



RAPPORT DE PROJET COLLABORATIF

DETECT'CHUTE

Florian Lefebvre, Sofiane Lhiyat, Haoran Dong, Arthur Le Breton

05/03/2025

Master professionnel HANDI
Technologie et handicap

I- Introduction	3
I-1 Problématique	3
I-2 Enjeux et solutions	4
I-3 Matériels et Logiciels	5
I-4 Matrice de RACI	7
II- Partie sofiane	8
II-1 Architecture globale du système	8
II-2 Développement logiciel	9
II-3 Communication et système d'alerte	11
II-4 Application Android	13
II-5 Sécurité et fiabilité	16
III- Détection de chute avec Mediapipe	19
III-1 La détection d'une chute	19
III-2 L'approche basée sur la vision par ordinateur	19
III-3 Notre approche : le framework Mediapipe	20
III-4 Le processus utilisé par Mediapipe	21
III-5 Première version avec Mediapipe et Python	23
III-6 Utilisation d'une bibliothèque de vidéo de chute ou non chute	24
III-7 Paramétrage externe du programme	25
III-8 Analyse de la dernière version du projet	27
IV- Rapport Arthur suite Florian	29
IV-1 Limites de la première version du système de détection	29
IV-2 Vers une solution hybride : intégration d'un modèle DNN	29
IV-3 Prétraitement des données et constitution du dataset	29
IV-4 Conception et entraînement du modèle DNN	30
IV-5 Détection en temps réel et intégration sur plateforme embarquée	32
IV-6 Évaluation des performances et résultats	33
V- Détection de chute avec les capteurs de pression au sol	34
V-1 Le choix du matériel	34
V-2 Le fonctionnement des capteurs	35
V-3 Le montage final	36
V-4 Logiciel, Code	37
V-5 Algorithmes pour la détection de chute par les tapis	38
V-6 Mécanisme d'interaction tapis-détection	39
V-7 Interprétation et conclusion	40
VI - Conclusion et perspectives	41
VI- 1 Conclusion	41
VI-2 Perspectives	41

I- Introduction

I-1 Problématique

Selon les chiffres du ministère de la Santé, chaque année, 2 millions de chutes de personnes âgées de plus de 65 ans sont responsables de 10 000 décès, la première cause de mortalité accidentelle, et de plus de 150 000 hospitalisations. Les chutes ont des conséquences physiques, psychologiques, sociales et réduisent la qualité de vie des individus. Elles constituent par ailleurs une rupture dans le parcours de vie des sujets âgés sur le plan de l'autonomie.

Notre public cible est toutes les personnes susceptibles de chuter (personnes âgées, fragilisées, en situation de handicap). Ces personnes peuvent avoir une différence au niveau cognitif ou corporel, donc notre système doit leur permettre de signaler leur chute immédiatement et automatiquement à l'aidant(e) ou à leur proche sans nécessiter leurs actions.

L'objectif de ce projet est de détecter la chute d'une personne dans le cadre du domicile (lieu d'habitation, maison de retraite, Indoor). ça peut être dans la cuisine, au salon, dans leur chambre ou dans la salle de bain. On fait l'attention que c'est surtout dans la salle de bain que le problème de confidentialité est très critique, car c'est pas tout le monde qui accepte d'être filmé 24/24h, sauf s'ils en ont vraiment le besoin. La caméra normale est un capteur très intrusif.

On va relever des informations sur la personne à l'aide de caméras infrarouges et de capteurs environnementaux (capteurs de pression). Nous allons croiser ces données afin de détecter la chute avec fiabilité. Si une suspicion de chute est détectée, le système va envoyer lui-même l'alerte à un aidant décidé par le patient.

L'enjeu principal du projet est de développer une solution fiable et accessible pour détecter les chutes à domicile tout en répondant aux besoins de sécurité, d'autonomie et de respect de la vie privée des utilisateurs. Ce projet soulève également des défis éthiques, ergonomiques et technologiques, en intégrant des dispositifs peu intrusifs et adaptés aux capacités des utilisateurs, tout en assurant un haut niveau de fiabilité et d'acceptabilité.

I-2 Enjeux et solutions

La détection des chutes est essentielle pour garantir la sécurité des personnes vulnérables, notamment les personnes âgées ou à mobilité réduite. Pour cela, nous utilisons une combinaison de caméras infrarouges et de capteurs intégrés dans des tapis de pression au sol. Ces deux technologies permettent de croiser les données afin d'augmenter la fiabilité de la détection des chutes. Si une chute est suspectée, le système déclenche automatiquement une alerte vers un aidant désigné, permettant une intervention rapide.

Installation et fonctionnement des capteurs :

Le protocole d'installation est essentiel pour maximiser l'efficacité des dispositifs. Le tapis de pression sera placé dans des zones identifiées comme étant à risque, par exemple à côté du lit. En parallèle, la caméra sera positionnée sur un mur dans la même pièce, donnant de la visibilité sur le lit.

Le tapis dispose d'un câble avec quatre fils :

- Les deux fils à droite : normalement ouverts et se ferment sous l'effet d'une pression.
- Les deux fils à gauche : activent l'envoi d'une alarme en cas de besoin.

Algorithme d'analyse de la caméra et intégration du tapis

La caméra infrarouge capture l'image de la personne et le programme identifie des points spécifiques sur le corps (par exemple, la tête, les épaules, les hanches, et les pieds). Ces points sont utilisés pour analyser les mouvements et détecter des chutes avec précision.

Les critères analysés incluent :

- La différence entre deux images successives (I et $I+1$) pour repérer les changements rapides de position.
- La position verticale du corps : un changement brutal peut indiquer une chute.
- La position finale et l'orientation du corps par rapport au sol.
- La distance des points clés par rapport au sol.
- Les paramètres de vitesse et d'accélération.

I-3 Matériels et Logiciels

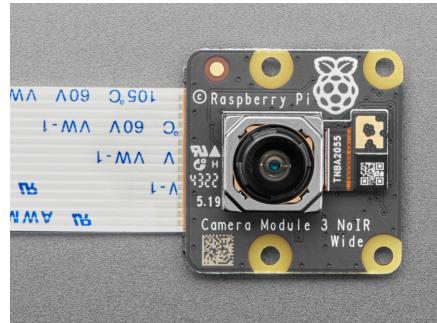


Figure 1: Raspberry Pi Camera Module 3 Wide NoIR - 12MP 120 Degree - Wide Angle Infrared Lens



Figure 2 : Tapis de capteur de pression utilisé dans le prototypage

<https://fr.farnell.com/defender-security/pm3-pk/pressure-mat-large-720x560mm/dp/1146764>
<https://fr.farnell.com/multicomp-pro/pm1-pk/pressure-mat-stair-595x170mm/dp/1146761>

Ces tapis peuvent soit confirmer avec la caméra, soit détecter une chute en son propre dans l'angle mort de la caméra. Dans la première commande, on a pris qu'un tapis de grande taille pour vérifier son bon fonctionnement dans le cadre de notre projet. Dans la seconde commande des tapis, nous avons choisi les tapis en petites tailles et en plus grande quantité afin d'augmenter le nombre de zones de détection et ainsi améliorer la précision du système.



Figure 3: breadboard, platine de prototypage fils résistance carte esp32 s3 mini Fer à souder

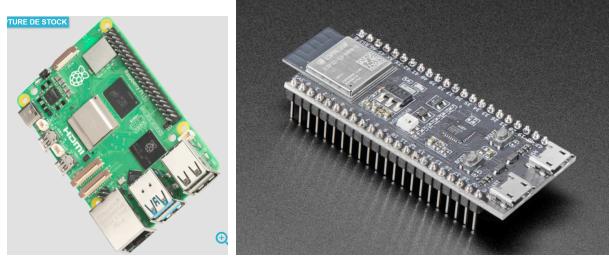


Figure 4 : Raspberry Pi 5 et ESP32 S3 Mini



Figure 5 : logiciel: Mu Editor



Figure 6 : bibliothèques : Mediapipe, Flask, gTTS, OpenCV

I-4 Matrice de RACI

Tâches	Projet : Objectifs & Activités	Arthur	Florian	Haoran	Sofiane
1	Expression du besoin	R	C	C	C
2	Rédaction du cahier des charges	A	C	C	R
3	Rédaction du dossier d'analyse	A	I	R	C
4	Programmation et test de l'algorithme de détection de chute	A	R	C	I
5	Programmation et test du tapis capteur de pression	A	C	R	I
6	Fusion des données	R	C	C	A
7	Mise en place de l'infrastructure réseaux	C	I	A	R
8	Test et validation finale	R	A	A	A

R : Réalisateur / Réalisatrice

A : Approbateur / Approbatrice

C : Consulté.e

I : Informé.e

II- Partie sofiane

II-1 Architecture globale du système

Vue d'ensemble du système

Le système de détection de chutes repose sur une architecture distribuée comprenant cinq composants principaux interconnectés. Au cœur du dispositif se trouve une unité de traitement composée d'un Raspberry Pi couplé à un ESP32. Cette unité centralise les données provenant d'une caméra et d'un ensemble de tapis de pression placés stratégiquement dans l'environnement de la personne. La caméra est chargée de l'analyse visuelle des postures et mouvements, tandis que les tapis captent les pressions au sol pouvant indiquer une chute. Un serveur Flask assure la communication entre l'unité de traitement et l'application mobile destinée à l'aidant. Cette architecture hybride combine deux méthodes de détection complémentaires pour maximiser la fiabilité et minimiser les faux positifs, tout en permettant une interaction bidirectionnelle entre tous les acteurs du système.

Flux de données

Le flux de données au sein du système suit plusieurs chemins précis et complémentaires. La caméra transmet en continu un flux vidéo au Raspberry Pi qui l'analyse grâce à MediaPipe pour détecter les anomalies de posture et de mouvement. Simultanément, les tapis de pression communiquent leur état à l'ESP32, signalant toute pression prolongée au sol. Lorsqu'une chute potentielle est identifiée, l'unité de traitement envoie une notification au serveur Flask accompagnée d'un message audio généré par Google Text-to-Speech. Le serveur Flask relaye ces informations vers l'application mobile sous forme d'alertes et de fichiers audio. En réponse, l'aidant peut confirmer la réception de l'alerte via l'application et enregistrer un message vocal qui sera transmis au serveur puis diffusé par le Raspberry Pi. Ce cycle bidirectionnel d'information se termine par une demande de réinitialisation que l'aidant peut initier une fois l'incident pris en charge, ramenant le système à son état de surveillance normal.

Communication entre les composants

La communication entre les différents éléments s'effectue via plusieurs protocoles adaptés à chaque type d'interaction. Au niveau local, la caméra est reliée au Raspberry Pi par une connexion USB directe permettant un transfert rapide et fiable du flux vidéo. Les tapis de pression sont connectés à l'ESP32 via des entrées GPIO, transmettant des signaux binaires indiquant leur état d'activation. Entre le Raspberry Pi, l'ESP32 et le serveur Flask, la communication s'établit sur le réseau local par des requêtes HTTP formatées selon une API REST spécifiquement conçue. Cette API expose plusieurs endpoints permettant l'échange de notifications, de fichiers audio et de commandes de contrôle. L'application mobile communique également avec le serveur Flask via des requêtes HTTP, récupérant les alertes et envoyant les confirmations d'écoute ainsi que les messages vocaux de l'aidant. L'ensemble de ces interactions est optimisé pour minimiser la latence, particulièrement critique dans un contexte d'urgence, tout en maintenant une fiabilité maximale dans la transmission des informations essentielles.

Modèle de fonctionnement

Le fonctionnement global du système s'articule autour d'un cycle de surveillance et d'alerte parfaitement orchestré. En mode normal, le système effectue une surveillance continue de l'environnement, analysant en temps réel les données de la caméra et des tapis de pression. L'algorithme de détection applique une logique de vérification croisée: lorsqu'un événement suspect est identifié par l'un des capteurs, une confirmation est recherchée auprès de l'autre source de données. Cette approche réduit considérablement les faux positifs tout en maintenant une haute sensibilité aux incidents réels. Une fois la chute confirmée, le système entre en phase d'alerte: un message vocal est diffusé localement pour rassurer la personne tandis qu'une notification est instantanément transmise à l'aide via l'application mobile. S'engage alors une phase d'interaction où l'aide peut communiquer vocalement avec la personne fragilisée par l'intermédiaire du système audio du Raspberry Pi. Cette communication bidirectionnelle permet d'évaluer la gravité de la situation et d'apporter un premier soutien psychologique en attendant une intervention physique si nécessaire. Après résolution de l'incident, l'aide peut déclencher la réinitialisation du système qui reprend alors son cycle de surveillance, enrichi des données collectées pour améliorer continuellement sa précision.

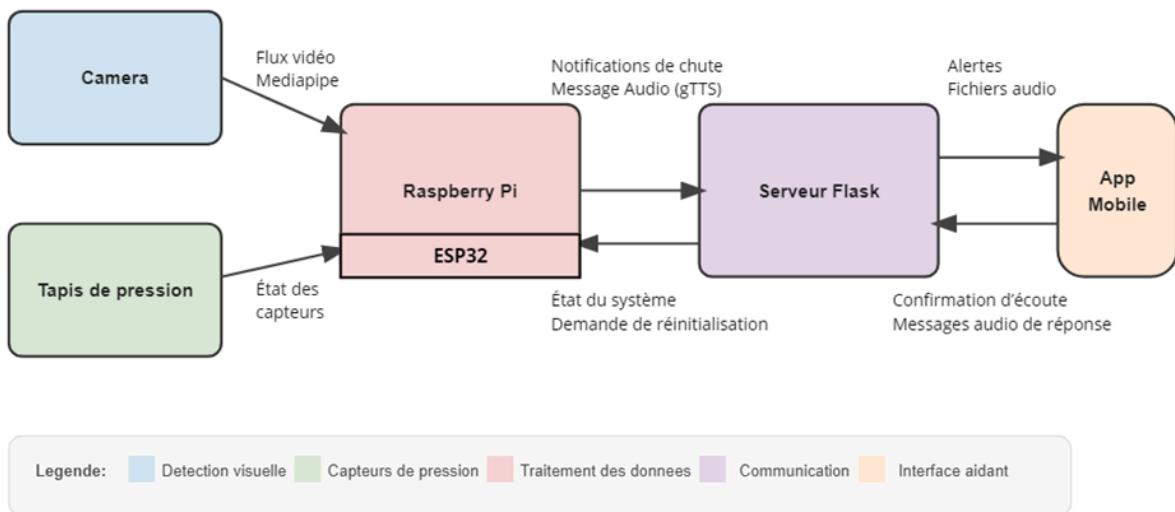


Figure 7: Architecture globale du système

II-2 Développement logiciel

Backend (Python/Flask)

Le backend du système repose sur un serveur Flask développé en Python, choisi pour sa légèreté et sa flexibilité. Ce serveur constitue le cœur logiciel du système, orchestrant les échanges entre tous les composants et gérant le traitement des données. L'architecture du serveur a été conçue selon un principe modulaire, séparant clairement les différentes

responsabilités en composants distincts. Le serveur gère notamment la réception des alertes de chute, leur traitement, et leur transmission vers l'application mobile. Il implémente également un système de threading pour gérer efficacement les opérations intensives comme la génération des messages vocaux et le traitement des fichiers audio, évitant ainsi tout blocage qui pourrait nuire à la réactivité du système en situation d'urgence. Une attention particulière a été portée à la gestion de la mémoire, avec un nettoyage périodique des ressources temporaires et une limitation du stockage aux informations essentielles, garantissant ainsi la stabilité du serveur même après de longues périodes d'activité.

API REST

L'API REST du système expose plusieurs endpoints soigneusement définis pour faciliter la communication entre les différents composants. Parmi les plus importants figurent l'endpoint "/health" qui permet de vérifier l'état du serveur, "/tapis_detection" qui reçoit et traite les données provenant des capteurs de pression, "/alert_available" qui informe la disponibilité d'une nouvelle alerte, "/get_audio" qui permet à l'application mobile de récupérer les fichiers audio d'alerte, et "/notify_listened" qui permet à l'aide de confirmer la prise en compte d'une chute. Les échanges de données suivent un format JSON standardisé, facilitant l'interopérabilité entre les différents composants du système. Chaque requête est validée à la réception pour assurer l'intégrité des données et prévenir toute erreur de traitement. Les réponses du serveur incluent systématiquement des codes d'état HTTP appropriés, accompagnés de messages explicites en cas d'erreur, permettant ainsi une gestion robuste des exceptions au niveau des clients.

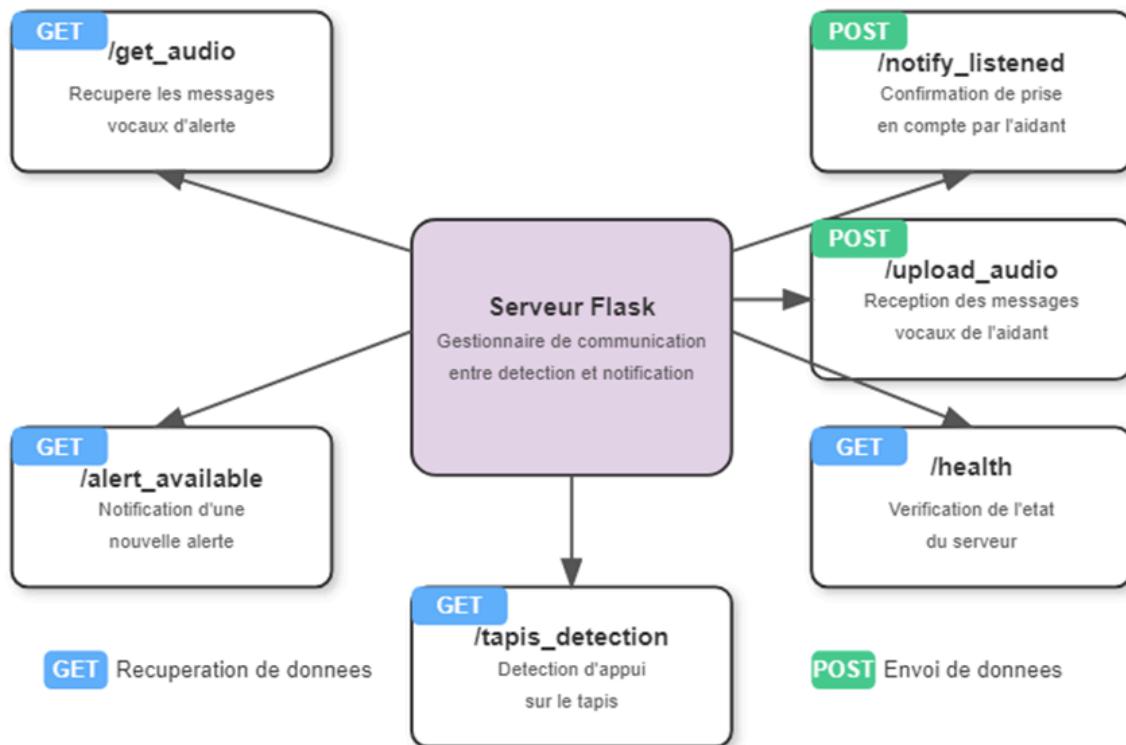


Figure 8 : Endpoints principaux

Gestion des requêtes

La gestion des requêtes a été conçue pour garantir robustesse et fiabilité, même dans des conditions défavorables. Chaque requête entrante est validée pour vérifier la présence et la cohérence des données attendues. Le système implémente des mécanismes sophistiqués de gestion des erreurs, avec différents niveaux de traitement adaptés à la nature de chaque problème potentiel. Pour les erreurs matérielles, comme les problèmes de caméra ou les dysfonctionnements des capteurs, le système peut basculer temporairement vers un mode dégradé privilégiant la source de données encore fonctionnelle. Pour les erreurs de communication réseau, un système de retransmission automatique avec backoff exponentiel permet de maintenir le service même en cas d'instabilité temporaire. La gestion de la concurrence est assurée par un système de verrouillage qui synchronise l'accès aux ressources partagées, particulièrement important lors du traitement simultané de plusieurs alertes. Des timeouts adaptatifs ont également été implémentés, s'ajustant automatiquement en fonction de la charge du système pour garantir une réactivité optimale tout en évitant les abandons prématurés.

II-3 Communication et système d'alerte

Protocoles de communication

Le système de détection de chutes implémente plusieurs protocoles de communication soigneusement sélectionnés pour garantir fiabilité et rapidité dans la transmission des alertes. Au niveau local, la communication entre le Raspberry Pi et l'ESP32 s'effectue via une connexion série UART, offrant un canal de communication bidirectionnel avec une faible latence. Les échanges entre le système de traitement et le serveur Flask reposent sur le protocole HTTP avec une API REST, choisi pour sa robustesse et sa compatibilité universelle. Les requêtes sont standardisées et utilisent des verbes HTTP appropriés selon la nature de l'opération : GET pour récupérer des données comme l'état du système, POST pour envoyer des informations comme les alertes de chute, et PUT pour les mises à jour d'état. Pour garantir la fiabilité des communications en conditions réelles, le système implémente des mécanismes de reconnexion automatique avec backoff exponentiel, permettant de maintenir le service même en cas d'instabilité temporaire du réseau. Une gestion avancée des timeouts a également été mise en place, optimisant la réactivité du système tout en évitant les abandons prématurés de requêtes critiques.

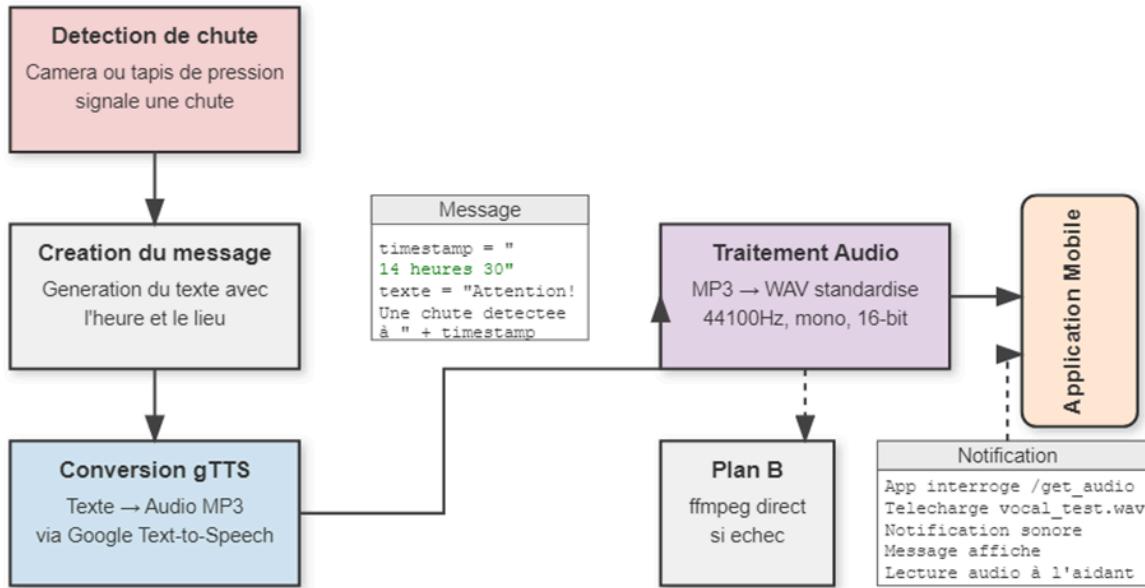


Figure 9 : Gestion des notifications vocales

Génération des messages d'alerte

La génération des messages d'alerte constitue un élément central du système, permettant une communication claire et efficace entre tous les acteurs impliqués. Lorsqu'une chute est détectée et confirmée, le système déclenche immédiatement la création d'un message d'alerte comprenant plusieurs informations essentielles : l'horodatage précis de l'événement, la localisation de la chute si plusieurs zones sont surveillées, et des détails sur le mouvement détecté pour aider à évaluer la gravité potentielle de l'incident. Le message textuel est formaté de manière claire et concise pour une compréhension immédiate par l'aideant. Il est ensuite transmis à l'application mobile où il apparaît sous forme de notification prioritaire, conçue pour attirer l'attention même si le téléphone est en mode silencieux. Parallèlement, un message vocal est généré pour communiquer directement avec la personne ayant chuté, la rassurant sur le fait qu'une alerte a été envoyée et qu'une assistance est en route. Ce double canal de communication assure que l'information circule efficacement vers tous les destinataires concernés, maximisant ainsi les chances d'une intervention rapide et adaptée.

Conversion texte-parole

Le système intègre un module sophistiqué de conversion texte-parole utilisant la bibliothèque Google Text-to-Speech (gTTS) pour générer des messages audio de haute qualité. Cette fonctionnalité est essentielle pour communiquer efficacement avec la personne ayant chuté et pour transmettre les messages vocaux de l'aideant. Lorsqu'une chute est détectée, le système génère automatiquement un message textuel contextuel qui est converti en fichier audio au format MP3, puis transformé en WAV pour une compatibilité optimale avec tous les composants. La voix synthétisée est configurée en français avec des paramètres optimisés pour la clarté et l'intelligibilité, particulièrement importantes dans un contexte de stress. Le processus de conversion intègre également des mécanismes de gestion d'erreurs robustes, avec des solutions de repli en cas d'échec de la conversion

primaire. Par exemple, si la conversion directe en WAV échoue, le système tente une approche alternative utilisant une conversion en deux étapes via un fichier temporaire, assurant ainsi la production d'un message audio même dans des conditions défavorables. Une attention particulière a été portée à l'optimisation des temps de traitement, cruciale pour maintenir la réactivité globale du système en situation d'urgence.

Mécanismes de confirmation

Les mécanismes de confirmation constituent un aspect fondamental du système, assurant que chaque alerte est bien reçue et traitée. Lorsqu'une alerte est envoyée à l'application mobile, le système attend une confirmation explicite de l'aide pour s'assurer que l'information a bien été prise en compte. Cette confirmation s'effectue via l'interface de l'application à travers un bouton dédié "Confirmer écoute". Dès réception de cette confirmation, le serveur Flask enregistre l'événement et génère automatiquement un message vocal de confirmation qui est diffusé auprès de la personne ayant chuté, l'informant que son aide a bien reçu l'alerte et qu'une assistance est en cours. Ce cycle de confirmation assure une communication transparente entre tous les acteurs impliqués, réduisant significativement l'anxiété associée à l'attente d'aide après une chute. En l'absence de confirmation dans un délai configurable, le système peut intensifier progressivement les alertes, par exemple en augmentant le volume des notifications ou en élargissant le cercle des aidants contactés. Une fois l'incident résolu, le système nécessite une réinitialisation manuelle via l'application ou directement sur le dispositif physique, garantissant ainsi qu'aucune situation d'urgence ne reste non traitée par erreur.

II-4 Application Android

Architecture de l'application

L'application Android est conçue selon le modèle d'architecture MVVM (Model-View-ViewModel), offrant une séparation claire des responsabilités et facilitant la maintenance et l'évolution du code. La couche Model définit les structures de données fondamentales de l'application, notamment les classes VoiceMessage qui représente un message vocal avec ses métadonnées comme la date, l'heure et l'état d'écoute. La couche View comprend les Activities et Fragments qui constituent l'interface utilisateur visible, organisée autour d'une activité principale (MainActivity) qui coordonne les différentes fonctionnalités et gère les transitions entre les écrans. La couche ViewModel fait le lien entre les données et l'interface utilisateur, gérant la logique métier sans dépendance directe aux composants Android spécifiques. Cette séparation permet notamment de maintenir l'état de l'application lors des changements de configuration comme les rotations d'écran. La communication réseau est encapsulée dans une couche Service utilisant la bibliothèque Retrofit pour les appels API REST, isolant ainsi la complexité des échanges HTTP du reste de l'application. L'ensemble est complété par un service en arrière-plan (VoiceMessageService) qui fonctionne même lorsque l'application n'est pas au premier plan, assurant une surveillance continue des nouvelles alertes.

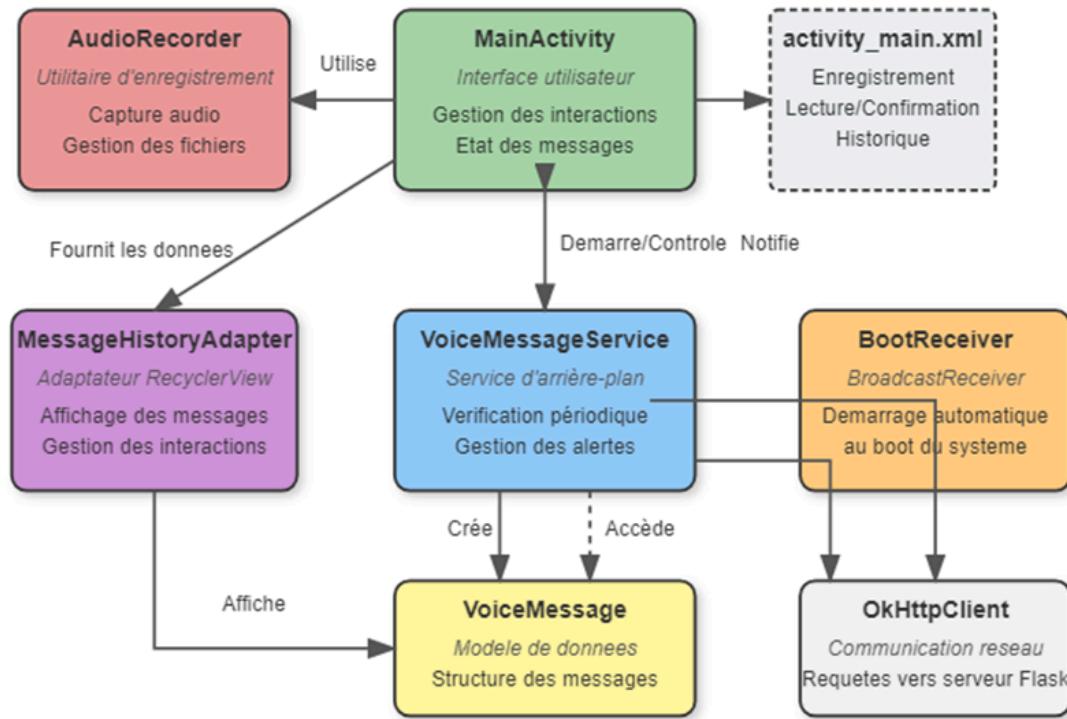


Figure 10 : Architecture Java

Interface utilisateur

L'interface utilisateur de l'application a été développée avec un accent particulier sur la simplicité et l'efficacité, garantissant une prise en main intuitive même en situation de stress. L'écran principal présente une vue épurée dominée par une notification claire en cas d'alerte de chute, accompagnée de boutons d'action aux fonctions explicites : "Écouter", "Arrêter l'alarme" et "Confirmer écoute". Ces éléments interactifs sont dimensionnés généreusement pour faciliter leur utilisation dans l'urgence. Sous cette section prioritaire se trouve un historique des messages organisé chronologiquement, permettant à l'aideant de consulter les incidents passés avec leurs horodatages précis. Chaque entrée de l'historique est interactive et ouvre, lorsque sélectionnée, un écran de détail avec les informations complètes sur l'événement et la possibilité de réécouter le message associé. Une attention particulière a été portée aux retours visuels et haptiques, avec des codes couleurs cohérents et des vibrations distinctives qui permettent d'identifier immédiatement la nature et l'urgence des notifications reçues. L'interface implémente également des animations subtiles qui guident l'utilisateur à travers les différentes étapes du processus d'assistance, fluidifiant ainsi l'expérience tout en maintenant l'attention sur les éléments essentiels.

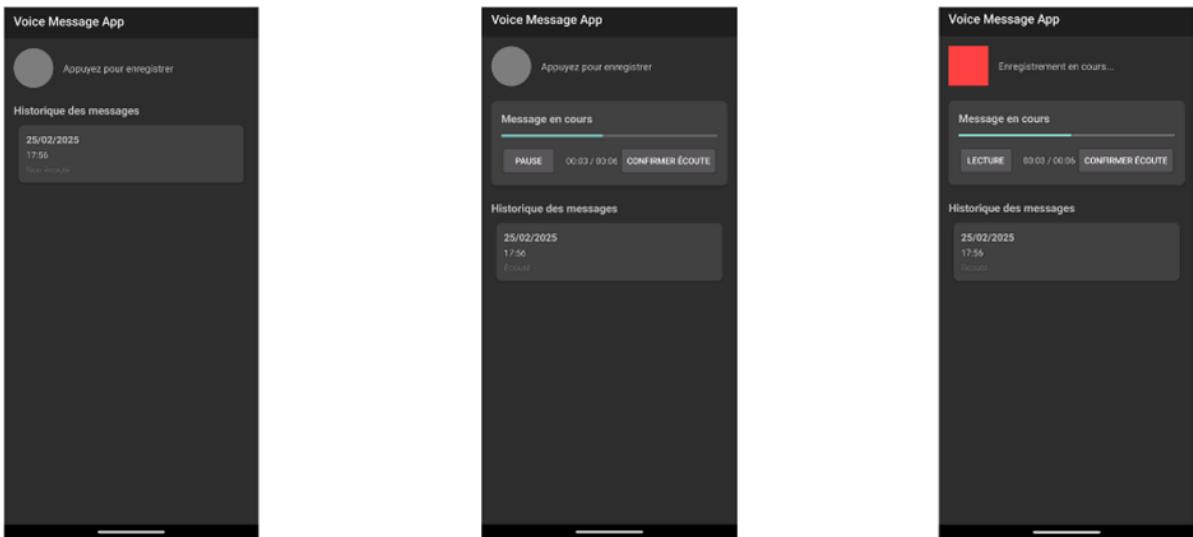


Figure 11 : Interface utilisateur

Communication avec le serveur

La communication entre l'application mobile et le serveur Flask est gérée par un système robuste basé sur la bibliothèque OkHttp, garantissant des échanges fiables et optimisés. L'architecture client-serveur implémente deux canaux de communication principaux : un mode pull où l'application interroge périodiquement le serveur pour vérifier la présence de nouvelles alertes via l'endpoint "/get_audio", et un mode notification où le serveur signale proactivement la disponibilité d'un nouveau message à travers un broadcast intent capté par un récepteur dédié dans l'application. La durée entre les vérifications périodiques est soigneusement calibrée pour offrir un bon compromis entre réactivité et consommation de ressources. Les requêtes HTTP sont optimisées avec des en-têtes appropriés pour le cache et la compression, minimisant ainsi le volume de données échangées. La gestion des erreurs réseau est particulièrement sophistiquée, avec des stratégies de reconnexion automatique en cas d'échec temporaire et des mécanismes de dégradation gracieuse permettant de maintenir un service minimal même dans des conditions de connectivité dégradée. Les réponses du serveur sont systématiquement validées pour détecter toute incohérence avant traitement, assurant ainsi la robustesse de l'application face à des données potentiellement corrompues ou incomplètes.

Gestion des alertes et notifications

Le système de gestion des alertes et notifications constitue l'élément central de l'application, conçu pour garantir qu'aucun incident ne passe inaperçu. Lorsqu'une nouvelle alerte de chute est détectée, l'application déclenche une séquence d'actions coordonnées pour maximiser les chances d'attirer l'attention de l'aide. Une notification prioritaire est affichée dans le système Android, utilisant le canal de notification "voice_message_alarm" configuré avec la priorité maximale. Cette notification s'accompagne d'une sonnerie d'alarme puissante qui ignore les paramètres de volume du téléphone, garantissant qu'elle sera entendue même si l'appareil est en mode silencieux. Un pattern de vibration distinct est également activé, offrant une alerte tactile supplémentaire. Si l'appareil le permet, les notifications utilisent également les LED de notification avec un code couleur dédié aux

alertes de chute. Pour les cas où l'alerte persisterait sans réponse, un mécanisme d'intensification progressive est implémenté, augmentant graduellement le volume et la fréquence des alertes sur une période configurable. L'application maintient également un état d'alerte visuel permanent jusqu'à confirmation explicite de l'aidant, évitant ainsi qu'une notification ne soit simplement écartée sans être traitée.

Historique des messages

L'historique des messages implémente un système sophistiqué de gestion et d'archivage des communications, offrant à l'aidant une vision complète des incidents passés. Chaque message vocal est automatiquement enregistré avec ses métadonnées essentielles : date et heure précises, durée du message, état d'écoute et éventuelles actions entreprises. Ces informations sont structurées au sein d'un modèle de données cohérent et persistant, sauvegardé localement dans une base SQLite. L'interface de consultation de l'historique présente ces données sous forme de liste chronologique inversée, mettant en évidence les messages non encore écoutés par un système visuel distinctif. L'implémentation utilise un RecyclerView optimisé avec un adaptateur personnalisé (MessageHistoryAdapter) qui assure des performances fluides même avec un grand nombre d'entrées. Chaque élément de la liste offre des actions contextuelles comme la réécoute du message, le marquage comme traité ou l'ajout de notes personnelles. L'historique implémente également des fonctionnalités de recherche et de filtrage permettant à l'aidant de retrouver rapidement un incident spécifique par date ou par type. Un mécanisme de synchronisation assure la cohérence des données entre l'application et le serveur, même après une période de déconnexion, garantissant ainsi qu'aucune information critique n'est perdue lors des problèmes temporaires de connectivité.

II-5 Sécurité et fiabilité

Périodes de cooldown

Pour éviter les alertes multiples concernant un même événement, le système implémente un mécanisme de périodes de cooldown soigneusement calibré. Un délai minimum de cinq minutes est automatiquement imposé entre deux alertes consécutives, période durant laquelle le système continue sa surveillance mais suspend temporairement la génération de nouvelles notifications. Ce délai est configurable selon les besoins spécifiques de chaque déploiement, permettant de l'adapter au profil de l'utilisateur et au contexte d'utilisation. Un système de hachage de vérification sophistiqué est également déployé pour identifier de manière unique chaque événement de chute, basé sur une combinaison de paramètres incluant l'horodatage, la localisation précise et les caractéristiques du mouvement détecté. Ce mécanisme permet d'éviter efficacement les doublons d'alertes qui pourraient survenir lors de perturbations réseau ou de redémarrages du système. Le système de cooldown s'adapte intelligemment au contexte, pouvant être temporairement désactivé lors de la détection de motifs suggérant des chutes multiples distinctes, une situation rare mais potentiellement critique nécessitant une attention immédiate.

Confirmation obligatoire

Le système de détection de chutes intègre un mécanisme de confirmation obligatoire qui constitue une couche de sécurité essentielle dans la chaîne de traitement des alertes. Dès qu'une chute est détectée, le système entre automatiquement en état de pause, maintenant l'alerte active jusqu'à ce qu'une intervention humaine explicite soit enregistrée. La confirmation par l'aidant s'effectue via l'endpoint spécifique "notify_listened" de l'API, qui enregistre non seulement la prise en compte de l'alerte mais également l'identité de la personne ayant confirmé et l'horodatage précis de cette action. Immédiatement après la confirmation, un retour vocal est automatiquement généré et diffusé à la personne ayant chuté, la rassurant sur le fait que son aidant a bien reçu l'alerte et qu'une assistance est en cours. Ce message vocal joue un rôle psychologique crucial, réduisant l'anxiété pendant l'attente de l'aide physique. La dernière étape du processus impose une réinitialisation manuelle du système, garantissant qu'aucune reprise de la surveillance ne peut s'effectuer par inadvertance avant que la situation ne soit entièrement résolue et que la personne ne soit en sécurité.

Récupération automatique

Pour garantir une disponibilité maximale du service même dans des conditions adverses, le système implémente des mécanismes avancés de récupération automatique. La fonctionnalité de démarrage au boot assure que le système de surveillance reprend automatiquement son fonctionnement après une coupure de courant ou un redémarrage non planifié, sans nécessiter d'intervention manuelle. Cette reprise s'effectue dans un état sécurisé, avec une vérification complète de l'intégrité des composants avant la réactivation des fonctions de détection. Un plan B sophistiqué pour la conversion audio a été implémenté pour garantir la génération des messages vocaux même en cas de défaillance du système principal. Ce plan alternatif utilise des méthodes de conversion différentes et peut même basculer vers des messages préenregistrés si nécessaire, assurant ainsi que la communication vocale avec la personne en détresse reste possible quelles que soient les circonstances. Le système maintient également une surveillance continue de son propre état de santé grâce à un endpoint dédié "/health" qui permet de vérifier instantanément l'état de tous les composants critiques. Des vérifications périodiques automatiques sont effectuées, et tout écart par rapport au fonctionnement nominal déclenche des procédures de récupération adaptées à la nature du problème détecté.

Intégration des mécanismes de sécurité

L'intégration harmonieuse des différents mécanismes de sécurité et de fiabilité forme un système cohérent où chaque composant renforce l'efficacité des autres. Le flux normal d'opération progresse de la prévention des faux positifs vers la génération d'alertes fiables, en passant par les mécanismes de cooldown qui filtrent les redondances potentielles. En cas d'alerte confirmée, le système bascule vers un mode de fonctionnement spécifique exigeant une confirmation humaine explicite avant toute reprise de la surveillance normale. Parallèlement, les mécanismes de récupération automatique opèrent en arrière-plan, assurant la résilience de l'infrastructure technique face aux perturbations environnementales ou matérielles. Cette architecture en couches superposées garantit qu'aucun point de défaillance unique ne peut compromettre l'ensemble du système, et que les incidents sont gérés au niveau le plus approprié. L'ensemble de ces mécanismes est régulièrement testé

par des procédures automatisées simulant diverses conditions de défaillance, assurant ainsi leur efficacité continue face à l'évolution de l'environnement d'exploitation.

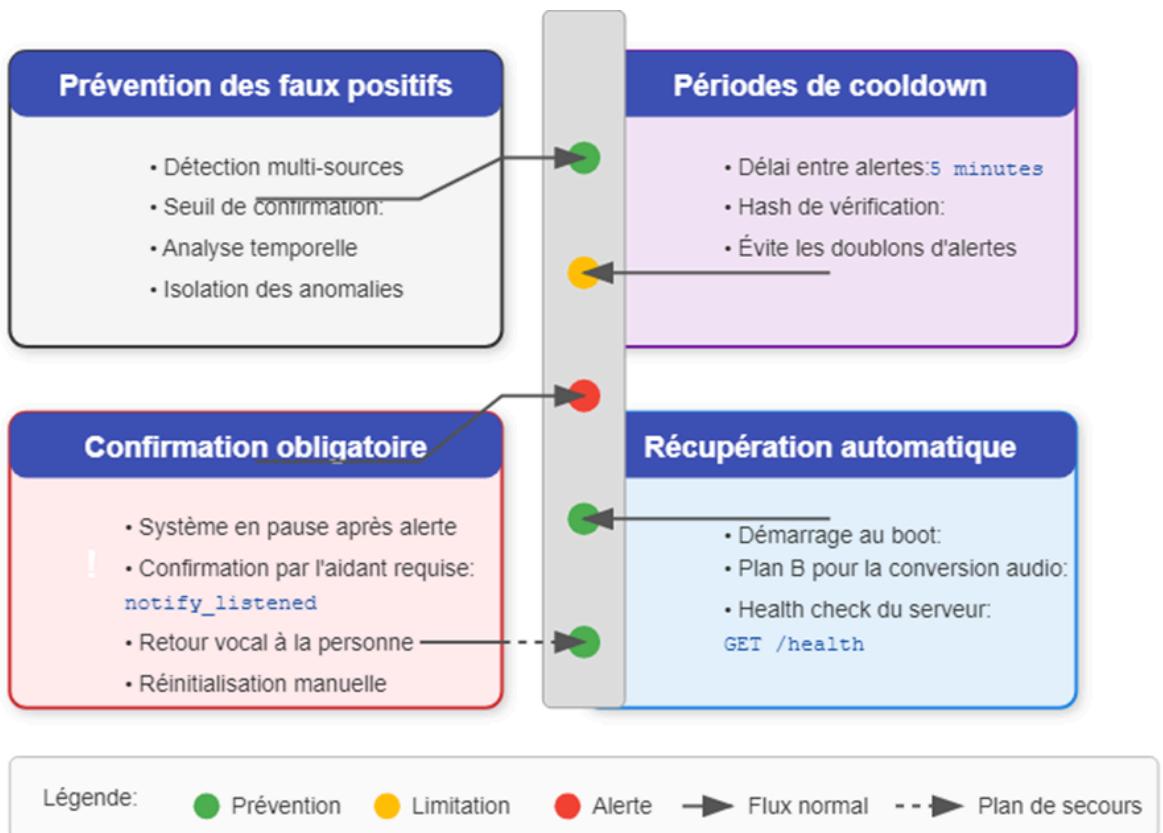


Figure 12 : Sécurité et fiabilité

III- Détection de chute avec Mediapipe

III-1 La détection d'une chute

Plusieurs approches existent pour la détection d'une chute. L'une d'entre elles est l'approche basée sur des capteurs. Elle repose sur l'analyse des données de mouvement et de position recueillies par divers capteurs placés sur une personne ou dans un environnement. Comme avantage, elle fonctionne n'importe où. Elle est peu coûteuse et peut être intégrée dans les smartphones et montres connectées. Elle est réactive et permet une détection instantanée de la chute. Comme inconvénients, elle doit être portée en permanence et peut donner de faux positifs (ex: saut brusque, course rapide).

Les principaux types de capteurs utilisés sont les Accéléromètres. Ceux-ci mesurent les accélérations dans différentes directions (généralement sur 3 axes : X, Y, Z) et permettent d'identifier les changements brusques de vitesse ou de position associés à une chute.

Les Gyroscopes mesurent la vitesse angulaire et permettent de détecter des rotations anormales du corps. Ils complètent les accéléromètres pour distinguer une chute d'un simple mouvement rapide. Les capteurs de Pression Intégrés dans des semelles ou des surfaces, ils détectent les impacts au sol, confirmant une chute.

Les capteurs de Proximité ou Ultrasoniques peuvent être utilisés pour détecter la distance par rapport au sol et suivre les changements rapides de hauteur. Les capteurs physiologiques mesurent des paramètres biométriques comme la fréquence cardiaque ou la pression sanguine, ce qui peut aider à distinguer une chute d'un simple mouvement brusque.

Les capteurs d'environnement reposent sur des dispositifs installés dans un espace de vie (maison, hôpital, maison de retraite) pour identifier une perte d'équilibre ou un impact au sol. Contrairement aux capteurs portables (accéléromètres, gyroscopes), ces capteurs ne nécessitent aucun dispositif porté par la personne.

L'approche la plus efficace repose sur une combinaison de plusieurs capteurs, permettant ainsi d'améliorer la précision et de réduire les faux positifs et négatifs. Les dispositifs portables (montres, bracelets connectés) sont les plus pratiques au quotidien, tandis que les solutions basées sur des capteurs fixes (tapis, capteurs de proximité) peuvent être adaptées à des environnements spécifiques comme les hôpitaux ou les maisons de retraite.

III-2 L'approche basée sur la vision par ordinateur

L'approche basée sur la **vision par ordinateur** repose sur l'analyse des **postures, mouvements et séquences vidéo** pour détecter une chute. La vision par ordinateur permet

de suivre l'évolution des postures corporelles et d'identifier des mouvements anormaux en utilisant des modèles d'apprentissage automatique et des techniques de vision artificielle.

Les principales étapes :

- L'acquisition de l'image (caméra en temps réel ou vidéo enregistrée)
- La détection et suivi du squelette humain
- L'analyse de la posture et des mouvements (trajectoire, vitesse, angles)
- La détection d'une chute en fonction de critères prédéfinis

L'utilisation de **caméras et algorithmes d'intelligence artificielle** pour analyser la posture et détecter les chutes en fonction de la trajectoire et de l'orientation du corps a de nombreux avantages. Elle fonctionne **sans capteur physique**. Elle peut détecter différents types de chutes. Néanmoins elle dépend de la qualité de la caméra et de l'éclairage. Elle pose des problèmes de confidentialité (caméras dans des espaces privés). Elle est idéale pour la télésurveillance et la détection intelligente des chutes dans des environnements contrôlés (hôpitaux, maisons de retraite).

L'approche par capteurs est idéale pour une utilisation individuelle et portable, avec un faible coût, une bonne autonomie et un respect de la vie privée. En revanche, l'approche par vision par ordinateur offre une meilleure précision, particulièrement dans des environnements surveillés, mais au prix d'un coût plus élevé et de préoccupations en matière de confidentialité.

Dans l'idéal, une combinaison des deux approches pourrait permettre une détection plus fiable et robuste, en tirant parti des forces de chacune : les capteurs pour une détection mobile et rapide, et la vision par ordinateur pour une analyse comportementale plus avancée.

III-3 Notre approche : le framework Mediapipe

Proposé par Google en open source, Mediapipe est l'un des frameworks de référence en vision par ordinateur. Il s'agit d'une boîte à outils de plusieurs modèles de vision par ordinateur permettant de réaliser des tâches diverses.

Ce framework offre une large gamme de solutions prêtées à l'emploi pour le traitement d'images et de vidéos, telles que la détection de visages, le suivi des mains, la segmentation de poses humaines, et bien plus encore.

Mediapipe repose sur une architecture modulaire permettant de traiter des flux multimédias en temps réel. Il utilise des graphes de traitement de données (appelés Mediapipe Graphs) qui enchaînent plusieurs étapes de traitement pour analyser les images ou des vidéos.

Un graphe Mediapipe est une chaîne de modules interconnectés, où chaque module effectue une tâche spécifique (ex: prétraitement d'image, détection de visage, suivi des mouvements). Cela permet une exécution optimisée, modulaire et efficace, particulièrement pour le traitement en temps réel sur CPU/GPU.

Un graphe dans Mediapipe est exécuté en pipeline.

L'image ou la vidéo entre dans le graphe via un nœud d'entrée.

Chaque nœud traite les données et envoie les résultats aux suivants.

Le dernier nœud fournit les résultats finaux, qui peuvent être affichés, stockés ou utilisés pour d'autres applications.

Par exemple, pour la détection de chute, le graphe engendré par Mediapipe pourrait être le suivant :

Le graphe suit ces étapes :

1. Capture de la vidéo (caméra ou fichier vidéo)
2. Prétraitement de l'image (conversion en RGB)
3. Détection de la posture avec Mediapipe Pose
4. Extraction des points clés du corps (tête, épaules, hanches, genoux, pieds)
5. Analyse des angles et distances pour détecter une chute
6. Génération d'une alerte si une chute est détectée

III-4 Le processus utilisé par Mediapipe

La détection de chute avec Mediapipe repose sur l'analyse des postures et des mouvements du corps en utilisant Mediapipe Pose. Cet outil permet d'extraire 33 points clés du squelette humain pour identifier des changements brusques d'orientation et d'immobilité, caractéristiques d'une chute. Voici ci-dessous ces 33 points clés :

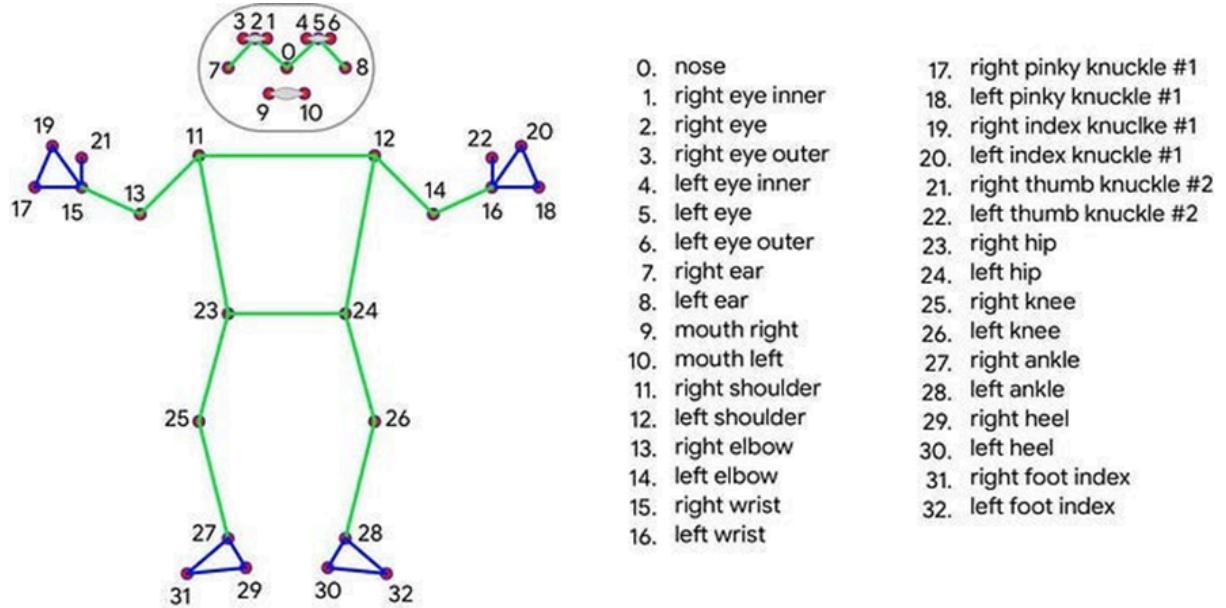


Figure 13: Les 33 points clés

Mediapipe Pose utilise des algorithmes de vision par ordinateur et d'intelligence artificielle pour suivre les articulations et les mouvements du corps à partir d'une vidéo en temps réel. Grâce à l'analyse des positions relatives des points clés (ex. tête, épaules, hanches, genoux), il est possible de détecter des chutes en identifiant :

Une perte soudaine d'équilibre (ex. inclinaison anormale du torse ou des jambes).

Un changement rapide de hauteur (ex. chute du centre de gravité du corps).

Une période d'immobilité prolongée après un mouvement brusque.

Avantages

Précision élevée : Permet d'identifier différents types de chutes grâce à l'analyse complète de la posture.

Traitements en temps réel : Adapté aux applications nécessitant une détection immédiate.

Utilisation sans capteur physique : Ne nécessite pas de dispositif porté par l'utilisateur, contrairement aux capteurs embarqués.

Limites et Contraintes

Dépendance à la qualité de la caméra : Une mauvaise résolution ou des angles de vue limités peuvent réduire l'efficacité de la détection.

Sensibilité aux conditions d'éclairage : Des ombres ou un faible contraste peuvent affecter l'identification des points clés.

Respect de la vie privée : Comme toute solution basée sur la vision par ordinateur, elle pose des questions sur la confidentialité des images capturées.

III-5 Première version avec Mediapipe et Python

Fonctionnement du programme

Ce programme utilise Mediapipe Pose et OpenCV pour détecter les chutes en analysant la position des hanches d'un individu capturé par une webcam.

Le programme commence par capturer la vidéo en temps réel à l'aide de la webcam, en appliquant un prétraitement qui inclut une inversion horizontale et une conversion en RGB pour être compatible avec Mediapipe. Ensuite, il utilise Mediapipe Pose pour détecter les 33 points clés du corps humain, notamment les hanches, et affiche les repères de posture sur l'image.

La détection de chute repose sur l'analyse de la position verticale des hanches. Le programme compare la position actuelle avec la précédente et, si une baisse soudaine de plus de 10% de la hauteur de l'image est détectée, une chute est suspectée. Lorsqu'une chute est identifiée, un message d'alerte s'affiche à l'écran et dans la console. L'alerte reste active pendant 30 images (environ 1 seconde) avant d'être réinitialisée pour éviter les faux positifs.

Enfin, le programme affiche la vidéo annotée en temps réel et s'arrête lorsque l'utilisateur appuie sur la touche Esc.

Analyse critique du programme

L'approche utilisée pour détecter les chutes repose sur l'analyse de la position verticale des hanches à l'aide de Mediapipe Pose et OpenCV. Cette méthode offre plusieurs avantages, notamment sa simplicité de mise en œuvre, sa faible consommation de ressources et sa capacité à fonctionner en temps réel sans nécessiter de capteurs physiques. L'utilisation d'une webcam permet d'obtenir un système peu coûteux et facilement déployable, rendant cette solution accessible à un large public.

Cependant, cette approche présente plusieurs limitations qui peuvent affecter sa fiabilité. Tout d'abord, elle est sujette aux faux positifs et faux négatifs : des mouvements brusques comme s'asseoir rapidement peuvent être interprétés comme une chute, tandis que certaines chutes lentes peuvent ne pas être détectées. De plus, la précision du système dépend fortement de la position et de l'angle de la caméra, ce qui peut entraîner des erreurs de détection si l'utilisateur se trouve partiellement hors du champ de vision.

Un autre point critique concerne l'absence d'analyse du comportement après la chute. Le programme détecte uniquement un changement soudain de hauteur des hanches, sans vérifier si la personne est immobile ou se relève après l'événement. Cela limite la pertinence des alertes, qui pourraient être déclenchées inutilement.

III-6 Utilisation d'une bibliothèque de vidéo de chute ou non chute

Pour pouvoir approfondir les techniques de détection de chute avec Mediapipe, un ensemble de vidéo, une centaine qui ont été réalisés par *Michał Kępski*, chercheur en vision par ordinateur, ont été utilisés. Elles sont regroupées en deux catégories. La première contient des vidéos qui sont des chutes et la seconde des non chutes.

Michael Kepsi a écrit un article « Event-driven system for fall detection using body-worn accelerometer and depth sensor », qui présente un système de détection des chutes basé sur des événements, combinant un accéléromètre porté sur le corps et un capteur de profondeur. L'objectif est d'améliorer la précision et la réactivité de la détection des chutes en exploitant des données multimodales pour distinguer les chutes accidentelles des mouvements normaux du quotidien.

En utilisant les vidéos sur le site de *Michael Kepsi*, notre programme de détection de chute a pu tester des cas réels et nombreux. Ces vidéos montrent la diversité des situations. Elles n'ont pas toujours été filmées dans le même lieu. La caméra est positionnée soit en face, soit au plafond. Leur durée est souvent très courte. Elles sont d'une qualité plus ou moins bonne. La démarche de notre approche va donc d'être de tester notre programme avec toutes ces vidéos, et de voir s'il est capable de bien reconnaître une vidéo de chute classée dans cette catégorie ou bien une vidéo de non chute. Plusieurs versions de ce programme ont été développées pour améliorer ce taux de reconnaissance.

La deuxième version du programme a permis de tester le programme sur toutes les vidéos. Une matrice de confusion ci-dessous a permis de montrer que le programme détectait assez bien les vidéos de chute mais était très mauvais dans les vidéos de non chute.

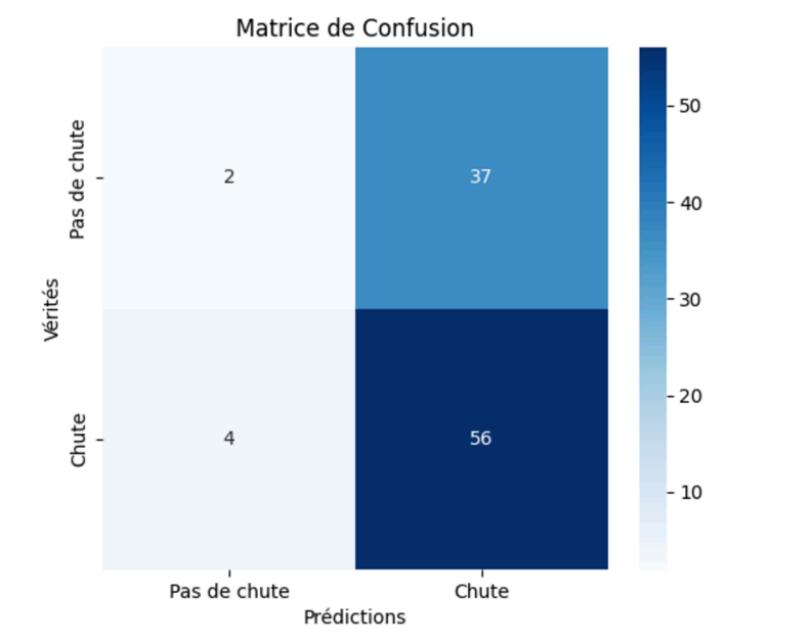


Figure 14: Résultat de la matrice de confusion

Résumé des résultats :

- ✓ 56 chutes détectées sur 60 vidéos dans 'chutes_video' (93.33% de précision)
- ✗ 2 non-chutes détectées sur 39 vidéos dans 'pas_de_chute_video' (5.13% correctes)
 - ◆ Traitement terminé.

Ce programme utilise OpenCV et MediaPipe pour détecter des chutes dans des vidéos. Son principe repose sur l'analyse de la position des épaules et des hanches afin de déterminer si une personne est tombée. Pour cela, il extrait les coordonnées des landmarks corporels en utilisant MediaPipe Pose, puis il compare la hauteur des épaules et des hanches. Si cette différence devient inférieure à un seuil de 0.14, une chute est considérée comme détectée.

Le programme fonctionne en parcourant image par image une vidéo donnée, traitant chaque frame pour identifier les points clés du corps. Il applique ensuite ce critère simple pour décider si une chute a eu lieu. Une fois cette analyse effectuée sur un ensemble de vidéos, les résultats sont stockés et une matrice de confusion est générée pour évaluer les performances du modèle.

Ce programme parvient bien à détecter les chutes, car lorsque quelqu'un tombe, la hauteur des épaules se rapproche significativement de celle des hanches. Ce critère est généralement fiable pour des chutes où le corps s'effondre ou se recroqueville sur lui-même.

Cependant, il montre des difficultés à bien identifier les non-chutes. Plusieurs facteurs expliquent cette faiblesse. Tout d'abord, certains mouvements normaux, comme s'accroupir, s'asseoir ou se pencher, peuvent provoquer une diminution temporaire de la distance entre les épaules et les hanches, conduisant à des fausses alertes. Ainsi, un simple mouvement rapide vers le bas peut être confondu avec une chute.

En conclusion, bien que le programme parvienne à détecter efficacement certaines chutes, son approche simplifiée basée sur la seule distance épaules-hanches génère de nombreuses fausses alertes. D'autre part, certaines vidéos où la caméra était au plafond ne pouvaient pas être analysées par notre programme. Il a fallu les effacer. Certaines vidéos n'étaient pas filmées dans les mêmes lieux. Du coup le paramétrage du programme ne correspondait pas forcément au lieu.

III-7 Paramétrage externe du programme

Ce qui est apparu évident lors des tests faits avec chaque vidéo, est que les paramètres pour bien détecter une chute sont à adapter à chaque lieu. D'autre part, les tests sur les vidéos, bien qu'elles puissent être très utiles, peuvent aussi fausser les résultats. Par exemple, de nombreuses vidéos de chutes étaient très courtes. Or, il a été constaté que quelqu'un qui chute, va rester un certain délai dans une position quasi-allongée. Si la vidéo est trop courte, ce paramètre n'est pas validé.

Pour pouvoir externaliser des paramètres utiles pour la détection de chute, le format JSON (JavaScript Object Notation) a été choisi. Le format JSON peut être utile pour la détection des chutes en permettant le stockage et l'échange structuré des données issues de l'analyse vidéo. Ce format repose sur des paires clé-valeur et est largement utilisé pour structurer des informations de manière légère et lisible.

L'intérêt principal de JSON réside dans sa compatibilité avec de nombreux langages de programmation et sa facilité de manipulation. Il permet aussi de transmettre efficacement les données entre différentes parties d'un système, par exemple entre un module de détection et une interface utilisateur affichant les résultats en temps réel.

Le programme final présente cette approche. Une interface graphique permet de charger un fichier de configuration JSON, de modifier les paramètres et de lancer le script de détection de chutes. Plusieurs choix s'offrent à l'utilisateur. Soit il applique ce programme à une vidéo ou à des vidéos situées dans un répertoire. Dans ce cas, le programme dit s'il reconnaît une chute ou non pour la ou les vidéos d'un répertoire. Soit il indique qu'il souhaite utiliser une webcam, et il applique les paramètres définis dans le fichier JSON pour détecter une chute. Voici ci-dessous l'interface graphique du programme.

Deux fichiers de configuration sont proposés. Le premier sert à tester les vidéos dont nous avons parlés auparavant. Avec les valeurs données dans l'interface ci-dessous, 25 vidéos de chutes sur 30 ont été reconnues. Nous avons constaté que si les vidéos durent un peu plus longtemps, 28 vidéos de chutes auraient pu être reconnues. En ce qui concerne le répertoire des non-chute, 32 vidéos de non-chute ont été reconnus sur 39 vidéos.

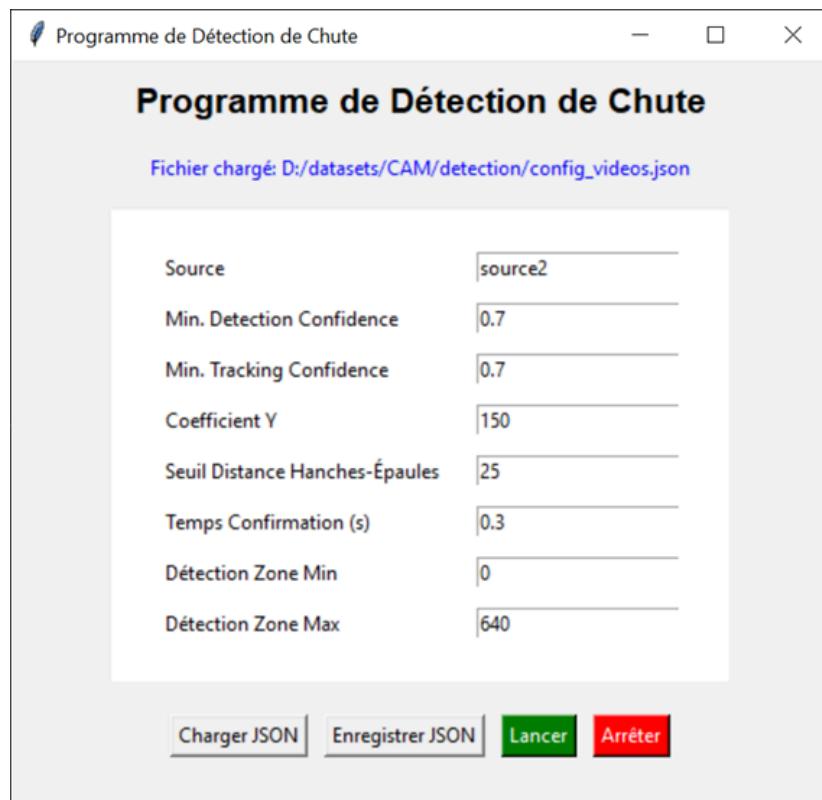


Figure 15: un exemple de paramétrage externe

Un deuxième fichier de configuration spécifique à la webcam est également proposé et nécessitera une configuration adaptée à l'environnement dans lequel nos tests en réel s'effectueront.

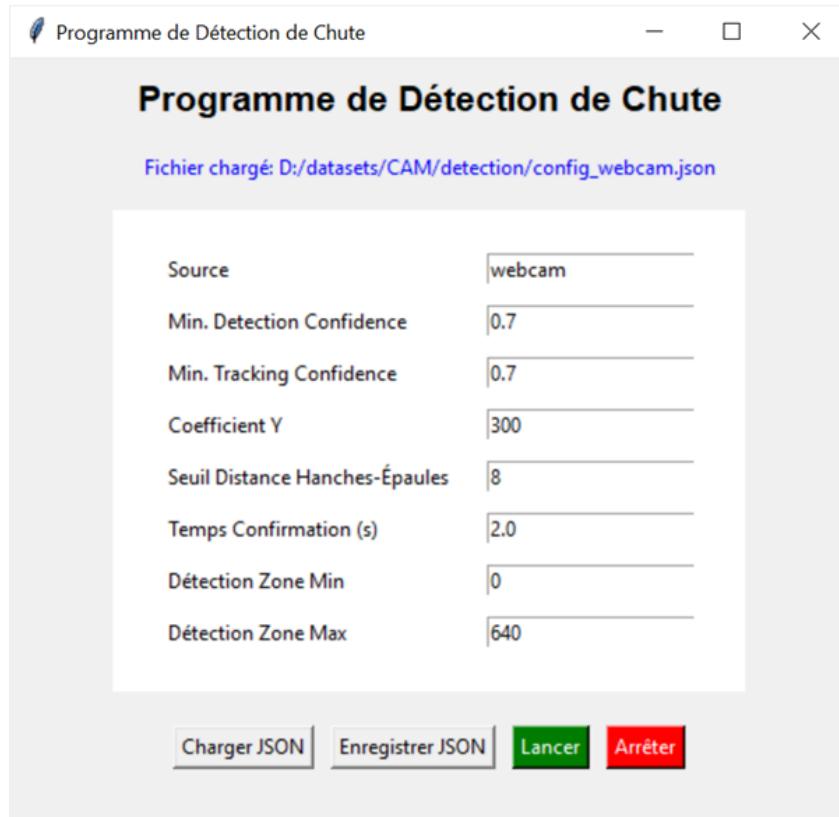


Figure 16: paramétrage pour la webcam

III-8 Analyse de la dernière version du projet

Ce programme intègre également une interface graphique développée avec la bibliothèque Tkinter pour permettre à l'utilisateur de charger, modifier et sauvegarder un fichier de configuration au format JSON. Ce fichier contient des paramètres essentiels comme le seuil de détection, les coefficients de tracking et les options d'alerte sonore. Grâce à JSON, la gestion des paramètres devient plus flexible et modifiable sans altérer le code principal.

L'interface permet aussi de lancer et d'arrêter l'exécution d'un script de détection de chute en utilisant le module subprocess. Cette interaction facilite l'exécution du programme sans nécessiter de manipulation manuelle en ligne de commande.

En résumé, l'utilisation de JSON facilite la personnalisation des paramètres, tandis que l'interface graphique améliore l'accessibilité du programme.

En ce qui concerne la dernière version du programme de détection de chute, commence par charger un fichier de configuration JSON contenant divers paramètres, tels que la

source de la vidéo (fichier ou webcam), les seuils de détection. Cette flexibilité permet d'adapter le programme à différents contextes d'utilisation.

Ensuite, il initialise MediaPipe Pose pour le suivi des mouvements du corps. Les images sont extraites image par image à partir du flux vidéo et converties en format RGB pour être traitées par MediaPipe. Si des landmarks corporels sont détectés, le programme trace les points clés du squelette humain et extrait les coordonnées des épaules et des hanches.

La détection de chute repose sur trois critères principaux :

1. **La hauteur des épaules par rapport à une ligne de référence (coef_y)** : Une épaule sous cette ligne est un indicateur potentiel de chute.
2. **La distance entre les épaules et les hanches** : Une distance trop faible peut signifier que la personne est allongée.
3. **La durée de maintien de cette position** : Une chute confirmée nécessite que ces conditions restent vraies pendant un temps défini.

Si ces conditions sont remplies, une alerte peut être déclenchée (sonore ou visuelle), et la chute est comptabilisée.

Pour conclure, L'utilisation d'un fichier JSON permet de modifier les paramètres sans altérer le code, rendant le programme flexible et adaptable. De plus, sa modularité lui permet de traiter plusieurs vidéos d'un répertoire ou de fonctionner en temps réel via une webcam. Il est également efficace pour détecter les chutes classiques où la personne tombe et s'effondre sur elle-même.

Ce programme constitue une base robuste pour la détection de chutes en vidéo, avec une logique claire et une bonne flexibilité grâce à la configuration JSON. Cependant, ses critères de détection peuvent être améliorés par une meilleure analyse temporelle et l'utilisation d'autres points clés du corps. Une évolution vers un modèle basé sur l'intelligence artificielle permettrait d'optimiser la précision et d'adapter le programme à des cas plus variés.

L'utilisation d'un tapis avec capteur pourrait considérablement améliorer la détection des chutes. Ce dispositif permettrait de capter directement les impacts au sol, réduisant ainsi les faux positifs liés aux mouvements similaires à une chute. Il offrirait également une validation complémentaire à l'analyse visuelle, en détectant précisément la force et la localisation de l'impact. De plus, un tapis avec capteur pourrait détecter une personne restant allongée au sol après une chute, renforçant ainsi la fiabilité du système. En combinant les données du tapis et celles issues de la vision par ordinateur, le programme gagnerait en précision.

IV- Rapport Arthur suite Florian

IV-1 Limites de la première version du système de détection

Après quelques semaines de tests avec la première version de l'application utilisant uniquement Mediapipe sur un paramétrage manuel, nous avons choisi les pistes d'amélioration possible de notre programme.

Tout d'abord, l'efficacité de la bibliothèque Mediapipe dépend beaucoup de l'angle sous lequel la caméra est positionnée, par exemple si la personne chute face à l'axe de la caméra, celle-ci pourrait avoir des difficultés à la détecter.

Dans la version actuelle du programme, Mediapipe analyse principalement les changements brusques de position et d'orientation du corps. Cette approche est efficace pour détecter des chutes brutales, mais elle peut rencontrer des difficultés avec des chutes progressives. Par exemple, une personne qui glisse lentement d'une chaise. De plus, certaines postures, comme le fait de s'agenouiller de ramasser un objet au sol, peuvent être confondues avec une chute, générant des faux positifs.

Enfin, le programme actuel basé sur Mediapipe se concentre sur la détection instantanée d'un mouvement semblable à une chute. Toutefois, il n'intègre pas une analyse du comportement après la chute. Or, une personne qui tombe accidentellement peut rester immobile un certain temps, tandis qu'une personne effectuant un mouvement rapide peut se relever immédiatement. L'absence de cette analyse post-chute peut donc entraîner des faux positifs.

IV-2 Vers une solution hybride : intégration d'un modèle DNN

Pour répondre à notre problématique, nous avons développé une nouvelle approche qui conserve les fonctionnalités de base tout en le rendant plus flexible et robuste. Cette solution hybride combine l'efficacité de Mediapipe avec la puissance de classification d'un réseau de neurones profond.

L'utilisation d'un modèle basé sur un DNN permettrait de supprimer ces limitations en rendant la détection plus fiable. Une approche qui combinerait Mediapipe et un modèle DNN offrirait ainsi une solution adaptée aux défis de la détection des chutes.

Pour répondre à notre problématique, nous avons développé une nouvelle approche qui conserve les fonctionnalités de base du système initial tout en le rendant plus flexible et robuste. Cette solution hybride, présentée dans les sections suivantes, combine l'efficacité de Mediapipe pour l'extraction de points anatomiques avec la puissance de classification d'un réseau de neurones profond.

IV-3 Prétraitement des données et constitution du dataset

Prétraitement des données

La partie prétraitement des données est gérée par la classe DatasetBuilder. Cette classe est responsable de la construction et de la préparation du jeu de données d'entraînement, avec plusieurs fonctionnalités essentielles.

Extraction des caractéristiques anatomiques

Notre classe ImageProcessor (intégrée dans dataset_builder.py) utilise Médiapipe avec des paramètres de confiance configurables, ce qui nous permet d'ajuster la sensibilité de la détection selon les conditions d'environnement.

Pour chaque image, nous appliquons le processus suivant:

1. Chargement de l'image et conversion au format RGB (MediaPipe ne travaille pas avec le format BGR d'OpenCV)
2. Application du détecteur de pose pour identifier les points clés anatomiques
3. Extraction d'un vecteur à 99 dimensions (33 points clés × 3 coordonnées x,y,z) qui représente la position complète du corps

Cette représentation vectorielle du squelette humain constitue l'entrée idéale pour notre réseau de neurones, car elle capture l'essentiel de la posture sans être affectée par les variations d'éclairage ou de textures.

IV-4 Conception et entraînement du modèle DNN

Constitution et équilibrage du jeu de données

Nous avons construit une base de données d'environ 4600 images réparties en quatre catégories :

- 1200 images de posture debout
- 1000 images de marche
- 400 images de squat (accroupissement)
- 2000 images de chute

Ce dataset a été établi avec le site web “Kaggle”.

Cette distribution, bien que déséquilibrée en faveur des chutes, nous permet d'obtenir une meilleure sensibilité pour les situations critiques tout en maintenant une bonne spécificité.

Normalisation et augmentation

Pour améliorer la robustesse du modèle, nous appliquons plusieurs techniques de prétraitement:

1. Normalisation des coordonnées squelettiques (centrage autour de la moyenne et mise à l'échelle par l'écart-type)
2. Mélange aléatoire (shuffle) des exemples pour éviter les biais d'ordre lors de l'entraînement
3. Division des données en ensembles d'entraînement (80%) et de validation (20%)

Ces étapes sont cruciales pour garantir un apprentissage stable et une bonne généralisation du modèle à des situations nouvelles.

Conception du modèle DNN (Deep Neural Network)

Au cœur de notre système se trouve la classe FallDetectionDNN, qui implémente un réseau de neurones profond spécialement conçu pour la classification des actions humaines à partir des données squelettiques.

Architecture du réseau

Notre modèle utilise une architecture séquentielle avec plusieurs couches, comme représenté dans la figure X ci-dessous.

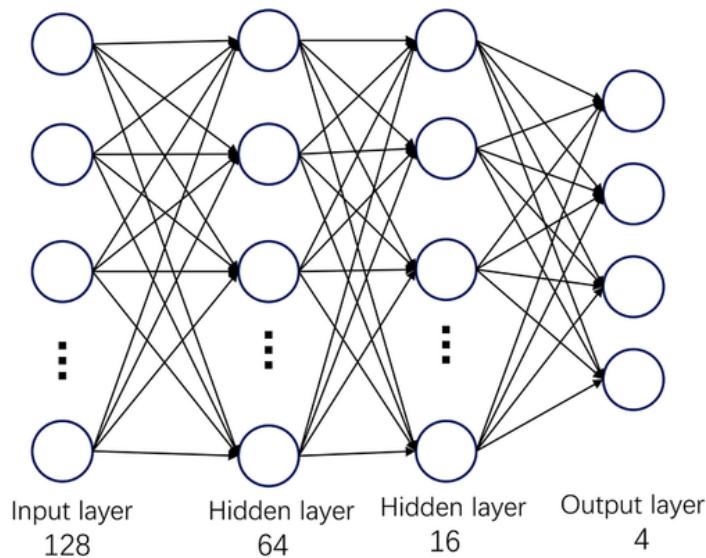


Figure 17 : architecture du réseau de neurones

1. Une couche d'entrée dense de 128 neurones recevant le vecteur de 99 caractéristiques
2. Une couche de normalisation par lots pour stabiliser l'apprentissage
3. Une couche cachée de 64 neurones avec activation ReLU (
4. Une couche de Dropout (25%) pour réduire le surapprentissage
5. Une seconde couche cachée de 32 neurones avec activation ReLU
6. Une nouvelle couche de normalisation par lots
7. Une couche de sortie de 4 neurones avec activation Softmax (correspondant à nos 4 classes)

Les trois premières couches utilisent la **fonction d'activation ReLU (Rectified Linear Unit)**, qui est connue pour éviter le problème de disparition du gradient et qui va améliorer la convergence de l'apprentissage.

La couche de sortie emploie la **fonction d'activation softmax**, idéale pour les tâches de **classification multiconcours**. Elle va générer des probabilités normalisées à partir des résultats précédents

Stratégies d'optimisation de l'apprentissage

Nous avons implémenté plusieurs techniques avancées pour optimiser l'entraînement:

1. **Compilation optimisée:** Nous utilisons l'optimiseur Adam avec un taux d'apprentissage initial de 0.001 pour débuter l'apprentissage.
2. **Callbacks stratégiques:**
 - EarlyStopping: Arrête l'entraînement lorsque la précision sur l'ensemble de validation cesse de s'améliorer pendant 10 époques consécutives, pour éviter le surapprentissage.
 - ReduceLROnPlateau: Réduit dynamiquement le taux d'apprentissage lorsque les performances stagnent, permettant un ajustement des poids.
 - ModelCheckpoint: Sauvegarde les meilleurs poids du modèle en fonction de la précision de validation.
3. **Paramètres ajustés:**
 - Taille de lot (batch size) de 32 pour un bon compromis entre vitesse et stabilité
 - Nombre d'époques maximal de 100, bien que l'arrêt précoce intervient généralement avant

Ces stratégies nous permettent d'atteindre une précision supérieure à 98% sur notre ensemble de validation, démontrant l'efficacité de notre approche.

IV-5 Détection en temps réel et intégration sur plateforme embarquée

La phase finale de notre système concerne l'intégration du modèle entraîné dans un système de surveillance en temps réel, spécifiquement optimisé pour fonctionner sur des plateformes à ressources limitées comme le Raspberry Pi.

La classe DetectionChuteAutomatique constitue l'interface principale pour la détection en temps réel. Elle intègre plusieurs fonctionnalités:

1. **Initialisation du détecteur:** Chargement du modèle Keras (.h5) entraîné
2. **Configuration d'un mécanisme de confirmation:** Implémentation d'un compteur et d'un seuil pour éviter les faux positifs
3. **Gestion des alertes:** Déclenchement d'alertes via un thread séparé pour maintenir la continuité de surveillance

Optimisations pour environnement embarqué

Nous avons implémenté plusieurs optimisations critiques pour assurer un fonctionnement fluide sur Raspberry Pi:

1. **Échantillonnage temporel:** Traitement d'une frame sur deux pour réduire la charge de calcul
2. **Redimensionnement spatial:** Réduction de la résolution des images pour accélérer le traitement

3. **Parallélisation:** Utilisation de threads séparés pour la détection et la gestion des alertes

Filtrage temporel à plusieurs niveaux

Un aspect particulièrement innovant de notre système est l'implémentation d'un filtrage temporel à deux niveaux:

1. **Niveau DNN:** Notre détecteur utilise déjà en interne une fenêtre glissante de 5 prédictions pour stabiliser les résultats
2. **Niveau système:** Le programme principal exige 3 détections positives consécutives avant de déclencher une alerte, afin d'ajouter une sécurité supplémentaire aux faux-positifs.

Cette approche double nous permet de réduire considérablement le taux de fausses détections tout en maintenant une sensibilité élevée pour les véritables situations de chute.

IV-6 Évaluation des performances et résultats

Notre modèle DNN a atteint une précision de 98% sur l'ensemble de validation

Sur notre ensemble de test indépendant de 920 images (20% des données totales), le modèle a maintenu une précision globale de 97,8%, démontrant sa capacité à généraliser sur des données non vues.

Sur le Raspberry Pi 5 (4 GB RAM), notre système optimisé atteint les performances suivantes :

- Fréquence de traitement : 20~30 FPS
- Latence de détection : ~200ms
- Consommation CPU moyenne : 65%
- Consommation mémoire : 420 MB

Le système reste fonctionnel même avec une résolution d'entrée réduite à 640x480 pixels, ce qui le rend adapté à une utilisation en continu.

V- Détection de chute avec les capteurs de pression au sol

V-1 Le choix du matériel

Dans le cadre de notre projet Detect Chute, en complément de la caméra, nous avons utilisé en plus 1 tapis de taille 72 x 56 cm et 10 tapis de 59.5 x 17cm, achetés sur le site Farnell. Ces tapis existent en tant que capteurs de pression au sol. Ils peuvent détecter une chute dans le sens où si il y a plusieurs zones activées simultanément pendant une durée prolongée, cela suggère une forte probabilité qu'une personne soit tombée au sol plutôt que debout, en mouvement ou en train de se baisser. Ces dernières actions n'activent généralement pas plus que deux zones, et/ou ne peuvent pas durer plus de 5 secondes.

Dans le prototype, nous avons envisagé de détecter la chute dans la chambre de la personne fragilisée, à côté de son lit. Pour cela, nous plaçons 8 tapis dans l'angle mort de la caméra (car c'est là où le besoin de détection est le plus important), et 3 autres dans l'autre côté du lit. La majorité des tapis sont placés sur la partie supérieure du corps, parce que quand les gens chutent, ils ont tendance à se supporter avec les bras pour se soutenir.

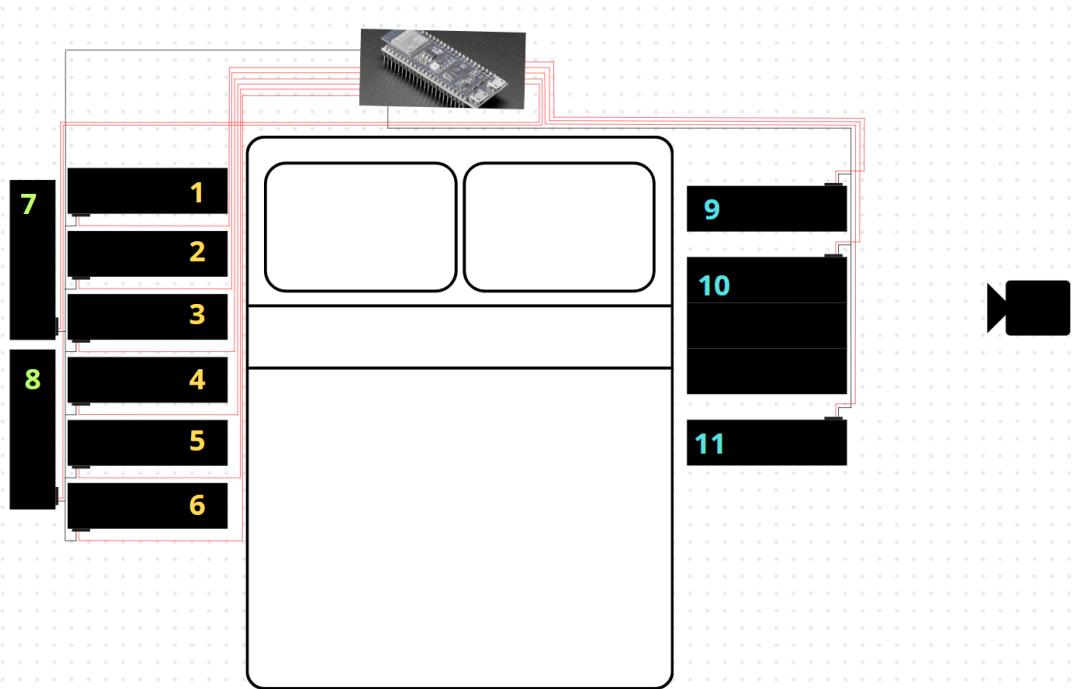


Figure 18: Le cas d'utilisation avec l'emplacement des tapis

N'importe la taille du tapis, le principe de fonctionnement est le même. Chaque tapis a quatre fils, mais il y a que les deux fils à droite qui nous sont nécessaires. Ils forment un bouton qui est normalement ouvert, et ils seraient connectés seulement s'il y a une pression importante. Sur la fiche de produit, il s'agit de 16kg appliqués sur la surface du tapis d'une

largeur de 50 mm. Mais en réalité même si on appuie seulement avec une main et un peu de force, le tapis détecte quand même une pression. Quand selon moi il est plus sensitif que sur la fiche.

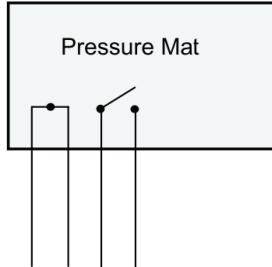


Figure 19: La structure interne du tapis

V-2 Le fonctionnement des capteurs

Dans un premier temps, j'ai testé le tapis avec une carte Arduino et l'application Arduino IDE afin de comprendre son fonctionnement et le type de données qu'il génère. Les résultats montrent que le tapis ne fournit que des valeurs binaires (0 ou 1), et ça fonctionne comme un bouton. Cela signifie qu'il détecte uniquement la présence ou l'absence de pression, sans différencier une position debout ou chute. Le tapis n'est donc pas résistif, ce qui signifie qu'il ne modifie pas le courant en fonction de la pression appliquée. Il peut être connecté aussi bien à des entrées analogiques qu'à des entrées numériques. (Dans le cas contraire, il devrait être exclusivement connecté aux broches analogiques).

Cependant le prototype final a été réalisé sur une carte ESP32-S3 Mini. J'ai d'abord utilisé une carte Arduino sur un breadboard pour les tests préliminaires, car l'ESP 32 nécessite des soudures pour son intégration. Cette approche m'a permis de valider le fonctionnement du tapis avant de l'adapter à la carte définitive. En plus de cet aspect pratique, l'ESP 32 offre une connectivité sans fil via WiFi, ce qui permet d'envoyer des alertes à notre serveur Flask sans nécessiter de matériel supplémentaire. À l'inverse, une carte Arduino classique aurait besoin d'un module WiFi externe.

Pour faire fonctionner un tapis, on va le connecter sur une carte esp32 s3 mini, qui est lui souder sur une plaque d'essai. Cette carte est connectée à l'ordinateur avec une cable USB-C. L'ordinateur existe comme une batterie externe pour alimenter le système. Il contient aussi le code python dans Mu Editor pour savoir quels sont les tapis activés, et l'algorithme pour identifier une chute. Dans les prochains paragraphes, je vais développer chaque partie en détail.

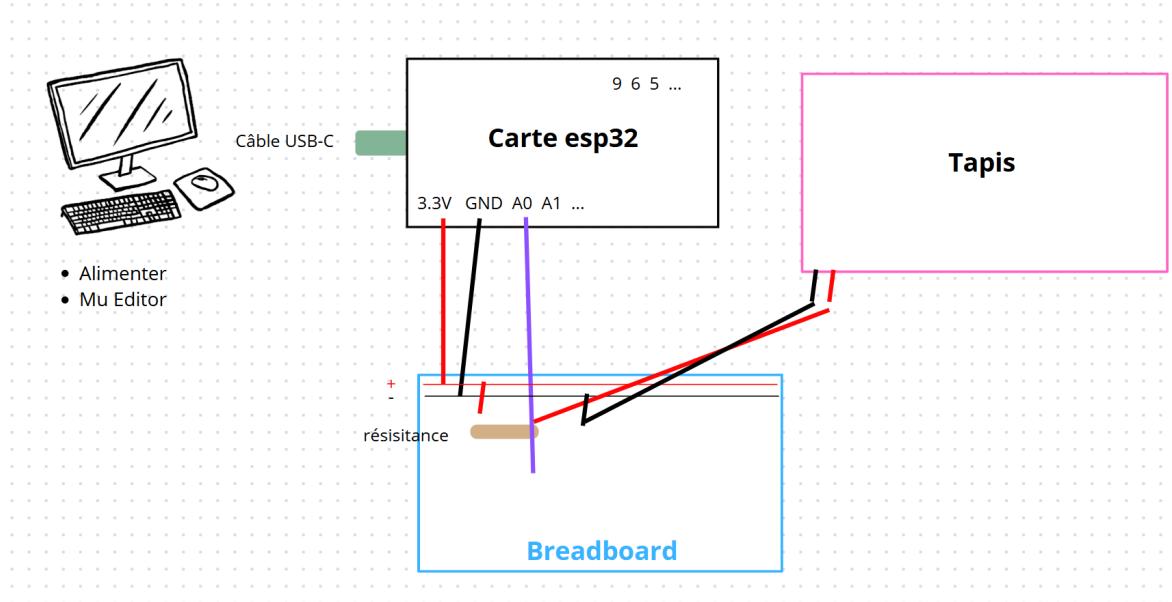


Figure 20: Le principe de fonctionnement pour un tapis

V-3 Le montage final

J'ai configuré 11 tapis de pression, en utilisant des broches analogiques (A0 à A5) et digitales (D5 -D11). Chaque tapis est dans une broche différente pour que chacun puisse détecter une pression et envoyer un signal indépendamment des autres.

Pour le câblage, j'ai exclusivement utilisé des fils rouges pour représenter le courant positif et des fils noirs pour le courant négatif, c'est pour avoir un schéma clair et facile à comprendre. La structure de câblage de chaque tapis est identique : nous acheminons d'abord le courant positif, qui traverse ensuite une résistance avant d'entrer dans une broche analogique ou digitale. Puis le courant circule à travers le tapis, en passant par son côté positif puis son côté négatif. À la fin du circuit, le courant retourne à la plaque, et c'est la fermeture du circuit électrique. Après il nous faudrait juste répéter ce même circuit 11 fois pour tous les tapis.

D'origine, les fils du tapis ont environ 5cm de longueur, ce qui est insuffisant pour notre cas d'utilisation souhaité. Car nous avons prévu de mettre la carte ESP32 au-dessous et au centre du lit, puis il y a les 11 tapis connectés dessus devaient être mis pour couvrir la surface autour du lit. Donc vers le milieu du projet, nous avons commandé en plus 15 m de câble blindé avec 2 fils rouge et noir dedans. Avec ça nous obtenons environ 1.2 mètre de fils allongés pour chaque tapis. Donc nous avons découpé puis soudé ces fils allongés sur les fils originaux du tapis, avant de commencer le montage final.

Avec la carte ESP 32 sur un breadboard, et j'ai connecté tous les 11 tapis dessus pour vérifier que chacun détecte une pression indépendamment et est capable d'envoyer un signal. Puis je soude cette carte sur la platine de prototypage avec le fer à souder. Il faut

vérifier que chaque broche est en contact avec un trou du module, et bien vérifier l'alignement avant de commencer à souder. Chaque fois il faut mettre une petite quantité d'étain pour remplir le trou, et s'assurer qu'aucune soudure ne fait de pont entre deux broches. Dans ce projet, j'ai mis la carte au centre et à l'extrémité de la plaque, pour gagner plus de place afin de tout connecter sur une seule plaque. En plus, j'ai mis une étiquette pour numérotter chaque tapis afin d'éviter les confusions.

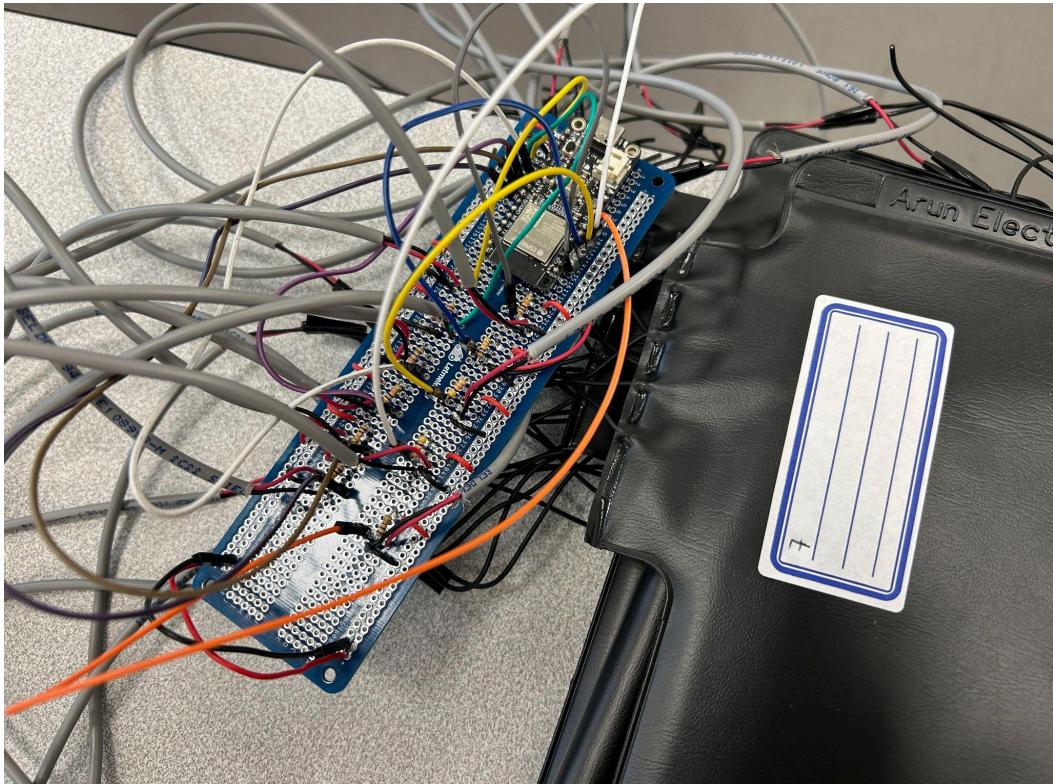


Figure 21: Le montage final

Dans ce montage, je n'ai pas utilisé de multiplexeur car tout peut entrer dans une seule plaque, la carte ESP32 S3 dispose de suffisamment de broches, et la connexion directe a moins de défaillance potentielle. Le dépannage et la maintenance sont aussi plus faciles. Pour ce prototype, quand j'ai fait le montage final, ça a fait plusieurs fois que je devrais re-souder les câbles électriques allongés. Et la connexion directe m'a aidé à identifier très vite quels sont les endroits cassés.

V-4 Logiciel, Code

Pour programmer la carte ESP32, j'ai utilisé Mu Editor en mode CircuitPython.

Ce code a pour l'objectif d'être un système de détection de chute utilisant 11 tapis sensibles à la pression. Le système est conçu pour détecter une chute potentielle lorsque d'au moins 3 tapis sont activés simultanément pendant une durée de 5 secondes, puis communique

cette information avec notre serveur Flask pour confirmer la détection. Le système repose sur le microcontrôleur CircuitPython avec une connexion WiFi.

Dans le code, d'abord j'importe plusieurs bibliothèques afin de préparer l'ESP32 à fonctionner avec des capteurs de pression, la connexion Wi-Fi, et la synchronisation de l'heure via un serveur NTP.

Board permet d'accéder aux broches de la carte ESP32-S3 Mini. Digitalio permet de configurer et contrôler les broches numériques pour lire l'état des capteurs de pression. Time est utilisé pour gérer des pauses et mesurer des durées. Wifi gère la connexion Wi-Fi de l'ESP32. Socketpool permet d'établir des connexions réseau. Adafruit_requests est pour envoyer et recevoir des requêtes HTTP. Ssl active la sécurité SSL pour les connexions HTTPS. Rtc gère l'horloge interne de la carte. Adafruit_npt.NPT permet de récupérer l'heure exacte depuis un serveur NTP (Network Time Protocol). Json est pour la gestion et le stockage des données au format JSON.

Ensuite, après la configuration du wifi et du serveur Flask, nous affichons une phrase qui indique la démarrage du système. Puis nous nous sommes connectés sur le wifi avec un message de réussite. Et nous synchronisons l'horloge interne de l'ESP32-S3 Mini avec un serveur NTP, avec un message de gestion des erreurs.

Prochainement, j'ai défini les broches pour chaque tapis, par exemple pour dire que le tapis numéro 1 est connecté sur la broche A0. Après la création de l'objet pour chaque tapis, j'ai attribué un emoji couleur différente à chaque tapis (les 8 dans l'angle mort de la caméra sont en carré, et les 3 autres en rond). Ce retour visuel avec ces emojis colorés nous permet d'identifier facilement chaque tapis.

Puis j'ai créé quelques variables qui définissent les conditions de détection de chute et la gestion des notifications. Précisément, nous stockons l'heure de début d'une chute potentielle, si une alerte a été envoyée ou pas, le nombre minimal de tapis qui doivent être activé (3), le temps minimum d'activation (5s), le temps entre 2 notifications pour éviter la redondance (60s), et l'heure de la dernière notification envoyée.

V-5 Algorithmes pour la détection de chute par les tapis

Une fois que nous avons ces variables, nos algorithmes comptent le nombre de tapis actifs. Si ce nombre dépasse le seuil (3), nous enregistrons l'heure de début; et si ce nombre re-devient insuffisant, la détection est annulée. Pour cette dernière situation, nous réinitialisons le début de la détection et notification envoyée.

Si le nombre de tapis activés reste suffisant et que la détection est en cours : nous calculons le temps écoulé. Si ce temps dépasse 5 secondes, nous considérons qu'une chute réelle a eu lieu, et une notification sera envoyée.

Puis nous calculons le temps écoulé depuis la dernière notification. Si ce temps dépasse le temps d'attente confirmation (60 sec), on considère la confirmation comme non reçue. En conséquence : nous réinitialisons le système et on prépare un nouvel envoi de notification.

Pour démarrer le programme, nous cliquons sur "Série". Et si tout est correct, dans la console, nous voyons quels sont les tapis appuyés en temps réel, avec tous les messages qui informe l'état de ce système.

V-6 Mécanisme d'interaction tapis-détection

Processus de communication

Lorsque les capteurs de pression intégrés dans les tapis détectent une activité significative (présence, mouvement inhabituel ou pression anormale), le microcontrôleur ESP32 initie une requête HTTP GET vers l'endpoint /tapis_detection du serveur Flask hébergé dans le script alerte_chute.py.

Cette requête sert de signal d'alerte précoce, indiquant au système de détection visuelle qu'une situation potentiellement à risque pourrait se développer.

Envoi de la requête HTTP par l'ESP32

L'ESP32 utilise un module de communication sans fil pour envoyer une requête HTTP GET au serveur de détection. Le processus d'envoi s'effectue via une fonction dédiée qui s'assure que les conditions préalables sont remplies, notamment la connectivité Wi-Fi et la synchronisation temporelle.

La requête est formatée selon un modèle prédéfini, incorporant l'adresse IP et le port du serveur cible. L'ESP32 s'assure de la validité des données temporelles avant l'envoi pour garantir la cohérence des horodatages. Si la connexion est établie avec succès, le microcontrôleur envoie la requête et attend une confirmation du serveur.

Un mécanisme de délai entre les requêtes est également implémenté pour éviter la surcharge du réseau et du serveur. Ce système assure un équilibre entre réactivité et efficacité des ressources réseau.

Adaptation dynamique de la sensibilité

À la réception de cette requête, le serveur Flask exécute la fonction tapis_detection() qui modifie immédiatement le comportement du détecteur de chutes à travers l'appel à la fonction set_falling_multiplier(1.2). Cette fonction ajuste la variable globale FALLING_CONFIDENCE_MULTIPLIER, augmentant sa valeur de 1.0 (valeur par défaut) à 1.2.

Cette modification a pour effet direct de renforcer la confiance associée à la classe 'falling' dans le modèle de détection. Concrètement, lors de l'analyse d'images où une chute pourrait

se produire, le système devient plus sensible aux mouvements susceptibles d'être des chutes, sans pour autant compromettre la précision globale du système.

Confirmation et réponse du système

Une fois la modification du multiplicateur effectuée, le serveur renvoie une réponse JSON à l'ESP32 confirmant que le paramètre a bien été ajusté. Cette réponse contient le statut de l'opération ainsi que la nouvelle valeur du multiplicateur.

Cette confirmation permet au microcontrôleur ESP32 de s'assurer que la sensibilité du système de détection visuelle a bien été ajustée en fonction de l'activité détectée sur le tapis, complétant ainsi la boucle de communication entre les deux composants du système.

V-7 Interprétation et conclusion

Implications et bénéfices

Cette approche d'adaptation dynamique présente plusieurs avantages significatifs :

1. **Détection contextualisée** : Le système ajuste sa sensibilité en fonction des conditions environnementales réelles, réduisant les faux négatifs dans les situations à risque.
2. **Réactivité accrue** : En augmentant temporairement la sensibilité de détection suite à une alerte des capteurs de pression, le système peut identifier une chute plus rapidement et avec plus de fiabilité.
3. **Réduction des fausses alertes** : L'intégration de multiples sources de données (vision et capteurs de pression) permet de corroborer les informations, diminuant ainsi le taux de faux positifs.
4. **Optimisation des ressources** : En fonctionnement normal, le système maintient une sensibilité standard, n'augmentant celle-ci que lorsque les capteurs de tapis indiquent un risque potentiel.

Conclusion

L'interaction entre les capteurs de tapis et le système de détection de chutes comble parfaitement les lacunes de la détection vidéo qui étaient la gestion des angles morts. Celle-ci apporte un précision supplémentaire sans pour autant être décisive dans la détection de chute car elle reste moins fiable que la vidéo. Cette synergie entre les différentes technologies rend le système plus robuste et réactif, capable de s'adapter dynamiquement aux situations à risque.

VI - Conclusion et perspectives

VI- 1 Conclusion

Le projet Detect'Chute a permis de développer une solution combinée d'une caméra et des capteurs de pression pour la détection des chutes, avec un serveur Flask pour la gestion des alertes. En intégrant Mediapipe pour l'analyse des postures et les capteurs de pression au sol , le système a démontré une très bonne précision et efficacité pour détecter les chutes, tout en réduisant les faux positifs. Toutes les données sont analysées en temps réel, puis en cas d'une chute détectée, une alerte à l'aidant via l'application Android sera envoyée automatiquement. L'alerte est accompagnée d'un message vocal généré par Google Text-to-Speech (gTTS) pour rassurer la personne en détresse. L'aidant peut ensuite interagir avec le système via l'application en écoutant l'alerte, envoyant une confirmation ou un message vocal, ce qui garantit une communication bi-directionnelle.

VI-2 Perspectives

La caméra reste un capteur très inclusif et cela pose des problèmes éthiques et de confidentialité de la vie privée, surtout dans le cadre du domicile, dans des espaces critiques comme la salle de bain ou la chambre à coucher. Son utilisation peut être perçue comme une atteinte à l'intimité des utilisateurs, ce qui limite son acceptabilité. Cependant, en s'appuyant uniquement sur des capteurs environnementaux comme les tapis de pression, la fiabilité du système n'est pas garantie. Ces capteurs peuvent détecter une présence au sol, mais ils ne peuvent pas toujours distinguer une chute d'une simple position allongée.

En plus, ça serait idéal de faire des tests de notre système en conditions réelles, permettrait d'évaluer son efficacité dans des conditions variées et d'adapter son fonctionnement aux besoins des utilisateurs.