de base qui pourrait répondre aux commandes vocales simples (par exemple, "Qu'y a-t-il devant moi ?") ou fournir des informations sur les objets reconnus. Cet assistant pourrait également être utilisé pour les retours audio sur la reconnaissance d'objet.

1.2 La mise en place du projet

Pour ce projet, il a fallu faire l'achat de matériel dont la liste est la suivante :

- Raspberry Pi (modèle avec capacité de traitement suffisant, comme le Raspberry Pi 4).
- Caméra 3D Intel RealSense D435i
- Capteur à ultrasons (pour détecter les obstacles proches).
- Modules audio (haut-parleur ou casque Bluetooth pour les retours audio).
- Moteur de vibration (pour le retour haptique).
- Batterie portable (pour alimenter le Raspberry Pi).
- Boutons ou interrupteurs (pour contrôler certaines fonctionnalités).

Les logiciels nécessaires pour ce projet sont :

- Raspbian OS ou une autre distribution compatible avec le Raspberry Pi.
- OpenCV et Yolo pour la vision artificielle.
- Libraries Python pour les capteurs (comme RPi.GPIO ou pigpio pour les capteurs à ultrasons).
- Text-to-Speech (TTS) (comme eSpeak ou Google Text-to-Speech) pour le retour audio.
- Speech Recognition (optionnel, pour les commandes vocales).

2 Etat de l'art

Plusieurs études ont été mené pour concevoir une canne pour aveugle dotée de possibilités innovantes comme la détection d'obstacles. Nous présentons ici certaines de ces études.

2.1 La canne EVAL multi-capteurs

Cette étude [WAN2021] propose une canne intelligente peu coûteuse, non intrusive et multifonctionnelle, pour des personnes aveugles ou des personnes malvoyantes. Plusieurs capteurs tels qu'un accéléromètre/gyromètre, un capteur à ultrasons, une résistance sensible à la force (FSR) et un capteur de température ont été installés sur la canne pour collecter les données de la démarche de l'utilisateur et de son environnement de marche. Bien que ces capteurs n'aient pas été installés directement sur le corps humain, la fonction de marche de l'utilisateur peut être décrite par certains d'entre eux, tels que l'accéléromètre et le FSR, ce qui signifie que la démarche de l'utilisateur peut être analysée et évaluée pendant la marche avec la canne. En outre, sur la base de l'analyse des données de l'accéléromètre, il est également possible de détecter des événements indésirables tels que des chutes pendant la marche, ce qui est utile pour le suivi de la santé de l'utilisateur. De plus, la canne dispose d'un capteur à ultrasons utilisé pour détecter un obstacle devant la canne et ce capteur fonctionnera avec un buzzer afin de donner un signal sonore à l'utilisateur lorsqu'un obstacle est déteint.

La canne EVAL représente une avancée significative pour la mobilité des personnes malvoyantes ou aveugles grâce à ses fonctionnalités technologiques et ses capacités étendues. Cependant, son adoption est freinée par son coût,

sa complexité et des limitations techniques. Elle est idéale pour les personnes ayant des besoins spécifiques et capables de s'adapter à ces outils modernes, mais elle ne remplace pas complètement la canne blanche traditionnelle, simple, fiable et économique.

2.2 Une canne connectée intelligente multi-technologie radio

Cette étude [VAL2021] présente le développement d'une canne intelligente, dotée de plusieurs technologies de communication sans fil, visant à améliorer la mobilité et la sécurité des personnes âgées ou à mobilité réduite. L'objectif principal de cette canne est de fournir une aide supplémentaire aux utilisateurs en intégrant des fonctionnalités de détection et de communication. L'étude porte sur plusieurs aspects techniques, allant de la conception à la réalisation, ainsi que sur les tests effectués pour s'assurer de son efficacité. La canne est équipée de différentes technologies radio (comme le Bluetooth et le Wi-Fi), permettant de collecter et de transmettre des données relatives à la position, à l'environnement et à l'état de santé de l'utilisateur. La canne intègre des capteurs pour surveiller divers paramètres tels que la distance, l'orientation et les obstacles, et pour fournir une aide en cas de chute ou d'incident. De plus, elle permet une communication avec des appareils externes, tels que des smartphones ou des dispositifs médicaux, facilitant ainsi un suivi à distance par des proches ou des professionnels de la santé.

Les tests réalisés ont permis de valider la fonctionnalité de la canne et son efficacité dans la détection d'obstacles et dans la fourniture d'assistance en cas d'urgence. L'article présente également les défis techniques rencontrés pendant le développement de la canne, ainsi que les solutions apportées. En résumé, cette étude propose une solution innovante et technologique visant à améliorer la sécurité des utilisateurs et à faciliter la gestion de leur mobilité, tout en assurant une surveillance à distance grâce aux technologies de communication sans fil.

2.3 Le projet ICanne

Le projet ICanne [4] est une canne intelligente conçue pour améliorer la mobilité et l'autonomie des personnes malvoyantes. Elle intègre une caméra et des capteurs intelligents permettant de détecter et d'identifier en temps réel les obstacles et objets environnants.

Grâce à l'intelligence artificielle et la vision par ordinateur, ICanne analyse l'environnement et alerte l'utilisateur via des signaux haptique (vibra-

tions) et audio (guidage vocal). Elle permet ainsi d'éviter les dangers tels que les trottoirs, escaliers, poteaux ou véhicules en mouvement.

ICanne répond à un besoin réel : améliorer l'autonomie des personnes aveugles ou malvoyantes. Actuellement, les cannes classiques ne permettent pas d'identifier précisément les obstacles complexes (véhicules en mouvement, objets suspendus, etc.). En intégrant une vision artificielle et des capteurs intelligents, cette innovation apporte une solution proactive aux défis de mobilité urbaine.

2.4 C'est quoi YOLO ?

YOLO [SUN2020] est l'acronyme de You only look once est l'une des architectures les plus révolutionnaires dans le domaine de la détection d'objets. Introduit pour la première fois en 2015, YOLO a transformé la manière dont les objets sont détectés dans les images et les vidéos contrairement aux approches traditionnelles de détection d'objets qui divisent l'image en région et passe chaque région à travers un classificateur séparé, YOLO adopte une approche radicalement différente en traitant la détection comme un problème de détection unique. Ce qui distingue YOLO, c'est sa capacité à traiter une image entière en une seule passe à travers le réseau de neurones d'où son nom ! You only look once. Cette approche permet non seulement de réaliser des détection extrèmement rapides mais aussi d'obtenir une grande précision même dans des contextes en temps réel comme la surveillance vidéo, la conduite autonome, et les applications mobiles. Au fil des années YOLO a évolué à travers plusieurs versions, chacune apportant des améliorations significatives en terme de vitesse, de précision et de capacité à détecter des objets dans des environnements variés et complexes.

- La version de YOLOV1 introduit l'idée novatrice d'utiliser un réseau de neurones unique pour la détection d'objets en temps réel révolutionnant ainsi la façon dont ces tâches étaient abordées.

- Cette première version a rapidement été suivi par YOLOv2 également connu sous le nom de YOLO9000. Elle a amélioré la précision du modèle tout en introduisant des techniques avancées telles que les ancres (anchor boxes) permettant une meilleure gestion des objets de différentes tailles.

- YOLOV3 a marqué une nouvelle étape importante en introduisant la détection multi-échelle, une fonctionnalité clé qui a permis de mieux détecté les objets à différentes tailles tout en améliorant la performance globale et la flexibilité du modèle.

- Avec YOLOV4, l'accent a été mis sur l'optimisation pour la vitesse. Cette version a combiné diverses techniques d'optimisation architecturale pour maximiser l'efficacité du modèle le rendant encore plus adapté aux application en temps réel.

- Ensuite, YOLOV5 est apparu bien qu'il n'ai pas été développé par les auteurs originaux, il a rapidement gagné en popularité car grace à ces performances remarquables, et à sa facilité d'utilisation. Cette version a été largement adopté par la communauté pour sa capacité à s'adapter à différents cas d'utilisation.

- Les versions YOLOV6 à YOLOV8 ont continué sur cette lancée en apportant des optimisations supplémentaires pour diverses applications. Ces versions ont vu des améliorations significatives en terme de structures réseaux et de techniques de formation renforçant la précision et la rapidité du modèle.

- YOLOV9 a poursuivi cet effort en intégrant des avancées récentes dans le domaine de la détection d'objets offrant un bonne équilibre entre rapidité et précision.

- Cependant, il existe des versions plus récentes comme YOLOV10 qui continuent à perfectionner ces aspects et à introduire de nouvelles architectures techniques de formation consolidant YOLO comme l'un des modèles les plus performants pour les tâches de détection en temps réel.

3 Le déroulement du projet

Pour mener à bien ce projet, j'ai dû réaliser un ensemble de mini-projets permettant d'avancer progressivement dans l'apprentissage des technologies et une intégration sécurisée de chacune d'entre elles. Je vais présenter ici les différentes versions que j'ai mis en place.

3.1 Première version : test et choix d'une version de YOLO

Au début de ce projet, j'ai dû choisir un outil me permettant de détecter des objets à l'aide d'une caméra. Il fallait prendre en compte le contexte technique de ce projet qui était l'utilisation d'un Raspberry version 5. J'ai fait le choix de YOLO car il a été optimisé pour des appareils à ressources

limitées comme le Raspberry Pi, les drones, ou les smartphones. Des variantes légères comme Tiny YOLO sont conçues spécifiquement pour des environnements avec des contraintes matérielles. YOLO est conçu pour être rapide, ce qui en fait un choix idéal pour des applications en temps réel. Contrairement à d'autres modèles de détection comme R-CNN, YOLO traite une image entière en une seule passe du réseau, ce qui réduit considérablement le temps de traitement. Les dernières versions de YOLO (comme YOLOV5 ou YOLOV8) sont optimisées pour le calcul sur GPU, offrant une performance exceptionnelle même sur des dispositifs embarqués. La version que j'ai choisi pour ce projet est YOLOv7. YOLOv7 est une évolution des modèles YOLO (You Only Look Once) pour la détection d'objets en temps réel. Publié en 2022, il améliore les performances par rapport à ses prédécesseurs comme YOLOv5 et YOLOv4. La figure suivante extraite du site de github : `https:github.com/WongKinYiu/yolov7/tree/main` permet de comparer quelques différentes versions. Sur ce schéma, chaque YOLO a des performances. Cela indique qui est le plus ou le moins rapide et le plus ou le moins précis.

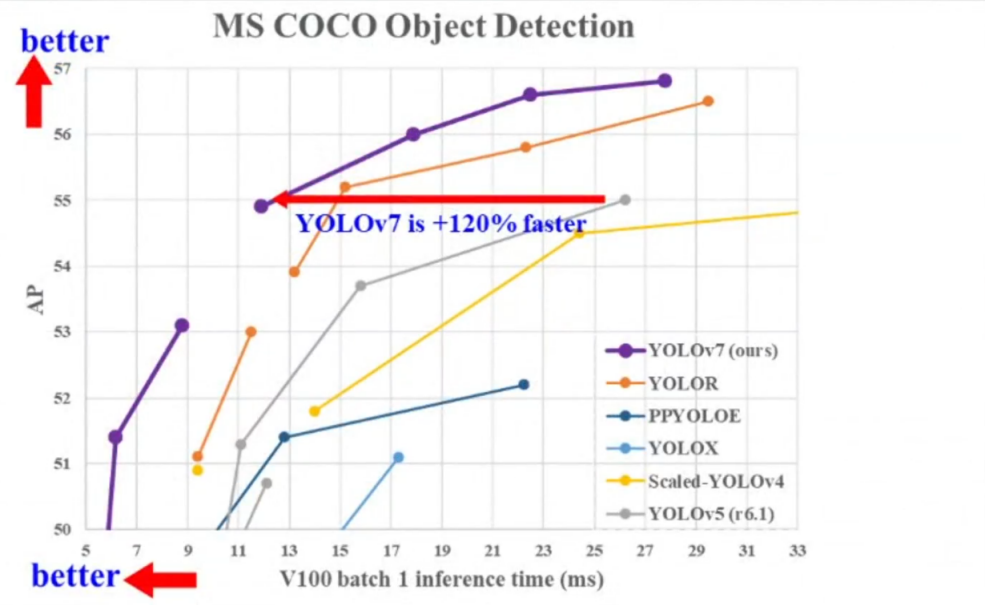

Figure 1: Comparaison de versions de YOLO

Dans le tableau ci-dessous, on voit que la précision de YOLOV7 est de 55.9

L'annexe 1 donne le programme de cette version et ci-dessous ses principales etapes:

Model	Test Size	AP^{test}	AP_{50}^{test}	AP_{75}^{test}	batch 1 fps	batch 32 average time
YOLOv7	640	51.4%	69.7%	55.9%	161 *fps*	2.8 *ms*
YOLOv7-X	640	53.1%	71.2%	57.8%	114 *fps*	4.3 *ms*
YOLOv7-W6	1280	54.9%	72.6%	60.1%	84 *fps*	7.6 *ms*
YOLOv7-E6	1280	56.0%	73.5%	61.2%	56 *fps*	12.3 *ms*
YOLOv7-D6	1280	56.6%	74.0%	61.8%	44 *fps*	15.0 *ms*
YOLOv7-E6E	1280	56.8%	74.4%	62.1%	36 *fps*	18.7 *ms*

Figure 2: Comparaison de versions de YOLO

1. **Chargement du Modèle YOLO**
   - Chargement des fichiers de configuration `yolov7-tiny.cfg` et des poids du réseau `yolov7-tiny.weights`.
   - Initialisation du réseau neuronal.
2. **Lecture des Classes d'Objets**
   - Chargement du fichier `coco.names`, contenant les noms des classes (ex. personne, voiture, chien...).
   - Stockage des classes dans une liste.
3. **Identification des Couches de Sortie**
   - Récupération des noms des couches du réseau.
   - Détermination des couches utilisées pour les prédictions.
4. **Capture Vidéo en Temps Réel**
   - Ouverture de la caméra pour capturer des images en continu.
   - Vérification de la connexion vidéo.
5. **Prétraitement de l'Image**
   - Conversion de l'image en un format compatible avec YOLO (redimensionnement à 416x416 pixels, normalisation).
   - Passage de l'image au réseau neuronal.
6. **Exécution de YOLO et Analyse des Détections**

- Extraction des scores de confiance pour chaque classe détectée.
- Filtrage des détections avec un seuil minimal de 50%.

7. **Calcul des Coordonnées des Boîtes Englobantes**
   - Conversion des coordonnées normalisées en pixels réels.
   - Détermination de la position et de la taille des objets détectés.
8. **Affichage des Résultats**
   - Dessin des boîtes englobantes sur l'image.
   - Affichage des noms des objets et du niveau de confiance.
9. **Gestion des Entrées Utilisateur**
   - Affichage de la vidéo avec les détections en temps réel.
   - Arrêt du programme lorsqu'on appuie sur la touche Échap.
10. **Libération des Ressources**
    - Fermeture de la capture vidéo.
    - Destruction des fenêtres OpenCV pour libérer la mémoire.

3.2 Deuxième version : Intégration de cette première version avec la caméra 3d

La caméra utilisée dans ce projet est le modèle 3d Intel RealSense D435i. Elle est une solution de choix pour les projets nécessitant une perception 3D avancée couplée à des fonctionnalités de suivi spatial. Sa polyvalence et sa facilité d'intégration en font un outil prisé dans des domaines variés allant de la recherche académique aux applications industrielles avancées.

Pour faire cette deuxième version, il a fallu utiliser le kit de développement SDK 2.0. Ce kit de développement est une solution clé en main conçue pour les développeurs et ingénieurs souhaitant exploiter les capacités avancées de la caméra D435i dans des applications de vision par ordinateur, robotique, ou réalité augmentée. Ce kit intègre la caméra RealSense D435i et les outils logiciels nécessaires pour faciliter son utilisation et son intégration dans divers projets. L'annexe 2 de ce document contient le code en Python de cette intégration de YOLO avec la caméra utilisée dans ce projet. Voici ci-dessous les étapes générales de ce programme :

1. **Chargement du modèle YOLOv7**

- Chargement du fichier de poids (`model.pt`) sur le **CPU**.
- Récupération des **noms des classes** que YOLO peut détecter.

2. **Initialisation de la caméra RealSense**
   - Activation des flux **RGB** et **profondeur** avec une résolution de 640×480 à 30 FPS.
3. **Boucle de capture d'images en temps réel**
   - Acquisition des frames (image couleur + profondeur).
   - Prétraitement des images :
     - Redimensionnement
     - Normalisation
     - Conversion en tenseur PyTorch
   - Détection des objets avec YOLOv7.
   - Filtrage des détections avec **Non-Max Suppression (NMS)** pour éviter les doublons.
4. **Affichage des résultats**
   - Pour chaque objet détecté :
     - Récupération des **coordonnées de la boîte englobante**.
     - Calcul de la **distance de l'objet** par rapport à la caméra.
     - Affichage des **boîtes vertes et labels** sur l'image.
5. **Interaction utilisateur**
   - Pose d'une question pour identifier un objet.
   - Comparaison avec `"quel objet"`.
   - Appel de la fonction de détection et affichage des objets détectés.
6. **Gestion de la fermeture du programme**
   - Si la touche `Échap (ESC)` est pressée, la capture s'arrête.
   - Fermeture de la caméra et des fenêtres OpenCV.

3.3 Troisième version : Développement d'une version avec Reconnaissance vocale et synthèse vocale sur PC

Cette troisième version a pour objectif d'offrir à l'utilisateur le moyen d'intéragir avec les objets reconnus par YOLO. Pour cela, il est interessant de noter que le langage utilisé dans notre interface est dit opératif [FAL1986]. Un langage opératif est orienté vers l'exécution d'actions concrètes et mesurables. Contrairement à un langage purement théorique ou descriptif, il permet de décrire des processus qui mènent directement à des résultats pratiques, comme l'exécution d'une tâche ou la résolution d'un problème. Dans un langage opératif, chaque action doit être clairement définie, de manière à ce que le programme ou le système sache exactement ce qu'il doit faire. Cela se traduit par une syntaxe simple et précise pour exprimer des opérations à réaliser. La syntaxe opérative en reconnaissance vocale est essentielle pour structurer l'interaction voix-machine de manière fluide et efficace. Elle permet de contrôler précisément l'écoute de l'utilisateur, l'analyse des mots-clés, l'activation des fonctionnalités associées, le retour vocal ou visuel.

Dans le contexte de notre étude, le vocabulaire choisi correspond en général à des verbes qui sont associées à ce que peut renvoyer la caméra sur son champ d'observation. Par exemple, des mots comme voir, aider, loin, proche peuvent déclencher des actions tels que voir les objets devant la caméra, demander de l'aide sur les possibilités du dialogue, donne l'objet le plus proche dans le champ de la caméra. La nature du dialogue est donc très limitée. De plus, le contexte de modalité choisi est la reconnaissance de la parole puisque notre sujet concerne des personnes aveugles ou malvoyants. Or, cette modalité est malheureusement très délicate à gérer de part le bruit ambiant, la qualité de la reconnaissance. Il faut par conséquent permettre d'interpréter des phrases avec malheureusement des mots non reconnus, des erreurs de reconnaissance. Les phrases peuvent être non respectueuse d'une syntaxe du français. A titre d'exemple, voici une phrase prononcé plusieurs fois et le résultat de la reconnaissance. La phrase est : "montre moi les objets que tu voix" et elle est prononcée 7 fois de suite :

```
1        montre-moi que tu vois
2        montre-moi les objets que tu vois
3        a des moutons et les objets que tu vois
4        montre-moi les objets que tu vois
5        montre-moi les objets que tu vois
6        a un mois de que tu as que tu vois
7        tu peux tu vois
```

Le résultat montre clairement que sur 7 répétions, trois sont correctes (2, 4, 5) et les autres ont une syntaxe ne respectant pas le français(1, 3, 6, 7). Par conséquent, utiliser un parseur qui permettrait de valider la syntaxe de la phrase reconnue vocalement, ne serait pas d'une grande utilité.

La technique que j'ai choisie est l'utilisation d'une grammaire à îlots de confiance[1][2]. Un îlot de confiance en langage naturel fait référence à une portion de texte ou de discours où certaines informations sont considérées comme fiables et établies, souvent dans un contexte plus large où des éléments de doute ou d'incertitude peuvent exister. En reconnaissance de la parole, il est fréquent qu'une partie de la transcription soit erronée à cause de bruits externes, d'accents ou de variations dans la parole. Les îlots de confiance aident à identifier ces erreurs rapidement et à les corriger en se basant sur les parties du discours où la certitude est plus élevée, facilitant ainsi les processus de traitement.

Les étapes générales pour implémenter un îlot de confiance sont :

1. Capture et validation initiale de l'entrée vocale.

2. Prétraitement et détection de confiance sur l'entrée vocale.

3. Validation des commandes dans un îlot sécurisé où seule une commande valide est exécutée.

4. Réponse ou action sécurisée en fonction des résultats.

Dans cet exemple, l'implémentation d'un îlot de confiance en Python sert à valider une commande vocale avant qu'elle ne soit exécutée. Ce système garantit que seules des commandes fiables et sûres sont prises en compte, protégeant ainsi l'intégrité du système. L'îlot de confiance joue un rôle de filtre avant de laisser certaines actions se produire, en fonction de la reconnaissance de la parole et d'une vérification préalable.

L'algorithme qui permet d'identifier l'action la plus probable à partir d'une phrase dite vocalement en analysant les mots qu'elle contient est le suivant :

1. Création d'une table d'actions : Chaque action est associée à une liste de mots synonymes.

2. Découpage de la phrase en mots : La phrase d'entrée est divisée en une liste de mots individuels.

3. Pour chaque action, on compte combien de mots de la phrase figurent dans la liste des synonymes de l'action.

4. Un score est attribué à chaque action en fonction du nombre de mots correspondants.
5. Sélection de l'action la plus pertinente :
6. L'action ayant le plus de mots en commun avec la phrase est retenue.
7. Si aucun mot pertinent n'est trouvé, la phrase est considérée comme du "bruit".

Phrase d'entrée	**Action identifiée**
Je veux apercevoir et observer le paysage	**voir**
Peux-tu écouter et percevoir ce son	**entendre**
Je vais courir et me déplacer rapidement	**marcher**
Cette phrase n'a aucun mot pertinent	**Bruit détecté**

Table 1: Exemple de fonctionnement de l'algorithme

Cet algorithme est décrit en pseudo-langage dans l'annexe 3 de ce document. En ce qui concerne la synthèse vocale et la reconnaissance vocale, plusieurs solutions étaient envisageable. Le tableau ci-dessous extrait de ChatGPT compare ces solutions :

Outil	**Précision**	**Langues**	**Hors ligne**	**Facilité**	**Coût**
Google Speech-to-Text	****	120+	Non	****	Payant
Microsoft Azure	****	70+	Non	****	Payant
IBM Watson	***	10+	Non	***	Payant
OpenAI Whisper	*****	50+	Oui	**	Gratuit
Deepgram	****	30+	Non	****	Payant
VOSK	***	20+	Oui	***	Gratuit

Table 2: Comparaison des outils de reconnaissance vocale

J'ai choisi VOSK qui a l'avantage d'être gratuit, d'être simple d'utilisation. VOSK est une bibliothèque open-source développée pour la reconnaissance vocale en temps réel et hors ligne. Basée sur le moteur Kaldi, elle est optimisée pour fonctionner sur des appareils à faible puissance, tels que les Raspberry Pi, les ordinateurs embarqués, et les serveurs locaux.

En ce qui concerne la synthèse vocale, j'ai choisi PyAudio qui est une bibliothèque Python qui permet d'interagir avec les périphériques audio pour enregistrer et lire des fichiers sonores. Elle est basée sur PortAudio, une API open-source pour la gestion des entrées et sorties audio sur plusieurs plateformes.

L'une des difficultés rencontrées est la gestion de la reconnaissance vocale et de la reconnaissance d'objets avec YOLO en parallèle. Comme le traitement de YOLO se fait en continu sur les images envoyées par la caméra, il a fallu gérer également la reconnaissance de la parole en parallèle. A l'aide des threads de Python, cela a été possible mais il a été nécessaire de vider le canal de communication utilisée par la reconnaissance vocale pour ne pas que la synthèse vocale soit interprétée par la reconnaissance vocale. L'annexe 5 contient le code de cette version.

3.4 Quatrième version : Portage de la version précédente sur Raspberry

Dans cette quatrième version, la problématique a été l'installation de l'interface pour Python. Le code de l'application n'a pas changé mais la camera 3d n'était pas reconnue par Python. De nombreux échanges avec la hotline du constructeur de la caméra ont fini par aboutir à un fonctionnement de l'application. Toutes le versions de Python ne sont pas compatibles avec cette caméra. Mais la hotline m'a indiqué que Python n'était pas le langage favori et stable pour cette caméra. D'ailleurs, ils convenaient eux-mêmes que je n'étais pas le seul à avoir des problèmes. Les langages favories pour cette caméra étant C ou C++, mais la gestion de YOLO se faisant généralement en Python, il semblait naturel de gérer également la caméra en Python. Voici un extrait de cette discussion avec la hotline :

"*Although the librealsense SDK can be made to work with Raspberry Pi using installation methods such as libuvc, Intel do not provide official support for the SDK on Pi. I also do not use it on Pi myself. It is not the ideal platform to use RealSense on, as at best a Pi can provide access to basic RealSense depth and color streams but not more advanced features such as IMU data, alignment, post-processing filters. It is possible to use libuvc to install librealsense, but adding wrappers such as pyrealsense2 or ROS on Pi can be problematic.*"

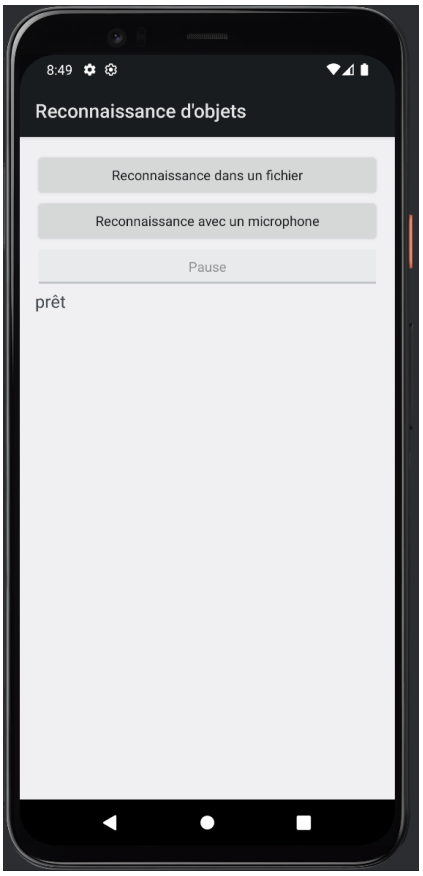

Figure 3: Interface de base avec l'application Android

Figure 4: Ouverture du microphone pour lancer une requête

3.5 Cinquième version : Développement d'un client Androïd

L'application Androïd (Figure 3 et 4) est un client qui permet de prononcer une commande vocalement. Une fois le microphone ouvert, l'utilisateur dit avec la voix ce qu'il souhaite demander à la caméra. Cette requête est envoyée au Raspberry par le protocôle WIFI. Le Raspberry sert donc de serveur. L'application Python est lancée sur celui-ci. Au démarrage, on demande à l'utilisateur s'il souhaite interagir avec la caméra sur le Raspberry ou bien sur le client Androïd. Ce client Androïd ne sert donc que d'interface client. Toute l'analyse de la requête et la détection des objets se réalisent sur l'ordinateur Raspberry. Cette application Androïd a été réalisé à l'aide d'un tutoriel [3] montrant l'intégration de la bibliothèque VOSK sur Androïd auquel j'ai ajouté le protocôle de communication pour communiquer avec le Raspberry.

Le code de cette application est fournie en annexe 4. Ci-dessous le code Python qui gère l'attente des requêtes :

```
1  with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       as server_socket:
2                  server_socket.bind((HOST, PORT))
3                  server_socket.listen()
4                  while True:
5                      client_socket, client_address =
                           server_socket.accept()
6                      with client_socket:
7                          data =
                               client_socket.recv(1024).decode('utf-8')
8                          if data :
9                              try:
10                                 parsed_data =
                                       json.loads(data)
11                                 extracted_value =
                                       parsed_data.get("text",
                                       None)
12                                 if extracted_value:
13                                     q.put(extracted_value.lower())
14                             except json.JSONDecodeError:
15                                 print("Failed to parse
                                       data as JSON.")
```

Le programme fonctionne de la manière suivante :

1. **Démarrage du serveur :**
   - Le serveur démarre et attend des connexions entrantes sur l'adresse `HOST` et le port `PORT`.
2. **Connexion d'un client :**
   - Un client se connecte au serveur via une connexion **TCP**.
   - Le serveur accepte la connexion et crée un *socket* dédié à la communication avec ce client.
3. **Réception et traitement des données :**
   - Le client envoie un message contenant des données au format **JSON**.
   - Le serveur reçoit ces données et les convertit en texte.

- Il extrait la valeur associée à la clé `"text"` si elle est présente.

4. **Stockage des données :**
   - Si la clé `"text"` est trouvée, son contenu est converti en **minuscule**.
   - Ce texte est ensuite ajouté à la file d'attente `q` pour un traitement ultérieur.
5. **Gestion des erreurs JSON :**
   - Si le message reçu n'est pas un JSON valide, une erreur est affichée sans interrompre le serveur.
6. **Maintien en écoute :**
   - Le serveur continue d'écouter et peut traiter de nouvelles connexions clients.

Côté Androïd, voici le code concerné par cette envoie de requête vers le Raspberry :

```
1  private void sendToServer(String text) {
2          try (Socket socket = new Socket(RASPBERRY_PI_IP,
               PORT);
3               OutputStream outputStream =
                    socket.getOutputStream()) {
4
5              outputStream.write(text.getBytes());
6              outputStream.flush();
7
8              Log.d(TAG, "Texte envoye au serveur : " +
                   text);
9
10         } catch (Exception e) {
11             Log.e(TAG, "Erreur lors de l'envoi au
                   serveur : " + e.getMessage());
12         }
13     }
```

3.6 Sixième version : Prise en compte de certaines remarques de Mr Uzan

Monsieur Uzan a eu la gentillesse de tester mon travail en me donnant des pistes d'améliorations. L'une d'entre elles est une meilleure gestion du micro.

En effet, le micro n'est pas contrôlable par l'utilisateur et la reconnaissance vocale peut déclencher le traitement sur ce qui vient d'être dit. Ce qui provoque la répétition d'une commande. Pour éviter cela, il est désormais possible vocalement d'ouvrir le micro, de le fermer et de connaître l'état du micro. D'autre part, lorsque la synthèse fonctionne, l'information qu'elle donne est enlevé du canal où le thread de la reconnaissance de la parole va chercher les informations à traiter.

Un autre conseil de Mr Uzan est de ne pas abuser de la synthèse vocale car elle peut à force devenir une charge cognitive importante pour l'utilisateur. C'est vrai que la tendance de mon application est de confirmer ce qui a été compris vocalement, ce qui devient trop redondant avec la reconnaissance de la parole. Un certain nombre de messages vocaux ont été supprimés pour alléger l'interface vocale.

Un autre conseil très intéressant mais pour plus tard, serait pour Monsieurs Uzan de pouvoir scénariser l'application. En effet, on pourrait imaginer être dans une salle de bain, et se demander où est le dentifrice, est-ce que la baignoire est remplie etc..

Un autre conseil serait de rendre le dialogue moins prévisible pour l'utilisateur comme par exemple, tirer au hasard une réponse parmi plusieurs ayant le même sens. Cela permet de générer un dialogue plus naturel et moins "robotisé". Cela n'est pas encore intégré dans l'application.

3.7 Perspectives : Integration de Rasa et de chatGPT

J'ai commencé à étudier le framework Rasa qui permet de développer un chatbot. Rasa est un framework open-source pour construire des chatbots et des assistants virtuels alimentés par l'intelligence artificielle (IA). Il est principalement utilisé pour développer des systèmes de traitement du langage naturel (NLP) et permet aux entreprises de créer des assistants conversationnels personnalisés.

En effet, l'interaction actuelle de mon système reste limitée et ne permet pas à l'utilisateur de construire un dialogue contextuel. Rasa servira à gérer les interactions avec l'utilisateur :

- Ex: *"Décris-moi ce qu'il y a devant moi."*
- Ex: *"Y a-t-il un obstacle devant moi ?"*
- Ex: *"Quel est l'objet à 2 mètres sur ma droite ?"*
- Rasa peut aussi gérer des commandes vocales et envoyer les requêtes au modèle de détection.

ChatGPT peut être utilisé pour :

- Améliorer la description des objets détectés (expliquer avec plus de détails).
- Adapter la réponse selon le contexte et le langage naturel.
- Offrir un support conversationnel avancé.

Ce type d'architecture serait conditionnée par la capacité du Raspberry a supporté ce type d'outils avec un FPS raisonnable. Bien que le Pi est utilisé pour des projets d'IA depuis un certain temps maintenant, une nouvelle carte pour le Raspberry : le PI IA kit pourrait lui fournit le coup de pouce nécessaire pour exécuter des modèles avancés et lui offre la capacité de faire fonctionner des modèles plus simples plus rapidement.

4 Conclusions

Ce projet m'a permis de découvrir de nombreux domaines que je ne connaissais pas du tout, telles que la reconnaissance vocale, la vision par ordinateur à l'aide d'un framework réputé comme YOLO. La mise au point technique de ce projet est complexe mais m'a permis d'entrevoir des possibilités très encourageantes au service du handicap.

Les apports de ce projet me permettront d'aborder mon stage en entreprise avec plus de sereinité car sa problèmatique rejoint un peu celle de ce projet en utilisant la vision par ordinateur pour apprendre à rééduquer certaines personnes souffrant d'handicap moteur.

5 Bibliographie

References

[BER2024] Somnath Bera, *fonctions vocales avec le Raspberry Pi Zero, Revue Elektor édition spéciale IA, p 58 à 65, 2024*

[FAL1986] Pierre Falzon, *Langages opératifs et compréhension opérative. Thèse de doctorat, Université Paris V – La Sorbonne, 1986*

[SUN2020] Anu Suneja, *A comprehensive review of YOLO architectures in computer vision: fromYOLOv1toYOLOv10, Independently published, 2020*

[PAN2016] Tanay Pant, *Building a virtual assistant for RaspBerry Pi, APRESS, 2016*

[VAL2021] Thierry Val, Réjane Dalcé, Imen Megdiche, Oussema Fakhfakh, Khawla Ltif, *Etude, conception, réalisation et tests d'une nouvelle canne connectée intelligente multi-technologie radio, HAL open science, 2021*

[WAN2021] Ting Wang, Rupert Grobler, Eric Monacelli, *The Developpment of EVAL Cane-A Smart Cane for the Evaluation of Walking Gait and Walking Environmenti, HAL open science, 2021*

6 Webographie

References

[1] Office québécois de la langue française. *Îlots de confiance en langage naturel*. Disponible en ligne : `https://vitrinelinguistique.oqlf.gouv.qc.ca/fiche-gdt/fiche/8394138/ilot-de-confiance`

[2] *Islands of Reliability for Regular Morphology*. MIT. Disponible en ligne : `https://web.mit.edu/albright/www/papers/Albright-IslandsOfReliability.pdf`

[3] *Site de démo Vosk sur Androïd*. Disponible en ligne : `https://alphacephei.com/vosk/`

[4] *Le projet icane*. Disponible en ligne : `https://www.transtech.fr/projets/icane/`

7 Annexes

7.1 Annexe 1: Test de Yolo v7

```
import cv2 as cv
import numpy as np

# Charger les poids et la configuration YOLOv4-tiny
weights_path = 'yolov7-tiny.weights'
config_path = 'yolov7-tiny.cfg'
net = cv.dnn.readNet(weights_path, config_path)

```

```
# Charger les noms des classes (fichier coco.names)
with open('coco.names', 'r') as f:
    classes = [line.strip() for line in f.readlines()]

# Obtenir les noms des couches de sortie
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in
   net.getUnconnectedOutLayers().flatten()]

cap = cv.VideoCapture(2)

while 1:
    ret, img = cap.read()
    if not ret:
        break
    img = cv.flip(img,1)
    height, width, channels = img.shape

    # Pretraitement de l'image de la camera
    blob = cv.dnn.blobFromImage(img, 0.00392, (416,
       416), (0, 0, 0), swapRB=True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    # Analyser les detections
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Obtenir les coordonnees de la boite
                   englobante
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                label = f"{classes[class_id]}:
                   {confidence:.2f}"

                # Dessiner la boite et l'etiquette sur
```

```
                    l'image
48                  color = (0, 255, 0)
49                  cv.rectangle(img, (x, y), (x + w, y +
                        h), color, 2)
50                  cv.putText(img, label, (x, y - 10),
                        cv.FONT_HERSHEY_SIMPLEX, 1, color, 2)
51
52      cv.imshow('Detection Camera', img)
53      if cv.waitKey(1) == 27:
54          break
55
56  cap.release()
57  cv.destroyAllWindows()
```

7.2 Annexe 2: Yolo v7 et la camera 3d

```
1   # Ceci est un commentaire
2   import cv2
3   import torch
4   import pyrealsense2 as rs
5   import numpy as np
6   from models.experimental import attempt_load
7   from utils.general import non_max_suppression
8
9   \# Charger le modele de YOLOOV7
10  weights = 'model.pt'  \# Remplacez par le chemin vers
        votre modele
11  model = attempt_load(weights, map_location='cpu')  #
        Charger le modele sur CPU
12  names = model.names  # Classes d'objets
13
14  # Initialisation de la camera Intel RealSense
15  pipeline = rs.pipeline()
16  config = rs.config()
17  config.enable_stream(rs.stream.color, 640, 480,
        rs.format.bgr8, 30)  # Flux RGB
18  config.enable_stream(rs.stream.depth, 640, 480,
        rs.format.z16, 30)   # Flux de profondeur
19  pipeline.start(config)
20
21  try:
22      while True:
```

```
23          # Capture des frames de la camera
24          frames = pipeline.wait_for_frames()
25          color_frame = frames.get_color_frame()
26          depth_frame = frames.get_depth_frame()
27
28          if not color_frame or not depth_frame:
29              continue
30
31          # Convertir les frames en images OpenCV
32          color_image =
               np.asanyarray(color_frame.get_data())
33
34          # Pretraitement pour YOLOv7
35          img = cv2.resize(color_image, (640, 480))
36          img = torch.from_numpy(img).permute(2, 0,
               1).float()
37          img /= 255.0  # Normalisation
38          if img.ndimension() == 3:
39              img = img.unsqueeze(0)
40
41          # Prediction avec YOLOv7
               mettre dans une fonction de ligne 40 a 61
42          pred = model(img, augment=False)[0]
43          pred = non_max_suppression(pred, 0.25, 0.45)  #
               Conf = 0.25, IoU = 0.45
44
45          # Dessiner les boites englobantes et afficher la
               distance
46          for i, det in enumerate(pred):  # Parcourir les
               detections
47              if len(det):
48                  for d in det:  # d = (x1, y1, x2, y2,
                       conf, cls)
49                      x1, y1, x2, y2, conf, cls = d[:6]
50                      x1, y1, x2, y2 = int(x1), int(y1),
                           int(x2), int(y2)
51
52                      # Calculer le centre de la boite
53                      center_x = (x1 + x2) // 2
54                      center_y = (y1 + y2) // 2
55
56                      # Obtenir la distance de l'objet
57                      distance =
```

```
                        depth_frame.get_distance(center_x,
                        center_y)
58
59                      # Afficher la boite et les
                        informations
60                      label = f"{names[int(cls)]}
                        {conf:.2f} ({distance:.2f}m)"
61                      cv2.rectangle(color_image, (x1, y1),
                        (x2, y2), (0, 255, 0), 2)
62                      cv2.putText(color_image, label, (x1,
                        y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                        (0, 255, 0), 2)
63  # ici poser la question
64          # comparer la question avec "quel objet"
65          # appeler la fonction et dire les objets
66          cv2.imshow("RealSense YOLOv7", color_image)
67
68          if cv2.waitKey(1) == 27:
69              break
70
71  finally:
72      # Liberer les ressources
73      pipeline.stop()
74      cv2.destroyAllWindows()
```

7.3 Annexe 3: Algorithme de la gestion des îlots de confiance

```
1  Debut
2
3  // Definition de la table des actions et leurs mots
       associes
4  Table Actions_Mots = {
5      "voir" = ["apercevoir", "regarder", "observer",
           "montrer"],
6      "entendre" = ["ecouter", "percevoir", "ouir"],
7      "parler" = ["dire", "exprimer", "communiquer",
           "prononcer"],
8      "marcher" = ["avancer", "se deplacer", "courir"]
9  }
10
```

```
// Fonction qui identifie l'action la plus probable en comparant les mots
Fonction Identifier_Action(phrase)
    // Diviser la phrase en mots individuels
    mots_phrase = Diviser(phrase, " ")

    // Initialisation des scores pour chaque action
    Scores_Actions = Table Vide

    // Parcourir chaque action et ses mots associes
    Pour chaque action, synonymes dans Actions_Mots Faire
        // Calcul de l'intersection des mots de la phrase et des synonymes
        score = 0
        Pour chaque mot dans mots_phrase Faire
            Si mot appartient a synonymes Alors
                score = score + 1
            FinSi
        FinPour
        // Stocker le score pour l'action
        Scores_Actions[action] = score
    FinPour

    // Trouver l'action avec le score maximal
    Meilleure_Action = ""
    Max_Score = 0

    Pour chaque action, score dans Scores_Actions Faire
        Si score > Max_Score Alors
            Max_Score = score
            Meilleure_Action = action
        FinSi
    FinPour

    // Retourner l'action la plus probable ou "" si aucune correspondance
    Retourner Meilleure_Action
FinFonction

// Execution de l'algorithme avec des exemples
Afficher(Identifier_Action("Je veux apercevoir et observer le paysage"))  // Sortie : "voir"
Afficher(Identifier_Action("Peux-tu ecouter et percevoir
```

```
    ce son"))        // Sortie : "entendre"
50  Afficher(Identifier_Action("Je vais courir et me
    deplacer rapidement"))  // Sortie : "marcher"
51  Afficher(Identifier_Action("Cette phrase n'a aucun mot
    pertinent"))       // Sortie : ""
52
53  Fin
```

7.4 Annexe 4: Client Androïd pour la reconnaissance vocale

```
1   package org.vosk.demo;
2
3   import android.Manifest;
4   import android.app.Activity;
5   import android.content.pm.PackageManager;
6   import android.os.Bundle;
7   import android.text.method.ScrollingMovementMethod;
8   import android.util.Log;
9   import android.widget.Button;
10  import android.widget.TextView;
11  import android.widget.ToggleButton;
12
13  import org.vosk.LibVosk;
14  import org.vosk.LogLevel;
15  import org.vosk.Model;
16  import org.vosk.Recognizer;
17  import org.vosk.android.RecognitionListener;
18  import org.vosk.android.SpeechService;
19  import org.vosk.android.SpeechStreamService;
20  import org.vosk.android.StorageService;
21
22  import java.io.IOException;
23  import java.io.InputStream;
24
25  import androidx.annotation.NonNull;
26  import androidx.core.app.ActivityCompat;
27  import androidx.core.content.ContextCompat;
28
29  import android.os.Bundle;
30  import android.util.Log;
31
32  import androidx.appcompat.app.AppCompatActivity;
33
34  import java.io.OutputStream;
35  import java.net.Socket;
```

```
36
37
38  public class VoskActivity extends Activity implements
39          RecognitionListener {
40
41      static private final int STATE_START = 0;
42      static private final int STATE_READY = 1;
43      static private final int STATE_DONE = 2;
44      static private final int STATE_FILE = 3;
45      static private final int STATE_MIC = 4;
46
47      /* Used to handle permission request */
48      private static final int
            PERMISSIONS_REQUEST_RECORD_AUDIO = 1;
49
50      private Model model;
51      private SpeechService speechService;
52      private SpeechStreamService speechStreamService;
53      private TextView resultView;
54
55
56      private static final String TAG = "SocketClient";
57      private static final String RASPBERRY_PI_IP =
            "192.168.1.27"; // Remplacez par l'IP du Raspberry Pi
58      // private static final String RASPBERRY_PI_IP =
            "192.168.1.30 "; // IP du raspberry ecran
59      private static final int PORT = 12345; // Doit
            correspondre au port du serveur
60      private OutputStream outputStream;
61      private Socket socket;
62
63      @Override
64      public void onCreate(Bundle state) {
65          super.onCreate(state);
66          setContentView(R.layout.main);
67
68          // Setup layout
69          resultView = findViewById(R.id.result_text);
70
71          setUiState(STATE_START);
72
73          findViewById(R.id.recognize_file).setOnClickListener(view
                -> recognizeFile());
74          findViewById(R.id.recognize_mic).setOnClickListener(view
                -> recognizeMicrophone());
75          ((ToggleButton)
                findViewById(R.id.pause)).setOnCheckedChangeListener((view,
                isChecked) -> pause(isChecked));
76
```

```
77          LibVosk.setLogLevel(LogLevel.INFO);
78
79          // Check if user has given permission to record
                audio, init the model after permission is granted
80          int permissionCheck =
                ContextCompat.checkSelfPermission(getApplicationContext(),
                Manifest.permission.RECORD_AUDIO);
81          if (permissionCheck !=
                PackageManager.PERMISSION_GRANTED) {
82              ActivityCompat.requestPermissions(this, new
                    String[]{Manifest.permission.RECORD_AUDIO},
                    PERMISSIONS_REQUEST_RECORD_AUDIO);
83          } else {
84              initModel();
85          }
86      }
87
88
89      private void initModel() {
90          StorageService.unpack(this, "model-en-us", "model",
91                  (model) -> {
92                      this.model = model;
93                      setUiState(STATE_READY);
94                  },
95                  (exception) -> setErrorState("Failed to
                        unpack the model" +
                        exception.getMessage()));
96      }
97
98      @Override
99      public void onRequestPermissionsResult(int requestCode,
100                                             @NonNull String[]
                                                    permissions,
                                                    @NonNull int[]
                                                    grantResults) {
101         super.onRequestPermissionsResult(requestCode,
                permissions, grantResults);
102
103         if (requestCode == PERMISSIONS_REQUEST_RECORD_AUDIO)
                {
104             if (grantResults.length > 0 && grantResults[0]
                    == PackageManager.PERMISSION_GRANTED) {
105                 // Recognizer initialization is a
                        time-consuming and it involves IO,
106                 // so we execute it in async task
107                 initModel();
108             } else {
109                 finish();
110             }
```

```
111             }
112         }
113 
114         @Override
115         public void onDestroy() {
116             super.onDestroy();
117 
118             if (speechService != null) {
119                 speechService.stop();
120                 speechService.shutdown();
121             }
122 
123             if (speechStreamService != null) {
124                 speechStreamService.stop();
125             }
126         }
127 
128         @Override
129         public void onResult(String hypothesis) {
130             resultView.append(hypothesis + "\n");
131             new Thread(() -> sendToServer(hypothesis)).start();
132         }
133 
134         @Override
135         public void onFinalResult(String hypothesis) {
136             resultView.append(hypothesis + "\n");
137             new Thread(() -> sendToServer(hypothesis)).start();
138             setUiState(STATE_DONE);
139             if (speechStreamService != null) {
140                 speechStreamService = null;
141             }
142         }
143 
144         @Override
145         public void onPartialResult(String hypothesis) {
146             // resultView.append(hypothesis + "aaa\n");
147         }
148 
149         @Override
150         public void onError(Exception e) {
151             setErrorState(e.getMessage());
152         }
153 
154         @Override
155         public void onTimeout() {
156             setUiState(STATE_DONE);
157         }
158 
159         private void setUiState(int state) {
```

```
160         switch (state) {
161             case STATE_START:
162                 resultView.setText(R.string.preparing);
163                 resultView.setMovementMethod(new
                        ScrollingMovementMethod());
164                 findViewById(R.id.recognize_file).setEnabled(false);
165                 findViewById(R.id.recognize_mic).setEnabled(false);
166                 findViewById(R.id.pause).setEnabled((false));
167                 break;
168             case STATE_READY:
169                 resultView.setText(R.string.ready);
170                 ((Button)
                        findViewById(R.id.recognize_mic)).setText(R.string.recogniz
171                 findViewById(R.id.recognize_file).setEnabled(true);
172                 findViewById(R.id.recognize_mic).setEnabled(true);
173                 findViewById(R.id.pause).setEnabled((false));
174                 break;
175             case STATE_DONE:
176                 ((Button)
                        findViewById(R.id.recognize_file)).setText(R.string.recogni
177                 ((Button)
                        findViewById(R.id.recognize_mic)).setText(R.string.recogniz
178                 findViewById(R.id.recognize_file).setEnabled(true);
179                 findViewById(R.id.recognize_mic).setEnabled(true);
180                 findViewById(R.id.pause).setEnabled((false));
181                 ((ToggleButton)
                        findViewById(R.id.pause)).setChecked(false);
182                 break;
183             case STATE_FILE:
184                 ((Button)
                        findViewById(R.id.recognize_file)).setText(R.string.stop_fi
185                 resultView.setText(getString(R.string.starting));
186                 findViewById(R.id.recognize_mic).setEnabled(false);
187                 findViewById(R.id.recognize_file).setEnabled(true);
188                 findViewById(R.id.pause).setEnabled((false));
189                 break;
190             case STATE_MIC:
191                 ((Button)
                        findViewById(R.id.recognize_mic)).setText(R.string.stop_mic
192                 resultView.setText(getString(R.string.say_something));
193                 findViewById(R.id.recognize_file).setEnabled(false);
194                 findViewById(R.id.recognize_mic).setEnabled(true);
195                 findViewById(R.id.pause).setEnabled((true));
196                 break;
197             default:
198                 throw new IllegalStateException("Unexpected
                        value: " + state);
199         }
200     }
```

```
201
202     private void setErrorState(String message) {
203         resultView.setText(message);
204         ((Button)
                findViewById(R.id.recognize_mic)).setText(R.string.recognize_micropl
205         findViewById(R.id.recognize_file).setEnabled(false);
206         findViewById(R.id.recognize_mic).setEnabled(false);
207     }
208
209     private void recognizeFile() {
210         if (speechStreamService != null) {
211             setUiState(STATE_DONE);
212             speechStreamService.stop();
213             speechStreamService = null;
214         } else {
215             setUiState(STATE_FILE);
216             try {
217                 Recognizer rec = new Recognizer(model,
                        16000.f, "[\"one zero zero zero one\", " +
218                         "\"oh zero one two three four five
                                six seven eight nine\",
                                \"[unk]\"]");
219
220                 InputStream ais = getAssets().open(
221                         "10001-90210-01803.wav");
222                 if (ais.skip(44) != 44) throw new
                        IOException("File too short");
223
224                 speechStreamService = new
                        SpeechStreamService(rec, ais, 16000);
225                 speechStreamService.start(this);
226             } catch (IOException e) {
227                 setErrorState(e.getMessage());
228             }
229         }
230     }
231
232     private void recognizeMicrophone() {
233         if (speechService != null) {
234             setUiState(STATE_DONE);
235             speechService.stop();
236             speechService = null;
237         } else {
238             setUiState(STATE_MIC);
239             try {
240                 Recognizer rec = new Recognizer(model,
                        16000.0f);
241                 speechService = new SpeechService(rec,
                        16000.0f);
```

```
                speechService.startListening(this);
            } catch (IOException e) {
                setErrorState(e.getMessage());
            }
        }
    }
    private void sendToServer(String text) {
        try (Socket socket = new Socket(RASPBERRY_PI_IP,
            PORT);
             OutputStream outputStream =
                socket.getOutputStream()) {

            outputStream.write(text.getBytes());
            outputStream.flush();

            Log.d(TAG, "Texte envoye au serveur : " + text);

        } catch (Exception e) {
            Log.e(TAG, "Erreur lors de l'envoi au serveur :
                " + e.getMessage());
        }
    }

    private void pause(boolean checked) {
        if (speechService != null) {
            speechService.setPause(checked);
        }
    }

}
```

Listing 1: Client Android

7.5 Annexe 5: Application Python sur Raspberry ou PC

```
import cv2
import torch
import pyrealsense2 as rs
import numpy as np
import pyttsx3  # synthese vocale
import os
import pyaudio
import threading
import queue
from models.experimental import attempt_load
```

```
11 from utils.general import non_max_suppression
12 from vosk import Model, KaldiRecognizer
13 import socket
14 import json
15
16 ##########################################################################
17 #                                                dictionnaire
       des objets detectable par YOLO
     #
18 ##########################################################################
19 translations = {
20     "person": "personne",
21     "bicycle": "velo",
22     "car": "voiture",
23     "motorbike": "moto",
24     "aeroplane": "avion",
25     "bus": "bus",
26     "train": "train",
27     "truck": "camion",
28     "boat": "bateau",
29     "traffic light": "feu de signalisation",
30     "fire hydrant": "borne d'incendie",
31     "stop sign": "panneau stop",
32     "parking meter": "parcmetre",
33     "bench": "banc",
34     "bird": "oiseau",
35     "cat": "chat",
36     "couch": "canape",
37     "dog": "chien",
38     "horse": "cheval",
39     "sheep": "mouton",
40     "cow": "vache",
41     "elephant": "elephant",
42     "bear": "ours",
43     "zebra": "zebre",
44     "giraffe": "girafe",
45     "backpack": "sac a dos",
46     "umbrella": "parapluie",
47     "handbag": "sac a main",
48     "tie": "cravate",
49     "suitcase": "valise",
50     "frisbee": "frisbee",
51     "skis": "skis",
```

```
    "snowboard": "planche de snowboard",
    "sports ball": "balle de sport",
    "kite": "cerf-volant",
    "baseball bat": "batte de baseball",
    "baseball glove": "gant de baseball",
    "skateboard": "skateboard",
    "surfboard": "planche de surf",
    "tennis racket": "raquette de tennis",
    "bottle": "bouteille",
    "wine glass": "verre a vin",
    "cup": "tasse",
    "fork": "fourchette",
    "knife": "couteau",
    "spoon": "cuillere",
    "bowl": "bol",
    "banana": "banane",
    "apple": "pomme",
    "sandwich": "sandwich",
    "orange": "orange",
    "broccoli": "brocoli",
    "carrot": "carotte",
    "hot dog": "hot dog",
    "pizza": "pizza",
    "donut": "beignet",
    "cake": "gateau",
    "chair": "chaise",
    "sofa": "canape",
    "pottedplant": "plante en pot",
    "bed": "lit",
    "diningtable": "table a manger",
    "toilet": "toilettes",
    "tvmonitor": "ecran de television",
    "laptop": "ordinateur portable",
    "mouse": "souris",
    "remote": "telecommande",
    "keyboard": "clavier",
    "cell phone": "telephone portable",
    "microwave": "micro-ondes",
    "oven": "four",
    "toaster": "grille-pain",
    "sink": "evier",
    "refrigerator": "refrigerateur",
    "book": "livre",
```

```
    "clock": "horloge",
    "vase": "vase",
    "scissors": "ciseaux",
    "teddy bear": "ours en peluche",
    "hair drier": "seche-cheveux",
    "toothbrush": "brosse a dents",
    "tv": "television"
}


#############################################################################
#                                        Les ilots de confiance pour la reconnaissance vocale                          #
#############################################################################


ilots_de_confiance={
    "voir": ["voir", "apercevoir", "objets", "detecter", "montrer", "autour"],
    "aider" : ["aider", "expliquer", "aide"],
    "quitter" : ["quitter", "stop", "sortir", "arreter"],
    "proche" : ["proche", "pres"],
    "loin" : ["loin", "lointain", "eloigne"],
    "personne": ["personne", "vois-tu"],
    "girafe": ["girafe", "vois-tu"],
    "oiseau": ["oiseau", "vois-tu"],
    "clavier": ["clavier", "vois-tu"],
    "souris": ["souris", "vois-tu"],
    "voiture": ["voiture", "vois-tu"],
    "cheval": ["cheval", "vois-tu"],
    "ciseaux": ["ciseaux", "vois-tu"],
    "bouteille": ["bouteille", "vois-tu"],
    "bonjour": ["bonjour", "salut"],
    "micro": ["micro", "microphone"],
    "micro_ouvert": ["micro", "ouvert"],
    "micro_ferme": ["micro","ferme"]

}

#############################################################################
#        Initialisation du modele de la bibliotheque Vosk pour la reconnaissance de la parole
```

```
                                #
134 ##############################################################################
135
136
137 def initialize_vosk_model(model_path):
138     if not os.path.exists(model_path):
139         raise FileNotFoundError(f"Le modele specifie a
                {model_path} est introuvable.")
140     print("Chargement du modele Vosk...")
141     model = Model(model_path)
142     print("Modele charge avec succes.")
143     return model
144
145
146 ##############################################################################
147 #                                        Dire une phrase
    par synthese vocale
                                                 #
148 ##############################################################################
149
150 def speak(text):
151     engine = pyttsx3.init()
152     engine.setProperty("voice", "french")
153     engine.setProperty("rate", 150)
154     engine.say(text)
155     engine.runAndWait()
156
157
158 ##############################################################################
159 #
    Raspberry ou Android

    #
160 ##############################################################################
161
162 def recognize_speech(vosk_model,q, choix):
163     if choix == 1:
164         recognizer = KaldiRecognizer(vosk_model, 16000)
165         mic = pyaudio.PyAudio()
166         stream = mic.open(format=pyaudio.paInt16,
167                           channels=1,
168                           rate=16000,
169                           input=True,
```

```
170                             frames_per_buffer=8192)
171         stream.start_stream()
172         print("Parlez dans le microphone...")
173         try:
174             while True:
175                 data = stream.read(4096,
                        exception_on_overflow=False)
176                 if recognizer.AcceptWaveform(data):
177                     result =
                            json.loads(recognizer.Result())
178                     q.put(result.get("text", "").lower())
179         except KeyboardInterrupt:
180             print("Arret de la reconnaissance vocale.")
181         finally:
182             stream.stop_stream()
183             stream.close()
184             mic.terminate()
185     elif choix == 2:
186         with socket.socket(socket.AF_INET,
                socket.SOCK_STREAM) as server_socket:
187             server_socket.bind((HOST, PORT))
188             server_socket.listen()
189             while True:
190                 client_socket, client_address =
                        server_socket.accept()
191                 with client_socket:
192                     data =
                            client_socket.recv(1024).decode('utf-8')
193                     if data :
194                         try:
195                             parsed_data =
                                    json.loads(data)
196                             extracted_value =
                                    parsed_data.get("text",
                                    None)
197                             if extracted_value:
198                                 q.put(extracted_value.lower())
199                         except json.JSONDecodeError:
200                             print("Failed to parse
                                    data as JSON.")
201
202
203 ##########################################################################
```

```
#
    Detection d'objets par YOLO
                                                                    #
################################################################################

def detection_objet(objets_courant) :
    global distance_min
    global distance_max
    global objet_min
    global objet_max
    global compteur_frame

    # Pretraitement pour YOLOv7
    img = cv2.resize(color_image, (640, 480))
    img = torch.from_numpy(img).permute(2, 0, 1).float()
    img /= 255.0  # Normalisation
    if img.ndimension() == 3:
        img = img.unsqueeze(0)
    # Prediction avec YOLOv7
    pred = model(img, augment=False)[0]
    pred = non_max_suppression(pred, 0.3, 0.45)  # Conf
       = 0.25, IoU = 0.45
    # Dessiner les boites englobantes et afficher la
       distance
    for i, det in enumerate(pred):  # Parcourir les
       detections
        if len(det):
            for d in det:  # d = (x1, y1, x2, y2, conf,
               cls)
                x1, y1, x2, y2, conf, cls = d[:6]
                x1, y1, x2, y2 = int(x1), int(y1),
                   int(x2), int(y2)
                # Calculer le centre de la boite
                center_x = (x1 + x2) // 2
                center_y = (y1 + y2) // 2
                # Obtenir la distance de l'objet
                distance =
                   depth_frame.get_distance(center_x,
                   center_y)
                # Afficher la boite et les informations
                label = f"{names[int(cls)]} {conf:.2f}
                   ({distance:.2f}m)"
                objet = names[int(cls)]
```

```
                    if distance <  distance_min:
                        distance_min = distance
                        objet_min = objet
                    if distance > distance_max:
                        distance_max = distance
                        objet_max = objet
                    if objet not in objets_courants:
                        objets_courants.append( objet )
                    translation = translations.get(objet,
                       "Mot non trouve")
                    cv2.rectangle(color_image, (x1, y1),
                       (x2, y2), (0, 255, 0), 2)
                    cv2.putText(color_image, label, (x1, y1
                       - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                       (0, 255, 0), 2)
        cv2.imshow("RealSense YOLOv7", color_image)


##########################################################################
#                                                    Programme
    principal
    #
##########################################################################

# Charger le modele YOLOv7
weights = 'model.pt'  # Remplacez par le chemin vers
    votre modele
model = attempt_load(weights, map_location='cpu')  #
    Charger le modele sur CPU
names = model.names  # Classes d'objets

# Initialisation de la camera Intel RealSense
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.color, 640, 480,
    rs.format.bgr8, 30)  # Flux RGB
config.enable_stream(rs.stream.depth, 640, 480,
    rs.format.z16, 30)   # Flux de profondeur
pipeline.start(config)

vosk_model_path = "vosk-model-small-fr-0.22"
#vosk_model_path =
    "vosk-model-small-fr-0.22/vosk-model-small-fr-0.22"
```

```
    # Chemin vers le modele Vosk
vosk_model = initialize_vosk_model(vosk_model_path)

# Initialiser le traducteur
# translator = Translator()
engine = pyttsx3.init()
data=""
distance_min=1.0
distance_max=0.0
objet_min=""
objets_courants=[]
sauvegarde_liste=[]
compteur_frame=0
etat_micro=True

HOST = '0.0.0.0'  # ecoutez sur toutes les interfaces reseau
PORT = 12345        # Assurez-vous que ce port correspond a celui de l'application Android

try:
    objet=""
    nb_objets=0
    q = queue.Queue()
    print("Assistant active. Parlez pour interagir avec la camera.")
    speak("Assistant active. Parlez pour interagir avec la camera.")
    # Thread pour la reconnaissance vocale
    while(True):
        choix=int(input("voulez-vous utiliser le raspBerry (1) ou Android (2) pour la reconnaissance vocale ?"))
        if choix !=1 and choix != 2:
            print("Il faut taper un ou 2")
            continue
        else :
            break
    speech_thread = threading.Thread(target=recognize_speech, args=(vosk_model,q,choix,))
    speech_thread.start()
    while True:
```

```
303             action=""
304             while not q.empty():
305                 remaining_data = q.get_nowait()
306                 action=""
307             try:
308                 data=""
309                 data = q.get(timeout=1)  # Timeout pour
                        eviter de rester bloque
310                 if data is None:  # Signal d'arret
311                     break
312             except queue.Empty:
313                 pass  # Continuer a tourner sans bloquer
314             # Capture des frames de la camera
315             frames = pipeline.wait_for_frames()
316             color_frame = frames.get_color_frame()
317             depth_frame = frames.get_depth_frame()
318             if not color_frame or not depth_frame:
319                 continue
320             # Convertir les frames en images OpenCV
321             color_image =
                    np.asanyarray(color_frame.get_data())
322             # parseur d'ilots de confiance
323             mots_vocaux = data.split()
324             action=""
325             ensemble1=set(mots_vocaux)
326             ensemble2=set()
327             nb_elements_commun=0
328             if(len(mots_vocaux) > 0):
329                 print("reconnaissance=", data)
330                 for cle, valeurs in
                        ilots_de_confiance.items():
331                     for valeur in valeurs:
332                         ensemble2.add(valeur)
333                     # print("ensemble1=", ensemble1,
                            "ensemble2=", ensemble2)
334                     communs=ensemble1 & ensemble2
335                     if len(communs) > nb_elements_commun:
336                         nb_elements_commun =
                                nb_elements_commun + 1
337                         action=cle
338                     ensemble2=set()
339             # action engendre
340             print("action = ", action)
```

```
341         if action == "micro":
342             if etat_micro == True:
343                 speak("le microphone est disponible")
344                 action=""
345             else :
346                 speak("le microphone n'est pas
                        disponible")
347                 action=""
348         elif action=="micro_ferme":
349             etat_micro=False
350             action=""
351             speak("ferme")
352         elif action=="micro_ouvert":
353             etat_micro=True
354             speak("ouvert")
355             action=""
356             data=""
357         if etat_micro==False :
358             continue
359         elif action=="voir":
360             speak("vous voulez savoir ce que l'on voit
                    ?")
361             print("les objets que l'on voit sont ")
362             liste_mots_a_dire=""
363             for value in objets_courants:
364                 nom=translations.get(value, "mot non
                        trouve")
365                 print(nom)
366                 liste_mots_a_dire+=nom
367                 liste_mots_a_dire += " "
368             speak(liste_mots_a_dire)
369             print("fin des objets reconnus")
370             data=""
371         elif action=="aider":
372             speak("""
373             Je peux detecter les objets devant vous et
                    indiquer leur distance.
374             Dites 'les objets' pour obtenir une liste
                    des objets detectes.
375             Dites 'quel est l'objet le plus proche' pour
                    connaitre l'objet le plus proche.
376             Dites 'quitte l'assistant' pour fermer
                    l'assitant.
```

```
377                 """)
378                 data=""
379             elif action=="quitter":
380                 speak("Assistant desactive. A bientot.")
381                 pipeline.stop()
382                 cv2.destroyAllWindows()
383                 speech_thread.join()
384                 exit()
385             elif action=="proche":
386                 speak("L'objet le plus proche est ")
387                 speak(translations.get(objet_min, "Mot non
                        trouve"))
388                 data=""
389                 action=""
390             elif action == "loin":
391                 speak("L'objet le plus eloigne est ")
392                 speak(translations.get(objet_max, "Mot non
                        trouve"))
393                 data=""
394                 action=""
395             elif action=="personne":
396                 speak("Vous voulez savoir si je vois une
                        personne ?")
397                 if "person" in objets_courants :
398                     speak("Oui, j'en vois une")
399                 else :
400                     speak("non, je n'en vois pas")
401             elif action=="girafe":
402                 speak("Vous me demandez s'il y a une girafe
                        ?")
403                 if  "giraffe" in objets_courants :
404                     speak("Oui, j'en vois une")
405                 else :
406                     speak("non, je n'en vois pas")
407             elif action == "voiture":
408                 speak("vous voulez savoir  si je vois une
                        voiture ?")
409                 if "car" in objets_courants:
410                     speak("Oui, j'en vois une")
411                 else:
412                     speak("non, je ne vois pas de voiture")
413             elif action == "cheval":
414                 speak("Vous me demandez s'il y a un cheval?")
```

```
415             if "horse" in objets_courants:
416                 speak("Oui, j'en vois un")
417             else:
418                 speak("non, je ne vois pas de cheval")
419         elif action=="oiseau":
420                 speak("Vous me demandez si je vois un
                        oiseau ?")
421                 if "bird" in objets_courants:
422                     speak("Oui, j'en vois un")
423                 else:
424                     speak("non, je n'en vois pas")
425         elif action=="clavier":
426             speak("Vous me demandez si je vois un
                    clavier ?")
427             if "keyboard" in objets_courants:
428                 speak("Oui, je vois un clavier")
429             else:
430                 speak("non, je ne vois pas de clavier")
431         elif action=="souris":
432             speak("Vous me demandez si je vois une
                    souris ?")
433             if "mouse" in objets_courants:
434                 speak("Oui, je vois une souris")
435             else:
436                 speak("non, je ne vois pas de souris")
437         elif action=="ciseaux":
438             speak("Vous me demandez si je vois un ciseau
                    ?")
439             if "scissors" in objets_courants:
440                 speak("Oui, je vois une paire de ciseau")
441             else:
442                 speak("non, je ne vois pas de ciseau")
443         elif action=="bouteille":
444             speak("Vous me demandez si je vois une
                    bouteille ?")
445             if "bottle" in objets_courants:
446                 speak("Oui, j'en vois une")
447             else:
448                 speak("non, je n'en vois pas")
449         elif action=="bonjour":
450             speak("bonjour, je suis votre assistant")
451         elif cv2.waitKey(1) == 27:
452             break
```

```
453             else :
454                 distance_min=1.0
455                 distance_max=0.0
456                 objet_min=""
457                 objet_max=""
458                 detection_objet(objets_courants)
459                 print("Les objets reconnus en cours sont :")
460                 for value in objets_courants:
461                     print(translations.get(value, "mot non
                        trouve"))
462                     compteur_frame = compteur_frame + 1
463                     if compteur_frame == 20:
464                         compteur_frame = 0
465                         objets_courants=[]
466             data=""
467             action=""
468     finally:
469         cv2.destroyAllWindows()
```