Radar Piéton : d’un capteur radar à la reconnaissance d’image en temps réel

Le point de départ

L’idée initiale était simple : permettre à un piéton de savoir si un vélo, une trottinette ou un cycliste approche par l’arrière, sans avoir à se retourner régulièrement. Le cas d’usage visé était la marche ou la course sur des voies partagées, où un usager plus rapide peut arriver sans que le piéton l’entende ou l’anticipe.

La première piste envisagée reprenait le principe des radars déjà utilisés côté cyclistes, comme le Garmin Varia : un capteur monté à l’arrière qui détecte les véhicules approchants et déclenche une alerte progressive selon la distance. L’objectif était d’adapter cette logique à un usage piéton, avec un budget et une complexité de fabrication réduits.

Exploration des capteurs radar, ultrason et LiDAR

Plusieurs familles de capteurs ont été testées ou évaluées pour cette première approche :

- Ultrason (HC-SR04) : peu coûteux et simple à mettre en œuvre, mais avec une portée et un angle de détection trop limités pour capter un objet en approche rapide à plusieurs mètres de distance.
- Radar Doppler (RCWL-0516) : détecte le mouvement mais ne fournit aucune information de distance ni de direction, ce qui le rend inutilisable pour distinguer un objet qui s’approche d’un simple mouvement latéral (piéton croisant, feuillage, etc.).
- Radar mmWave (HLK-LD2410 / LD2450) : plus prometteur, avec une

détection de présence et de mouvement plus fine, mais une portée et une résolution angulaire qui restaient insuffisantes pour un usage fiable en extérieur à faible coût.
- LiDAR (TF-Luna) : bonne précision de distance sur un axe, mais un champ de détection ponctuel (un seul point, pas une zone), ce qui aurait nécessité un mécanisme de balayage pour couvrir un angle utile derrière le piéton.

Aucune de ces solutions, prise isolément et à ce niveau de budget, n'offrait le compromis nécessaire entre portée, angle de couverture et fiabilité en conditions réelles de marche. Le choix s'est donc porté vers une approche différente : utiliser une caméra et un traitement d'image, plutôt qu'un capteur de distance dédié.

Le changement d'approche : caméra et intelligence artificielle embarquée

Le nouveau principe repose sur l'utilisation de la caméra d'un smartphone, associée à un modèle de reconnaissance d'objets exécuté directement sur l'appareil, sans envoi d'image vers un serveur distant. Le projet a pris la forme d'une PWA (Progressive Web App), hébergée sur GitHub Pages, utilisant TensorFlow.js et le modèle pré-entraîné COCO-SSD (variante légère `lite_mobilenet_v2`) pour analyser en continu le flux de la caméra arrière du téléphone.

Le principe de fonctionnement retenu est le suivant :

- la caméra du téléphone, portée dans le dos, filme en continu ;
- chaque image est réduite et analysée par le modèle de détection

d’objets ;
- la taille de la silhouette détectée dans l’image, et sa vitesse de grossissement d’une image à l’autre, servent de proxy pour estimer la proximité et la vitesse de rapprochement d’un objet ;
- selon ces valeurs, l’application déclenche une alerte progressive (vibration, son, puis annonce vocale).

Cette approche ne mesure pas de distance réelle au sens strict : elle s’appuie sur un modèle géométrique simplifié (une projection de type sténopé), en supposant une taille humaine moyenne et un champ de vision de caméra connu, pour convertir la taille apparente d’une personne dans l’image en une distance approximative, puis en une vitesse de rapprochement.

Premiers ajustements après tests de terrain

Les premiers tests en conditions réelles ont fait apparaître un problème inattendu : le modèle de détection, initialement configuré pour reconnaître spécifiquement la classe « vélo » (`bicycle`), ne parvenait à identifier un vélo que lorsqu’il était vu de profil. Un vélo vu de face ou de trois-quarts, configuration la plus fréquente pour un objet qui approche par l’arrière, n’était simplement pas reconnu comme tel.

La solution retenue a été de changer de classe cible : plutôt que de chercher à reconnaître le vélo lui-même, l’application détecte désormais toute silhouette humaine (classe `person`), que la personne soit à pied ou à vélo. Cette classe reste identifiable sous la plupart des angles, contrairement à la forme d’un vélo. Une estimation de vitesse de rapprochement, calculée à partir du modèle de distance décrit plus haut, permet ensuite de formuler une hypothèse : au-delà d’un certain seuil de vitesse relative (5 km/h, au-dessus d’une allure de

marche normale), l'application suppose qu'il s'agit d'un vélo plutôt que d'un piéton, et l'annonce vocalement.

Un second problème est apparu au cours des mêmes tests : une surchauffe notable du téléphone après quelques minutes d'utilisation continue. Plusieurs optimisations ont été mises en place pour y répondre :

- réduction de la résolution de capture caméra et de l'image effectivement transmise au modèle de détection ;
- cadence de détection adaptative, ralentie tant qu'aucune personne n'est détectée, accélérée dès qu'un suivi est en cours ;
- arrêt complet de la caméra et de la boucle de détection lorsque l'application passe en arrière-plan.

Adaptation à un usage sans regarder l'écran

L'usage réel du dispositif — téléphone porté dans une pochette dans le dos — a orienté plusieurs évolutions vers une interface pensée pour être entendue plutôt que vue :

- ajout d'annonces vocales (« Piéton » / « Vélo ») via la synthèse vocale du navigateur, qui transitent par la sortie audio active du téléphone, écouteurs Bluetooth compris ;
- conservation des vibrations en complément du son ;
- ajout d'un mode « écran éteint » masquant l'aperçu caméra et l'affichage des détections, ne conservant qu'un indicateur de statut minimal, pour réduire la consommation liée au rendu visuel ;
- persistance des réglages (seuils de sensibilité, de confiance de détection, préférences son/vibration) d'une session à l'autre.

Un point de vigilance structurel est apparu au cours de ces tests : que ce soit sur Android ou iOS, le système d'exploitation suspend l'exécution du code JavaScript et coupe l'accès à la caméra dès que l'écran s'éteint ou que l'application passe réellement en arrière-plan. L'écran doit donc rester allumé (bien que non regardé) pendant toute la durée d'utilisation, ce qui constitue une contrainte incontournable pour une application fonctionnant dans un navigateur.

Vers une caméra déportée

Pour ne pas mobiliser en permanence le téléphone comme unique support de la caméra, plusieurs pistes ont été explorées afin de déporter la fonction caméra vers un module externe, connecté en USB :

- la réutilisation d'une caméra CSI (Arducam IMX708) déjà utilisée sur un autre projet a été envisagée, mais nécessite un adaptateur CSI vers USB dédié, dont le coût rejoint celui d'une caméra USB neuve ;
- il a été établi qu'aucune caméra USB, aussi bien reconnue soit-elle par le système Android, n'est directement accessible depuis un navigateur : Chrome pour Android ne remonte pas les caméras UVC externes à l'API `getUserMedia`, contrairement à Chrome sur ordinateur. Seule une application native (ou un empaquetage via un framework comme Capacitor) peut accéder à ce type de périphérique.

Une caméra USB compacte, UVC, à focale fixe (3,6 mm, environ 90° de champ de vision, capteur 2MP) a finalement été choisie et achetée, en remplacement de la piste de réutilisation de la caméra existante. Le passage à une version native de l'application, seule à même d'exploiter cette caméra externe, reste l'étape suivante, une fois les tests de calibration terminés sur la version

actuelle en PWA.

Où en est le projet

À ce stade, l'approche de reconnaissance en temps réel — détection de personnes par caméra et modèle embarqué, estimation de proximité et de vitesse de rapprochement, distinction piéton/vélo par la vitesse relative, alerte progressive sonore et vibratoire — fonctionne et a été validée par des premiers tests en conditions réelles, jugés fiables et robustes. Les évolutions en cours portent désormais sur la partie matérielle : le passage à une caméra externe déportée, et l'empaquetage de l'application en version native pour lever les limitations propres au navigateur.

Last version of prototype RadarPieton

Author

Patrick

GPM Factory