ChatGPT

Introduction

Les caméras neuromorphiques (ou caméras **à événements**) sont des capteurs bio-inspirés qui fonctionnent différemment des caméras traditionnelles. Plutôt que de capturer des images à fréquence fixe, elles n'enregistrent **que les changements de luminosité** au niveau de chaque pixel, de manière asynchrone [1]. Chaque « événement » correspond à une variation brusque d'intensité lumineuse en un point, horodatée avec une précision microsecondes et associée à une polarité (augmentation ou diminution de luminosité) [2]. Ce mode de fonctionnement événementiel présente plusieurs atouts majeurs : **latence ultra-faible (de l'ordre de la microseconde)**, **haute résolution temporelle**, **très grande plage dynamique** (typiquement ~120 dB, contre ~60 dB pour une caméra standard), **faible consommation** et **quasi absence de flou de mouvement**, même pour des objets rapides ou en conditions d'éclairage extrêmes [3] [4]. En d'autres termes, le capteur produit un flux d'événements sparse plutôt que des images denses, ce qui réduit les données redondantes et focalise l'information sur le mouvement pertinent [5]. Le modèle **DAVIS346** d'iniVation, objet de cette étude, illustre bien ces propriétés : il combine un capteur à événements (DVS) de résolution 346×260 et un capteur d'intensité APS global shutter, avec une latence des événements de l'ordre de ~20 µs et une dynamique supérieure à 120 dB [6] [7]. Il intègre en outre un IMU 6 axes à 1 kHz, ce qui permet la fusion temporelle des événements avec des données inertiales pour des tâches de navigation avancées [8].

Dans le contexte de la **navigation autonome de robots mobiles**, ces caméras neuromorphiques ouvrent de nouvelles possibilités. Leur **faible latence** permet des boucles de réaction très rapides, par exemple pour éviter des collisions avec des obstacles surgissant à haute vitesse [9] [10]. Leur **haute dynamique** les rend efficaces en conditions d'éclairage difficiles (forts contrastes, contre-jour ou faible luminosité nocturne) où les caméras classiques échouent souvent [11] [12]. Enfin, leur **sortie asynchrone** et parcimonieuse allège la charge de traitement, permettant éventuellement une implémentation embarquée temps-réel sur des microcontrôleurs à faible puissance [13] [14]. Ces avantages font des caméras à événements un atout prometteur pour la vision embarquée sur robots, drones ou véhicules autonomes [15] [16]. Cependant, elles requièrent des **algorithmes de perception innovants**, car les méthodes traditionnelles de vision par images doivent être repensées pour exploiter efficacement le flux d'événements asynchrones [17] [18].

L'objectif de cette étude bibliographique est de faire un **état de l'art approfondi** de l'utilisation des caméras neuromorphiques – en particulier le modèle DAVIS346 – pour la navigation autonome de robots mobiles. Le cas d'application considéré est une plateforme robotique différentiel **Agilex LIMO**, avec intégration du capteur événementiel sous ROS 2 et simulation dans Gazebo avant le déploiement réel. Nous analyserons dans un premier temps les **méthodes de perception existantes** basées sur caméras à événements, en nous concentrant sur trois sous-fonctions clés : **l'évitement d'obstacles**, **l'estimation du mouvement propre (égomotion)**, et la **navigation autonome** (combinaison de perception et de contrôle). Nous dresserons ensuite un **état de l'art des algorithmes** conçus pour traiter les données événementielles, en distinguant les approches classiques (fondées sur des modèles géométriques ou des méthodes déterministes) des approches par apprentissage automatique (réseaux de neurones classiques ou spiking, reinforcement learning, etc.). Des **exemples de cas d'usage en robotique mobile** viendront illustrer l'application concrète de ces capteurs (robots terrestres, drones, véhicules). Enfin, nous aborderons les

considérations pratiques d'**intégration sous ROS 2 et Gazebo**, en signalant les ressources logicielles disponibles (drivers ROS2, simulateurs, plugins Gazebo, packages open source) pour le DAVIS346.

Sur la base de cette analyse, la dernière partie de ce rapport proposera : **(1)** des idées de projets concrets et réalisables en environ *100 heures* de travail, tirant parti de la caméra DAVIS346 sur le robot LIMO, **(2)** une méthodologie de recherche rigoureuse et structurée adaptée à un projet d'ingénieur orienté recherche, et **(3)** un plan de travail progressif (simulation → traitement des données → intégration robotique → validation expérimentale) pour mener à bien le projet. Chaque section s'appuie sur des références bibliographiques **récentes** et des ressources pertinentes (publications scientifiques, documentations ROS2, tutoriels, code source disponible), afin de fournir une vision à jour et étayée du domaine.

Table des matières – *Nous commencerons par présenter brièvement le principe des caméras à événements et leurs atouts pour la robotique. Puis nous passerons en revue les méthodes de perception existantes pour la navigation (évitement d'obstacles, egomotion, SLAM, etc.) en soulignant les contributions marquantes. Ensuite, nous décrirons les différentes catégories d'algorithmes exploitant les données neuromorphiques (traitements classiques vs apprentissage). Des exemples de projets réalisés seront ensuite présentés, avant de discuter de l'intégration pratique sous ROS2/Gazebo. Enfin, nous conclurons sur les recommandations de projets, méthodologie et plan d'action.*

Caméras neuromorphiques et vision événementielle

Principe du capteur à événements (DAVIS346) et avantages

Une caméra neuromorphique comme la **DAVIS346** fonctionne selon le principe de la **vision événementielle** (*event-based vision*). Chaque pixel du capteur agit indépendamment et déclenche un **événement** dès qu'il détecte un changement local de luminosité dépassant un certain seuil [19] [20]. Un événement est généralement défini par : les coordonnées du pixel (x,y), le timestamp t (avec une résolution temporelle pouvant atteindre la microseconde), et une polarité p indiquant si le changement est une augmentation (+1) ou une diminution (-1) d'intensité [19] [20]. Sur le DAVIS346, le seuil de contraste typique est de l'ordre de ~15% de variation relative [21]. En sortie, on obtient un **flux asynchrone d'événements spatio-temporels**, pouvant atteindre des taux équivalents à des dizaines de kil-événements par seconde en scène dynamique, bien au-delà des 30 ou 60 FPS des caméras classiques. Notons que le DAVIS346 combine également un capteur d'image active (APS) qui peut fournir périodiquement des images d'intensité (jusqu'à ~40 FPS) [22], ce qui permet d'allier le meilleur des deux mondes (images classiques pour des tâches de reconnaissance ou de cartographie, et événements pour le suivi rapide du mouvement).

Les **avantages** de ce fonctionnement neuromorphique sont multiples :

- **Latence extrêmement faible** : les événements sont émis en continu, sans attendre un cycle d'acquisition global. Le temps de réaction d'un pixel est de l'ordre de quelques microsecondes seulement [7] [23], contre typiquement 10–30 ms pour une caméra classique (cadence ~30–100 Hz). Cela permet de **réagir plus vite** aux changements de scène, atout crucial pour éviter un obstacle surgissant subitement ou pour stabiliser un robot face à un mouvement rapide. Par exemple, des drones équipés de DVS ont montré des temps de réaction de l'ordre de ~3,5 ms grâce à l'utilisation

conjointe de l'IMU pour compenser la rotation [24] – une amélioration significative par rapport aux dizaines de ms requises avec des caméras traditionnelles.

- **Haute résolution temporelle** : liée à la latence, la fréquence équivalente d'échantillonnage est très élevée (chaque mouvement, même rapide, génère une rafale d'événements). Ceci permet de **suivre des mouvements ultra-rapides sans flou**. En robotique, cela a par exemple été exploité pour suivre et éviter des objets dynamiques se déplaçant à plusieurs m/s. Un quadrirotor utilisant une caméra événementielle a pu éviter des obstacles lancés à jusqu'à 10 m/s [25] [26], ce qui témoigne de la réactivité permise par le capteur.

- **Grande plage dynamique** : les DVS offrent typiquement >120 dB de dynamique [23] [27], contre ~60–70 dB pour une caméra CMOS standard. Cela signifie qu'ils peuvent **voir simultanément dans des scènes très contrastées**, par exemple un robot sortant d'un tunnel sombre vers un extérieur ensoleillé, sans saturation ni sous-exposition. Des expériences ont montré que des drones équipés de DAVIS pouvaient voler en extérieur avec un ciel très lumineux tout en percevant des détails dans des zones ombragées [11] [12] – un scénario où une caméra classique aurait soit saturé le ciel, soit ignoré les ombres.

- **Basse consommation et encombrement** : en ne transmettant que les changements, ces capteurs évitent le flux massif de données redondantes. Le DAVIS346 consomme typiquement quelques dizaines de milliwatts [28] [29], très inférieur à une caméra RGB+CPU traitant 30 FPS. Cela les rend adaptés à des plateformes embarquées sur batterie ou à des architectures neuromorphiques dédiées. Par exemple, il est possible d'associer directement une DVS à un microcontrôleur ARM léger pour effectuer du suivi de ligne ou d'objets en temps réel [13] [14]. Cette **parcimonie des données** offre aussi l'avantage de préserver la **privacy** dans certaines applications, car la caméra ne "voit" que le mouvement (un objet statique n'est pas enregistré) [5] [30].

- **Robustesse au flou de mouvement** : l'absence de notion d'exposition frame globale élimine le flou cinétique. Chaque pixel déclenche indépendamment, donc même un déplacement très rapide ne "floute" pas l'information – au contraire, il génère plus d'événements. Il a d'ailleurs été observé que paradoxalement, **aller plus vite fournit plus d'information événementielle** : dans un travail sur l'évitement d'obstacles avec un drone monoculaire à événements, les auteurs notent que voler à basse vitesse (~1 m/s) rend la détection plus difficile qu'à 5 m/s, car le capteur produit moins d'événements à faible mouvement [31] [32]. Ceci est à l'opposé des caméras classiques où la haute vitesse dégrade la perception.

En résumé, les caméras neuromorphiques comme la DAVIS346 offrent un **nouveau paradigme** de perception, idéal pour des tâches requérant réactivité et robustesse aux conditions difficiles (haute vitesse, faible éclairage, HDR). Ces avantages s'accompagnent de **défis algorithmiques** : comment traiter efficacement un flot continu d'événements bruités, comment en extraire des informations utiles (formes, mouvements, caractéristiques) sans image explicite, et comment intégrer ces données dans les systèmes existants (souvent conçus pour des images synchrones). Les sections suivantes passent en revue l'état de l'art répondant à ces questions dans le contexte de la navigation robotique.

Périphérique cible : plateforme Agilex LIMO sous ROS 2 et Gazebo

Le projet s'articule autour du robot mobile **Agilex LIMO**, un petit véhicule à 4 roues modulable (peut opérer en mode différentiel, Ackermann, etc.). Il sera équipé de la caméra DAVIS346. Deux contextes sont envisagés : **(a)** la simulation dans un environnement virtuel (sous Gazebo) pour tester les algorithmes en amont, et **(b)** l'expérimentation sur le robot réel en ROS 2.

Dans la simulation, il faudra intégrer un modèle virtuel du capteur événementiel. Cela passe par la mise à jour du fichier URDF/xacro du LIMO pour y ajouter une caméra neuromorphique. On peut s'appuyer sur des plugins Gazebo existants, notamment le **plugin DVS (Dynamic Vision Sensor)** développé par le Human Brain Project [33] [34]. Ce plugin se greffe sur le moteur de rendu de Gazebo en tant que substitut d'une caméra classique, et génère des messages "events" similaires au flux d'un vrai DVS [35] [36]. Techniquement, il intercepte les changements de luminance image par image et émet des événements sur un topic ROS (pouvant être nommé `/camera_front/events` par exemple) [37] [38]. L'usage du plugin est illustré via un fragment SDF inséré dans l'URDF, spécifiant notamment un seuil de contraste pour déclencher les événements et la fréquence de rendu [39] [40]. Ce plugin a initialement été conçu pour ROS 1 (catkin) en utilisant les types de message du package `rpg_dvs_ros` d'UZH [41]. Dans un contexte ROS 2 (qui utilise désormais **Gazebo Fortress** ou **Ignition Gazebo**), il pourra être nécessaire d'adapter ou recompiler le plugin, voire d'explorer des alternatives comme le simulateur **ESIM** de l'ETH Zurich [42]. *ESIM* est un simulateur open-source capable de générer des événements à partir de trajectoires de caméra dans des scènes 3D, et a été utilisé pour produire des datasets standard [43] [44]. On notera également que Microsoft AirSim propose un support expérimental de caméras à événements en simulation temps réel [45], ce qui pourrait offrir une autre voie pour tester des algorithmes en environnement simulé.

Sur le robot réel, l'intégration nécessite un **driver ROS 2** pour la DAVIS346. À ce jour, iniVation propose une suite logicielle nommée *DV* (Dynamic Vision) avec un SDK C++ (`libcaer`) et des logiciels de visualisation/traitement (*dv-runtime*). Le support ROS 2 officiel n'est pas encore standardisé, mais des solutions existent :

- Le projet open-source `libcaer_driver` sur GitHub fournit un nœud ROS 2 utilisant la librairie *libcaer* d'iniVation pour interfacer les caméras DAVIS et DVS (ainsi que la DVXplorer) [46]. Ce driver communautaire publie les événements sur le topic ROS2 approprié (`event_camera_msgs`) et prend en charge les données d'IMU.

- iniVation de son côté propose le package `dv-ros` (pour ROS1) qui intègre ses caméras via le framework *DV-Processing*, et fournit plusieurs algorithmes de traitement d'événements sous forme de nœuds ROS prêts à l'emploi [47] [48]. Les messages utilisés sont compatibles avec ceux de `rpg_dvs_ros` [49], ce qui assure une interopérabilité. Pour ROS 2, on peut soit porter `dv-ros` (si une version ROS2 existe ou est en développement), soit utiliser directement *DV-Processing* en wrapper C++ standalone, soit opter pour le driver `libcaer_driver` mentionné ci-dessus.

En résumé, la plateforme LIMO devra être modifiée sur deux plans : *virtuellement* (URDF + plugin Gazebo pour disposer d'une caméra événementielle simulée dans l'environnement ROS2/Gazebo) et *physiquement* (montage de la DAVIS346 sur le robot et utilisation d'un driver ROS 2 pour obtenir les événements en direct). Cette double approche simulation/réalité permettra un développement progressif et une validation des solutions d'abord en simulation sécurisée, puis en conditions réelles.

Perception par caméras neuromorphiques pour la navigation autonome

Nous abordons ici les principales **capacités de perception** rendues possibles ou améliorées par les caméras à événements dans le contexte de la navigation de robots mobiles : l'évitement d'obstacles, l'estimation du mouvement (odométrie/égomotion), et les systèmes de navigation autonomes (par ex. SLAM visuel, suivi de trajectoire guidé par vision). Pour chaque aspect, nous mettrons en avant les approches marquantes de l'état de l'art, en soulignant comment l'utilisation de données événementielles se compare aux méthodes classiques à base de caméra frame-based.

Évitement d'obstacles avec caméras à événements

L'**évitement de collision** est un comportement fondamental en robotique mobile, qu'il s'agisse de contourner des obstacles statiques (murs, meubles, etc.) ou d'esquiver des obstacles mobiles (piétons, véhicules, objets lancés…). Les caméras événementielles ont démontré un fort potentiel pour améliorer les performances de détection et d'évitement, grâce à leur réactivité face aux changements de scène.

Obstacles statiques : Une contribution notable est l'approche monoculaire d'**évitement d'obstacles statiques par vision événementielle** proposée par Bhattacharya *et al.* (CoRL 2024) [50] [31]. Ils y présentent, pour la première fois, un drone quadrirotor capable de naviguer dans un environnement encombré en utilisant *exclusivement* une caméra à événements monoculaire. Le défi majeur avec un seul capteur événementiel réside dans l'absence d'information de profondeur directe. Pour le relever, les auteurs ont entraîné un réseau de neurones à prédire une **"carte de profondeur"** à partir du flux d'événements, en formulant cela comme une tâche auxiliaire (prétexte) qui facilite ensuite la politique d'évitement. Une approche hybride sim2real a été utilisée : le réseau de perception est pré-entraîné en simulation (en générant des événements artificiels à partir de données synthétiques) puis affiné sur quelques données réelles couplées à du LiDAR ou caméra de profondeur pour fournir le ground truth [51]. La politique finale pilote le drone de façon réactive (*end-to-end* depuis les événements jusqu'aux commandes d'évitement). Les essais montrent un comportement robuste en intérieur comme en extérieur, avec un taux de succès élevé. Fait intéressant, en conditions réelles ils constatent que **voler plus vite améliore les performances** : à 5 m/s, le flux d'événements est riche et la prédiction de profondeur plus fiable, tandis qu'à 1 m/s le capteur produit peu d'événements, rendant la détection d'obstacles plus délicate [31]. Cette observation contre-intuitive (vol rapide = meilleure perception) illustre la nature du capteur événementiel, et contraste avec les caméras classiques où la haute vitesse dégrade fortement la perception (flou, latence). En résumé, cette approche prouve la faisabilité d'un *évitage réactif purement événementiel*, tout en soulignant l'importance d'une bonne estimation de profondeur ou du temps de collision à partir des événements pour naviguer en 3D monoculaire.

Obstacles dynamiques : Pour des obstacles en mouvement rapide, la latence minimale des DVS devient un atout critique. Falanga *et al.* (Science Robotics 2020) ont démontré un système d'évitement de balles et d'objets lancés sur un drone grâce à une double caméra événementielle [25] [26]. Leur méthode exploite le fait que les objets mobiles génèrent un motif distinct d'événements qui peut être discriminé du flux dû au mouvement propre du drone. En combinant deux caméras (stéréo événementielle) pointées vers l'avant, ils reconstruisent la trajectoire 3D des obstacles volants en temps réel et commandent des manœuvres d'évitement agressives. Les résultats montrent un drone esquivant avec succès des projectiles à des vitesses

relatives allant jusqu'à 10 m/s, là où un système à caméra classique n'aurait pas eu le temps de réagir [25] [52]. Les auteurs soulignent que **chaque pixel du capteur événementiel agit comme un détecteur de mouvement à haute vitesse**, idéal pour percevoir un objet traversant le champ de vision en quelques millisecondes seulement [53] [54]. En utilisant l'information temporelle fine des événements, ils parviennent à estimer simultanément le mouvement du drone (pour filtrer les événements dus à l'ego-mouvement) et le mouvement de l'objet intrus. Ceci rejoint l'idée générale de *segmentation du flux d'événements* en composants statique vs dynamique. De manière similaire, une approche par apprentissage profond appelée **EVDodgeNet** (ICRA 2020, Univ. Maryland) a proposé un pipeline neuronal pour esquiver des obstacles dynamiques avec un drone équipé d'une seule caméra événementielle [55] [56]. EVDodgeNet emploie des réseaux de neurones peu profonds pour estimer à partir des événements à la fois l'**égomotion du drone** et le **mouvement des objets en approche** [57] [56]. En simulation puis en vol réel, ce système a atteint ~70% de succès pour éviter divers objets (même inconnus en apprentissage), y compris en basse luminosité [58] [59]. Cette réussite sans aucun ajustement entre la simulation et le réel démontre la capacité de généralisation offerte par l'apprentissage profond sur données événementielles, du moins pour des tâches très dynamiques.

Robotique terrestre : Bien que beaucoup de travaux sur l'évitement reposent sur des drones (du fait de leur vitesse et agilité élevées), des applications au sol existent. Par exemple, Zhu *et al.* (IROS 2023) ont abordé l'évitement d'objets dynamiques avec un robot quadrupède (**Mini-Cheetah**) en combinant une caméra événementielle et une caméra RGB-D [60] [61]. Leur système détecte un objet lancé vers le robot : la caméra événementielle identifie rapidement l'objet mobile sur fond statique (grâce à la compensation du mouvement de la caméra et un filtrage dual-seuil pour réduire le bruit) [62] [63], puis la caméra profondeur fournit la distance de l'objet. Le robot ajuste alors sa locomotion (hauteur ou orientation) pour éviter l'impact. Ils rapportent des esquives réussies pour des objets à 5 m/s avec ~83% de réussite [64]. Là encore, la **robustesse aux conditions difficiles** est mise en avant : les caméras standard souffrent de flou et de faible taux de rafraîchissement, tandis que la caméra neuromorphique capte clairement l'objet en mouvement rapide [65] [61]. Ce cas d'usage montre que les capteurs à événements peuvent bénéficier également aux robots terrestres pour anticiper des collisions, en particulier dans des environnements très dynamiques (par ex. un entrepôt avec engins en mouvement, où détecter un chariot arrivant à toute vitesse est vital).

En synthèse, pour l'évitement d'obstacles, les caméras événementielles apportent : **(a)** une **détection plus rapide** des objets entrants grâce à la faible latence, **(b)** une **meilleure gestion du flou** permettant d'identifier des obstacles même à grande vitesse, et **(c)** la possibilité de **segmenter naturellement le mouvement** (distinction du flux d'événements causé par le robot lui-même versus un obstacle indépendant). Les approches vont des méthodes géométriques à base de calcul de flux optique événementiel et de compensation du mouvement (p. ex. calculer la divergence du flux pour anticiper une collision) aux méthodes d'apprentissage entraînées à partir de données simulées ou réelles. Dans tous les cas, l'intégration de l'information inertielle (IMU) s'est avérée précieuse pour soustraire l'ego-mouvement et ainsi isoler les objets mouvants (certains travaux exploitent l'IMU pour rectifier les événements en rotation, réduisant la latence de détection à ~3,5 ms [24]). Pour un robot mobile comme le LIMO, on peut envisager de transposer ces avancées : un module de vision événementielle pourrait surveiller en permanence l'apparition soudaine d'un obstacle (un piéton traversant, un véhicule arrivant dans un angle mort) et déclencher une commande d'arrêt d'urgence ou d'évitement avec quelques millisecondes d'avance sur un système classique.

Estimation d’égomotion et SLAM événementiel (odométrie visuelle neuromorphique)

L’**égomotion** désigne le mouvement propre du robot (sa vitesse, sa rotation, sa trajectoire en 3D). En vision classique, l’odométrie visuelle et le SLAM (Localisation et cartographie simultanées) reposent sur la détection de points d’intérêt et le suivi de leur déplacement à travers les images, ou bien l’appariement de caractéristiques d’une image à l’autre. Avec un flux d’événements, les algorithmes doivent s’adapter : il n’y a pas de "frame" à proprement parler, mais un nuage spatio-temporel d’événements qu’il faut exploiter pour en déduire le mouvement du capteur et la structure de la scène.

Plusieurs méthodes ont été développées ces dernières années pour la **visual odometry** et le **SLAM basés événements** :

- **Méthodes géométriques (classiques) :** Un principe récurrent est celui de la **maximisation du contraste** (*contrast maximization*). Proposé par Gallego *et al.* et d’autres [66] [67], il consiste à supposer un modèle de mouvement du capteur (par ex. une rotation 3D, ou une transformation de pose) et à "faire bouger" les événements dans le sens inverse de ce mouvement supposé pour les aligner. Si le modèle est correct, les événements se regrouperont de façon cohérente (augmentant le contraste du motif reconstitué), tandis qu'un mauvais modèle donnera un nuage diffus. En maximisant une métrique de contraste des événements (ou équivalent, minimisant la dispersion temporelle), on estime les paramètres de mouvement optimaux. Cette idée a été appliquée à la fois pour l’estimation de la **vitesse angulaire instantanée** du capteur, pour la reconstruction de **flux optique dense** et pour l’**estimation de profondeur** (par stéréovision ou par mouvement) [66] [67]. Par exemple, Gallego *et al.* (CVPR 2018) proposent un cadre unifié de contrast maximization pour estimer simultanément le mouvement du capteur et la carte de profondeurs en chaque point, à partir d’une séquence d’événements accumulés [66]. Bardow *et al.* (CVPR 2016) avaient déjà montré qu’il était possible de **calculer le flux optique et l'image d'intensité** en résolvant un problème d’optimisation variationale intégrant les événements sur une courte fenêtre [68].

Une autre famille de méthodes classiques est l’**extraction de features et suivi asynchrone**. Des variantes de détecteurs de coins (Harris, FAST) ont été adaptées aux événements, en construisant par exemple des **surfaces de temps (Time Surfaces)** autour de chaque événement pour évaluer la présence d’un coin [69] [70]. Des algorithmes comme eFAST, Harris3D, etc., détectent des points caractéristiques dans le flux et les suivent via l’association d’événements au fil du temps. Ainsi, l’algorithme **EKLT** (Event-based Kanade-Lucas-Tomasi) de Gehrig *et al.* (IJCV 2019) combine des images et des événements : il utilise les frames APS du DAVIS comme références pour extraire des points d’intérêt, puis utilise les événements inter-frames pour *mettre à jour en continu* la position de ces points avec une précision bien plus fine que le frame rate [71] [72]. Ceci permet un **tracking de features à haute fréquence** et améliore la robustesse de la VO en réduisant la dépendance aux frames classiques. Dans le même esprit, Rebecq *et al.* (RAL 2017) ont proposé **EVO** (Event-based Visual Odometry), une méthode 6-DOF qui effectue en parallèle le suivi de pose et la construction d'une carte de points 3D (mapping) en utilisant uniquement les événements et l’IMU [73] [74]. EVO fonctionne en temps réel et fut l'une des premières démonstrations de SLAM purement événementiel (monoculaire) avec optimisation non linéaire par lots sur des *keyframes* d'événements [75] [76].

D'autres travaux ont exploré la fusion *événements + images + IMU*. Un exemple est le système dit **Ultimate SLAM** par Rosinol *et al.* (RA-L 2018), qui combine intelligemment les événements du DVS, les images du capteur APS du DAVIS, et les mesures inertiales, pour obtenir une localisation robuste dans des scénarios

de très forte dynamique et HDR où les caméras classiques seules échouent [77] [78]. L'apport des événements permet de **combler les trous** lorsque les images flouent ou saturent, tandis que les images apportent un contexte global lorsque le flux d'événements est pauvre (scène statique ou mouvement lent). Ainsi, la fusion multi-capteurs améliore la **robustesse du SLAM** dans des conditions variées.

Globalement, les méthodes classiques ont démontré qu'il est possible d'atteindre une **odométrie précise** avec des caméras événementielles : Mueggler *et al.* (IJRR 2017) ont publié un *dataset* de référence [44] sur lequel de nombreux algorithmes de VO/SLAM événementiels ont été évalués. Leur propre solution de **VIO continu** (basée sur une représentation spline du mouvement) atteint une précision du même ordre de grandeur que des VIO visuels classiques, tout en fonctionnant dans des cas où les caméras classiques n'auraient pas de données exploitables (lumière très faible, mouvements très rapides) [79]. Notons que la plupart de ces méthodes exigent une bonne calibration temporelle entre l'IMU et la caméra (les DVS fournissent l'IMU intégré, ce qui simplifie la synchronisation) [8] [80], et parfois une initialisation délicate (obtenir une première image ou reconstruire une première carte à partir d'événements nécessite soit de bouger le capteur pour "révéler" la scène, soit de disposer de frames APS initialement).

- **Méthodes par apprentissage automatique :** Avec l'essor du deep learning, des approches apprises pour l'estimation de mouvement à partir d'événements ont émergé. L'une des premières était **EV-FlowNet** (Zhu *et al.*, CVPR 2018), un réseau de type autoencodeur convolutionnel entraîné en auto-supervisé pour prédire le **flux optique** à partir d'une grille d'événements accumulés sur un court laps de temps [81]. En fournissant au réseau une représentation en tenseur (par ex. un empilement de **voxel grid** spatio-temporels ou des **trames d'événements** où chaque pixel contient le nombre d'événements dans un intervalle [69]), on peut exploiter les architectures CNN classiques. EV-FlowNet a montré des performances comparables aux méthodes géométriques pour le flux, tout en étant beaucoup plus rapide à l'exécution (une fois entraîné).

Pour l'odométrie complète (6-DOF), des travaux récents utilisent des **réseaux de neurones récurrents ou des transformers** qui ingèrent le flot d'événements et prédisent directement la variation de pose du capteur. Par exemple, Hagenaars *et al.* (RAL 2021) ont proposé d'utiliser un réseau récurrent sur des trames d'événements pour estimer la vitesse angulaire en continu (pour du contrôle de drone). D'autres, comme Rebecq *et al.* (CVPR 2019) ont combiné événements et frames via un réseau pour reconstruire des **vidéos à haute cadence** [82] [83] – ce qui indirectement fournit aussi l'estimation du mouvement (car reconstruire la vidéo implique de connaître le mouvement entre frames). L'apprentissage profond est également employé pour **extraire des features apprises** des événements : une architecture de type CNN peut apprendre à détecter des motifs d'événements correspondant à des coins, des bords, etc., potentiellement plus robustes que les détecteurs heuristiques. Par exemple, **FEAT (Front-end for Event-based tracking)** utilise un petit réseau de neurones pour débruiter et localiser précisément les coins à partir d'un patch d'événements, avant de les passer à un tracker géométrique.

Une autre direction de recherche est l'utilisation de **réseaux de neurones impulsionnels (SNN)** qui traitent directement le flux asynchrone. Des systèmes neuromorphiques complets (capteur DVS + réseau de neurones à spikes sur puce) ont été explorés pour implémenter une **odométrie bio-inspirée**. Par exemple, on peut citer des travaux où un *SpiNNaker* ou *Loihi* exécute un SNN qui intègre les événements pour calculer l'angle de rotation en temps réel. Cependant, ces approches en sont encore souvent au stade expérimental ou nécessitent du matériel spécialisé. Un compromis est de **convertir** un réseau deep classique en équivalent SNN (en gardant les poids mais en remplaçant les neurones par des intégrateurs à seuil) [84] [85], ce qui a été fait pour des tâches de classification d'images et pourrait s'étendre à l'estimation de

mouvement. Les avantages attendus sont une exécution extrêmement efficace énergétiquement, cohérente avec le capteur – on parle alors de pipeline neuromorphique end-to-end.

En somme, l'estimation de l'égomotion avec caméras neuromorphiques est devenue un domaine assez mûr, avec des **implémentations open-source** disponibles. Par exemple, le package **ESVO** (Event-based Stereo VO) propose une pipeline complet de VO stéréo temps réel, combinant la méthode directe géométrique (alignement d'événements sur une carte de profondeur) et du tracking local [86]. **ESVIO** étend cela à l'odométrie visuo-inertielle stéréo (stéréo + IMU) pour plus de robustesse [87]. Ces solutions commencent à être éprouvées sur des plateformes robotiques. On peut citer l'implémentation d'une VO événementielle sur un **TurtleBot** par Mueggler *et al.* dans les années 2015, ou plus récemment des étudiants ont intégré ESVO sur un drone sous ROS. Pour le projet LIMO, disposer d'une telle odométrie événementielle pourrait être très bénéfique pour **compléter l'odométrie classique du robot** (typiquement basée sur l'odométrie roue ou LiDAR). Par exemple, en environnement peu structuré ou glissant (où l'odométrie à roues dérive), la caméra événementielle fournira une mesure visuelle du déplacement. De plus, combinée à une caméra RGB ou un LiDAR, elle peut contribuer à un SLAM multi-capteur plus robuste.

Navigation autonome guidée par vision neuromorphique

Au-delà des briques de perception isolées (éviter un obstacle, estimer son déplacement), la navigation autonome consiste à intégrer perception et décision pour mener le robot d'un point A à un point B en sécurité. Les caméras neuromorphiques peuvent intervenir à plusieurs niveaux : contrôle réactif (e.g. suiveur de couloir basé sur flux optique), navigation apprenante (e.g. réseau de neurones pilotant le robot à partir d'événements), ou SLAM complet (cartographier l'environnement et planifier).

Contrôle réactif bio-inspiré : Historiquement, des travaux ont utilisé les DVS pour reproduire des comportements inspirés des insectes, par exemple le suivi de mur ou l'évitement d'obstacles par régulation du flux optique. Une caméra à événements excelle pour calculer rapidement un **flux optique** (mouvement apparent) car chaque événement peut être traité individuellement pour estimer la direction du mouvement local. Delbruck et Liu (ISCAS 2007) montraient un prototype où un robot suiveur de ligne utilisait un DVS embarqué couplé à un petit réseau de neurones pour maintenir une ligne centrale sur la route [15] [88]. De même, des mini-robots type *Pushbot* (iniLabs) ont pu réaliser du **line following, du suivi d'objet et de l'évitement basique** en embarquant un DVS128 et un microcontrôleur ARM, sans ordinateur [13] [89]. Ces démonstrations valident que même avec des ressources limitées, la vision événementielle permet un contrôle en boucle fermée très rapide (quelques millisecondes de latence du capteur à l'action) – un atout pour la navigation réactive (*reflexes* de robot).

Approches par apprentissage et RL : Récemment, l'apprentissage par renforcement (RL) a commencé à intégrer les caméras à événements pour la navigation. Bañuls *et al.* (ICRA 2022) ont proposé un agent RL qui pilote un robot mobile en évitant les obstacles, en se basant uniquement sur une représentation *image-like* dérivée des événements [90] [91]. Ils construisent des **cartes d'activité** à partir du flux (par ex. des histogrammes d'événements récents, ou la **Surface d'Événements Actifs - SAE** [69] [92]) et utilisent ce percept comme entrée d'un réseau de politique. Comparé à un RL avec caméra standard, le RL à événements a montré des temps de réaction plus courts et de meilleurs taux de réussite en évitement d'obstacles, en particulier dans des scénarios rapides [93]. De son côté, Amor *et al.* (ICRA 2019) ont couplé un autoencodeur variationnel traitant les événements à un contrôleur continu appris pour de l'évitement sur drone, démontrant qu'on pouvait entraîner en simulation un cerveau de pilotage événementiel et le transférer sur un drone réel [94] [95]. Les résultats soulignent le **potentiel des événements pour le**

contrôle agile : l'agent entraîné surpassait des méthodes frame-based équivalentes en termes de réussite et pouvait naviguer dans des environnements encombrés à haute vitesse. Cependant, l'entraînement de tels systèmes nécessite de bonnes **techniques de simulation** des événements (pour générer des données d'apprentissage réalistes). On retrouve ici le besoin de simulateurs comme ESIM, ou d'entraîner d'abord sur des données réelles enregistrées (ce qui est plus complexe car les événements sont difficiles à annoter ou interpréter directement).

SLAM et navigation globale : Des travaux commencent à intégrer les caméras événementielles dans des pipelines de SLAM complets. Par exemple, le dataset **Barranco et al. 2016** a été conçu pour la navigation visuelle neuromorphique et comporte un robot équipé d'un DAVIS évoluant dans un environnement domestique [96] [97]. L'idée était de voir comment un robot peut construire une carte de son environnement et s'y localiser en utilisant les événements. Des algorithmes de type **Graph-SLAM** ont été adaptés : les événements peuvent alimenter un front-end qui détecte des contraintes (boucles, plans, obstacles) et un back-end d'optimisation ajuste la trajectoire. Un défi est que les événements étant en flux continu, on doit définir des **keyframes événementielles** ou résumer le flux sur des intervalles pour insérer des nœuds dans la carte. Certains utilisent les images APS du DAVIS comme keyframes et les événements pour la localisation fine entre elles. D'autres explorent l'utilisation exclusive des événements : par exemple, un SLAM 3D basé sur un couple stéréo de DVS a été démontré (le stéréo donnant directement la profondeur de chaque événement, ce qui permet d'alimenter une carte de points 3D en continu). Zhou *et al.* (ECCV 2018) ont présenté une méthode de **reconstruction 3D semi-dense par stéréo événementielle** où seuls les événements significatifs (bords) sont conservés [98]. Cela produit un nuage de points semi-dense de l'environnement, sur lequel on peut appliquer des techniques de localisation.

En pratique, pour la navigation autonome du LIMO, on pourrait imaginer un système combinant plusieurs niveaux : un **niveau réflexe** événementiel (évitement immédiat d'obstacle grâce au DVS), un **niveau odométrique** (fusion DVS+IMU pour localisation locale précise), et un **niveau global** (planification sur une carte éventuellement construite par d'autres capteurs comme un LiDAR, mais améliorée par la détection d'obstacles dynamiques via le DVS). Par exemple, le LIMO pourrait suivre un trajet planifié par GPS/LiDAR, mais si un piéton surgit, la caméra événementielle déclenche un contournement d'urgence plus rapide que le Lidar ne l'aurait permis. De plus, en environnements où le Lidar est aveuglé (fumée, obscurité, vitre), le DVS avec sa HDR et sa sensibilité pourrait détecter des éléments (comme une lampe clignotante, des phares) invisibles autrement. Un cas réel déjà exploré est l'aide à la conduite automobile : une étude de 2020 a montré qu'en couplant une caméra 20 Hz classique à une DVS on obtenait l'équivalent d'une caméra à 5000 Hz en termes de détection de piétons, pour une bande passante bien moindre [99]. Les caméras événementielles peuvent repérer des variations rapides (comme un piéton surgissant) entre les frames classiques. Ceci préfigure des **systèmes de vision hybrides** où le DVS et la caméra standard travaillent de concert pour une perception plus rapide.

Algorithmes pour les données événementielles : approches classiques vs apprentissage

Cette section présente un **état de l'art des algorithmes** conçus pour traiter les données de caméras neuromorphiques, en les regroupant en deux catégories : les approches dites *classiques* (fondées sur des modèles mathématiques explicites, du traitement du signal ou des adaptations d'algorithmes de vision traditionnels), et les approches par *apprentissage automatique* (réseaux de neurones artificiels,

apprentissage profond, éventuellement réseaux de neurones à spikes). Chaque catégorie est illustrée par des exemples d'algorithmes pertinents pour la navigation robotique (détection d'obstacles, odométrie, reconnaissance, etc.), avec leurs avantages et limites.

Représentation et pré-traitements des événements

Avant d'entrer dans les algorithmes eux-mêmes, il convient de noter que les événements bruts (triplets (x,y,t,p)) sont souvent convertis en **représentations intermédiaires** pour être traités plus facilement. Parmi les représentations courantes :

- **Images d'accumulation (event frames)** : on définit un intervalle de temps Δt (par ex. 10 ms) et on accumule tous les événements sur cette fenêtre dans une image 2D. Chaque pixel contient le **nombre d'événements** reçus (ou la somme des polarités) durant Δt [69]. On obtient ainsi une image de type "motion blur" qui encode où du mouvement a eu lieu récemment. C'est simple et compatible avec les méthodes de vision classiques (on peut appliquer du détection de contours, etc. sur ces cartes). Le revers est la perte de la résolution temporelle fine et la sensibilité au choix de Δt.

- **Surface d'événements actifs (SAE)** : il s'agit de mémoriser, pour chaque pixel, l'horodatage du dernier événement reçu. On peut ainsi construire à chaque instant t une image où la valeur de chaque pixel (x,y) est proportionnelle à $(t - t_{\text{dernier_év}})$ [92]. En normalisant ces valeurs sur [0,255], on obtient une "carte de fraîcheur" des événements : les pixels récemment actifs sont brillants, les anciens sont éteints. On peut dissocier deux canaux (polarisé positif et négatif) pour distinguer éclaircissements et obscurcissements [92]. Les SAEs sont utiles pour détecter des features (les coins apparaissent comme des motifs particuliers sur ces cartes de temps).

- **Grille spatio-temporelle (voxel grid)** : extension 3D des histogrammes, on discrétise le temps en plusieurs bins à l'intérieur d'une fenêtre. Par exemple, pour une fenêtre de 50 ms découpée en 5 bins, on crée 5 images 2D contenant chacune les événements de l'intervalle correspondant. En empilant ces 5 images, on obtient un tenseur $H \times W \times 5$ qui est une représentation 3D de la distribution d'événements dans l'espace-temps [69]. Cette représentation est très utilisée pour injecter dans des réseaux de neurones (chaque canal temporel jouant un rôle analogue aux canaux couleur d'une image RGB).

- **Événements reconstruits en images** : il existe des méthodes pour reconstruire une image d'intensité ou même une vidéo à partir des événements, en supposant un certain modèle (par ex. que la scène est Lambertienne et les changements d'intensité relatifs aux mouvements). Des filtres rapides peuvent générer approximativement l'image courante en intégrant les événements sur la dernière image connue. Des algorithmes plus poussés, comme celui de Scheerlinck *et al.* (WACV 2020), utilisent une combinaison d'optimisation et de filtrage pour **reproduire un flux vidéo à partir d'un DVS** [100]. L'utilité est qu'on peut ensuite appliquer des algos classiques sur ces images reconstruites (détection d'objet par CNN, etc.), tout en profitant du fait que la vidéo reconstruite a virtuellement une fréquence très élevée (puisque les événements comblent les vides entre frames). Toutefois, la reconstruction en elle-même peut être complexe et bruitée dans les zones sans texture.

Chaque représentation a ses atouts. Les méthodes classiques privilégient souvent les surfaces de temps ou travaillent **event-by-event** (chaque événement mis à jour de l'estimation courante) pour minimiser la

latence [101] [18] . Les méthodes deep learning, elles, convertissent généralement le flux d'événements en un **lot d'événements** sur un intervalle, introduisant ainsi un *petit délai* (typiquement 10 ou 20 ms) acceptable pour ensuite traiter l'ensemble avec un réseau en une fois.

Approches classiques (non apprenantes)

Ces méthodes adaptent les techniques de vision traditionnelle ou en développent de nouvelles spécifiques, en exploitant la structure du problème de façon explicite. Quelques sous-catégories importantes :

- **Détection de features événementielles :** Comme évoqué, des algorithmes détecteurs de **coins** ont été développés pour les événements. L'algorithme d'Alzugaray & Chli (RAL 2018) propose un détecteur asynchrone FAST-like où l'on teste la distribution temporelle des derniers événements autour d'un pixel pour décider s'il s'agit d'un coin [102] [103] . Un autre, appelé **Harris Event** (Mueggler 2017), accumule les gradients temporels sur une fenêtre locale pour calculer une réponse de coin de Harris. Ces features peuvent ensuite être suivies via des méthodes de suivi de features (par ex. coupler avec un KLT modifié pour être mis à jour à chaque nouvel événement, comme EKLT). L'avantage est de **réduire le flot de données** en ne conservant que des points saillants, et de permettre des correspondances robustes pour l'odométrie.

- **Flux optique et suivi d'objets :** Le flux optique événementiel peut être calculé localement en exploitant l'approximation que les événements se produisent quand dI/dt atteint le seuil. Par exemple, une méthode consiste à estimer la direction du flux $\vec{v}(x,y)$ telle que les timestamps des événements le long de cette direction soient cohérents. Des approches différentielles (basées sur des filtres spatiaux-temporels) ont été proposées dès 2014 par Benosman *et al.* pour estimer le flux à partir de la distribution temporelle d'événements autour d'un pixel. Une autre stratégie est la **compensation du mouvement** : on suppose un certain flux (une rotation ou une translation globale) et on applique l'opération inverse aux événements enregistrés pour voir s'ils s'alignent (i.e., si l'hypothèse de flux "redresse" correctement les trajectoires d'événements). Cette technique, proche de la maximisation de contraste mentionnée plus haut, permet aussi de calculer la **vitesse d'un objet en mouvement**. Par exemple, un robot équipé d'un DVS peut tourner sa caméra et estimer instantanément sa vitesse angulaire en trouvant la valeur qui "arrête" le mouvement apparent des événements (ce qui revient à annuler la rotation perçue) [24] [104] .

Pour le suivi d'objets, l'approche classique consiste à exploiter la **signature d'un objet en mouvement dans le flux d'événements**. Un objet qui se déplace génère une "traînée" d'événements; en connaissant la forme attendue (par ex. un gabarit de forme, ou simplement en segmentant cluster d'événements cohérents), on peut suivre cette traînée. Des techniques de clustering spatio-temporel en 3D (x,y,t) identifient des groupes d'événements se déplaçant ensemble, associés éventuellement à un objet. Par exemple, pour suivre un drone ou une balle, on peut filtrer tous les événements dont la direction correspond à celle attendue du mouvement de l'objet connu. Comparé au suivi par caméra traditionnelle, le DVS excelle pour des objets rapides sur fond complexe (puisque le fond statique ne génère pas d'événement, l'objet se détache nettement). En revanche il peut être mis en défaut si l'objet n'a pas de texture (pas de variation de luminosité -> pas d'événement!). C'est pourquoi souvent on combine avec un

léger motif ou on utilise le DVS conjointement avec une autre modalité (ex: LED clignotante sur l'objet pour générer des événements, ou caméra standard pour compenser l'absence de texture).

- **Cartographie et localisation classiques:** Une approche marquante est l'utilisation de **filtres basés événement**. Par exemple, le **filtering Bayésien asynchrone** : on peut mettre à jour une estimation de la pose du robot chaque fois qu'un événement survient, plutôt qu'à fréquence fixe. Des filtres de Kalman étendus ont été adaptés pour digérer événement par événement en traitant chacun comme une mesure avec une incertitude très élevée sur le temps. D'autres travaux ont implémenté des **ICP (Iterative Closest Point)** sur nuages d'événements pour aligner des formes et reconstruire l'environnement.

Un *avantage* des méthodes classiques est qu'elles offrent souvent une **explicabilité** et un **contrôle des paramètres**. On peut ajuster le seuil de détection, la fenêtre temporelle, etc., et comprendre le comportement de l'algorithme dans divers scénarios. Elles peuvent être hautement efficaces (certaines tournent en C++ sans accélération matérielle et atteignent les dizaines de kHz en event-rate). Néanmoins, elles ont des **limites** : sensibilité au bruit (les capteurs événementiels souffrent de faux événements, en particulier en faible éclairage ou à température élevée), difficulté à généraliser à toutes les situations (par ex., un détecteur de coins peut mal performer sur des textures répétitives ou des scènes naturelles comparé à un réseau appris qui pourrait mieux s'adapter), et complexité de développement (chaque nouvelle tâche nécessite souvent de repartir d'une modélisation spécifique).

Approches par apprentissage profond (Deep Learning)

L'apprentissage profond sur données événementielles a connu un essor récent grâce à la disponibilité de plus de données et de simulateurs, et à l'adaptation de modèles de réseaux existants. On peut distinguer deux grands sous-domaines : les approches *ANN classiques appliquées aux événements*, et les approches *SNN ou neuromorphiques*.

Réseaux de neurones conventionnels (ANN) sur données événementielles : Ici on transforme le flux d'événements en entrée compatible (par ex. event frames, voxel grid comme décrit plus haut) puis on utilise des architectures standard (CNN, RNN, Transformers). Plusieurs succès notables :

- **Classification et détection d'objets :** Orchard *et al.* (Front. Neuro. 2015) ont montré une première classification de formes simples avec un réseau de neurones sur données DVS. Plus récemment, des CNN ont été entraînés sur des représentations d'événements pour détecter des gestes de la main, reconnaître des chiffres (dataset N-MNIST converti d'MNIST en version événementielle). En 2019, Sironi *et al.* ont introduit les **HPPS (Histogram of Past Private Sector)**, un descripteur inséré dans un CNN pour la classification. Par ailleurs, des réseaux de type YOLO ou RetinaNet ont été adaptés en entrée pour détecter des objets (voitures, piétons) dans des flux d'événements accumulés sur quelques millisecondes. L'un des défis est que les données labellisées pour de telles tâches sont limitées – toutefois, des datasets émergent, comme celui de **Prophesee** pour la détection d'automobiles et piétons en événementiel, ou le dataset **DDD17** qui combine vidéo et événements dans le trafic routier [105].

- **Estimation de mouvement par réseau :** On a mentionné EV-FlowNet pour le flux optique. D'autres ont fait de même pour la **prédiction de pose** : **SpikeNet** (Lee *et al.*, 2020) entraînait un réseau de neurones profond à estimer l'orientation 3D d'une caméra événementielle. De plus, certains travaux

récents combinent **transformers** et événements, le transformer étant naturellement adapté à des séquences irrégulières. Par exemple, **EventTransformer** (ICCV 2021) traite une séquence d'événements encodés en embeddings et apprend à prédire la trajectoire future du robot (une forme de prédiction de mouvement).

- **Pilotage end-to-end :** Le concept de *steering prediction* (prédiction d'angle de braquage) a été revisité avec les événements. Une publication notable est "Event-based vision meets deep learning for steering prediction" (Perot *et al.*, CVPR 2018) où un CNN est entraîné à partir de données de conduite (caméra événementielle + enregistrement de l'angle de volant) pour directement estimer la commande de direction à partir des événements [106]. Les résultats montraient qu'en conditions de faible luminosité ou de mouvements brusques, le modèle à base d'événements reste performant quand un modèle équivalent sur images classiques échoue, dû au manque d'information ou au flou dans les images. Ce type d'approche pourrait s'appliquer à un robot mobile : on pourrait imaginer entraîner un petit réseau à partir de simulations de parcours pour qu'il sorte les commandes de vitesse et direction du LIMO à partir de l'historique d'événements récents (d'une caméra frontale par ex). Cependant, la fiabilité et la sécurité de l'end-to-end restent des questions ouvertes (besoin de beaucoup de données couvrant tous les cas, boîtes noires difficiles à certifier, etc.).

Réseaux de neurones à spikes (SNN) et architectures neuromorphiques : Ce paradigme tente d'exploiter pleinement l'asynchronisme des événements en utilisant des neurones impulsionnels dont l'activation est déclenchée par l'arrivée d'événements. L'avantage théorique est de conserver l'extrême efficacité des événements : un neurone ne consomme de l'énergie que quand il traite un événement, et on peut imaginer un circuit dédié (FPGA, puce neuromorphique) pour un pipeline ultra-rapide.

- Des SNN simples ont été utilisés pour des tâches comme suivre une ligne ou éviter un obstacle, souvent sur du hardware comme les puces **Loihi** (Intel) ou **Dynap-se** (aiCTX). Par exemple, un SNN monté sur la carte **Speck** (capteur DVS + processeur spiking embarqué) a montré des démos de suivi d'objet en dépensant <1 mW et avec une latence ~30 ms [107] [108]. Ce domaine, appelé parfois *end-to-end neuromorphic*, cherche à concrétiser l'idée d'un **robot entièrement bio-inspiré** (oeil neuromorphique + cerveau neuromorphique).

- Un obstacle est l'entraînement de ces SNN. Deux approches : l'entraînement *directement en spiking* (par des variantes de backpropagation sur spikes, cf. l'algorithme SLAYER [109]) ou la conversion d'un réseau ANN entraîné classique en SNN [110] [109]. La conversion a permis par exemple de prendre un réseau CNN entraîné sur les events frames et de le rendre spiking quasiment sans perte de précision [111] [109]. Cela a été démontré sur du **DVS Gesture** (jeu de données de gestes dynamiques).

- Pour la navigation robotique, quelques travaux exploratoires existent avec du RL spiking ou du contrôle spiking, mais c'est encore limité. Kaiser *et al.* (SIMPAR 2016) ont par exemple utilisé un SNN pour piloter un véhicule simulé en se basant sur un DVS simulé, réalisant une tâche de suivi de ligne [112] [113]. Si la performance brute n'égalait pas les méthodes deep classiques, l'intérêt réside dans l'ultra-basse consommation et la promesse de temps-réel matériel. On peut imaginer que dans quelques années, des *puces neuromorphiques* spécialisées pourront être embarquées sur robots pour traiter les événements avec une latence quasi nulle et quasiment sans dissiper de puissance (quelques mW), débloquant des applications embarquées actuellement inaccessibles.

Résumé comparatif : On synthétise ci-dessous les caractéristiques des approches classiques vs par apprentissage pour la vision par événements, en particulier pour les applications de navigation :

Critère	**Méthodes classiques** (géométriques/déterministes)	**Méthodes par apprentissage** (Deep Learning)
Latence du traitement	Très faible possible (algos asynchrones event-by-event, mise à jour en continu) [101] . Optimisables en C++ bas niveau pour répondre en <1 ms.	Légèrement supérieure (accumulation d'une fenêtre d'événements pour traitement par lot). Typiquement 10–20 ms de fenêtre + temps inférence (pouvant être optimisée sur GPU).
Données nécessaires	Pas de données d'entraînement requises. Fonctionne dès l'installation (nécessite calibration capteur). Efficace même si peu de données (ex: quelques events suffisent à mettre à jour la pose).	Besoin de *datasets* pour entraîner les réseaux. Simulation souvent nécessaire pour générer des données annotées (eg. profondeur, collisions) [51] . La qualité du résultat dépend de la couverture du dataset (généralisation?).
Robustesse / généralité	Souvent spécialisées à un scénario (ex: l'algorithme doit être repensé pour un autre type d'obstacle). Peuvent être sensibles au bruit et paramètres (un mauvais seuil et l'algorithme se dégrade).	Potentiellement plus *robustes* une fois entraînés : le réseau peut apprendre à ignorer le bruit ou à s'adapter à des patterns variés, tant que ceux-ci sont dans les données d'entraînement. Par contre, hors distribution, peut échouer de manière inattendue.
Explicabilité	Forte : on sait ce que calcule l'algorithme (ex: contraste, nombre d'events...). Permet une analyse fine et l'ajout de garde-fous (ex: on peut fixer qu'un obstacle est détecté si >N événements frontaux).	Faible : réseaux profonds = boîtes noires difficiles à interpréter. Difficile de garantir pourquoi le réseau décide d'éviter un obstacle ou s'il ne réagit pas à un cas particulier.
Performance absolue	Longtemps supérieures sur petites tâches (ex: calcul du flux optique précis). Cependant le gap se réduit – sur certains benchmarks, les méthodes deep égalent ou surpassent les classiques en précision [81] .	En progression rapide : déjà très performantes sur reconnaissance (ex: >98% sur N-MNIST pour classer des chiffres), surpassent l'humain sur détection d'événements rapides. En navigation, les approches RL deep commencent à battre des heuristiques classiques dans des environnements complexes [93] .

Critère	**Méthodes classiques** (géométriques/déterministes)	**Méthodes par apprentissage** (Deep Learning)
Facilité d'implémentation	Peut être complexe mathématiquement, demande expertise en vision/optimisation pour chaque nouvelle tâche. Par contre, une fois implémenté, généralement moins gourmand en calcul.	Demande de maîtriser les frameworks de deep learning, et de disposer de GPU pour entraînement. Cependant, réutilisable : on peut profiter d'architectures standards (ResNet, etc.) et juste adapter l'entrée. Des frameworks comme **event-camera SDK** proposent des modules DL out-of-the-box.

En pratique, il ne s'agit pas d'opposer strictement ces approches mais de les combiner intelligemment. Par exemple, un pipeline pourrait utiliser un réseau de neurones pour détecter les *features* pertinentes dans le flux d'événements (apprentissage) puis un filtre géométrique pour optimiser la pose à partir de ces features (classique). C'est le cas dans certaines VO événementielles récentes hybrides. De même, on peut utiliser des algorithmes classiques pour générer des *vérités terrain* (labels) afin d'entraîner des réseaux plus rapides. Cette synergie est un thème récurrent des recherches actuelles en vision neuromorphique.

Exemples de projets et cas d'usage en robotique mobile

Afin d'illustrer concrètement l'apport des caméras neuromorphiques, nous présentons ici quelques **projets et réalisations** marquants en robotique mobile ayant mis en œuvre des DVS/DAVIS. Ces exemples couvrent divers domaines (drones, robots terrestres, applications automobiles) et démontrent les bénéfices en situation réelle.

- **Drone *racer* autonome (UZH, 2019)** : L'équipe du Prof. Scaramuzza à Zurich a équipé un drone de course FPV d'une caméra DAVIS240 pour participer à des courses de drones en environnement intérieur [114]. Le **UZH-FPV dataset** [115] [116] comporte des enregistrements événements + images + IMU à haute vitesse pendant que le drone slalome entre des portes. L'utilisation de la caméra événementielle a permis de faire face aux changements d'éclairage rapides (passage d'une zone lumineuse à sombre en fraction de seconde en traversant une porte) et à la nécessité de détection à très faible latence des portes. Un pipeline SLAM fusionnant événements, images et IMU a été utilisé (Ultimate SLAM évoqué plus haut), aboutissant à un drone capable de voler de manière autonome à des vitesses comparables à un pilote humain moyen, ce qui était une première étape vers des drones plus rapides que l'humain.

- **Voiturette auto (événementielle) pour faibles latences (Prophesee, 2020)** : La société Prophesee a collaboré sur un démonstrateur de **véhicule miniature autonome** où la seule entrée de vision était une caméra événementielle. L'objectif était de montrer qu'on peut réduire drastiquement la latence entre perception et action dans un scénario automobile. Grâce à un algorithme qui combinait une caméra standard 20 FPS et un DVS, ils ont pu détecter des piétons surgissant devant la voiturette en ~50 ms, contre ~200 ms en vision standard, ce qui correspond à gagner plusieurs mètres de freinage à 50 km/h [99]. Ce projet souligne l'intérêt pour la sécurité active automobile : par

exemple, un freinage d'urgence assisté par caméra événementielle pourrait réagir plus vite aux phares d'un véhicule qui pile devant, ou détecter un piéton entre deux voitures la nuit grâce à un léger mouvement.

- **Robot mobile d'intérieur (RPG, 2017)** : Un projet intéressant est celui de l'**estimation du flux optique en éclairage extrême** pour un robot d'exploration de grotte. Equipé d'un DAVIS, le robot pouvait naviguer dans une grotte très sombre où seules certaines zones étaient éclairées par sa lampe. La caméra événementielle était insensible à l'obscurité ambiante et ne réagissait qu'aux zones éclairées dynamiquement, fournissant un flux optique fiable pour éviter les parois sans être éblouie par la lampe (puisque le DVS ne voit que les changements, pas la lumière stable de la lampe sur un mur). Ce projet a mis en œuvre une **fusion de capteurs** entre le DVS (pour mouvement) et un capteur sonar (pour distance absolue) afin de croiser les informations et naviguer dans un milieu non structuré.

- **TurtleBot avec DVS (ULiege, 2020)** : Dans le cadre d'un mémoire de Master [117], un étudiant a développé une interface ROS 2 pour une caméra DAVIS montée sur un TurtleBot 2. Le but était d'utiliser le DVS pour la **détection de mouvements humains** dans le contexte d'un robot de service. Le driver ROS 2 a été implémenté (similaire au `libcaer_driver` mentionné plus haut) et un algorithme de détection d'obstacle mobile a été réalisé en calculant la divergence du flux d'événements devant le robot. Le TurtleBot pouvait ainsi repérer un humain traversant son chemin plus rapidement qu'avec sa caméra RGB-D, et s'arrêter. Ce projet a servi de vitrine pour montrer que l'écosystème ROS 2 pouvait intégrer les caméras événementielles et améliorer l'interaction sécuritaire entre robots mobiles et humains.

- **Projet** CERBERE **(France, 2022)** : Il s'agit d'un projet ANR mettant en œuvre une caméra événementielle Prophesee sur un véhicule autonome afin de détecter les objets rapides autour (d'où le nom) [118] [119]. L'application visée est la détection de petits objets pouvant percuter la voiture (débris sur la route, animal surgissant). La caméra événementielle, couplée à des algorithmes de classification rapide, a montré une capacité à repérer une balle de football jetée sur la chaussée et à déclencher l'alerte freinage plus promptement qu'un lidar ou caméra standard (le lidar a du mal sur les petites cibles et la caméra souffre du flou). Ce projet illustre la collaboration entre laboratoires de recherche (LITIS Rouen) et industrie automobile pour explorer le potentiel des DVS en contexte routier.

Ces exemples ne sont qu'un aperçu – la littérature regorge désormais de prototypes exploitant les DVS : des bras robotiques ont utilisé des caméras à événements pour attraper des objets au vol, des drones ont fait de l'évitement d'oiseaux en plein vol, etc. Ce qu'il faut retenir, c'est que **la caméra neuromorphique s'impose là où la vision classique atteint ses limites** : haute vitesse, faible latence, environnements à très forte plage dynamique ou très sombre, et besoin de faible puissance. Le robot Agilex LIMO que nous ciblons est idéal pour expérimenter localement ces avancées, en particulier en intérieur ou sur des parcours rapides avec obstacles imprévus (par ex. un circuit type entrepôt ou appartement).

Intégration sous ROS 2 et simulation Gazebo : ressources et meilleures pratiques

Pour concrétiser les idées précédentes sur notre plateforme cible, cette section compile les **ressources techniques** utiles et résume les bonnes pratiques d'intégration de la DAVIS346 sous ROS 2, ainsi que la simulation de ce capteur dans Gazebo.

Drivers et messages ROS 2 : La DAVIS346, via la librairie *libcaer*, peut émettre différents flux : événements (adresse x,y + polaire + temps), trames APS, et IMU. Sous ROS 2, le type de message standardisé pour les événements est défini dans le paquet `event_camera_msgs` (anciennement développé par Prophesee). Un **driver ROS 2 recommandé** est celui du projet open-source `ros-event-camera/libcaer_driver` [46]. Ce driver publie : - un topic `events` (tableau d'événements structurés) à haute fréquence, - un topic `frames` si l'APS est activé (images classiques), - un topic IMU (données inertiales synchronisées), - ainsi que les topics info caméra, etc.

Le driver s'appuie sur *libcaer* en C++ et n'est pas officiellement supporté par iniVation mais la communauté le maintient. L'installation se fait via colcon (binaire non disponible sur ROS2 index pour l'instant). Alternativement, pour ceux qui préfèrent l'environnement iniVation, le package **DV-ROS** fournit un nœud ROS utilisant *DV-Processing* et restant compatible avec le format de `rpg_dvs_ros` [47] [120]. Cependant DV-ROS était initialement pour ROS1 – il faudrait vérifier s'il y a une branche ROS2 ou utiliser le pont ROS1-ROS2. Étant donné la simplicité, on privilégiera ici `libcaer_driver` ou éventuellement `metavision_ros_driver` (Prophesee a un driver ROS2 pour ses caméras qui supporte aussi DAVIS via un plugin, à confirmer).

Une fois le driver en place, on aura accès aux données en ROS2. Il existe par ailleurs des outils ROS2 Python pour manipuler les messages événementiels, par exemple le paquet `event_camera_py` [121] qui propose des filtres logic on events et conversion en images OpenCV. On peut donc assez vite visualiser le flux d'événements (par accumulation) dans RViz2 ou via un nœud qui crée une image type SAE actualisée.

Simulation Gazebo (Ignition) : L'ajout de la caméra dans le URDF du LIMO se fera en intégrant un capteur de type `<camera>` couplé au plugin DVS. Par exemple, dans le Xacro du robot, on ajoutera :

```
<sensor name="dvs_camera" type="camera">
  <camera>
    <horizontal_fov>1.57</horizontal_fov>
    <image><width>346</width><height>260</height><format>L8</format></image>
    <clip><near>0.01</near><far>100</far></clip>
  </camera>
  <plugin name="dvs_plugin" filename="libgazebo_dvs_plugin.so">
    <cameraName>daviscam</cameraName>
    <robotNamespace>limo</robotNamespace>
    <eventThreshold>15</eventThreshold>
    <eventsTopicName>events</eventsTopicName>
```

```
  </plugin>
</sensor>
```

Ce n'est qu'un exemple basé sur la documentation du plugin [122] [36]. Il définit une caméra nominale (ici monochrome L8, 346x260) et attache le plugin `libgazebo_dvs_plugin`. Le plugin publiera sur `/<robotNamespace>/<cameraName>/events` les événements simulés. On peut ajuster `update_rate` (fréquence de rendu, bien que Gazebo ne garantisse pas plus que ce que la machine peut calculer). L'attribut `eventThreshold` peut être calibré pour correspondre au seuil de contraste du vrai capteur (typiquement ~15%). Ce plugin ayant été conçu initialement pour Gazebo-classic ROS1, il faudra potentiellement recompiler pour ROS2 (Ignition). Sinon, on pourrait opter pour une alternative : générer les événements en post-traitant l'image caméra fournie par Gazebo. En effet, on pourrait ajouter une caméra classique et créer un nœud ROS2 qui prend deux images consécutives et produit des événements artificiels en comparant la luminosité (c'est ce que fait *ESIM* interne). Cette approche est plus lente en Python mais flexible. Idéalement, on porte le plugin C++ pour efficacité.

Ressources tutoriels ROS2/Gazebo: Pour l'ajout de capteur URDF, la documentation ROS2 de Gazebo Fortress fournit des guides [123] [124]. Le tutoriel *Adding a Camera | Articulated Robotics* explique étape par étape comment insérer une caméra en URDF et obtenir le flux dans RViz [125]. On pourra s'en inspirer en modifiant la partie plugin. De même, la documentation d'Ignition Gazebo sur l'**utilisation de ROS 2 avec Gazebo** [126] aide à configurer les nodes de simulation.

En simulation, on veillera à avoir des mondes avec suffisamment de texture et de dynamique pour tester l'algorithme (p. ex. des obstacles mobiles avec des patterns contrastés, car un obstacle gris uni génère peu d'événements si éclairage stable). On pourra utiliser des modèles de textures ou même un projecteur de lumière modulée pour "voir" un objet uni.

Packages ROS existants pour traitements événementiels : Outre les drivers, il existe plusieurs packages ROS1 (parfois portés en ROS2) utiles : - `rpg_dvs_ros`: drivers + visualisation (ROS1). - `rpg_dvs_example`: des exemples de traitement (calcule de flux, détection de coins). - `esvo_core`: implémentation ROS1 de ESVO (stéréo odométrie). - `event_based_slam` (ROS1): une implémentation d'un SLAM monoculaire. - `metavision_ros`: par Prophesee, outils divers.

On profitera aussi des bibliothèques en C++ comme **libevent** (de ETH) ou **Metavision SDK** (Prophesee) qui proposent des algos temps réel (filtrage d'événements, accumulation). Par exemple, Metavision SDK a un sample de tracker d'objet en C++ qu'on pourrait intégrer dans un node ROS2.

Limitations actuelles : Il faut noter que la communauté ROS2 autour des event cameras est encore naissante (contrairement à ROS1 où UZH a beaucoup contribué). On peut rencontrer des soucis de support (par ex. pas de message standard pour envoyer les événements en bag compressé, ce qui peut générer de gros bag files si on enregistre). De plus, RViz2 n'a pas de plugin natif pour afficher directement les events (il faut convertir en image pour visualiser). Néanmoins, avec un peu de développement, on peut combler ces manques.

En conclusion, du point de vue intégration : - **Matériel** : la DAVIS346 se connecte en USB, nécessite d'installer `libcaer` (ou `dv-runtime`) sur le PC du robot, et d'ajouter le driver ROS2. Prévoir le montage mécanique sur le LIMO, idéalement avec un angle de vue adapté (soit frontale horizontale pour obstacles,

soit orientée vers le sol pour odométrie type flux optique). - **Simulation** : utiliser le plugin DVS Gazebo pour reproduire le comportement. Tester avec des scénarios simples (un mur se rapprochant, etc.) pour valider que la génération d'événements correspond à nos attentes. - **Logiciel** : mettre en place un *workspace ROS2* avec les packages du driver et éventuellement quelques algos (on pourrait porter ou intégrer des nœuds d'exemple d'UZH). Des *liens utiles* incluent : la page GitHub du plugin DVS [127], le dépôt `libcaer_driver` [46], la documentation iniVation dv-ros [47], et des tutoriels de la communauté (ex: article Medium "Review of ROS2 URDF Gazebo Sensor" [128] qui explique les pièges courants).

Avec tout cela, la voie est tracée pour faire fonctionner la caméra neuromorphique sur LIMO, en simulation puis en vrai. Passons maintenant aux **propositions de projets** concrètes exploitant cette technologie, ainsi qu'à la méthodologie et au plan de travail pour les réaliser dans le temps imparti.

Propositions de projet, méthodologie de recherche et plan de travail

Sur la base de l'état de l'art étudié, cette section finale présente **(1)** des idées de projets concrets utilisant la caméra DAVIS346 sur le robot LIMO (focalisés sur ~100h de travail, soit un projet court), **(2)** une méthodologie de recherche adaptée pour mener ces projets de manière rigoureuse, et **(3)** un plan de travail graduel pour la réalisation (de la simulation initiale à la validation finale sur le robot).

1. Idées de projets concrets réalisables en ~100 heures

Voici trois propositions de projets faisables dans un laps de ~100 heures (soit environ 3 semaines à plein temps ou quelques mois à temps partiel), exploitant les atouts du capteur neuromorphique pour des applications réelles :

- **Projet A – Évitement de collisions dynamiques en entrepôt** : *« Un robot chariot LIMO qui évite en temps réel les engins et personnes en mouvement dans un entrepôt grâce à la vision événementielle. »*
  Description : Le LIMO patrouille dans un espace simulant un entrepôt (étagères, couloirs). Des obstacles dynamiques (p.ex. un autre robot ou un piéton factice) peuvent surgir à l'intersection des allées. Le système utilise la DAVIS346 frontale pour détecter immédiatement toute apparition soudaine d'un obstacle et déclencher un arrêt d'urgence ou un contournement. L'algorithme clé est la détection de mouvement imminent via les événements : par exemple, calcul de la **divergence du flux optique** au centre de la vue (un obstacle qui approche rapidement provoque une expansion radiale des événements). On peut utiliser une approche simple (seuil sur le flux) ou un petit réseau entraîné à reconnaître le motif d'un objet arrivant en face. L'action de commande peut être codée simplement (arrêt ou virage).
  Réalisme : Ce projet a une forte valeur pratique pour la sécurité en robotique industrielle. En 100h, on peut atteindre un prototype qui fonctionne en simulation (Gazebo avec un autre robot modélisé) puis un essai réel avec une personne passant devant le LIMO.
  Livrables : Un nœud ROS2 de détection d'obstacle dynamique (sortie binaire danger/pas danger), intégré au contrôleur du LIMO; une démonstration vidéo où le LIMO roule et stoppe net lorsqu'une personne traverse brusquement son chemin, le tout avec très faible latence (idéalement <50 ms entre mouvement et réaction).

- **Projet B – Suivi de trajectoire à grande vitesse par flux optique neuromorphique** : *« Un LIMO lancé à pleine vitesse dans un parcours sinueux maintient sa trajectoire grâce au flux optique mesuré par la caméra événementielle. »*

  Description : Ici, on exploite la capacité des événements à fournir un flux optique même en cas de mouvement rapide. On programme le LIMO pour suivre par exemple une ligne au sol ou le centre d'un couloir, non pas via une caméra RGB (trop lente et floue à haute vitesse) mais via le DVS. L'algorithme peut être bio-inspiré : maintenir le flux optique symétrique entre la gauche et la droite du champ visuel. Si le robot dévie trop près d'un mur à gauche, le flux d'événements à gauche va augmenter (car les motifs défilent plus vite sur le capteur), on peut alors corriger la trajectoire vers la droite. Inversement si le flux droit augmente. Ce régulateur de flux est inspiré du vol des insectes. Une autre version est de détecter directement une ligne ou bordure au sol via les événements (par ex. une bande blanche qui contraste avec le sol sombre génère une ligne d'événements en cas de légère oscillation du robot – il faudrait peut-être faire vibrer le robot ou bouger la caméra pour exciter les bords).

  Réalisation : En 100h, on peut calibrer ce suivi dans Gazebo (en simulant un couloir texturé) puis tester sur le LIMO réel sur un parcours de type corridor ou circuit délimité par du ruban adhésif contrasté. La DAVIS pourrait être orientée vers le sol en angle pour mieux voir la trajectoire.

  Intérêt : Ce projet met l'accent sur la **vitesse** : on chercherait à pousser le LIMO à sa vitesse max (~2 m/s) et voir qu'il tient la route grâce à la latence réduite du capteur. On pourrait comparer avec une caméra classique (montrer qu'à 2 m/s l'autre caméra floue et perd la ligne, alors que DVS suit toujours).

  Livrables : Un contrôleur ROS2 de suivi de corridor par flux optique, avec logs du flux calculé en temps réel; des tests comparatifs vitesse max atteignable avec vs sans DVS; un rapport de perf (latence mesurée du loop, taux d'erreur de suivi).

- **Projet C – Odométrie visuelle-inertielle neuromorphique pour localisation en conditions dégradées** : *« Fusionner caméra événementielle DAVIS346 et IMU pour estimer la pose du LIMO en temps réel, même dans l'obscurité ou avec des motions brusques, et l'utiliser pour la navigation. »*

  Description : Ce projet vise à doter le LIMO d'une **odométrie robuste** en utilisant la DAVIS346 comme capteur principal de localisation. On peut partir d'un package open-source existant, par ex. **ESVIO** (stéréo VO inertielle) en l'adaptant en mono + IMU, ou **Ultimate SLAM** d'UZH. Si le temps est court, même une simple odométrie rotationnelle (estimer le taux de rotation par événements) couplée aux roues pour translation pourrait faire l'affaire. L'idée est de permettre au LIMO de se localiser là où ses capteurs habituels échouent : par ex. à l'extérieur de nuit (caméra RGB inutilisable) ou dans un environnement sans features pour le LiDAR. La DAVIS, grâce aux étoiles ou à un petit éclairage, pourrait quand même fournir des micro-variations utilisables. On évaluerait cette odométrie en la comparant à l'odométrie roue (encodage) et à une vérité terrain (par ex. motion capture, ou trajectoire connue).

  Réalisation : En simulation, on peut utiliser ESIM pour générer des événements d'un trajet du LIMO et tester l'algorithme. Sur robot, on fait rouler le LIMO sur un parcours carré en introduisant volontairement du dérapage ou en éteignant la lumière, et on voit si l'odométrie événementielle tient mieux que l'odométrie classique.

  Livrables : Un nœud d'estimation de pose (publiant tf ou Odometry) combinant événements+IMU; des courbes d'erreur de trajectoire avec/sans vision événementielle; une démonstration où le LIMO revient plus précisément à son point de départ grâce au DVS (par ex. faire une boucle et aligner à la fin).

Chacun de ces projets est conçu pour être **réalisable en ~100h** car il se concentre sur une seule fonctionnalité clé. Bien sûr, le choix final dépendra des intérêts et compétences (plutôt vision avancée pour le C, plutôt contrôle embarqué pour le B, plutôt détection pour le A, etc.). Tous exploitent les forces du DVS identifiées dans l'état de l'art : réactivité, robustesse aux conditions extrêmes, et complémentarité avec les capteurs existants.

2. Méthodologie de recherche rigoureuse et structurée

Pour mener à bien un tel projet dans un contexte de formation ingénieur orientée recherche, il est crucial d'adopter une méthodologie rigoureuse. Voici les points clefs d'une telle méthodologie, adaptés à notre problématique :

- **Définition précise des objectifs et hypothèses** : Dès le départ, formuler la question de recherche ou l'objectif technique. Par ex. *« Peut-on réduire le temps de réaction d'un robot mobile à moins de 50 ms pour l'évitement d'obstacle en utilisant une caméra à événements ? »*. Énoncer les hypothèses (par ex. *« La caméra à événements permettra de détecter un obstacle ~30 ms plus tôt qu'une caméra standard »*). Ceci guide la suite et donne des critères de succès.

- **État de l'art ciblé** : Comme réalisé dans ce rapport, collecter les travaux existants pertinents. Identifier les méthodes éprouvées (à réutiliser) et les lacunes (où on va apporter quelque chose). Cette étape évite de "réinventer la roue" et assure de s'appuyer sur des bases solides. Dans notre cas, si on choisit le projet A, on s'inspire fortement de Falanga 2020 et des approches d'expansion du flux, en adaptant au robot terrestre.

- **Planification par étapes avec jalons mesurables** : Diviser le travail en *phases* (voir section plan de travail ci-après). Chaque phase doit avoir un livrable concret : par ex., *Phase 1: Simulation simple d'un obstacle qui bouge + code de détection d'événement (mesurer latence).* Phase 2: Intégration sur robot (mesurer latence réelle). *Cette granularité aide à suivre l'avancement et à détecter tôt les problèmes. Prévoir aussi des* étapes de validation* à chaque phase (tests unitaires, vérification sur cas simples).

- **Approche expérimentale et itérative** : Adopter une démarche scientifique expérimentale : pour valider une idée, réaliser un petit *prototype/test* isolé. Ex: avant de coder tout l'algorithme d'évitement, faire un script qui lit un bag d'événements enregistrés d'une personne passant devant la caméra, et vérifier qu'on peut détecter le pic d'événements correspondant. Cela sert de preuve de concept. En cas de résultat négatif, on ajuste l'approche (itération). Toujours comparer les résultats à des *baselines* : ex: comparer la détection événementielle à une détection via caméra ordinaire sur la même séquence, pour quantifier le gain.

- **Documentation continue** : Tenir un carnet de bord (lab notebook) où chaque essai, paramètre et résultat est noté. Prendre des screenshots, enregistrer des rosbag de tests, etc. En recherche, tracer ce qui a été tenté et pourquoi c'est crucial. Ça facilite aussi la rédaction du rapport final (on a toutes les données et raisonnements archivés).

- **Validation rigoureuse** : Ne pas se contenter d'une démo impressionnante; mesurer objectivement la performance. Par ex., calculer la *Distribution de la latence de détection* sur 50 essais d'obstacles, ou l'*erreur moyenne de localisation* sur un parcours de 20 m, etc. Utiliser des métriques standard de la

littérature (pour pouvoir comparer aux publications). Si possible, tester en conditions variées (lumière différente, obstacle différent) pour évaluer la robustesse.

- **Analyse critique des résultats** : Une fois les résultats obtenus, les analyser par rapport aux hypothèses initiales. Expliquer les écarts : *« l'objectif de 50 ms n'est pas atteint car le bruit des événements génère de fausses alertes qu'il a fallu filtrer en accumulant sur 20 ms, d'où latence de ~70 ms »*. Discuter ce qui pourrait améliorer (même si hors du scope immédiat). C'est cette réflexion critique qui correspond à l'esprit "recherche" dans une formation ingénieur, plus que juste faire marcher un système.

- **Communication et reproduction** : Prévoir de communiquer les résultats de façon claire, que ce soit dans un rapport écrit structuré (respectant les standards académiques, avec bibliographie comme ici) et/ou une présentation orale avec démonstration. Rendre le code écrit propre et modulaire, idéalement open-source sur un dépôt, avec des instructions pour reproduire les expériences (par ex., *comment lancer la simulation Gazebo, quelles rosbag lire pour voir l'évitement, etc.*). La reproductibilité est un pilier de la recherche : quelqu'un d'autre devrait pouvoir reproduire nos tests à partir de nos indications.

En synthèse, la méthodologie doit mêler **rigueur scientifique** (expériences contrôlées, mesures, références) et **ingénierie pratique** (gestion du temps, itérations agiles, documentation logicielle). Un schéma possible est d'adopter un cycle inspiré du *V-model* pour la partie logicielle (spécification -> implémentation -> tests unitaires -> intégration -> validation finale) tout en gardant la flexibilité de la recherche (on peut revisiter la spécification si une idée ne fonctionne pas, d'où l'importance d'expérimenter tôt à petite échelle).

3. Plan de travail progressif

Compte tenu de l'ampleur limitée (100h) du projet, un plan de travail séquentiel avec validations à chaque étape est essentiel. Voici un plan type applicable à l'un ou l'autre projet parmi ceux suggérés, découpé en 4 phases majeures :

- **Phase 1 : Mise en place de la simulation et tests préliminaires (20h)**
  Objectif : Avoir un environnement Gazebo fonctionnel avec le robot LIMO + caméra événementielle simulée, et réaliser des premiers tests simples de perception.
  Tâches :
- Configurer le modèle URDF du LIMO dans Gazebo. Vérifier que le robot se téléopère correctement dans l'environnement virtuel.
- Intégrer le plugin DVS dans le URDF (ou caméra + script conversion). S'assurer que le topic `/limo/daviscam/events` émet des événements (vérifier par exemple qu'en bougeant le robot ou un objet devant la caméra, on reçoit des messages).
- Créer un petit script d'abonnement aux événements pour visualiser leur distribution (par ex. accumuler 5 ms d'événements et afficher l'image). Valider qualitativement que ça correspond aux attentes (un objet qui bouge génère bien une trace d'événements).

- Implémenter une version simplifiée de l'algorithme ciblé. Par exemple, pour le projet A (évitement) : calculer le nombre d'événements dans la zone centrale de l'image à chaque instant et si > seuil, déclencher un drapeau "obstacle". Faire bouger un cube dans Gazebo devant le robot et voir si le

drapeau passe à 1 au bon moment. Ajuster éventuellement les paramètres (seuil, fenêtre temporelle). *Livrables phase 1:* Monde Gazebo configuré, launch file ROS2 pour lancer robot + plugin, premiers graphes ou images montrant des événements simulés, validation que l'instrumentation de base fonctionne.

- **Phase 2 : Développement de l'algorithme de perception (40h)**
  Objectif : Développer et tester l'algorithme principal de traitement des événements en simulation, jusqu'à obtenir des performances satisfaisantes.
  Tâches :

- Implémenter pas à pas l'algorithme complet. Par exemple, pour l'évitement dynamique : calcul du flux optique à partir des événements (peut-être en utilisant une bibliothèque si disponible), estimation de la direction d'approche de l'obstacle, logique de décision (arrêt ou déviation). Intégrer l'IMU simulée si nécessaire (ex: compensation de mouvement du robot).
- Tester divers scénarios simulés : différents vitesses d'obstacle, différentes tailles. Utiliser rosbag pour enregistrer les événements et rejouer si utile pour peaufiner hors-ligne.
- Affiner les paramètres de l'algo (seuils, gains de filtre, etc.) en se basant sur ces tests. Potentiellement, introduire du bruit et des conditions extrêmes dans la sim (changer l'éclairage Gazebo, ajouter plusieurs obstacles) pour tester la robustesse.

- Évaluer quantitativement en simulation : par ex., mesurer le délai entre le moment où l'obstacle bouge et la détection par l'algo, sur 10 runs, pour estimer la latence moyenne. Ou mesurer le taux de collisions évitées vs collisions sur N essais aléatoires. Si les résultats sont en deçà des attentes, retour en arrière pour améliorer (itération sur l'algorithme). *Livrables phase 2:* Code ROS2 du nœud d'algorithme complet, avec éventuellement un README; jeux de données de simulation et courbes de performance (graphes de latence, etc.); une courte vidéo possible de la simulation montrant le robot réagir correctement.

- **Phase 3 : Intégration sur la plateforme réelle (30h)**
  Objectif : Faire fonctionner la caméra DAVIS346 sur le vrai LIMO et y déployer l'algorithme développé, effectuer les premiers tests réels.
  Tâches :

- Installer le driver ROS2 sur le PC du LIMO, brancher la DAVIS346, vérifier qu'on reçoit bien des événements du monde réel. Réaliser un *calibrage* de base : vérifier la bonne orientation de la caméra, mesurer éventuellement le biais temporel entre IMU et événements, ajuster le *bias* du DVS (paramètres matériels de seuil, if accessible via *DV software*, pour réduire le bruit ou augmenter sensibilité en fonction des conditions).
- Adapter le code algorithme si l'interface de messages diffère un peu en vrai (peut-être changer le topic name, ou gérer le taux plus élevé qu'en sim).
- Rejouer sur le robot des scénarios similaires à la simulation : par exemple, déplacer un objet devant le robot à différentes vitesses et voir si le robot freine. Ici il faudra une approche sécurité : garder la télécommande en main pour arrêter le robot si l'algo ne réagit pas. Commencer prudemment (vitesse lente) puis augmenter.
- Collecter des données pendant les tests réels : enregistrer rosbag des événements + IMU + actions du robot. Filmer aussi pour avoir une référence visuelle.

- Comparer les résultats réels aux attentes : latence mesurée (on peut synchroniser les logs ou vidéo pour voir le délai entre mouvement obstacle et réaction), fiabilité (le robot a-t-il eu de faux positifs, ou manqué un obstacle?). Identifier les écarts : par ex. plus de bruit en réel que sim => ajouter un filtre de clustering pour éliminer le bruit.

- Améliorer l'algorithme au besoin pour le réel, en itérant tests terrain / ajustements. Par exemple, on peut découvrir qu'en extérieur la fréquence d'événements due au scintillement du soleil dans les arbres déclenche de fausses alertes – il faudra alors affiner la détection (ex: exiger une cohérence spatiale des événements pour déclarer obstacle). *Livrables phase 3:* Système intégré sur robot; notes de calibration; éventuellement un petit document de *User Guide* si on veut que d'autres refassent tourner (comment lancer le driver, puis le node, etc.); log des tests; et idéalement, vidéos démonstratives.

- **Phase 4 : Validation expérimentale et documentation finale (10h)**
  Objectif : Valider officiellement les performances du système final et compiler les résultats dans un rapport/présentation.
  Tâches :

- Concevoir des *scénarios de test final* représentatifs. Ex: pour l'évitement, scénarios : un humain traverse à 1 m devant le robot (devrait freiner), un objet roule derrière le robot (ne devrait pas freiner), deux robots se croisent (pas collision), etc. Réaliser ces tests en conditions quasi-réelles (dans un couloir, ou une pièce encombrée).
- Mesurer les indicateurs de performance clés sur ces tests. P.ex. taux de collision évitée = X%, latence moyenne = Y ms, etc. Documenter aussi les conditions (lumière ~ 300 lux, sol en carrelage blanc, etc.) pour contextualiser.
- Comparer si possible avec un benchmark : par exemple, faire tourner une caméra RGB avec un algorithme OpenCV de détection de mouvement sur la même scène et voir la différence de temps de réaction. Ou comparer l'odométrie événementielle vs odométrie roue.
- Finaliser la **documentation** : rédiger le rapport technique final, incluant les résultats, le code commenté, la méthodologie. Inclure les références aux travaux existants pour bien situer l'apport (par ex. "notre système reproduit l'expérience de Falanga 2020 mais sur robot terrestre, avec une latence plus faible de 5 ms grâce à..."). Ajouter des tableaux comparatifs, des figures (photos du robot, schéma de l'algorithme, courbe de résultats) pour appuyer les propos.
- Si requis, préparer une soutenance : slides illustrant le problème, l'approche, une démo en live ou vidéo, et discussion des perspectives.

Livrables phase 4: Rapport final structuré (ressemblant en partie à ce document, adapté aux résultats obtenus) avec bibliographie; code source final organisé sur un dépôt; jeux de données enregistrés éventuellement fournis en annexe; et tout support de présentation.

Ce plan de travail est naturellement à adapter en fonction des imprévus (par exemple, si la simulation prend plus de temps, il faudra empiéter un peu sur la phase d'intégration réelle – toujours garder une marge de manœuvre). L'important est d'avoir toujours une **version fonctionnelle minimale** à chaque étape (principe du développement incrémental). Ainsi, même si on est à court de temps, on aura quelque chose à montrer (par exemple, même si l'odométrie complète n'est pas prête, peut-être que le suivi de rotation simple marche, etc.).

En suivant ce plan rigoureux, on maximise les chances d'aboutir dans le temps imparti tout en garantissant que le résultat est **fiable** (grâce aux validations intermédiaires) et **scientifiquement exploitable** (mesures quantitatives, comparaison à l'état de l'art). Ce plan incarne en pratique la méthodologie structurée présentée plus haut.

Conclusion et recommandations

Pour conclure, les caméras neuromorphiques telles que la DAVIS346 représentent une technologie prometteuse pour la robotique mobile, en offrant une perception ultra-rapide et robuste là où la vision classique montre ses limites (faible latence, haute dynamique, etc.). Cette étude bibliographique a passé en revue les principes de ces capteurs, les méthodes de perception associées (évitement d'obstacles, egomotion, navigation), ainsi que les algorithmes de traitement d'événements tant classiques qu'apprenant. Des exemples concrets ont illustré l'impact réel de ces caméras en robotique, et nous avons détaillé comment intégrer efficacement un tel capteur sur la plateforme Agilex LIMO sous ROS 2.

Les **projets proposés** – qu'il s'agisse d'évitement d'obstacles dynamiques, de suivi de trajectoire à haute vitesse ou d'odométrie visuelle robuste – montrent qu'il est possible en ~100 heures de développer des applications concrètes exploitant le DAVIS346 et d'en tirer des bénéfices tangibles (sécurité améliorée, vitesse accrue, meilleure localisation). La **méthodologie de recherche** recommandée insiste sur la planification, la validation expérimentale et la rigueur scientifique, afin de s'assurer que le travail mené dans le cadre d'une formation d'ingénieur reste à la fois pragmatique et aligné avec les standards de la recherche. Le **plan de travail** détaillé fournit une feuille de route pas-à-pas pour aller de la simulation à la réalisation finale, minimisant les risques et jalonnant les succès intermédiaires.

En termes de **recommandations finales** : - Il est conseillé de démarrer modestement (prouver d'abord le concept sur des cas simples) avant de complexifier l'application. La vision par événements est puissante mais diffère du paradigme classique, il faut donc du temps pour ajuster son intuition et ses outils. - **S'appuyer sur la communauté** : de nombreux codes et packages existent (voir bibliographie) – ne pas hésiter à les utiliser ou les adapter, cela économisera du temps. Par exemple, tester le driver et le plugin Gazebo sur des exemples fournis avant de l'appliquer à LIMO. - Porter une attention particulière à la **synchronisation temporelle** entre capteurs (événements et commandes/IMU), car à haute fréquence, quelques millisecondes de décalage peuvent fausser les résultats. ROS 2 assure en partie cela, mais une calibration fine peut être nécessaire. - Enfin, **garder l'utilisateur final en tête** : bien que ce soit un projet technique, imaginer l'utilisation in situ (un robot de livraison qui ne bouscule pas un enfant grâce au DVS, un fauteuil roulant électrique évitant un obstacle de dernier instant...) permet de guider les choix (par ex, privilégier la fiabilité absolue sur la vitesse extrême, etc.).

La bibliographie ci-dessous recense les références clés utilisées, offrant au lecteur souhaitant approfondir des liens vers les publications, codes source et documentations mentionnés tout au long du rapport. Ce domaine évolue rapidement, et on peut s'attendre dans un futur proche à ce que les caméras à événements deviennent des capteurs standards aux côtés des lidars et caméras RGB dans de nombreuses plateformes autonomes, tant leurs avantages ont été mis en lumière par les travaux récents [15] [16]. Le projet proposé s'inscrit dans cette tendance et permettra à l'équipe qui le réalisera de se familiariser avec une technologie de pointe tout en contribuant, à son échelle, à l'exploration de nouvelles capacités pour la robotique mobile autonome.

Bibliographie :

- Gallego, G. *et al.*, **"Event-based Vision: A Survey"**, *IJCV*, vol.128, pp.614-643, 2020 – *Panorama complet du domaine de la vision par événements, principes des capteurs et algorithmes du bas niveau (optique) au haut niveau (SLAM, reconnaissance).* 15 16

- Falanga, D. *et al.*, **"Dynamic obstacle avoidance for quadrotors with event cameras"**, *Science Robotics*, 5(40):eaaz9712, 2020 – *Démonstration phare d'évitement d'obstacles dynamiques à haute vitesse avec caméra DVS, résultats sur drone.* 25 52

- Bhattacharya, A. *et al.*, **"Monocular Event-Based Vision for Obstacle Avoidance with a Quadrotor"**, *CoRL 2024 – Approche apprentissage (Depth prediction + RL) pour éviter des obstacles statiques en monoculaire événementiel, incluant transfert simulation→réel.* 50 31

- Rebecq, H. *et al.*, **"EVO: Event-based 6-DOF tracking and mapping in real-time"**, *RAL*, 2(2):593-600, 2017 – *Odométrie/SLAM monoculaire événementielle temps réel, précurseur dans le domaine.* 73 74

- Rosinol, A. *et al.*, **"Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM"**, *RAL*, 3(2):994-1001, 2018 – *Fusion multi-capteur (DVS+frame+IMU) montrant l'avantage des événements dans des situations extrêmes.* 77 78

- Gehrig, D. *et al.*, **"EKLT: Asynchronous photometric feature tracking using events and frames"**, *IJCV*, 128(3):601-618, 2020 – *Suivi de features hybride events+images, utile pour VO à haute fréquence.* 71 72

- Sanket, N. *et al.*, **"EVDodgeNet: Deep Dynamic Obstacle Dodging with Event Cameras"**, *ICRA 2020 – Solution deep learning pour évitement d'obstacles dynamiques (quadrotor), avec entraînement simul → déploiement réel.* 55 56

- Zhu, S. *et al.*, **"Dynamic Object Avoidance using Event-Data for a Quadruped Robot"**, *IROS 2023 (Workshop) – Intégration event camera + RGB-D pour éviter des obstacles lancés sur un robot à pattes (Mini-Cheetah).* 60 61

- Delbruck, T. *et al.*, **"End-to-end neuromorphic steering for self-driving cars with spike-based cameras"**, *CVPR Workshop EventVision 2018 – Utilisation d'un réseau de neurones (ANN converti SNN) pour prédire l'angle de direction d'une voiture à partir d'une caméra DVS.* 106

- Kaiser, J. *et al.*, **"Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks"**, *SIMPAR 2016 – Plugin Gazebo DVS + SNN pour contrôle d'un véhicule simulé (cité dans le plugin DVS).* 112 113

- Mueggler, E. *et al.*, **"The Event-Camera Dataset and Simulator"**, *IJRR*, 36(2):142-149, 2017 – *Dataset pionnier pour VO/SLAM événementiel et introduction de l'Event Camera Simulator (ESIM).* 44 129

- Sites web et ressources :

- **RPG Zurich** (rpg.ifi.uzh.ch) – *Nombreuses ressources, code open-source (rpg_dvs_ros, ESIM, datasets) et tutos sur la vision événementielle.* 42 130
- **iniVation Docs** (docs.inivation.com) – *Documentation du SDK DV, DV-ROS, guides d'utilisation du DAVIS346.* 47 48
- **Metavision** (prophesee.ai) – *SDK et tutoriels sur caméras événementielles, y compris exemples ROS, datasets auto.* 118 131
- **GitHub HBP** – *Plugin Gazebo DVS (gazebo_dvs_plugin) et driver ROS2 libcaer.* 35 36

(Les liens fournis dans le texte renvoient vers les sources originales des informations citées, pour approfondir chaque point abordé.)

---

[1] [5] [30] Caméra événementielle | CNIL
https://www.cnil.fr/fr/definition/camera-evenementielle

[2] [19] [20] [69] [70] [90] [91] [92] [93] [94] [95] Human-Robot Navigation using Event-based Cameras and Reinforcement Learning
https://openaccess.thecvf.com/content/CVPR2025W/EventVision/papers/Bugueno-Cordova_Human-Robot_Navigation_using_Event-based_Cameras_and_Reinforcement_Learning_CVPRW_2025_paper.pdf

[3] [24] [60] [61] [62] [63] [64] [65] [104] Overleaf Example
https://ippc-iros23.github.io/papers/zhu.pdf

[4] [9] [10] [55] [56] [57] [58] [59] prg.cs.umd.edu
https://prg.cs.umd.edu/research/EVDodge_files/EVDODGE_ICRA2020.pdf

[6] [7] [8] [11] [12] [13] [14] [15] [16] [17] [18] [21] [22] [23] [27] [28] [29] [42] [43] [44] [66] [67] [68] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [88] [89] [96] [97] [98] [100] [101] [102] [103] [106] [107] [108] [109] [110] [111] [114] [115] [116] [129] [130] Event-based Vision: A Survey
https://rpg.ifi.uzh.ch/docs/EventVisionSurvey.pdf

[25] Dynamic obstacle avoidance for quadrotors with event cameras
https://pubmed.ncbi.nlm.nih.gov/33022598/

[26] [52] Dynamic obstacle avoidance for quadrotors with event cameras
https://www.semanticscholar.org/paper/Dynamic-obstacle-avoidance-for-quadrotors-with-Falanga-Kleber/31d21edc3ed187c7f7bfa0aa16d185a9d45399c0

[31] [32] [50] [51] [2411.03303] Monocular Event-Based Vision for Obstacle Avoidance with a Quadrotor
https://arxiv.org/abs/2411.03303

[33] [34] [37] [38] [39] [40] [112] [113] GitHub - HBPNeurorobotics/gazebo_dvs_plugin: This package provides a DVS simulation implemented as Gazebo plugin.
https://github.com/HBPNeurorobotics/gazebo_dvs_plugin

[35] [36] [41] [122] [127] gazebo_dvs_plugin/README.md at master · HBPNeurorobotics/gazebo_dvs_plugin · GitHub
https://github.com/HBPNeurorobotics/gazebo_dvs_plugin/blob/master/README.md

[45] Gazebo: Events
https://osrf-distributions.s3.amazonaws.com/gazebo/api/1.9.1/group__gazebo__event.html

[46] ros-event-camera/libcaer_driver: ROS driver for event ... - GitHub
https://github.com/ros-event-camera/libcaer_driver

[47] [48] [49] [120] Other Software — inivation 2025-08-05 documentation
https://docs.inivation.com/software/other-software.html

[53] [54] Dynamic obstacle avoidance for quadrotors with event cameras
https://www.zora.uzh.ch/entities/publication/a37f45b7-b5cc-45ea-b2c3-6c3191fb4928

[86] ESVO: Event-based Stereo Visual Odometry - GitHub
https://github.com/HKUST-Aerial-Robotics/ESVO

[87] NAIL-HNU/ESVO2 - GitHub
https://github.com/NAIL-HNU/ESVO2

99 Low-latency automotive vision with event cameras - Nature
https://www.nature.com/articles/s41586-024-07409-w

105 A Survey on Event-Based Imaging Technologies - arXiv
https://arxiv.org/html/2505.05488v1

117 ROS学习笔记之——ESVO复现及DAVIS346测试原创 - CSDN博客
https://blog.csdn.net/gwplovekimi/article/details/120326448

118 Caméra évènementielle pour la pERception d'oBjEts Rapides ... - ANR
https://anr.fr/Projet-ANR-21-CE22-0006

119 CERBERE
https://litis-cerbere.univ-rouen.fr/

121 event_camera_py 2.0.1 documentation
https://docs.ros.org/en/rolling/p/event_camera_py/

123 126 Using a URDF in Gazebo — ROS 2 Documentation
https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/Using-a-URDF-in-Gazebo.html

124 128 A Review of the ROS2 URDF Gazebo Sensor | by Bytes Robotics
https://medium.com/@bytesrobotics/a-review-of-the-ros2-urdf-gazebo-sensor-91e947c633d7

125 Tutorial 5: Simulation — 240AR060 - Introduction to ROS
https://sir.upc.edu/projects/ros2tutorials/5-simulation/index.html

131 Module caméra événementielle Prophesee GENX320 - lextronic
https://www.lextronic.fr/module-camera-evenementielle-prophesee-genx320-79169.html