Suivi d'objet en 6 degrés de liberté avec caméra événementielle

Mémoire

Etienne Dubeau

Maîtrise en génie électrique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Suivi d'objet en 6 degrés de liberté avec caméra événementielle

Mémoire

Etienne Dubeau

Sous la direction de:

Jean-François Lalonde, directeur de recherche
Raoul de Charette, codirecteur de recherche

Résumé

Actuellement, les méthodes de suivi d'objet utilisent majoritairement un capteur conventionnel doté d'une fréquence de capture limitée, par example : une caméra couleur RGB ou un capteur RGB-D qui fournit également la profondeur à chaque pixel. Ceux-ci ne sont pas idéaux lorsque l'objet se déplace à grande vitesse car des images floues sont produites. Augmenter la fréquence de capture est la solution naïve, mais cela a comme effet d'augmenter le nombre de données capturées et la complexité d'exécution des algorithmes. Ceci cause particulièrement problème dans un contexte de réalité augmentée qui utilise des systèmes embarqués ou mobiles qui ont des capacités de calcul limitées. D'un autre côté, la popularité des capteurs événementiels, qui mesurent les variations d'intensité dans la scène, est en augmentation dû à leur faible puissance d'utilisation, leur faible latence, leur capacité d'acquisition à grande vitesse et le fait qu'ils minimisent le nombre de données capturées.

Ce mémoire présente donc une méthode d'apprentissage profond de suivi d'objet à grande vitesse en six degrés de liberté en combinant deux capteurs distincts, soit un capteur RGB-D et une caméra événementielle. Pour permettre l'utilisation des capteurs conjointement, une méthode de calibration temporelle et spatiale est détaillée afin de mettre en registre les images capturées par les deux caméras. Par la suite, une méthode d'apprentissage profond de suivi d'objet est présentée. Celle-ci utilise uniquement des données synthétiques à l'entrainement et utilise les deux capteurs pour améliorer les performances de suivi d'objet en 6DOF, surtout dans les scénarios à grande vitesse. Pour terminer, un jeu de données RGB-D-E est capturé et annoté à la position réelle pour chaque trame. Ce jeu de données est accessible publiquement et peut être utilisé pour quantifier les performances de méthodes futures.

Table des matières

Liste des tableaux

Liste des figures

Avant-propos

Le chapitre 3 est l'article suivant :

Etienne Dubeau, Mathieu Garon, Benoit Debaque, Raoul de Charette et Jean-François Lalonde. RGB-D-E : Event camera calibration for fast 6-dof object tracking. *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2020

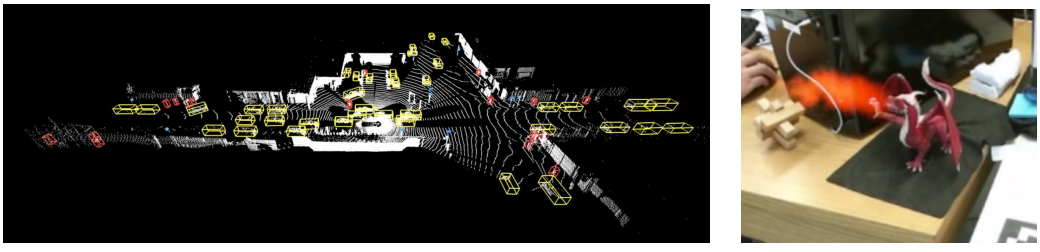
L'étudiant Etienne Dubeau est l'auteur de ce mémoire et contributeur principal de l'article. Les co-auteurs de l'articles sont Mathieu Garon étudiant à l'Université Laval, Benoit Debaque collaborateur chez Thales Digital Solutions, Raoul de Charette collaborateur chez Inria et Jean-François Lalonde professeur au département de génie électrique et de génie informatique à l'Université Laval. Jean-François Lalonde est superviseur de l'étudiant et Raoul de Charette est co-superviseur.

Introduction

Le suivi d'objet est un problème où la position d'un objet, soit la translation et la rotation, est déterminée à partir d'une source comme une caméra. Ce domaine est fondamental dans plusieurs applications variées, que ce soit pour la conduite autonome, pour la réalité augmentée ou pour la robotique mobile et pour les domaine industrielles. La figure 0.1 illustre des exemples de ces applications. Cependant, la majorité de ces problèmes sont traités à partir de caméras conventionnelles avec une capture de trames périodique. Ces caméras sont limitées dans leur fréquence de capture et les algorithmes de suivi voient ainsi limitée leur vitesse d'opération à celle de la caméra.

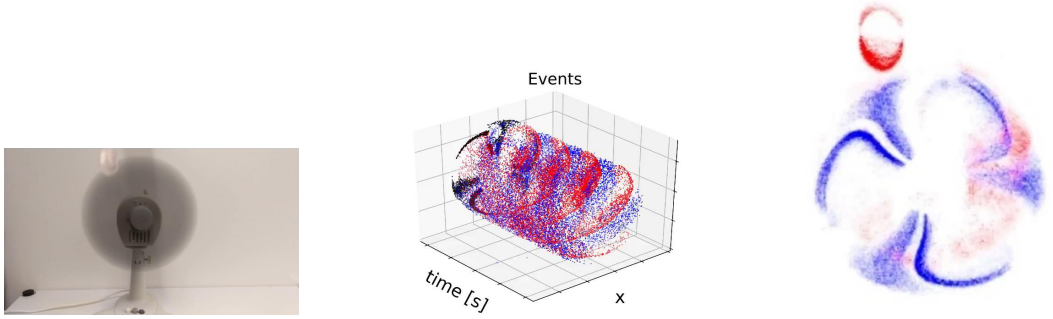
De par leur faible fréquence de capture, les caméras conventionnelles produisent des images floues si le déplacement des objets dans la scène est rapide. Celles-ci deviennent pratiquement inutilisables pour distinguer la position d'un objet, car elles superposent dans une trame une moyenne de toutes les positions durant l'exposition. Par exemple, sur la figure 0.2 (a), il est difficile de distinguer la forme des pales du ventilateur et leur déplacement car elles tournent trop rapidement par rapport au temps d'exposition de la caméra.

Une solution naïve à la fréquence de capture est d'utiliser une caméra à grande vitesse de capture (*High FPS camera*). Cependant, ces caméras sont généralement très coûteuses en énergie et sont limitées à des scènes très lumineuses, dû à leur courte durée d'exposition. De



(a) (b)

FIGURE 0.1 – Exemple d'application reliée au suivi d'objet. (a) Détection et suivi d'objets avec LIDAR dans un contexte de conduite autonome. Figure tirée de [64]. (b) Ajout d'élément virtuel, feu sur un dragon dans ce contexte, sur un objet suivi en temps réel. Figure tirée de [16].



(a) Trame conventionnelle (b) Événements selon le temps (c) Événements domaine image

FIGURE 0.2 – Exemple de capture avec caméra conventionnelle et événementielle sur une scène rapide. (a) Capture avec caméra conventionnelle. (b) Capture avec caméra événementielle affichée selon le temps et le domaine spatial (x et y). (c) Accumulation des événements durant 10 ms dans le domaine image (x et y). Tiré de [62].

FIGURE 0.3 – Reconstruction vidéo à partir de données d'une caméra événementielle. La vidéo est reconstruite jusqu'à 8000 trames par seconde dépendant du mouvement dans la scène. Tiré de [58].

plus, l'augmentation du nombre de trames capturées augmente la complexité de calcul et de mémoire utilisée par les algorithmes. Dans certains cas, un algorithme de suivi devient alors impossible à utiliser en temps réel à grande fréquence.

Une caméra événementielle ou caméra neuromorphique, est fondamentalement différente d'une caméra conventionnelle. Ce nouveau type de caméra n'enregistre pas d'images, mais plutôt des changements dans l'espace et le temps. Chacun de ses photorécepteurs est indépendant et a une résolution temporelle de l'ordre des microsecondes. Ceci permet de capturer uniquement les changements dans la scène à très grande vitesse tout en minimisant la quantité de données capturées. De plus, les caméras événementielles supportent une large plage dynamique (*High Dynamic Range "HDR"*) d'environ 120 dB versus 60 dB pour une caméra conventionnelle. La figure 0.2 compare la capture entre une caméra événementielle et une caméra conventionnelle. Sur la même figure, les événements sont représentés en 3D spatialement et temporellement ainsi qu'en 2D en accumulant les événements dans le temps.

Dû aux avantages des caméras événementielles, il est possible de les utiliser pour de la reconstruction d'images HDR (figure 0.4) ou de la reconstruction de vidéo à grande vitesse de capture (figure 0.3).

(a) Capture LDR (b) Reconstruction HDR

FIGURE 0.4 – Reconstruction HDR de trame à partir d'une capture d'événements. (a) Image capturée avec caméra conventionnelle LDR. (b) Image HDR reconstruite à partir d'une capture événementielle. Malgré que beaucoup d'artéfacts visuels sont présents dans la reconstruction, il est possible de distinguer des sections sous-exposées ou sur-exposées de l'image LDR. Tiré de [58].

Toutefois, une des difficultés de l'utilisation des caméras événementielles est que les algorithmes de vision pour caméra conventionnelle ne s'appliquent pas directement à ce nouveau type de capteur. De plus, dans une situation de suivi d'objet, la majorité des algorithmes sont développés avec des capteurs conventionnels parfois combinés avec de la profondeur (RGB-D). Cependant, ces techniques ne fonctionnent pas à grande vitesse tel que mentionné ci-haut. Il serait donc pertinent de pouvoir combiner les avantages de résolution temporelle d'une caméra événementielle aux méthodes de suivi actuelles.

Dans ce contexte, ce mémoire présente une méthode de calibration de caméra événementielle et RGB-D pour permettre le suivi d'objet connu avec déplacement à très grande vitesse en combinant ces modalités. Pour atteindre ce but, nous proposons les quatre contributions suivantes :

1. La première contribution est la calibration des deux caméras, la caméra RGB-D *Kinect Azure* et la caméra événementielle *DAVIS346*. Un support physique est créé sur mesure pour unir la caméra événementielle et celle RGB-D. De plus, une approche de calibration spatiale et temporelle est détaillée pour mettre en registre les images capturées par les deux caméras.
2. La seconde contribution est la capture d'un jeu de données RGB-D-E rendu public pour favoriser la recherche future. Celui-ci est capturé sur un objet avec modèle 3D connu sur 10 séquences avec déplacement à grande vitesse. Le jeu de données est annoté avec la position réelle pour chaque trame et contient un total de 2472 trames.
3. La troisième contribution est une méthode de suivi d'objet connu en utilisant les modalités des caméras événementielles et RGB-D. L'approche utilise des réseaux de neurones pour prédire le déplacement d'un objet en six degrée de liberté (6DOF) tout en utilisant les avantages des différentes modalités.
4. La dernière contribution est l'amélioration des performances de suivi d'objet en 6DOF dans la littérature. La fusion des modalités permet une amélioration dans des scénarios à grande vitesse où les caméras conventionnelles sont très affectées par le flou de mouvement capturé.

Le chapitre 1 présente la littérature sur l'utilisation des caméras événementielles et du suivi d'objet. Le chapitre 2 explique les caméras utilisées pour les travaux et détaille la calibration spatiale et temporelle. Le chapitre 3 est l'article publié à la conférence ISMAR 2020 dans lequel le suivi d'objet en RGB-D-E et la capture du jeu de données sont détaillés. Pour finir, le chapitre 4 discute d'approches pouvant être explorées afin d'améliorer les performances de la méthode du chapitre 3.

Chapitre 1

Revue de littérature

Ce chapitre présente les travaux antérieurs reliés aux contributions apportées dans ce mémoire. La première section se concentre sur les caméras événementielles en abordant leurs avantages, l'accès aux données et leurs applications. Dans la seconde section, le suivi d'objet avec caméra conventionnelle est présenté.

1.1 Caméra événementielle

Cette section présente les caméras événementielles en quatre parties différentes. Pour commencer, les attributs et le fonctionnement de ces caméras sont expliqués. Par la suite, on aborde l'accès au jeu de données et la génération synthétique des données. On termine avec la représentation des données dans un réseau de neurones et les autres applications de la littérature.

1.1.1 Caractéristiques des caméras événementielles

Les caméras événementielles [38] fonctionnent différemment des caméras conventionnelles. Chacun de leur pixel est indépendant et réagit continuellement au changement d'intensité relative capturée en émettant des événements quand le changement excède une certaine borne.

Ces événements sont obtenus comme un flux continu en 4D adressé spatialement (X et Y), temporellement et ils indiquent un changement d'intensité (positif ou négatif). La nature de ces données rend l'information capturée très compacte en minimisant la capture d'information redondante. Leur avantage est un temps de réaction <1 milliseconde avec une fréquence de capture d'environ un million de fois plus rapide qu'une caméra conventionnelle. Ceci les rend très propice à une utilisation avec scène dynamique où une caméra conventionnelle serait trop affectée par le flou de mouvement. De plus, leur capture à haute plage dynamique ("High-dynamic-range") d'environ 120 dB permet de capturer des scènes de luminosité faible ou élevé. Le dernier avantage est que ces caméras requièrent très peu d'énergie, inférieur à 1W en

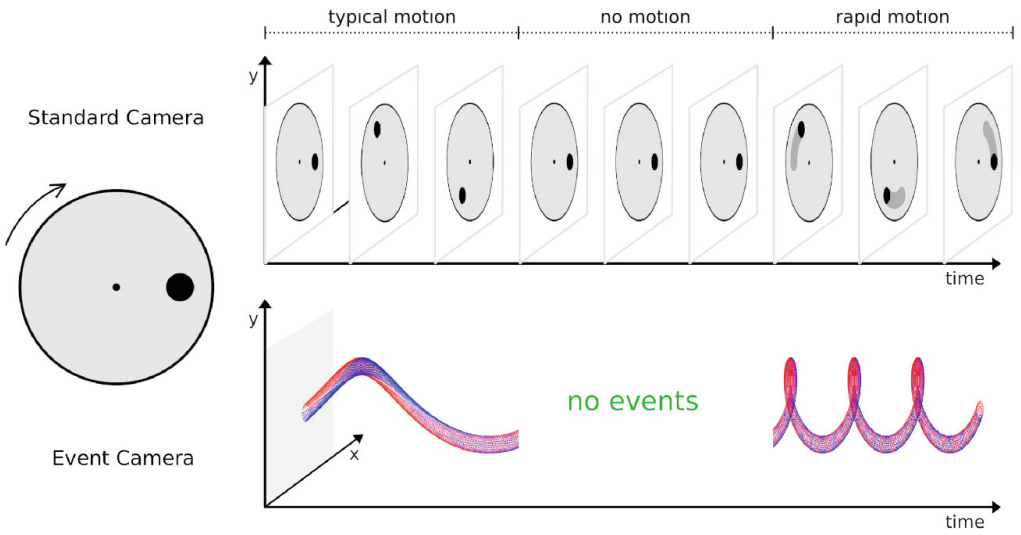


FIGURE 1.1 – Visualisation de la capture d'une caméra conventionnelle (haut) comparée à une caméra événementielle (bas). Les deux caméras capturent le même déplacement d'un cercle qui se déplace à plusieurs vitesses soit, vitesse normale ("typical motion"), vitesse nulle ("no motion") et vitesse rapide ("rapid motion"). Avec la caméra événementielle, on évite la capture d'information redondante. La polarité des événements est représentée en rouge et bleu pour indiquer une polarité positive et négative respectivement. Tiré de [31].

général, et sont donc très propices dans un contexte de robotique ou électronique portable [25] ("Wearable"). La figure 1.1 illustre la différence entre la capture d'une caméra événementielle et d'une caméra conventionnelle.

1.1.2 Jeu de données événementiel et génération synthétique

Une grande problématique de travailler avec des données événementielles est l'accès aux jeux de données. Malgré que plusieurs jeux de données sont disponibles, ceux-ci se limitent surtout à des tâches de localisation et d'odométrie [9; 35; 46] ou même le suivi d'objet en 2D [23]. Cependant, nous sommes très loin du nombre de jeux de données accessible avec caméras conventionnelles. De plus, les jeux de données pour suivi d'objet en 6 dégrées de liberté sont inexistants.

Cependant, plusieurs simulateurs de données sont disponibles. À la base, un simulateur prend la différence d'intensité entre deux trames consécutives pour créer des données similaires à celle d'une caméra événementielle. Ces trames peuvent être capturées par une caméra conventionnelle ou générées avec un engin de rendu. [28] contribue à une des premières techniques de simulation, mais comme reconnu par les auteurs, cette technique ne donne pas de résultats convaincants, car elle ne prend pas avantage des propriétés asynchrones de la caméra. Les événements générés sont limités à la fréquence fixe des rendus et donc, synchrone. [35]

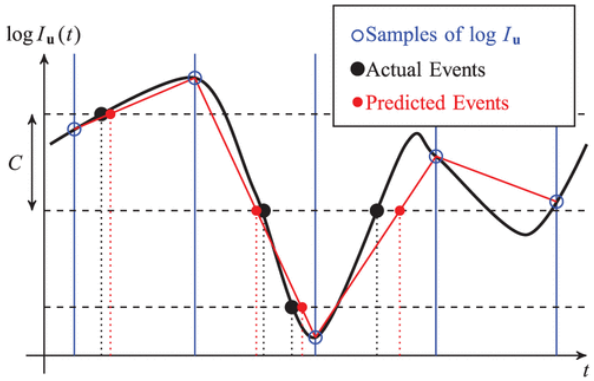


FIGURE 1.2 – Prédiction des événements asynchrones avec taux d'échantillonnage fixe. Les événements prédits ("Predicted Event") sont obtenus selon le seuil C de la caméra et l'intensité de l'événement précédent. Pour connaître l'intensité entre deux trames, une interpolation linéaire est appliquée. Tiré de [46].

et [46] proposent aussi une méthode similaire, mais produisent des résultats plus réalistes en augmentant la fréquence de génération des trames. Cette méthode peut fonctionner dans certaines scènes, mais dépend de la trajectoire de la scène car la vitesse des déplacements est saturée par la fréquence de génération des trames. Pour sa part, [46] introduit une technique d'interpolation basée sur le changement d'intensité lumineuse entre deux trames pour simuler le côté asynchrone de la caméra tout en gardant fixe la fréquence des rendus. La figure 1.2 illustre leur technique d'interpolation.

[54] a bonifié l'approche de [46] en générant les trames de manière adaptative au lieu d'utiliser un échantillonnage fixe. Deux techniques adaptatives sont explorées avec des résultats similaire. Soit se basant sur l'intensité lumineuse maximale ou sur la valeur maximale du flux optique, ils déterminent le temps de génération de la prochaine trame. Ceci augmente la précision des données générées en plus d'augmenter l'efficacité du simulateur. Cette solution est celle utilisée pour notre génération de données synthétiques.

1.1.3 Représentation des données pour traitement avec réseau de neurone

La représentation événementielle n'est pas intuitive avec une approche de réseau de neurones conventionnelle (CNN) car les données doivent être converties sur un système à coordonné fixe temporelle et spatiale. Ceci n'est pas évident car l'information obtenue par les caméras événementielles est asynchrone et représentée en 4D. Une approche par *Event Spiking Network* [40] est naturel pour le traitement de ces données, mais est présentement instable et difficile à entrainer [33]. Ces réseaux prennent en entrées des valeurs en temps et espace comme les données générées par les caméra événementielle. Chaque événement peut être traité individuellement et activer les "neurones" du réseau pour en modifier la prédiction. Plusieurs autres approches sont utilisées pour convertir l'information en tenseur 2D en discartant l'information temporelle

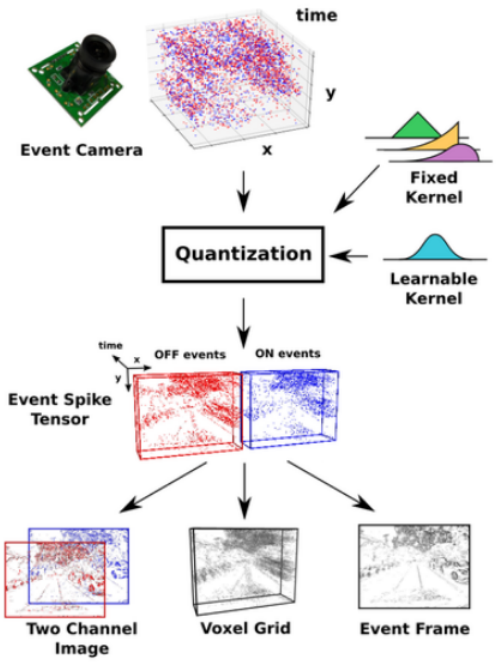


FIGURE 1.3 – Approche générale pour la conversion des données événementielles en tensor. Une convolution est appliquée sur les données capturées en accumulant sur le temps deux tensors 3D. Un tensor d'événements positifs et un tensor d'événements négatifs ("Event Spike Tensor"). Par la suite, cette représentation peut être simplifiée en une image 2 canaux ("Two Channel Image"), dans une grille de voxel ("Voxel Grid") ou sur une image 1 canal ("Event Frame"). Tiré de [18].

et la polarité [55] ou en tenseur 3D en discartant une seule de ces informations [43; 74].

L'approche de [18] permet de conserver l'information temporelle en accumulant l'information spatiale dans plusieurs fenêtres de temps discrètes. Leur technique commence par accumuler l'information dans un "Event Spike Tensor", soit deux tenseurs 3D (X, Y et temps), un par polarité. Le temps est discrétisé en plusieurs périodes et une convolution 1D, fixe ou entrainable, est appliquée sur l'axe du temps. Ceci permet de partager l'information temporelle avec les périodes adjacentes. Cette représentation peut être utilisée directement avec des réseaux de neurones conventionnels, mais il est aussi possible de la simplifier. Ainsi un "Event Spike Tensor" permet de compresser l'information en concaténant l'axe du temps ou la polarité. Cette approche est celle retenue dans le cadre de nos travaux. La figure 1.3 illustre l'approche utilisée par [18].

1.1.4 Autres applications

Plusieurs scénarios sont favorables aux caméras événementielles. On retrouve les caméras événementielles dans diverses applications. Par exemple, elles conviennent naturellement à des tâches de surveillance [39] ou d'estimation du flux optique [3; 73], car l'information redon-

dante est minimisée. De plus, leur plage HDR (High Dynamic Range) et leur faible latence les rendent très propices à la reconstruction de vidéo à grande fréquence (High FPS) [58; 30], le suivi d'objet en 2D [19; 45] ou la reconnaissance d'objets et de gestes [2; 34; 52].

Un domaine très actif dans la littérature est l'odométrie et le SLAM (Simultaneous Localisation and Mapping). La nature de ces caméras les rend incompatibles avec les algorithmes traditionnels, mais beaucoup sont adaptés pour supporter ce type de caméra. Par exemple, [69] utilise un filtre à particules pour réduire l'erreur de reprojection afin de déterminer la translation de la caméra, mais est limité à des scènes à haut contraste et à des mouvements planaire. [30; 14; 60] utilisent des techniques similaires, mais ils se concentrent sur la rotation de la caméra au lieu de la translation.

L'ajout d'un capteur RGB-D à un capteur événementiel permet à [68] de développer une technique de SLAM en 6 degrés de liberté (6DOF) avec reconstruction de la profondeur de la scène. Par contre, l'ajout d'un capteur à fréquence fixe entraîne les compromis associés au capteur conventionnel.

Basé sur les travaux de [30], [31] utilise une combinaison de filtres probabilistes pour déterminer la position de la caméra en 6DOF, la profondeur de la scène et la reconstruction d'image d'intensité à partir des événements uniquement. Malgré que la méthode fonctionne seulement avec une caméra événementielle, celle-ci est peu robuste aux changements d'intensité et aux mouvements rapides en plus d'exiger un GPU pour un traitement en temps réel. [56] améliore la technique précédente en démontrant que la reconstruction d'image d'intensité n'est pas nécessaire et ceci permet une exécution en temps réel sur CPU.

1.2 Suivi d'objet avec caméra conventionnelle

Cette section aborde les techniques employées dans la littérature pour le suivi d'objet. La première sous-section présente un survol général et la seconde explique en profondeur la méthode *Deeptrack* [16; 17] car celle-ci est la base de nos travaux.

1.2.1 Méthodes de suivi

Plusieurs travaux sont réalisés utilisant des réseaux de neurones pour du suivi d'objet 2D. Par exemple, [15; 4] utilisent des méthodes temporelles et [59] utilise une méthode de détection pour déterminer la position 2D d'un objet. Ces méthodes ont l'avantage d'être en temps-réel, et ce, même sur appareil mobile. Cependant, la position est limitée en 2D et il est donc impossible de déterminer la pose 6DOF d'un objet.

Des méthodes par ICP comme [48; 1] sont utilisés sur la profondeur pour obtenir la pose 6DOF. [8; 7] ajoutent un filtre à particules sur ces méthodes pour augmenter la précision et la stabilité avec des petits objets. [65; 27] utilisent une méthode d'apprentissage avec un arbre de

décision (*decision tree*) pour régresser un sous-ensemble de points 3D sur l'objet suivi. Pour supporter les scénarios d'occlusion, un sous-ensemble de points est sélectionné aux bordures de l'objet. Cependant, cela fonctionne uniquement quand ce sous-ensemble n'est pas affecté par l'occlusion. Les auteurs de [26] ont amélioré la méthode précédente en ajoutant un détecteur pour réinitialiser le suivi en cas de perte d'objet durant l'exécution. De plus, ils contribuent à la capture d'une unique scène d'évaluation RGB-D où la caméra est déplacée rapidement face à quatre objets statiques.

Plusieurs auteurs utilisent des réseaux de neurones pour résoudre des problèmes géométriques 3D. Par exemple, [10] infère l'homographie entre deux images et [51] identifie la pose d'une main. Dans le second cas, le réseau infère la position entre un rendu et l'image courante. Ceci est appliqué en boucle fermée et est similaire à la méthode détaillée à la section 1.2.2.

1.2.2 Deeptrack

La méthode de base utilisée pour nos travaux est *Deeptrack*. Cette méthode permet le suivi d'objet en 6DOF sur un objet avec modèle 3D connu. La première contribution [16] définit la méthode de tracking avec réseau de neurones profond. Leur technique consiste à entrainer un réseau à prédire la différence 6DOF entre deux trames. La première trame est un rendu de l'objet à l'itération précédente. La seconde est la position actuelle observée par la caméra. Les trames sont accompagnées de la profondeur (RGB-D) et les auteurs ont capturé leur propre objet 3D pour générer leurs rendus. La figure 1.6 schématise l'architecture du réseau utilisé.

Pour les données d'entraînement, les auteurs ont opté pour un entraînement synthétique en générant des paires de trames à différents déplacements. Dans le cas des deux rendus, ceux-ci sont augmentés avec du bruit et un éclairage réaliste aléatoire. Le second rendu est superposé à un arrière-plan RGB-D sélectionné aléatoirement depuis le jeu de données SUN3D [70]. La figure 1.4 illustre les données utilisées lors de l'entrainement.

Pour comparer leurs résultats, les auteurs ont capturé plusieurs scènes avec quatre objets différents. Les déplacements sont capturés sur une table rotative avec marqueurs de calibration. Les marqueurs permettent de trouver le déplacement 6DOF appliqué à l'objet entre chaque trame.

Dans une seconde publication [17], les auteurs s'attaquent au même problème, mais se concentrent sur la capture de données. Avant leur contribution, le jeu de données de [8] était le plus utilisé pour l'évaluation du suivi d'objet en 6DOF. Celui-ci est limité à quatre courtes séquences purement synthétiques et irréalistes. De plus, ce jeu de données est limité pour comparer plusieurs méthodes, car certaines atteignent une précision près de 100%.

Pour ces raisons, les auteurs ont capturé et publié un jeu de données d'évaluation plus complet. Celui-ci est capturé sur des données réelles en RGB-D et comprend la position précise des

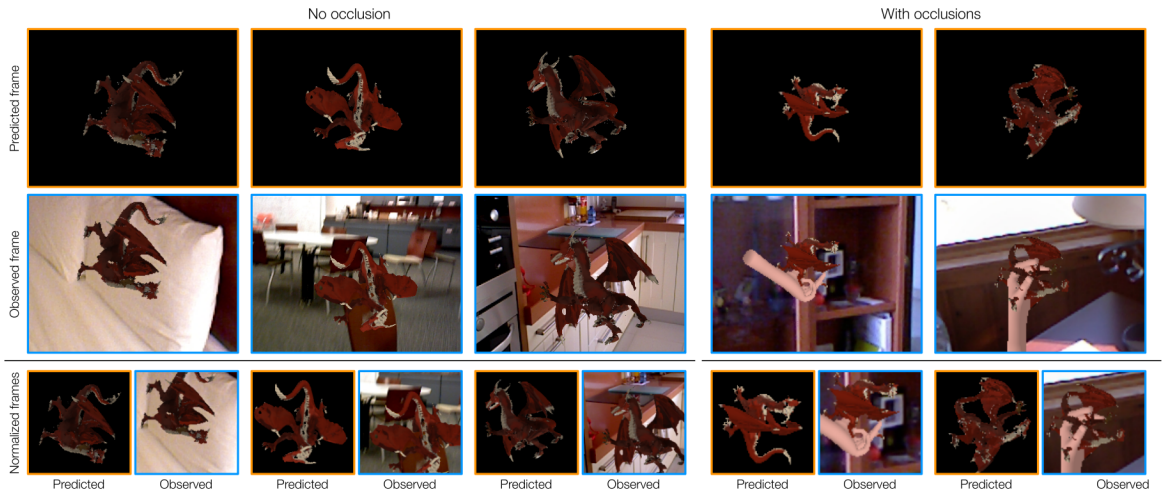


FIGURE 1.4 – Exemple de données d'entrainement synthétique. Exemple avec occlusion ("With occlusions") et sans occlusion ("No occlusion"). La première rangée illustre les trames de prédiction ("Predicted frame") et la seconde, les trames observées ("Observed frame"). Les trames observées sont composées sur un arrière-plan tirée du jeu de données SUN3D [70]. La dernière rangée illustre les trames normalisées et données au réseau. Figure tirée de [16].

objets à chaque trame. Au total, 297 séquences et 11 différents objets sont capturés sur trois scénarios pour quantifier plusieurs situations : la stabilité, l'occlusion et l'interaction.

Leur système de capture est composé d'un système de capture Vicon pour identifier la position précisément de l'objet dans la scène et d'une caméra Kinect pour la capture des données RGB-D. La figure 1.5 illustre le système de capture.

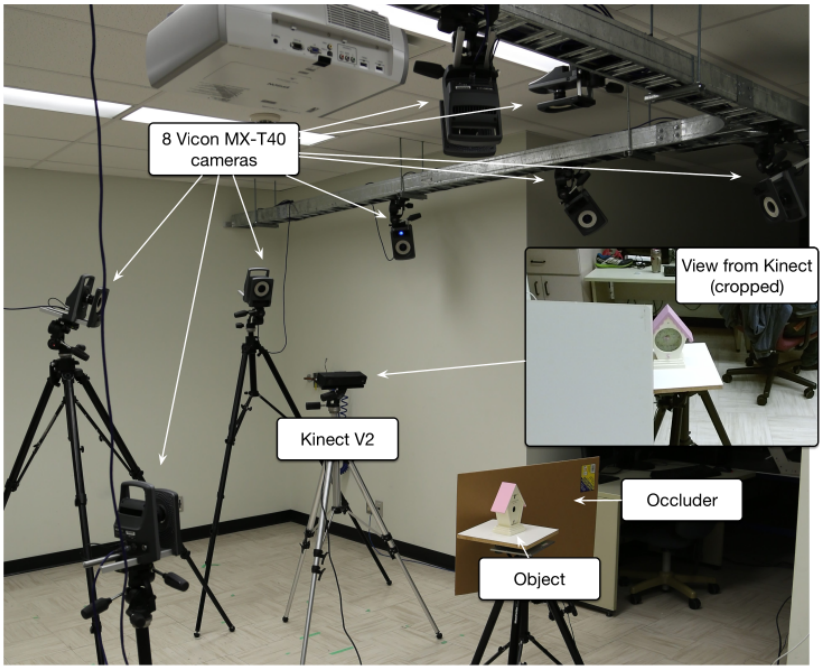


FIGURE 1.5 – Système d'acquisition des données utilisées par [17]. Aperçu du système complet avec les 8 caméras Vicon ("8 Vincon MX-T40 cameras") et une Kinect V2. On y voit l'objet capturé ("Object"), un panneau d'occlusion ("Occluder") ainsi que le point de vue capturé par la Kinect ("View from Kinect (cropped)"). Figure tirée de [17].

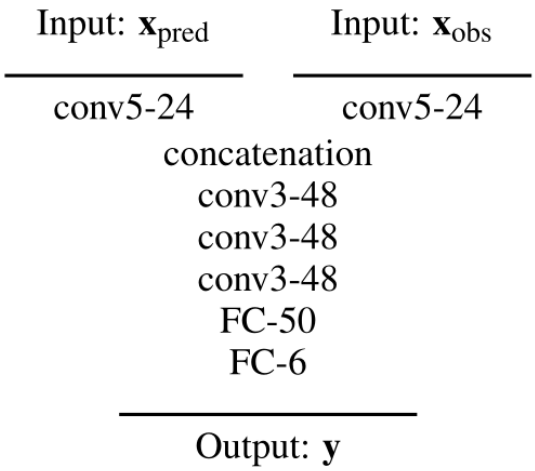


FIGURE 1.6 – Architecture du réseau de suivi d'objet de Deeptrack. La notation "conv$x-y$" indique une couche de convolution avec y filtres de dimension $x \times x$ et "FC-x" indique une couche entièrement connectée de x unités de sorties. Le réseau commence par convoluer le rendu de la position courante, $\mathbf{x}_{pred}$, et la trame observée par la caméra, $\mathbf{x}_{obs}$. Les sorties de ces convolutions sont concaténées et passées dans trois couches de convolution successive. Finalement, deux couches entièrement connectées estiment la transformation $\mathbf{y}$ entre les deux trames. Figure tirée de [16].

Chapitre 2

Calibrage des caméras

Cette section présente les caméras utilisées pour la capture des données ainsi que les méthodes de calibrage spatial et temporel entre les caméras.

2.1 Support matériel

Le système est composé de deux caméras, la caméra Azure Kinect de Microsoft pour capturer les images couleur et la profondeur ainsi qu'une caméra événementielle DAVIS346 de iniVation. Les deux caméras sont fixées sur un support spécialement réalisé pour cette application. Cela permet de conserver une position fixe entre les deux caméras. La figure 2.1 illustre les deux caméras avec le support.

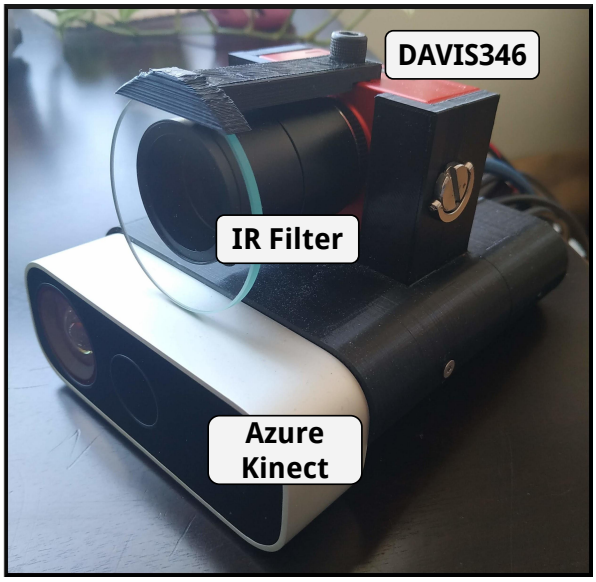


FIGURE 2.1 – Système de capture RGB, profondeur et événementiel avec le support personnalisé.

2.1.1 Azure Kinect

La Azure Kinect de Microsoft est une caméra qui capture des trames RGB et la profondeur. Le capteur de profondeur est de type *Time-of-Flight* et est composé d'un capteur et d'un émetteur infrarouge. Notez que l'image infrarouge peut aussi être capturée et elle est utilisé pour les captures de calibrations des sections 2.2.1, 2.2.2 et 2.2.3. Le capteur RGB a une résolution maximale de 4096×3072 pixels avec un champ visuel de $90° \times 59°$ tandis que le capteur de profondeur supporte une résolution jusqu'à 1024×1024 pixels avec un champ visuel de $120°$. Les deux capteurs supportent une fréquence de 30 trames par seconde et sont synchronisés temporellement au point central de leur exposition.

2.1.2 DAVIS346

La DAVIS346 est une caméra événementielle développée par iniVation. Celle-ci peut capturer les événements $E = \{t, p, x, y\}$ où t est l'estampille temporelle du déclenchement de l'événement, p est la polarité, x et y représentent la position spatiale sur l'image. La caméra peut aussi capturer des trames en niveaux de gris simultanément à une fréquence maximale de 40 Hz. Les deux capteurs sont spatialement alignés et ont une résolution de 346×260 pixels. Puisque les deux capteurs sont parfaitement alignés, le capteur en niveaux de gris est utilisé pour la calibration à la section 2.2.1 et à la section 2.2.2.

Les principales caractéristiques de ce capteur événementiel lorsque seuls les évènements sont utilisés et que l'image en niveaux de gris est désactivée : une faible latence d'environ $20\mu s$, un débit minimal, une consommation d'énergie inférieure à $30mW$ et une plage dynamique de $120dB$. De plus, il supporte la synchronisation temporelle matérielle.

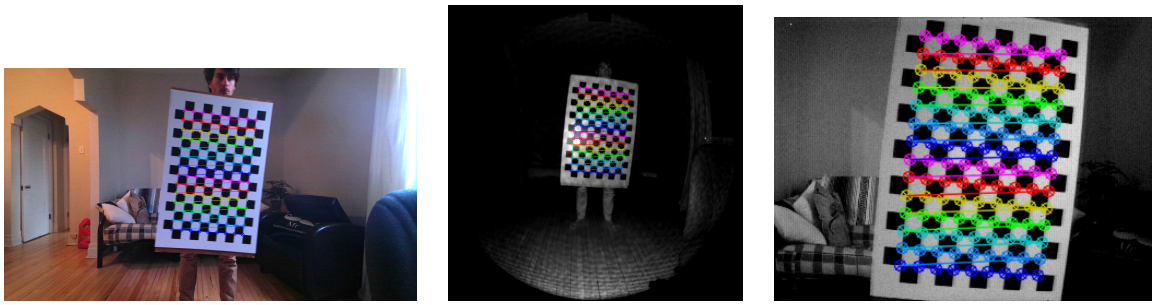
2.1.3 Filtre infrarouge

Nous avons remarqué que le nombre d'événements capturés est amplifié lors de capture avec les deux caméras simultanées, et ce même sur une scène statique. Ce bruit est dû au capteur de profondeur *Time-of-Flight* (ToF) de la caméra Kinect qui utilise un capteur actif et émet des pulsations de lumière infrarouge. Un filtre infrarouge est ajouté devant la lentille de la caméra événementielle pour éviter que celle-ci réagisse à ces pulsations. Ce filtre est montré à la figure 2.1.

2.2 Calibrage spatial

2.2.1 Calibrage intrinsèque

Les paramètres de la caméra sont définis comme un ensemble $I_c = \{C_c, D_c\}$ où C_c représente les paramètres intrinsèques de la caméra et D_c représente la distorsion de la lentille.



a) b) c)

FIGURE 2.2 – Exemple de capture du damier avec les différents capteurs. De gauche à droite, le capteur RGB, le capteur infrarouge de profondeur et le capteur en niveaux de gris de la caméra évémentielle. Les cercles de couleurs représente les coins du damier détectés. Les 3 captures sont prises en simultané et l'entièreté des trames est affichée.

$$C_c = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$D_c = \begin{bmatrix} k_1, k_2, k_3, k_4, k_5, k_6, p_1, p_2 \end{bmatrix} \tag{2.2}$$

Les paramètres intrinsèques et les paramètres de distorsion sont les suivants :

- Longueur focale de la lentille sur l'axe des x : f_x
- Longueur focale de la lentille sur l'axe des y : f_y
- Le centre optique de la lentille sur l'axe x : c_x
- Le centre optique de la lentille sur l'axe y : c_y
- Le coefficient de non-orthogonalité des axes x et y : γ
- Coefficient de distorsion radiale : k_1, k_2, k_3, k_4, k_5 et k_6
- Coefficient de distorsion tangentielle : p_1 et p_2

La valeur de γ est définie à 0 en assumant que les axes de l'image sont perpendiculaires. Pour estimer les valeurs des paramètres intrinsèques des caméras C_c et les coefficients de distorsion D_c, la méthode de Zhang [71] est utilisé avec un damier de taille connue. Le damier est composé de 9×14 carrés de 24 mm et la méthode de Harris [20] est utilisée pour obtenir les coordonnées des coins des carrés sur les images. Les captures sont prises afin de maximiser la couverture du champ de vision de la caméra en favorisant les bordures du champ de vision pour mieux estimer la distorsion.

Les paramètres sont estimés pour les trois capteurs, sois le capteur RGB (I_{RGB}) avec 199 captures, le capteur de profondeur (I_{Depth}) avec 112 captures et le capteur événementiel (I_{Event})

avec 50 captures. Des exemples de captures pour les 3 capteurs sont affichés à la figure 2.2. Les paramètres obtenus sont les suivants :

$$C_{RGB} = \begin{bmatrix} 619.53 & 0 & 641.15 \\ 0 & 619.84 & 357.30 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$D_{RGB} = \Big[-0.2123, -3.061, -5.192\text{e-}5, 2.378\text{e-}3, 2.769, -0.3400, -2.783, 2.595\Big] \tag{2.4}$$

$$C_{Depth} = \begin{bmatrix} 254.78 & 0 & 251.84 \\ 0 & 254.63 & 252.92 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$D_{Depth} = \Big[1.811, 0.8881, 6.936\text{e-}5, -9.228\text{e-}5, 0.04024, 2.1483, 1.442, 0.2277\Big] \tag{2.6}$$

$$C_{Event} = \begin{bmatrix} 380.61 & 0 & 187.31 \\ 0 & 380.723 & 131.01 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

$$D_{Event} = \Big[-11.62, 31.60, 8.396\text{e-}4, -7.040\text{e-}4, 14.45, -11.25, 27.20, 27.76\Big] \tag{2.8}$$

2.2.2 Calibrage extrinsèque

Dans sa forme générale, la transformation extrinsèque entre deux référentiels est définie comme $\mathbf{T}_{\mathrm{a}}^{\mathrm{b}}$ pour exprimer la transformation d'une coordonnée a en 3 dimensions vers un référentiel b. $\mathbf{T}_{\mathrm{a}}^{\mathrm{b}}$ est défini comme une matrice de 4×4 selon la matrice de rotation $\mathbf{R}_{\mathbf{a}}^{\mathbf{b}}$ et le vecteur de translation $\mathbf{t}_{\mathbf{a}}^{\mathbf{b}}$.

$$\mathbf{T}_{\mathrm{a}}^{\mathrm{b}} = \begin{bmatrix} \mathbf{R}_{\mathrm{a}_{3\times3}^{\mathrm{b}}} & \mathbf{t}_{\mathrm{a}_{3\times1}^{\mathrm{b}}} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \tag{2.9}$$

Des captures sont prises en exposant un damier sur les deux repères de caméra a et b simultanément en s'assurant que leur champs de vue est superposé. Le damier utilisé et l'approche de détection des coins sont les mêmes que ceux de la section 2.2.1. L'algorithme Perspective-N-Point (PnP) [11] est utilisé pour trouver les transformations extrinsèques depuis les coordonnées des coins du damier. Les positions du damier sont réparties uniformément dans la zone de capture, sois environ entre 0.7 m et 1.4 m.

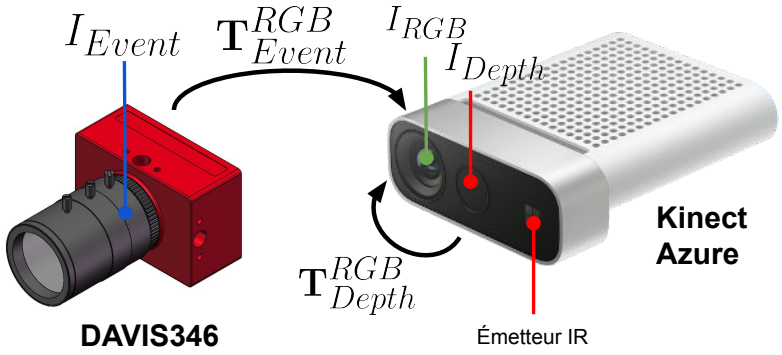


FIGURE 2.3 – Schéma des transformations entre les caméras.

Une partie des captures est utilisée pour estimer les valeurs des transformations et le reste est utilisé pour quantifier les erreurs de projection. Ici, l'erreur de projection est la distance euclidienne en pixel entre un point du référentiel a projeté vers le référentiel b et celle-ci est calculée sur tous les coins des images de test.

Dans ce contexte, deux transformations sont estimées, soit les paramètres extrinsèques entre le capteur de profondeur et le capteur RGB $\mathbf{T}^{\text{RGB}}_{\text{Depth}}$ et les paramètres extrinsèques entre le capteur événementiel et le capteur RGB $\mathbf{T}^{\text{RGB}}_{\text{Event}}$. Ces deux transformations permettent d'utiliser le référentiel du capteur RGB comme repère commun.

Pour la première transformation $\mathbf{T}^{\text{RGB}}_{\text{Depth}}$, 60 images sont capturées pour estimer les paramètres et une erreur de projection médiane de 0.8671 pixels est calculée sur 45 images. Pour la seconde transformation $\mathbf{T}^{\text{RGB}}_{\text{Event}}$, 36 images sont capturées pour estimer les paramètres et une erreur de projection médiane de 0.2548 pixels est calculée sur 30 images. La figure 2.3 résume les transformations entre les caméras et la figure 2.4 compare les erreurs de projection pour les deux transformations.

$$\mathbf{T}^{\text{RGB}}_{\text{Depth}} = \begin{bmatrix} 1 & -3.3\text{e-}3 & 8\text{e-}4 & -320.1 \\ 3.2\text{e-}3 & 1 & 8.49\text{e-}2 & -2.118 \\ -1.1\text{e-}3 & -8.49\text{e-}2 & 1 & 4.002 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

$$\mathbf{T}^{\text{RGB}}_{\text{Event}} = \begin{bmatrix} 1 & -1.6\text{e-}3 & -3.09\text{e-}2 & -30.60 \\ 1.7\text{e-}3 & 1 & 2\text{e-}3 & -46.98 \\ -3.09\text{e-}2 & -2\text{e-}3 & 1 & 3.24 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

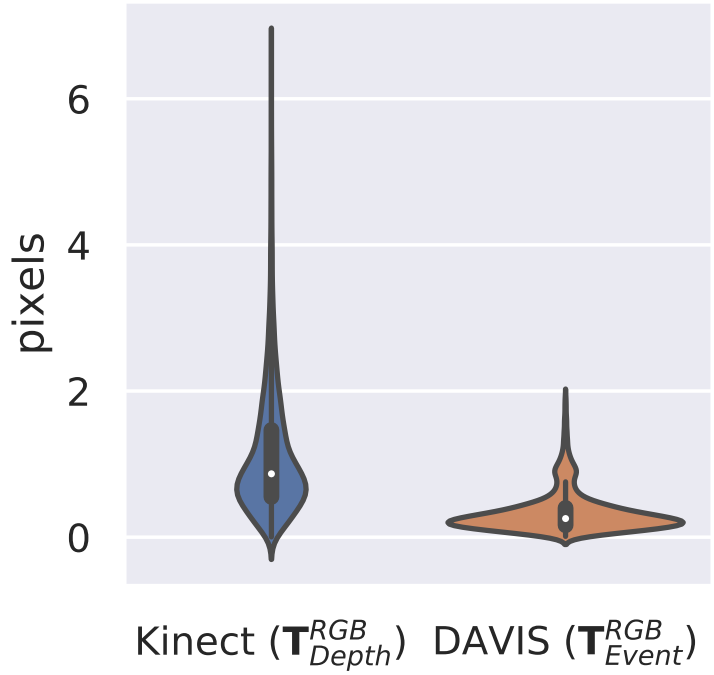


FIGURE 2.4 – Erreur euclidienne de projection de la profondeur et de la caméra événementielle dans le référentiel image.

2.2.3 Correction de la profondeur

Comme il est reporté dans [12; 17; 63; 22], les caméras de type *Time-of-Flight* ont une certaine imprécision dans leur capture de profondeur et celle-ci peut-être corrigée avec une correction polynomiale. Dans le cas de la caméra Azure Kinect, une divergence moyenne de 8.5 mm est mesurés entre la profondeur capturée d et la profondeur réelle d_e. La profondeur réelle est obtenue avec le problème Perspective-N-Point (PnP) [11] depuis les coordonnées des marqueurs d'images de damier prises avec le capteur infrarouge de la Kinect Azure. Ces coordonnées sont obtenues de manière similaire à la section 2.2.1. La profondeur capturée d est la profondeur aux marqueurs du damier calculé la Kinect Azure.

En utilisant la méthode des moindres carrés sur les paires d et d_e, la correction polynomiale d_c est calculée. Un total de 45 images avec des distances variant d'environ 0.5 à 2.5 m sont utilisées pour calculer la correction d_c. La correction linéaire suivante est obtenue :

$$d_c = 1.0112 * d - 1.7897 \tag{2.12}$$

Nous avons aussi comparé avec des polynomiaux à degré plus élevé, mais sommes restés avec un degré de premier ordre, car les résultats étaient similaires tout en évitant de surapprendre sur nos échantillons. La figure 2.5 compare la correction polynomiale avec différents dégrées testé sur 5143 marqueurs indépendants au calcul de la correction.

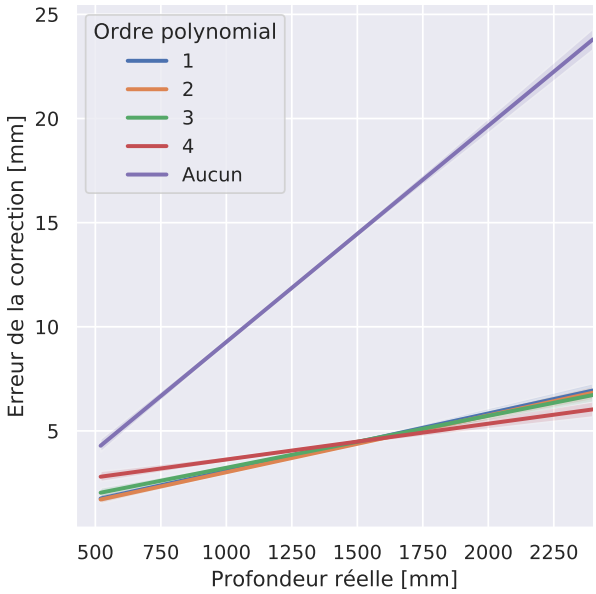


FIGURE 2.5 – Régression linéaire de l'erreur ($|d_c - d_e|$) selon la profondeur réelle d_e avec différents degrés polynomiaux pour la correction d_c. «*Aucun*» représente la profondeur d non corrigée obtenu par la Kinect.

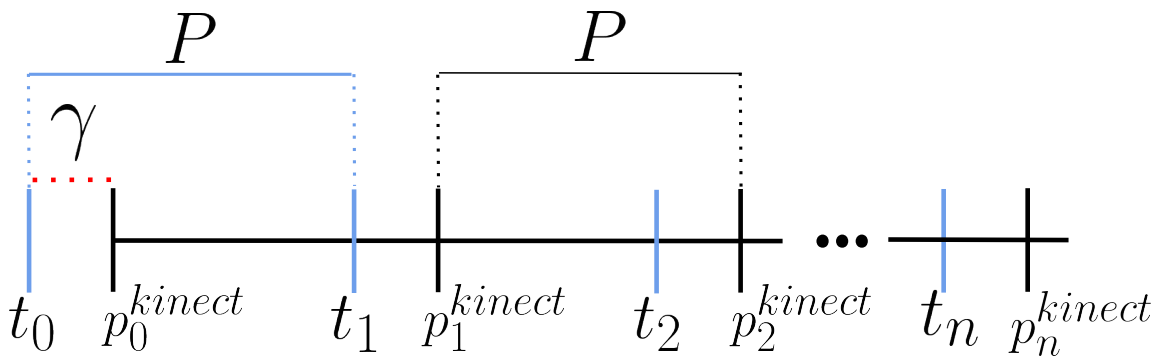


FIGURE 2.6 – Diagramme de synchronisation du signal émis par la caméra Kinect Azure

2.3 Calibrage temporel

Cette section explique la méthode utilisée pour calibrer les deux caméras dans un référentiel de temps commun.

Chaque caméra supporte le calibrage temporel matériel avec une approche maître-esclave où le maître émet une pulsation à une fréquence fixe et l'esclave reçoit la pulsation. Dans ce contexte, la Kinect Azure agit comme maître et émet une pulsation au début de l'exposition du capteur RGB tandis que la DAVIS346 agit comme esclave et reçoit la pulsation.

Chaque pulsation est associée à une estampille temporelle pour chacun des référentiels de

caméra soit p_n^{davis} pour a DAVIS346 et p_n^{kinect} pour la Kinect Azure où n représente l'index de la pulsation. Un décalage est observé entre les horloges internes des caméras. Ce décalage augmente avec le temps de capture et, pour cette raison, il ne peut pas être corrigé avec une valeur unique. Ainsi, seulement le référentiel de temps de la DAVIS346 est utilisé. La correspondance entre les deux référentiels est obtenue directement grâce à un alignement par rapport à l'index n.

À noter que dans le cas de la Kinect Azure, l'estampille temporelle obtenue est associée à la trame plutôt qu'à la pulsation et sa valeur absolue n'est pas documentée. Cette estampille est définie comme t_n. De plus, un nombre inconnu de pulsations sont émises avant de capturer la première trame et l'estampille t_n est déphasée par rapport à p_n^{kinect} car t_n/P n'est pas entier où P est le temps de la période entre deux expositions. Pour ces raisons, il n'est pas possible d'associer directement t_n avec p_n^{kinect}. La relation suivante est obtenue entre l'estampille de p_n^{kinect} et t_n :

$$p_n^{kinect} = t_n + \gamma + xP \tag{2.13}$$

où γ est la valeur inconnue du déphasage et x est le nombre de périodes de décalage par rapport à la première pulsation.

En expérimentant avec différents temps d'exposition ϕ de la caméra, nous constatons que $\gamma = -\phi$. De plus, en capturant une séquence d'images d'une diode électroluminescente allumée précisément entre deux pulsations, il est possible de déduire que $x = 0$. Notez que ces valeurs sont obtenues pour le micrologiciel en version 1.6.102 de la Kinect Azure et que ces valeurs peuvent différer avec une autre version.

La figure 2.6 illustre le signal de pulsation par rapport aux paramètres de la Kinect.

Chapitre 3

RGB-D-E : Event Camera Calibration for Fast 6-DOF Object Tracking

3.1 Résumé

Les appareils de réalité augmentée nécessitent plusieurs capteurs pour effectuer diverses tâches telles que la localisation et le suivi. Actuellement, les caméras populaires sont principalement basées sur des images (par exemple, RGB et profondeur) qui imposent une bande passante de données et une consommation d'énergie élevées. Avec la nécessité de systèmes de réalité augmentée à faible consommation et plus réactifs, l'utilisation de capteurs uniquement basés sur des trames impose des limites aux divers algorithmes qui nécessitent des données sur l'environnement à haute fréquence. En tant que tels, les capteurs basés sur les événements sont devenus de plus en plus populaires en raison de leur faible puissance, de leur bande passante et de leur latence, ainsi que de leur capacité d'acquisition de données à très haute fréquence. Dans cet article, nous proposons, pour la première fois, d'utiliser une caméra événementielle pour augmenter la vitesse de suivi d'objets 3D à 6 degrés de liberté. Cette application nécessite de gérer une vitesse d'objet très élevée pour transmettre des expériences de réalité augmentée convaincantes. Pour cela, nous proposons un nouveau système qui associe un capteur RGB-D récent (Kinect Azure) à une caméra événementielle (DAVIS346). Nous développons une approche d'apprentissage profond, qui combine un réseau RGB-D existant avec un nouveau réseau basé sur des événements en cascade De plus, nous démontrons que notre approche améliore considérablement la robustesse d'un système basé sur des trames à la pointe de la technologie. Notre code et notre jeu de données d'évaluation RGB-D-E sont disponibles sur https://github.com/lvsn/rgbde_tracking.

3.2 Abstract

Augmented reality devices require multiple sensors to perform various tasks such as localization and tracking. Currently, popular cameras are mostly frame-based (e.g. RGB and Depth) which impose a high data bandwidth and power usage. With the necessity for low power and more responsive augmented reality systems, using solely frame-based sensors imposes limits to the various algorithms that needs high frequency data from the environement. As such, event-based sensors have become increasingly popular due to their low power, bandwidth and latency, as well as their very high frequency data acquisition capabilities. In this paper, we propose, for the first time, to use an event-based camera to increase the speed of 3D object tracking in 6 degrees of freedom. This application requires handling very high object speed to convey compelling AR experiences. To this end, we propose a new system which combines a recent RGB-D sensor (Kinect Azure) with an event camera (DAVIS346). We develop a deep learning approach, which combines an existing RGB-D network along with a novel event-based network in a cascade fashion, and demonstrate that our approach significantly improves the robustness of a state-of-the-art frame-based 6-DOF object tracker using our RGB-D-E pipeline. Our code and our RGB-D-E evaluation dataset are available at https://github.com/lvsn/rgbde_tracking.

3.3 Introduction

Compelling augmented reality (AR) experiences are achieved through the successful execution of several tasks in parallel. Notably, simultaneous localization and mapping (SLAM) [49], hand tracking [47], and object tracking in 6 degrees of freedom (6-DOF) [17] must all be executed efficiently and concurrently with minimal latency on portable, energy-efficient devices.

This paper focuses on the task of 6-DOF rigid object tracking. In this scenario, successfully tracking the object at high speed is particularly important, since freely manipulating an object can easily result in translational and angular speeds of up to 1 m/s and 360°/s respectively. Despite recent progress on real time 6-DOF object tracking at 30 fps [17; 41; 36], these methods still have trouble with very high object motion and tracking failures are still common. Increasing the speed of 6-DOF object trackers is of paramount importance to bring this problem closer to real-world applications.

To increase the speed of object tracking, one can trivially employ cameras with frame rates higher than 30 fps. Indeed, 90 and even 120 fps off-the-shelf cameras are available and could be used as a drop-in replacement. However, this comes at significant practical disadvantages: higher data bandwidth, increased power consumption (since the algorithms must be executed more often), and the necessity to have sufficient light in the scene since exposure times for each frame is necessarily decreased.

In this work, we propose a system to increase the speed of 6-DOF object tracking applications with a minimal increase in bandwidth and power consumption. Specifically, we propose to *combine* an event camera (specifically, the DAVIS346 camera) with an RGB-D camera (the Kinect Azure) into a single "RGB-D-E" capture system. The event camera offers several key advantages: very low latency (20 μs), bandwidth, and power consumption (10–30 mW), all while having much greater dynamic range (120 dB vs 60 dB) than frame-based cameras.

This paper makes the following contributions. First, we show how to calibrate the setup both spatially and temporally. Second, we provide a new challenging publicly available 6-DOF evaluation dataset that contains approximately 2,500 RGB-D-E frames of a real-world object with high-speed motion with the corresponding ground truth pose at each frame. Third, we propose what we believe to be the first 6-DOF object tracker that uses event-based data. Similar to previous work [17; 41; 36], our approach assumes that the object to track must be rigid (non-deforming) and its textured 3D model must be known a priori. Finally, we demonstrate through a quantitative analysis on our real evaluation dataset that, using an extension of an existing deep learning approach for 6-DOF object tracking results in a threefold decrease in the number of tracking failures and achieves robust tracking results on fast free interaction motions. We believe this paper brings 6-DOF object tracking one step closer to real-world augmented reality consumer applications.

3.4 Related work

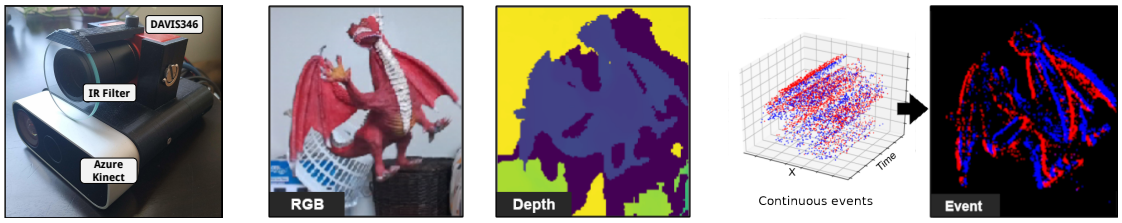
The majority of computer vision systems rely on established frame-based camera architectures, where the scene irradiance is captured synchronously at each pixel or in a rapid, rolling shutter sequence [37]. However, such cameras need to stream large amount of data (most of which redundant), making them power- and bandwidth-hungry. Recently, a newer camera architecture with a event-based paradigm [38] is gaining popularity. By triggering events on each pixel asynchronously when the brightness at that pixel changes by certain threshold, event-based camera can stream at a much higher frequency while consuming less power. A branch of computer vision research now focuses on developing algorithms to take advantage of this new type of data.

Event-based applications. Event-based sensors bring great promises in the field as their low power consumption makes them ideal for embedded systems such as virtual reality headset [13], drones [66; 9] or autonomous driving [43]. Their high-speed resolution also enables the design of robust high-frequency algorithms like SLAM [50; 67; 68; 56; 5; 13; 31] or fast 2D object tracking [19; 45]. While related to our work since we also focus on tracking, all related works are still restricted to tracking objects in the 2D image plane. In this paper, we extend the use of event cameras to the challenging task of fast 6-DOF object tracking by building over a state-of-the-art frame-based 6-DOF object tracker [16]. Different from other works, we benefit from RGB, Depth and Event data to propose the first RGB-D-E 6-DOF object tracker.

Deep learning with events. Using event-based data is not straightforward since the most efficient deep architectures for vision are designed for processing conventional image data (e.g. CNNs). In fact, it is still unclear how event-based data should be provided to networks since each event is a 4-dimensional vector storing time, 2D position, and event polarity. Experimental architectures such as spiking neural networks [40] holds great promises but are currently unstable or difficult to train [33]. With conventional deep frameworks, events can be converted to 2D tensors only by discarding both time and polarity dimensions [55] or to 3D tensors by discarding either of the two dimensions [43; 74]. Recent work [57; 61] has demonstrated that conventional grayscale frames can be reconstructed from event data, opening the way to the use of existing algorithms on these "generated" images. In this paper, we favor the Event Spike Tensor formulation from Gehrig et al. [18], where time dimension is binned. This allows us to exploit event data directly without requiring the synthesis of intermediate images, while maintaining a fast convolutional network architecture.

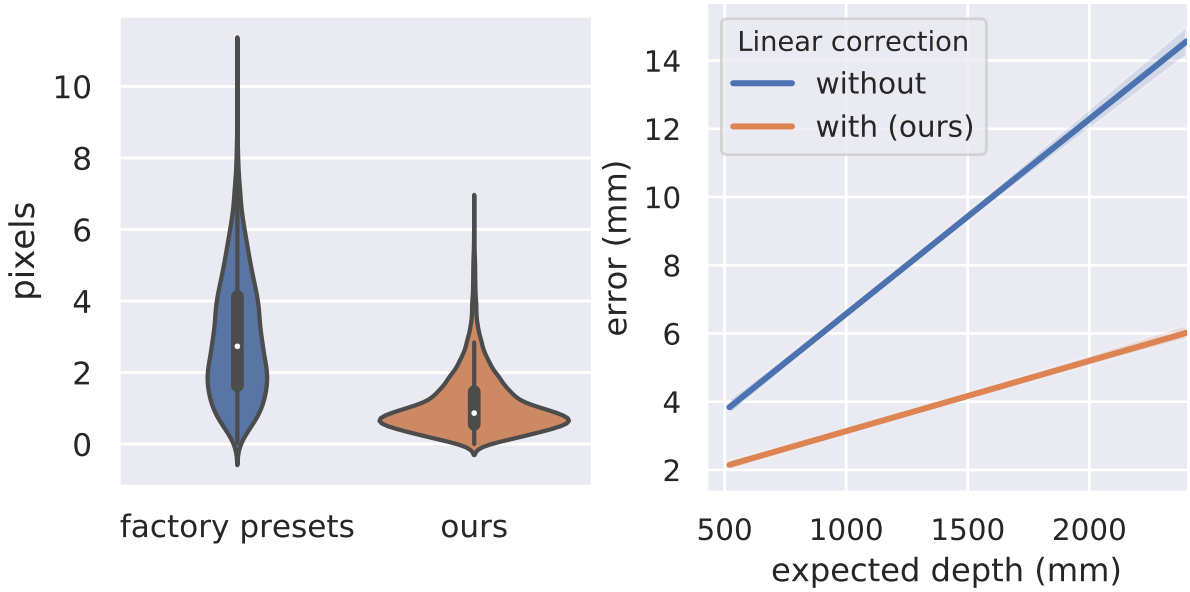
Event-based datasets. Finally, large amount of training data is required. While a few events datasets exist mostly for localization/odometry [9; 35; 46] or 2D object tracking [23], there are, as of yet, no 6-DOF object tracking dataset which contains event data. Instead, event data can be synthesized with a simulator such as [54] which allows various types of data augmentation [58]. Our experiments show that a network can be trained without using real

data and is not critically affected by the real-synthetic domain gap.



(a) Hardware setup (b) RGB (c) Depth (d) Event

FIGURE 3.1 – Our RGB-D-E hardware setup uses a Kinect Azure (RGB-D) and a DAVIS346 event-based camera (E) for continuous events which are temporally binned. RGB-D-E data streams are spatially and temporally calibrated and used for 6 degree of freedom object tracking.



(a) (b)

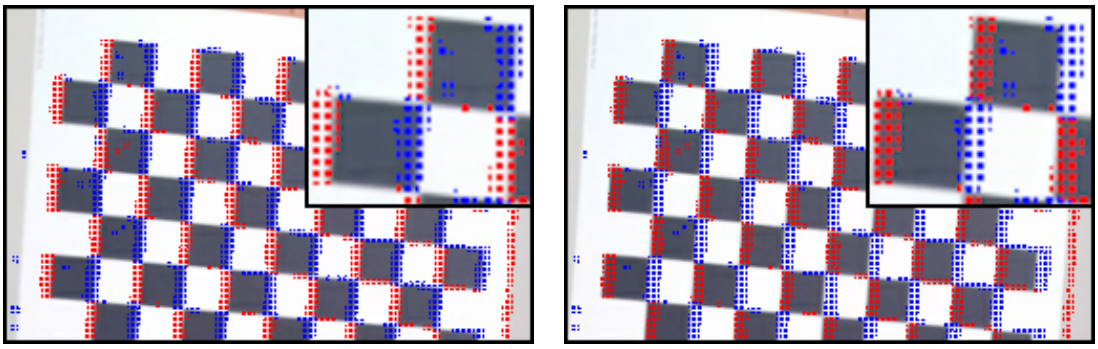
FIGURE 3.2 – Comparison between the factory presets and our calibration. (a) The reprojection error from the Depth to RGB image ($\mathbf{T}^{\text{RGB}}_{\text{Depth}}$) computed on 51 matching planar checkerboard images. (b) Linear regression of the Kinect depth map error compared to the expected depth, computed on calibration target corners.

3.5 System overview and calibration

In this section, we describe our novel RGB-D-E hardware setup, which combines a Microsoft Kinect Azure (RGB-D) with a DAVIS346 event camera (E).

3.5.1 System overview

As illustrated in Fig. 3.1, the DAVIS346 event camera is rigidly mounted over the Kinect Azure using a custom-designed, 3D-printed mount. We observed that the modulated IR signal projected on the scene by the Time-of-Flight (ToF) sensor in the Kinect triggered multiple



Frame t_0
Event $[t_0, t_0 + \Delta t]$

Frame $t_0 + \Delta t$
Event $[t_0, t_0 + \Delta t]$

FIGURE 3.3 – Events from the DAVIS346 camera projected on Kinect RGB frames on a moving calibration target. The projection is obtained using the calibrated transformation $\mathbf{T}^{\text{RGB}}_{\text{Event}}$. Events of a moving checkerboard are accumulated and represented as red (negative) and blue (positive) on both frames. Pixels with more than one event are presented to reduce distraction by noise. Frames are captured at a $\Delta t = 1/15$ s interval and events between t_0 and $t_0 + \Delta t$. A proper alignment of events with the checkerboard demonstrates that the system is calibrated both spatially and temporally.

events in the DAVIS346 camera. To remedy this limitation, an infrared filter is placed in front of the event camera lens.

3.5.2 Spatial calibration

Our system contains 3 cameras that must be calibrated: the Kinect RGB, the Kinect Depth and the DAVIS346 sensor. In this paper, we describe a coordinate system transformation with the notation $\mathbf{T}^b_a$, denoting a transformation matrix from coordinate frame a to b.

The intrinsic parameters of each camera can be computed with a standard method [71]. The checkerboard corners can easily be found using the color frame and the IR image from the Kinect Azure. Calibrating an event-based sensor is usually more difficult, however the DAVIS346 possesses an APS sensor (gray scale frame-based capture) that is spatially aligned with the event-based capture sensor. We thus use the APS sensor to detect the target corners that will be used for the intrinsic and extrinsic calibration.

Intrinsics. We capture images where a checkerboard target (9×14 with 54 mm squares) is positioned in a spatial uniform distribution in the frustum of each camera. To account for varying fields of view, 199 images were captured for the Kinect RGB, 112 for the Kinect Depth, and 50 for the DAVID346. For each sensor, we retrieve the intrinsic parameters (focal and image center) with a lens distortion model including 6 radial and 2 tangential parameters.

Extrinsics. We retrieve the rigid transformations $\mathbf{T}^{\text{Depth}}_{\text{RGB}}$ and $\mathbf{T}^{\text{Depth}}_{\text{Event}}$ by capturing images of the target in overlapping frustums. Once the 3D points are retrieved from the previously

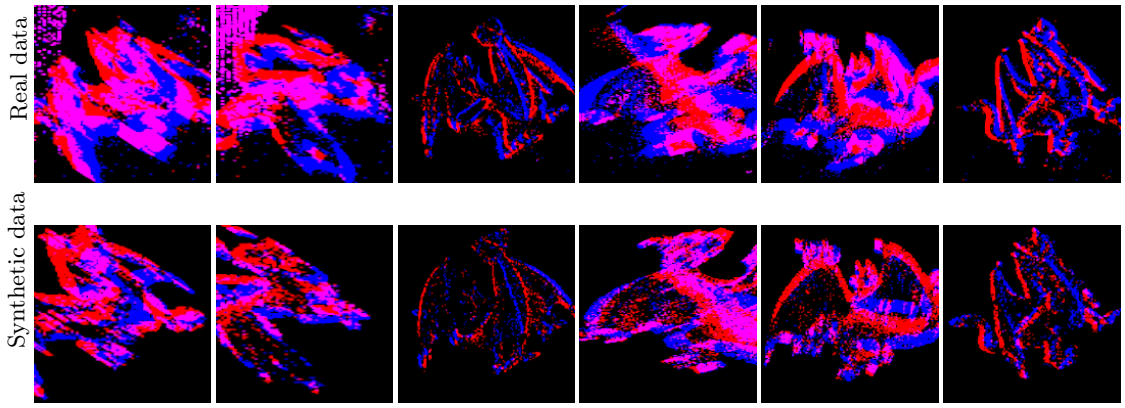


FIGURE 3.4 – Qualitative comparison between real (top) and synthetic events (bottom). Synthetic frames are generated with the event simulator of Rebecq et al. [54], where the pose of the synthetic object is adjusted to match the real. Event polarities are displayed as blue (positive) and red (negative).

computed camera intrinsic and the known checkerboard geometry, PnP [11] is used to retrieve the 6-DOF transformation between each camera.

Finally, we compare our calibration procedure with the factory presets of the Kinect Azure. Motivated by previous work [6] that demonstrate lower accuracy errors with factories presets calibration we capture a *test* dataset of 45 target images and show that we obtain a lower reprojection error in Fig. 3.2-(a).

3.5.3 Depth correction

As [22; 17] reported for the Kinect 2, we also found that the depth from the Kinect Azure has an offset that changes linearly w.r.t the depth distance and average in an error in the range of 8.5 mm. We compare the target points from the calibration dataset with the depth pixels in each frame and fit a 2nd-degree polynomial to the errors w.r.t to their distance to the camera. In Fig. 3.2b, we show the error with and without the polynomial correction on the test calibration set. Using the correction, the mean error on the test calibration set is less than 4 mm.

3.5.4 Temporal synchronization

In a multi-sensors setup, each sensor acquires data at its own frequency aligned with its inner clock. For time-critical applications, such as fast object tracking, it is required to synchronize the sensors clocks to ensure temporal alignment of the data. Technically, this is commonly addressed with synchronization pulses emitted by a *master* sensor at the beginning of each data frame acquisition, subsequently triggering the acquisition of other *slave* sensors.

In our setup, both Kinect and DAVIS346 support hardware synchronization but we found that

the Kinect (master) emits a variable number of pulses before the first RGB-D frame. This led to incorrect triggering of DAVIS346 (slave) and thus temporal misalignment of RGB-D and Event data. Because pulses are always emitted at the same frequency, we fix this by computing the pulses offset δ as

$$\delta = \lfloor \mathrm{RGBD}_0^t \times \mathrm{RGBD}^{\mathrm{fps}} \rfloor \,, \tag{3.1}$$

where RGBD_0^t is the timestamp of the first RGB-D frame and $\mathrm{RGBD}^{\mathrm{fps}}$ is the Kinect framerate (here, 30). Following this, we can pair RGBD and Event frames as $\left(\mathrm{RGBD}_i, \mathrm{E}_{i+\delta}\right)$. Fig. 3.3 illustrates the projection of events captured on a moving checkerboard. The events are captured between the two RGB frames. Alignment with the borders of the pattern shows the temporal and spatial calibration.

3.6 Fast 6-DOF object tracking

With the sensors spatio-temporally calibrated, we enhance an existing tracking framework by the addition of the new event modality (E). We build on the work of Garon et al. [16; 17] who propose a deep learning approach of robust 6-DOF object tracking, which relies on the refinement between a render of the object at the current pose estimate and the current Kinect RGB-D frame. While this method is robust to occlusion and small displacements, we notice that it is significantly impacted by larger motions (over 0.5 m/s), possibly because of the motion blur induced. Additionally, the network in [17] is fundamentally limited by a maximum pose translation of 2 cm between two frames. We note that increasing the sensor frame rate is also not a practical solution as the network computation time is the main bottleneck. In this section, we improve the tracker reactivity and robustness with the addition of an event-specific network. In the following, we first describe the generation of synthetic data for training and proceed to explain how frame-based and event-based trackers are jointly used.

3.6.1 Training data generation

Despite the existence of event datasets [23; 35; 72], none of them provide event data with 6-DOF object pose. Since capturing a dataset of sufficient magnitude and variety for training a deep network is prohibitive, we rely on synthetic data generated from an event camera simulator [54]. The engine renders a stream of events that represent changes in pixel brightness, thus mimicking event-based sensors. We build a training dataset by generating sequences of events where our target object (here, a toy dragon) is moved in front of a static camera. We acquire a textured 3D model of the dragon with a Creaform GoScan™ handheld 3D scanner at 1 mm voxel resolution, subsequently cleaned manually using Creaform VxElements™ to remove background and spurious vertices. As the camera remains stationary, we simulate the scene background with a random RGB texture from the SUN3D dataset [70] applied on a plane orthogonal to the virtual camera optical axis. We next describe the simulation setup followed by various data augmentation strategies applied to the data sample.

Simulation details. Event sequences are generated by first positioning the object in front of the camera at a random distance $d \sim \mathrm{U}(0.45\ \mathrm{m}, 0.8\ \mathrm{m})$ (where $\mathrm{U}(a, b)$ denotes a uniform distribution in the $[a, b]$ interval) and a random orientation. The center of mass of the object is aligned with the optical axis of the camera, so the object appears in the center of the frame. The object is then displaced by a random pose transformation over 33 ms and the generated events are recorded. The transformation is generated by first sampling two directions on the sphere using spherical coordinate (θ, ϕ) with $\theta \sim \mathrm{U}(-180°, 180°)$ and $\phi = \cos^{-1}(2x-1)$, where $x \sim \mathrm{U}(0, 1)$ as in [17] and then sample the magnitude of the translation and rotation with $\mathrm{U}(0\ \mathrm{m}, 0.04\ \mathrm{m})$ and $\mathrm{U}(0°, 35°)$ respectively. A 3D bounding box of size 0.207 m around the object is projected on the image plane. The event spatial axes are then cropped according to

the projected bounding box and resized with bilinear interpolation to a spatial resolution of 150×150. Each 33 ms pose transformation generates a set of N events storing $\{t, x, y, p\}_{i=1..N}$ where t is time, x and y are pixel coordinates and p the polarity of the event (positive or negative, indicating a brighter or darker transition respectively). A total of 10 such event sets are simulated for each background image, leading to 180,000 training and 18,000 validation sets.

Data augmentation. To maximize the reliability of our simulations, we randomize some parameters as in [58] to increase variability in the dataset and reduce the domain gap between synthetic and real data. The contrast threshold, which defines the desired change in brightness to generate an event, is difficult to precisely estimate on real sensors [13] and is instead sampled from a gaussian distribution $\mathrm{N}(0.18, 0.03)$ (where $\mathrm{N}(a, b)$ denotes a gaussian distribution with mean a and standard deviation b). Subsequently, the proportion of ambient lighting versus diffuse lighting for the OpenGL rendering engine (employed in the simulator) is randomly sampled from $\mathrm{U}(0, 1)$. To simulate tracking errors, the center of the bounding box is offset by a random displacement of magnitude $\mathrm{N}(0, 25)$ pixels. Finally, we notice the appearance of white noise captured by the DAVIS346. To quantify the noise, we capture a sequence of a static scene (which should generate no event) with the DAVIS346 and count the number of noisy events generated in each 33 ms window. A gaussian distribution is then fit to the number of noisy events. At training time, we sample a number k from the fitted distribution and randomly select k elements in the set (across t, x, and y) to add uniformly to the input volume. This process is done separately for each polarity (positive and negative). Fig. 3.4 shows the qualitative similarity between real samples acquired with the DAVIS346 and our synthetic samples at the same pose.

3.6.2 Approach overview

In this paper, we assume that the pose of the object in the previous frame, $\mathbf{P}_{t-1}$, is known. In a full system, it could be initialized by a 3D object detector (e.g. SSD-6D [29]) at the first frame ($t = 0$). The task of a temporal object tracker is to determine the relative pose *change* $\Delta\mathbf{P}$ between two frames such that an estimate of the current pose $\mathbf{P}_t$ can be obtained by

$$\mathbf{P}_t = \Delta\mathbf{P}\,\mathbf{P}_{t-1}\,. \tag{3.2}$$

Note that all poses $\mathbf{P}_i$ are expressed in the RGB camera coordinate system.

In this work, we rely on two deep networks to estimate $\Delta\mathbf{P}$. First, our novel event network $f_e(\mathbf{e}_{[t-1,t]})$ that takes event data $\mathbf{e}_{[t-1,t]}$ accumulated during the $[t-1, t]$ time interval, and cropped according to the previous object pose $\mathbf{T}_{\mathrm{RGB}}^{\mathrm{Event}}\mathbf{P}_{t-1}$. Here, $\mathbf{T}_{\mathrm{RGB}}^{\mathrm{Event}} = (\mathbf{T}_{\mathrm{Event}}^{\mathrm{Depth}})^{-1}\mathbf{T}_{\mathrm{RGB}}^{\mathrm{Depth}}$ is the extrinsic camera calibration matrix from sec. 3.5.2, necessary to transform the pose estimate in the event camera coordinate system.

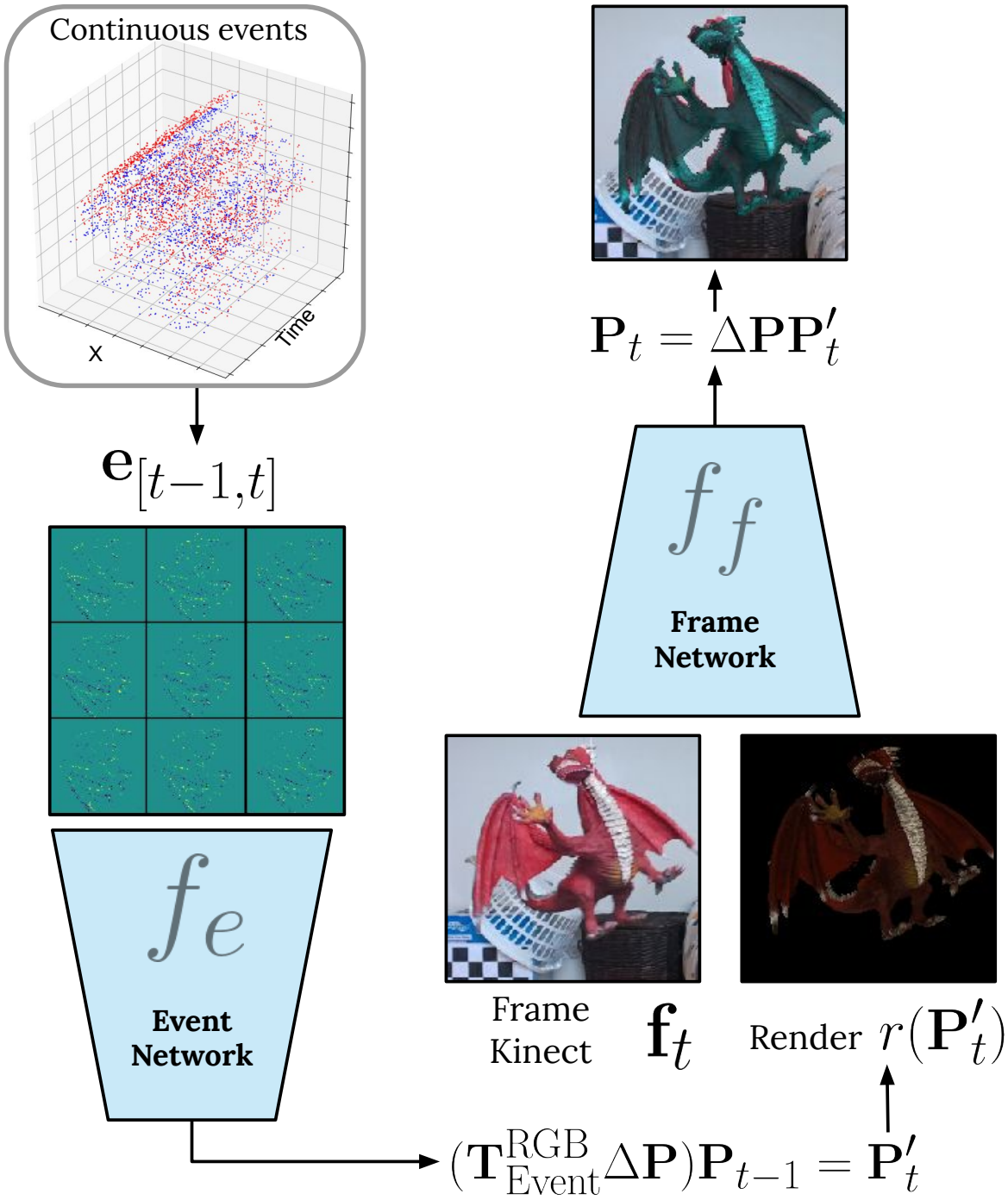


FIGURE 3.5 – Overview of our method for high-speed tracking with both sensors. On the left, our event network predicts the relative pose changes $\Delta\mathbf{P}$ from $\mathbf{e}_{[t-1,t]}$. The predicted pose is transformed to the RGB referential to estimate the current pose $\mathbf{P}'_t$. On the right side, the frame-based network [17] uses the improved pose $\mathbf{P}'_t$ to compute a final pose refinement.

Second, we also employ the RGB-D frame network of Garon et al. [17] $f_f(\mathbf{f}_t, \mathbf{P}_{t-1})$. Although more recent techniques exist, this choice was made because it is the only one providing both

Input: $\mathbf{e}_{[t-1,t]}$
kernel-5
conv-3-64
fire-32-64
fire-64-128
fire-128-256
fire-128-512
FC-500
FC-6
Output: ΔP

(a)

Input: $r(\mathbf{P}_{t-1})$	Input: $\mathbf{f}_t$
conv3-64	conv3-64
fire-32-64	fire-32-64
concatenation	
fire-64-256	
fire-128-512	
fire-256-1024	
FC-500	
FC-6	
Output: ΔP	

(b)

FIGURE 3.6 – Our deep network architecture for the (a) event and (b) RGB-D frames. We use the same network architecture as [17]. The main differences between both networks are that in (a) we only have one head and a learnable kernel is added to merge temporal information event input data $\mathbf{e}_{[t-1,t]}$ [18]. The notation "kernel-x" represents a learnable kernel of dimensions x convoluted on the temporal dimensions with the weights shared for every pixel. The notation "conv-x-y" represent a 2D convolution layer y filters of size $x \times x$, "fire-x-y" are "fire" modules [24] reducing the channels to x and expanding to y and "FC-x" are fully connected network of size x. Each fire module has skip-links followed by a 2×2 max pooling. Dropout of 30% is used after the activation function of each "fire" module and both "FC-500". All layers, (except "FC-6") are followed by an activation function.

training and inference code and offers robust performance. Alternatively, other approaches such as [27; 42; 44] could also be considered. However, [27] is already outperformed by [17], the inference code of [42] is limited to LineMOD objects [21], and [44] is an improvement over [17] for occlusion handling, which is not the focus here. Note that our method is not limited to this specific network and could be extended to any RGB or RGB-D frame based tracker.

Each network aims to estimate the relative 6-DOF pose of the object. Interestingly, while events are much more robust to fast displacement they carry less textural information than RGB-D data and we found that the event network used on its own is slightly less accurate. Therefore, we use a cascade approach where the event network first estimate $\mathbf{P}'_t$, and subsequently the frame network is provided with this new estimation for refinement:

$$\mathbf{P}'_t = (\mathbf{T}^{\text{RGB}}_{\text{Event}} f_e(\mathbf{e}_{[t-1,t]}))\, \mathbf{P}_{t-1}\,, \tag{3.3}$$

$$\mathbf{P}_t = f_f(\mathbf{f}_t, r(\mathbf{P}'_t))\, \mathbf{P}'_t\,, \tag{3.4}$$

with $\mathbf{T}^{\text{RGB}}_{\text{Event}}$ obtained from the extrinsic camera calibration matrices from sec. 3.5.2 as before. Note that $f_f()$ is an iterative method and can be run multiple time to refine its prediction. To simplify the notation we show a single iteration, in practice, 3 iterations are used as in the original implementation. A diagram overview of the method is provided in fig. 3.5.

3.6.3 Event network

Event data is fundamentally different than frame-based data as it possesses two extra dimensions for time and polarity ($T \times P \times X \times Y$, where T is discretized time and P is polarity.). We use the "Event Spike Tensor" representation from [18] where the time dimension is binned (in our case 9 bins for a 33 ms sample), and the polarity dimension is removed by simply subtracting the negative events from the positive ones. Finally, the spatial dimensions are resized as explained in the previous section. The final tensor has a shape of $9 \times 150 \times 150$ where each voxel represents the number of events recorded per time bin. We normalize that quantity between 0 and 1 by dividing each voxel by the maximum amount of events seen in a single voxel during training.

Event network architecture. While the event spike tensor can be processed by a standard CNN, we follow [18] and first learn a 1D filter in the time dimension and then apply a standard image convolution where the time dimension acts as different channels. In practice, we use the same backbone from [17] for the RGB-D frame network and event network but change only the first two input layers to match the event spike tensor. Fig. 3.6 (a) shows the event network architecture. The event network is optimized with ADAM [32] at a learning rate of 0.001 and a batch size of 256. We train for 40 epoch and apply a learning rate scheduling by multiplying the latter by 0.3 every 8 epochs.

3.6.4 RGB-D network

The RGB-D network (see [17] for more details) takes as input the current RGB-D frame cropped according to the previous pose $\mathbf{f}_t$ and a rendering of the object at the previous pose $r(\mathbf{P}_{t-1})$. Both inputs have a shape of $4 \times 184 \times 184$ and are normalized by subtracting the mean and dividing by the standard deviation of a subset of the training dataset. The last layer outputs the predicted 6-DOF pose difference between both inputs.

RGB-D network architecture. As shown in fig. 3.6-(b), each input is individually convoluted then passed to a "fire" module [24]. The module outputs are then concatenate before being max pooled. The single feature map is fed to multiple "fire" modules before being applied to two fully connected layers. The RGB-D network is trained with the same optimizer, hyper-parameters and data augmentations from the original work (see [17] for more details).

3.6.5 Processing Time

The average inference time is 29.02 ms split in 25.06 ms for the RGB-D network and 3.96 ms for the event network. Note that inference can be reduced close to 25 ms total by running networks in parallel since the total memory footprint is 152.48 MB (frame) + 54.96 MB (event) = 207.44 MB (total), which easily fits on a modern GPU. Runtimes are averaged over 100 samples and computed on an Intel i5 and Nvidia GeForce GTX 1060.

The numbers reported above include all prepossessing steps such as calculating the bounding box from the last known position and rendering the image for [17]. Note that building the "Event Spike Tensor" representation can be achieved in real time while the previous frame is being processed by the networks (average of 9ms). While our current, unoptimized implementation computes those steps serially, this operation could be trivially parallelized.

3.7 Experiments

We now proceed to evaluate our RGB-D-E system for the high-speed tracking of 3D objects in 6-DOF. We first describe our real test dataset, then present quantitative and qualitative results.

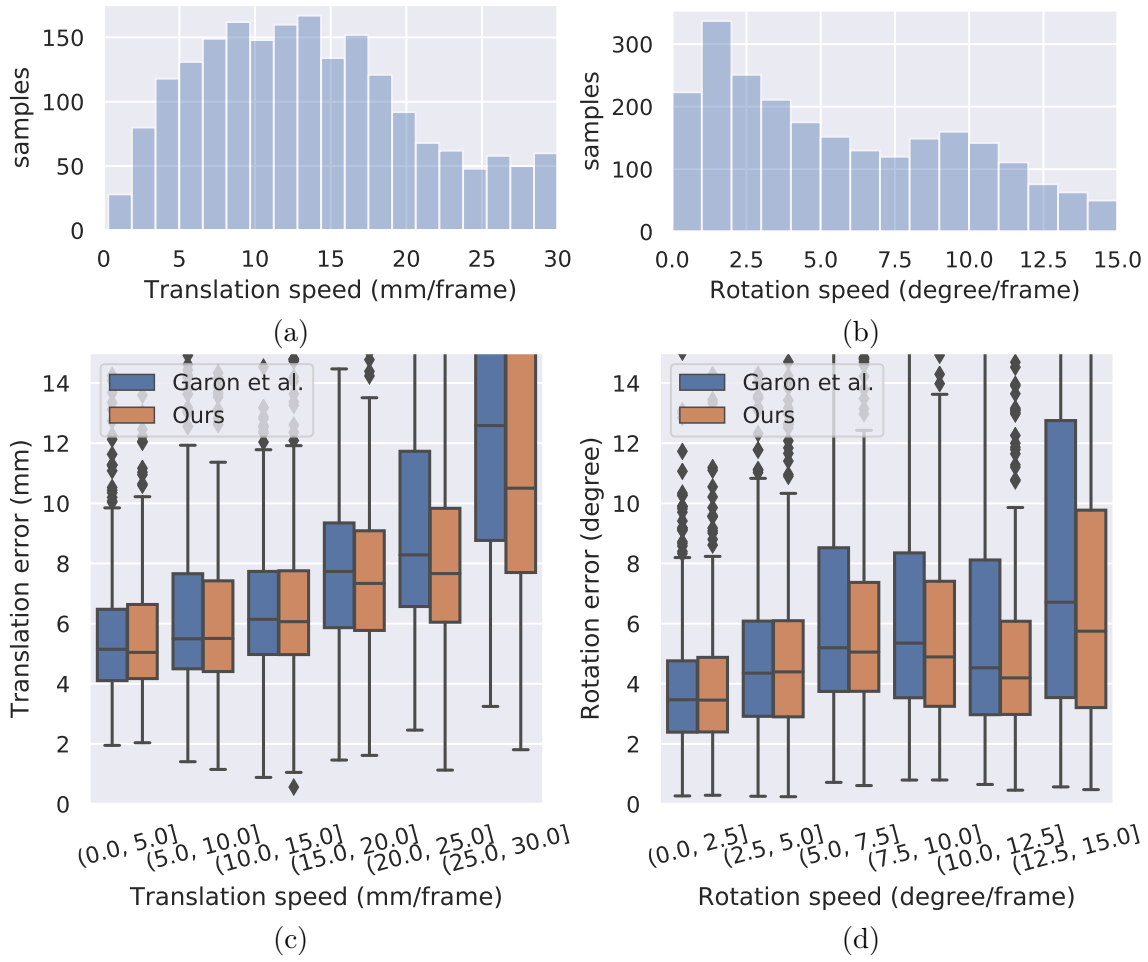


FIGURE 3.7 – Translation and rotation error as a function of object displacement speed, computed over two consecutive frames. The first row shows the test set distribution indicating the number of frames where the object has a particular (a) translation and (b) rotation speed. The second row plots the distribution of errors between the prediction and the ground truth, computed separately for (c) translation (eq. 3.5) and (d) rotation (eq. 3.6). Errors are computed on 10 sequences with a total of 2,472 frames.

3.7.1 Test dataset

In order to compare the RGB-D and the RGB-D-E trackers, we capture a series of real sequences of a rigid object freely moving at various speeds with different environment perturbation and record the corresponding RGB-D frames and events using our capture setup. To provide a quantitative evaluation, we obtain ground truth pose of the object at each frame

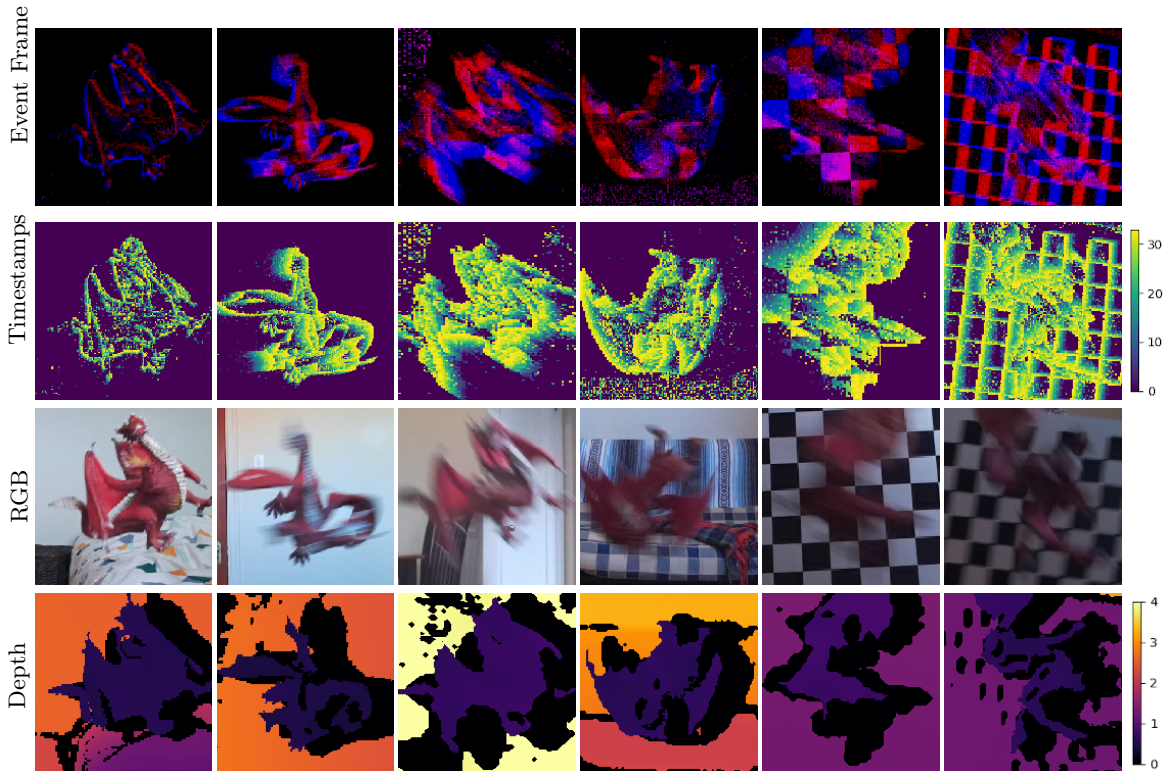


FIGURE 3.8 – Visualization of the RGB-D-E dataset. Event amplitude (top) from 0 to 5 events accumulate for 33 ms. Event polarities are displayed as blue (positive) and red (negative). Timestamp (second row) associated to each event. Only the last timestamp is displayed for each pixel and range from 0 to 33 ms. Synchronized RGB (third row) and depth map frame (bottom) from the Kinect. The depth map interval is from 0 to 4 m.

using the approach described below. We capture a total of 10 sequences with an average duration of 10 seconds, for a total of 2,472 frames and corresponding event data. Examples from different sequences of the dataset are shown in Fig. 3.8. The full dataset is publicly available.

For each sequence, we first manually align the 3D model with the object on the first RGB frame. Then, we use ICP [53] to align the visible 3D model vertices with the depth from the RGB-D frame, back-projected in 3D. To avoid back-projecting the entire depth frame, only a bounding box of $(280\text{ mm})^3$ centered around the initial pose is kept. Vertex visibility is computed using raytracing and updated at each iteration of ICP. If the angular pose difference between two successive iterations of ICP is less than 10°, it is deemed to have converged and that pose is kept. If that condition is not met after a maximum of 10 iterations, ICP diverges and the final pose is refined manually. For all subsequent frames in the sequence, ICP is initialized with the pose from the previous frame. In all, every frame in our test dataset is manually inspected to ensure a good quality pose is obtained, even when it has been determined automatically.

3.7.2 Evaluation

We quantitatively compare our RGB-D-E tracker with the RGB-D approach of Garon et al. [17], which is the current state-of-the-art in 6-DOF object tracking. We represent a pose $\mathbf{P} = [\mathbf{R}\ \mathbf{t}]$ by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$. The translation error $\delta_{\mathbf{t}}$ between a pose estimate and its ground truth (denoted by *) is reported as the L2 norm between the two translation vectors

$$\delta_{\mathbf{t}}(\mathbf{t}^*, \mathbf{t}) = ||\mathbf{t}^* - \mathbf{t}||_2 \,. \tag{3.5}$$

The rotation error between the two rotation matrices is computed using

$$\delta_{\mathbf{R}}(\mathbf{R}^*, \mathbf{R}) = \arccos\left(\frac{\mathrm{Tr}(\mathbf{R}^T\mathbf{R}^*) - 1}{2}\right) , \tag{3.6}$$

where $\mathrm{Tr}(\cdot)$ denotes the matrix trace.

Fig. 3.7 compares the translation and rotation errors obtained by both approaches. These plots report the error between two adjacent frames only: the trackers are initialized to their ground truth pose at the initial frame. Our method reports lower errors at translation speeds higher than 20 mm/frame, which corresponds to approximately 600 mm/s, and similar rotation errors overall. This is not surprising, given the fact that our method relies on the RGB-D network of Garon et al. [17] to obtain its final pose estimate.

However, visualizing the per-frame error does not tell the whole story. Indeed, in a practical scenario the trackers estimate a succession of predictions instead of being reset to the ground truth pose at every frame. Errors, even small, may therefore accumulate over time and result in tracking failure. Following [17], we consider a tracking failure when either $\delta_{\mathbf{t}}(\mathbf{t}_i^*, \mathbf{t}_i) > 3$ cm or $\delta_{\mathbf{R}}(\mathbf{R}_i^*, \mathbf{R}_i) > 20°$. Results of this analysis are presented in Tab. 3.1. The experiment is done at 30fps, 15fps, and 10fps, which is obtained by down-sampling the input frame-based frequency. To allow comparison, the event network is run at the same frequency with the same time window to accumulate the events of 33 ms. For all frame rates, our RGB-D-E approach has at least 61% fewer failures than the RGB-D approach of Garon et al. [17].

Fig. 3.9 shows representative qualitative results comparing both techniques with the ground truth. Those results show that the approach of Garon et al. [17] is affected by the strong motion blur which arises under fast object motion. In contrast, our approach remains stable and can follow the object through very fast motion. **Please see video results in the supplementary materials**.

Method	Failures		
	30fps	15fps	10fps
Garon et al. [17]	83	130	166
Ours	28	48	64

TABLE 3.1 – Number of tracking failures for each method with multiple RGB-D camera's frame rates. Failures are computed on 10 independent sequences with a total of 2,472 frames.

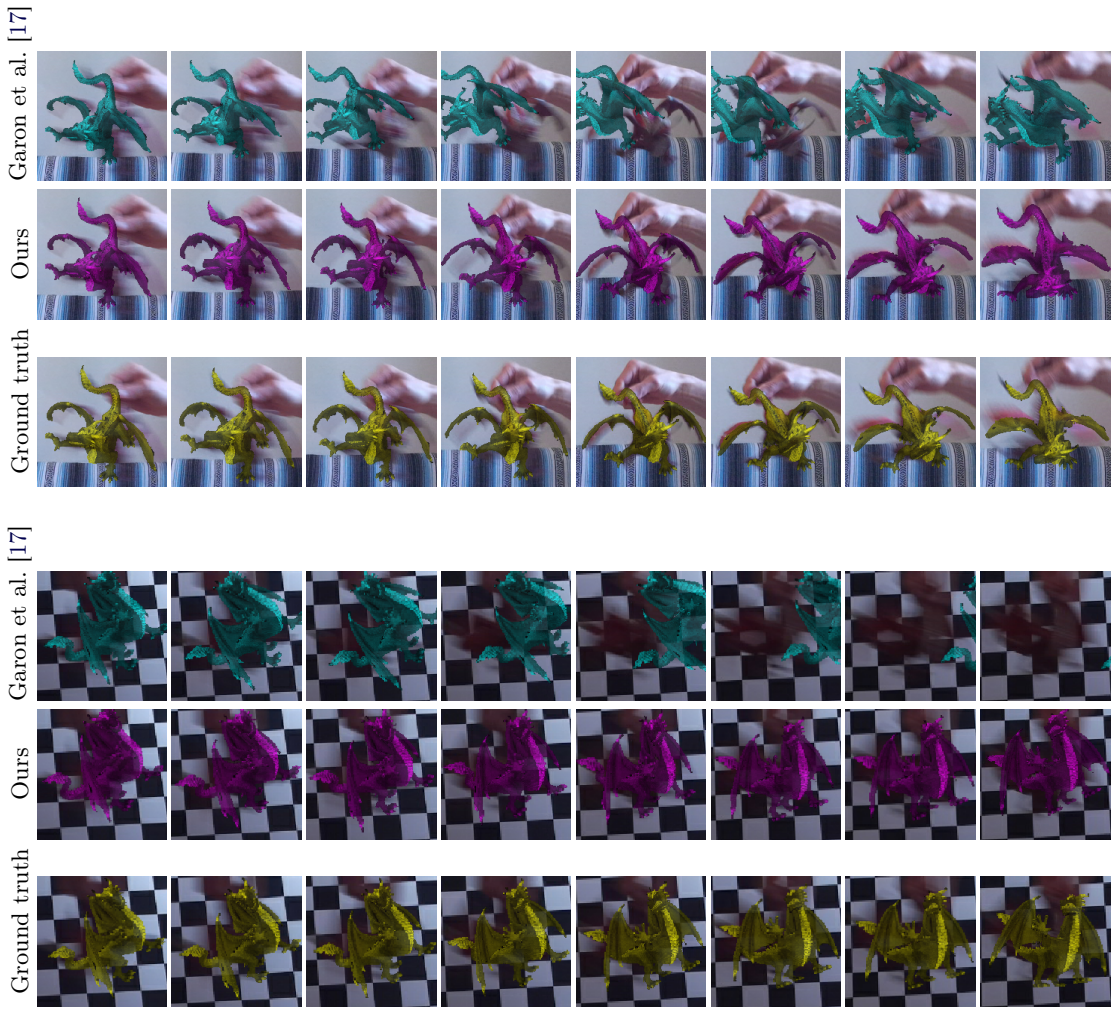


FIGURE 3.9 – Qualitative comparison between the different approaches on different sequences. The overlay is the position predicted by each method. From top to bottom, Garon et al. [17] (blue), ours (pink) and the ground truth (yellow). Each frame is continuous in the sequence and cropped according to the ground truth position.

3.8 Discussion

We present a novel acquisition setup for simultaneous RGB-D-E capture which combines a Kinect Azure camera with a DAVIS346 sensor. With the new event modality, we show that a state-of-the-art RGB-D 6-DOF object tracker can be significantly improved in terms of tracking speed. We capture an evaluation dataset with ground truth 3D object poses that mimics difficult scenarios typically encountered in augmented reality applications : a user manipulating a small object with fast free motions. Using this dataset, we demonstrate that our approach achieves a threefold decrease in loss of tracking over the previous state-of-the-art, thereby bringing 6-DOF object tracking closer to applicability in real-life scenarios.

Limitations and future work. First, capturing an evaluation dataset is time-consuming and obtaining the 6-DOF ground truth pose of the object is difficult, especially when fast motions are involved. While our semi-automatic approach provided a way to acquire a small number of sequences easily, scaling up to larger RGB-D-E datasets will require more sophisticated apparatus such as a motion capture (mocap) setup as in [17]. Indeed, mocap systems are ideal for this use case as they can track the object robustly at high frame rates. Second, while using a cascade scheme improves significantly the robustness to large motion of the tracker, it is still inherently limited in accuracy since it always relies on the frame network. The success of the cascade configuration motivates further exploration of better ways to fuse the Event modality with the previous frame-based modalities. Third, we notice that the trackers are still sensitive to dynamic backgrounds (see the last example in the supplementary video). We anticipate that this could be partially solved by generating training data with spurious structured events such as those that could be created by a dynamic background (or a moving camera). These represent exciting future research directions that we plan to investigate in order to achieve even more robust and accurate object tracking systems that can be used in real-world augmented reality applications.

Chapitre 4

Discussion : fusion des modalités

Avec la méthode de suivi d'objet du chapitre 3, deux réseaux de neurones sont employés et les données RGB-D sont séparées des événements. La position de l'objet est fusionnée à l'extérieur des réseaux en appliquant une transformation connue entre la pose des deux caméras. Les trames RGB-D permettent d'obtenir la pose absolue de l'objet tandis qu'une séquence d'événements donne le déplacement de l'objet. La plus grande limitation avec cette approche est qu'aucune des modalités n'est partagée entre les deux réseaux. Puisque les deux modalités sont fondamentalement différentes, nous estimons que leur union devrait donner un gain de performance sur les métriques de suivi d'objet du chapitre 3.

Pour cette raison, ce court chapitre se concentre sur la possibilité de fusionner ces deux modalités autrement. Nous commençons par explorer une méthode de fusion spatiale. Par la suite, une méthode avec réseau temporel est envisagée et nous terminons avec les détails d'implémentation de ces deux méthodes.

Malheureusement, malgré que ces approches semblent prometteuses, leurs performances de suivi sont inférieures ou égales à la méthode courante du chapitre 3 selon des expérimentations préliminaires. Les résultats ne sont pas détaillés ici et uniquement la méthodologie est abordée dans l'optique de guider des travaux ultérieurs.

4.1 Fusion spatiale

Pour la fusion spatiale des modalités, deux réseaux sont expérimentés, soit la fusion hâtive (*Early Fusion*) et la fusion tardive (*Late Fusion*). Dans le cas de la première méthode, les données sont fusionnées dans les premières couches du réseau à neurones profonds. Dans le cas de la seconde méthode, les données sont fusionnées dans les dernières couches.

Cette méthode est différente de la méthode courante par son architecture regroupant les modalités dans un même réseau au lieu d'avoir deux réseaux distincts et d'appliquer la transformation manuellement. L'architecture est basée sur la méthode du chapitre 3 en modifiant les

réseaux précédents au minimum. La figure 4.1 schématise l'architecture pour les deux réseaux.

Lors de l'inférence, l'information événementielle $\mathbf{e}_{[t-1,t]}$ est accumulée entre les trames $\mathbf{f}_{t_1}$ et $\mathbf{f}_t$. Lors de la première itération, $\mathbf{e}_{[t-1,t]}$ est fourni conjointement avec la trame courante f_t et un rendu $r(\mathbf{P}_{t-1})$ de la position précédente. Ici, l'information événementielle est ajoutée pour indiquer au réseau le déplacement de l'objet depuis la dernière trame. De plus, dans un suivi optimal, ce déplacement observé par $\mathbf{e}_{[t-1,t]}$ devrait être le même qu'entre $\mathbf{f}_t$ et $r(\mathbf{P}_{t-1})$. Pour les deux itérations successives, seulement $\mathbf{f}_t$ et le rendu de la nouvelle pose $r(\mathbf{P}_t)$ sont fournis pour raffiner la pose courante, similairement à la méthode du chapitre 3. La figure 4.3 illustre les itérations du réseau.

Durant l'entrainement, le réseau est entrainé avec les informations événementielles uniquement 1/3 du temps. Durant le reste des itérations, uniquement les trames RGB-D et les rendus sont fournis. Ceci simule le comportement observé à l'inférence sans forcer la boucle entière des trois itérations. Les détails sur les données d'entrainement sont discutés dans la section 4.3.

4.2 Fusion temporelle

Cette section expérimente une architecture récurente où les données événementielles, RGB et de profondeur sont combinées dans un même réseau. Ceci est similaire à la section 4.1 mais ce choix de réseau permet de capturer de l'information entre les itérations.

Le *Gated Recurrent Unit* (GRU) est un réseau de type *Recurrent Neural Network* (RNN) et est différent d'un réseau conventionnel car l'état caché *hidden state* est partagé entre chaque itération. En ajoutant une entrée et une sortie au réseau, il est possible de partager cette information entre les itérations. La figure 4.4 illustre l'architecture du réseau.

L'intuition derrière cette approche est de capturer la position absolue de l'objet entre les itérations. Ainsi, il n'est plus nécessaire de fournir la trame RGB à chaque itération et il est possible d'utiliser les derniers événements obtenus pour mettre à jour la position de l'objet. En enlevant la restriction liée à la fréquence de capture de la caméra, on se rapproche d'une prédiction asynchrone en modulant la taille des événements accumulés.

Pour la première itération, le réseau prends en entrée un rendu de la position courante du suivi $r(\mathbf{P}_{t-11})$, la trame RGB-D $\mathbf{f}_t$ et les événements associés à ce déplacement $\mathbf{e}_{[t-11,t]}$. Durant les itérations successives, seulement les événements sont fournis pour prédire le déplacement. Ceux-ci sont accumulés sur une période prédéterminée ($11ms$ en pratique) et représentée en *Event Spike Tensor* similaire à l'approche du chapitre 3. Contrairement à ce chapitre, seulement trois cannaux sont utilisés pour cette représentation au lieu de neuf. Ceci devrait conserver un ratio d'événements similaires par canal car les événements sont accumulés sur $11ms$ au lieu de $33ms$. La période d'accumulation des événements et le nombre d'itérations pourrait être modulés mais il faudrait en prendre compte durant l'entraînement du réseau. La

figure 4.4 illustre les itérations durant l'inférence.

Durant l'entraînement, les trois itérations sont exécutés séquentiellement. Chaque itération comprend l'information *hidden state* de l'itération précédente excepté pour la première itération où celle-ci est initialisée à zéro. Les événements accumulés des 11 dernières ms sont fournis à chaque itération mais le rendu et les trames RGB-D sont fournis uniquement pour la première itération. La fonction de perte et les poids du réseau sont mis à jour après chaque cycle de trois itérations. Ceci a pour but de ne pas pénaliser le réseau si une itération est erronée et seulement pénaliser la prédiction du cycle complet. Les autres détails d'implémentation sont discutés dans la section 4.3.

4.3 Implémentation

Cette sous-section décrit les détails d'implémentation supplémentaires pour les deux sous-sections précédentes. Toutes les expérimentations de cette section sont basées sur l'entraînement du réseau de suivi événementiel de la section 3.6.3. Ainsi, tous les paramètres non-spécifiés sont identiques à ceux de cette section.

Les données d'entraînement sont encore purement synthétiques. Cependant, en fusionnant les données au niveau du même réseau, nous assumons que la prédiction du réseau est basée sur le point de vue du capteur RGB. Ainsi, la transformation T_{Event}^{RGB} entre les deux caméras n'est jamais donnée directement au réseau mais les trames RGB et les données événementielles sont générées du même point de vue que leur capteur respectif. Ceci impacte surtout l'ajout des arrière-plans qui sont augmentés aléatoirement lors de l'entraînement pour le réseau RGB-D. Aussi, nous supposons que le réseau apprend la transformation entre les deux capteurs sans devoir l'appliquer manuellement durant la fusion des positions comme pour l'approche du chapitre 3.

Des expérimentations préléminaires ont été réalisées en les comparant avec la méthode du chapitre 3. Les erreurs sont calculées en boucle ouverte, soit entre deux trames, sur la translation et la rotation de manière similaire à l'expérimentation de la figure 3.7. Celles-ci sont calculés sur les méthodes explorées dans ce chapitre et sur le jeu de données RGB-D-E du chapitre 3. Nous n'avons malheureusement pas observé d'amélioration de performance comparativement à la méthode du chapitre 3. Pour cette raison, des travaux futurs seraient nécessaires pour améliorer les performances.

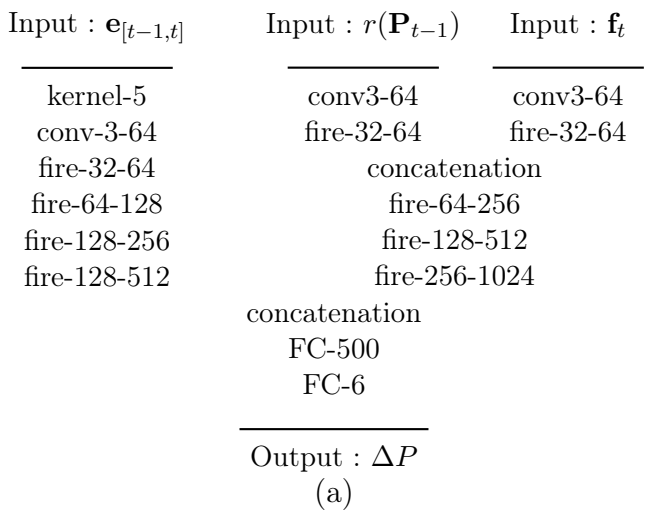


FIGURE 4.1 – Architecture de fusion tardive (a) et de fusion hâtive (b). La notation est la même que celle du chapitre 3. La différence principale est la fusion entre les deux têtes avec concatenation. La notation est la même que celle de la figure 3.6.

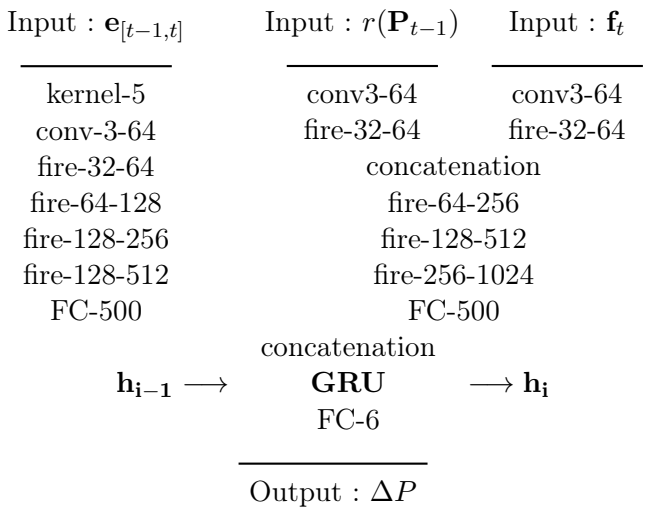


FIGURE 4.2 – Architecture de fusion avec réseau GRU. L'architecture est similaire à celle du chapitre 3 et la notation est la même que celle de la figure 3.6. La différence principale est la fusion entre les deux têtes avec concatenation et le réseau GRU avec son entrée $\mathbf{h_{i-1}}$ et sa sortie $\mathbf{h_i}$. L'entrée est connectée à la sortie de l'itération précédente.

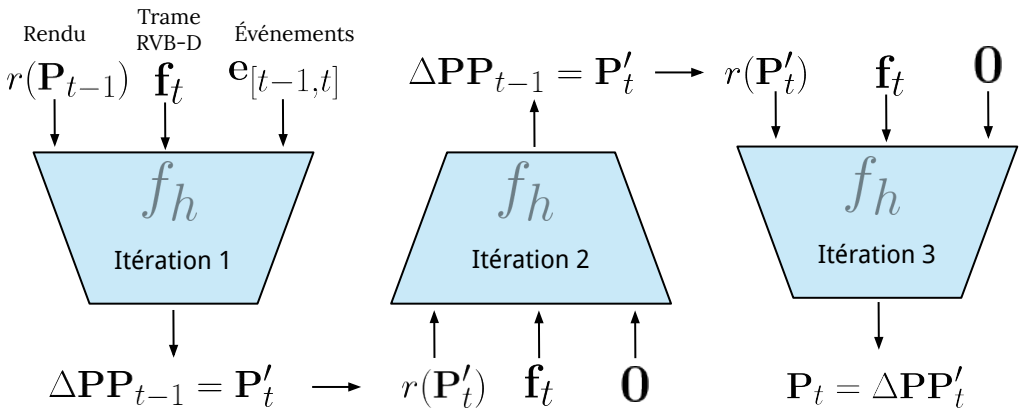


FIGURE 4.3 – Vue d'ensemble des itérations pour les méthodes de fusion hâtive et tardive. Seul la prédiction entre deux trames est illustrée. La première itération prédit le déplacement $\mathbf{\Delta P}$ entre la trame $\mathbf{f}_{t-1}$ et $\mathbf{f}_t$. Les itérations successives utilisent uniquement la position absolue de l'objet, fournie par la trame $\mathbf{f}_t$, pour raffiner la pose.

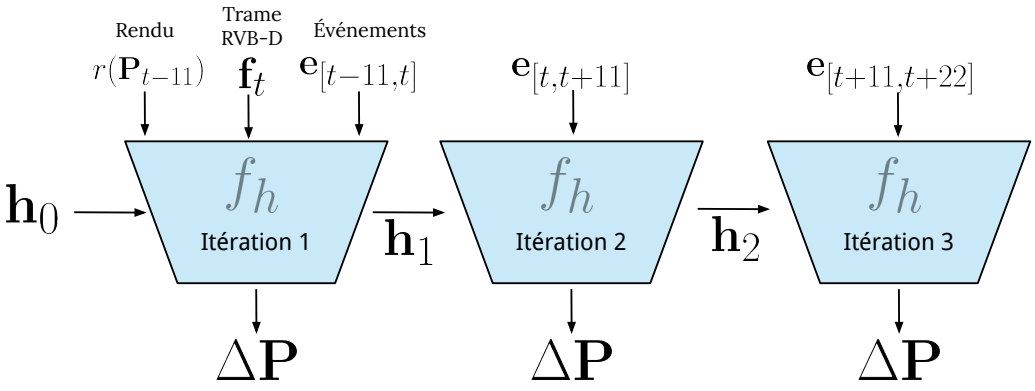


FIGURE 4.4 – Vue d'ensemble de la méthode de fusion GRU pour une prédiction entre deux trames. Seulement la prédiction entre deux trames est illustrée.

Conclusion

Le suivi d'objet demeure une tâche complexe. Malgré les améliorations des dernières années avec les algorithmes d'apprentissage profond, ceux-ci demeurent saturés en grande partie par le type de capteur utilisé. Ils utilisent majoritairement des caméras conventionnelles avec une fréquence de capture fixe. Ceci limite les performances à grande vitesse majoritairement dû au flou de mouvement capturé qui crée une ambiguïté dans la pose de l'objet.

L'objectif de ce mémoire est de présenter une méthode pour résoudre le problème de suivi d'objet à grande vitesse. Pour y parvenir, nous proposons une méthode combinant avec une caméra conventionnelle et une caméra événementielle.

Tout d'abord, le chapitre d'introduction décrit la technique de calibration utilisée pour calibrer les deux types de caméras ensemble. La méthode de calibration détaillée est accessible pour des expérimentations futures car elle requiert uniquement une cible de calibration standard et une imprimante 3D pour le support entre les caméras. De plus, celle-ci détaille une calibration spatiale et temporelle entre les deux caméras en plus de la correction de la profondeur.

Le chapitre 3 est la contribution principale du mémoire. Celui-ci démontre qu'il est possible d'améliorer les performances de suivi d'objet à grande vitesse. En combinant une caméra événementielle avec une caméra RGB-D, la vitesse de translation peut atteindre jusqu'à $600mm/s$. Ce chapitre se base sur une méthode existante et ajoute un second réseau pour raffiner la pose entre deux trames RGB-D. Le tout est entraîné uniquement avec des données synthétiques pour toutes les modalités.

Finalement, une contribution supplémentaire est le jeu de données capturé et rendu accessible publiquement à la communauté. Celui-ci capture près de 2500 trames en RGB-D accompagnées des événements correspondants dans différents scénarios. Ces scénarios sont axés sur les vitesses de déplacement angulaire et spatiale. Ce jeu de données n'est pas utilisable pour un entraînement de réseau de neurones dû à son nombre limité d'échantillons. Il peut toutefois être utilisé pour quantifier les performances d'une méthode dans de futurs travaux.

Malgré les gains de performance observés avec notre technique, nous pensons qu'il y a encore place à amélioration. Par exemple, le chapitre 4 discute d'approches non-successives qui pourraient potentiellement permettre d'avoir une meilleure fusion des modalités. Toutefois, afin

d'y parvenir, d'autres travaux seraient nécessaires. Cette fusion aurait le potentiel de mieux rassembler l'information et ainsi améliorer la prédiction finale. De plus, il serait intéressant d'utiliser le plein potentiel des caméras événementielles et d'avoir une méthode complètement asynchrone contrairement à notre méthode où les prédictions sont alignées avec les trames. Les réseaux de type *Spiking Neural Networks* [40] semblent prometteurs pour y parvenir, car la prédiction pourrait être raffinée à chaque nouveaux événements. En terminant, remplacer notre caméra conventionnelle par une caméra à capture rapide (*High FPS*) pourrait aussi donner un gain, mais augmenterait la complexité avec l'ajout de nouvelles données.

Nous espérons que cette recherche permettra l'avancement des travaux futurs dans le suivi d'objet en mouvement rapide et l'utilisation de caméras événementielles avec des caméras conventionnelles pour de multiples applications.

Bibliographie

[1] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. Di Stefano, and M. Vincze. Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In *2013 IEEE international conference on robotics and automation*, pp. 2104–2111. IEEE, 2013.

[2] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7243–7252, 2017.

[3] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi. Event-based visual flow. *IEEE transactions on neural networks and learning systems*, 25(2) :407–417, 2013.

[4] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pp. 850–865. Springer, 2016.

[5] S. Bryner, G. Gallego, H. Rebecq, and D. Scaramuzza. Event-based, direct camera tracking from a photometric 3d map using nonlinear optimization. In *International Conference on Robotics and Automation*, 2019.

[6] C. Chen, B. Yang, S. Song, M. Tian, J. Li, W. Dai, and L. Fang. Calibrate multiple consumer RGB-D cameras for low-cost and efficient 3D indoor mapping. *Remote Sensing*, 10(2) :328, 2018.

[7] M. Chitchian, A. S. van Amesfoort, A. Simonetto, T. Keviczky, and H. J. Sips. Adapting particle filter algorithms to many-core architectures. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 427–438. IEEE, 2013.

[8] C. Choi and H. I. Christensen. Rgb-d object tracking : A particle filter approach on gpu. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1084–1091. IEEE, 2013.

[9] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza. Are we ready for autonomous drone racing ? the uzh-fpv drone racing dataset. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6713–6719, 2019. doi : 10.1109/ICRA.2019.8793887

[10] D. DeTone, T. Malisiewicz, and A. Rabinovich. Deep image homography estimation. *arXiv preprint arXiv :1606.03798*, 2016.

[11] M. A. Fischler and R. C. Bolles. Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6) :381–395, 1981.

[12] S. Foix, G. Alenya, and C. Torras. Lock-in time-of-flight (tof) cameras : A survey. *IEEE Sensors Journal*, 11(9) :1917–1926, 2011.

[13] G. Gallego, J. E. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza. Event-based, 6-DOF camera tracking from photometric depth maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(10) :2402–2412, 2017.

[14] G. Gallego and D. Scaramuzza. Accurate angular velocity estimation with an event camera. *IEEE Robotics and Automation Letters*, 2(2) :632–639, 2017.

[15] Q. Gan, Q. Guo, Z. Zhang, and K. Cho. First step toward model-free, anonymous object tracking with recurrent neural networks. *arXiv preprint arXiv :1511.06425*, 2015.

[16] M. Garon and J.-F. Lalonde. Deep 6-DOF tracking. *IEEE Transactions on Visualization and Computer Graphics*, 23(11), Nov. 2017.

[17] M. Garon, D. Laurendeau, and J.-F. Lalonde. A framework for evaluating 6-DOF object trackers. In *European Conference on Computer Vision*, 2018.

[18] D. Gehrig, A. Loquercio, K. Derpanis, and D. Scaramuzza. End-to-end learning of representations for asynchronous event-based data. *IEEE/CVF International Conference on Computer Vision*, Oct 2019.

[19] A. Glover and C. Bartolozzi. Robust visual tracking with a freely-moving event camera. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[20] C. G. Harris, M. Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, vol. 15, pp. 10–5244. Citeseer, 1988.

[21] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pp. 548–562. Springer, 2012.

[22] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis. T-less : An RGB-D dataset for 6D pose estimation of texture-less objects. In *IEEE Winter Conference on Applications of Computer Vision*, 2017.

[23] Y. Hu, H. Liu, M. Pfeiffer, and T. Delbruck. Dvs benchmark datasets for object tracking, action recognition, and object recognition. *Frontiers in neuroscience*, 10 :405, 2016.

[24] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet : Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. *arXiv preprint arXiv :1602.07360*, 2016.

[25] G. Indiveri and R. Douglas. Neuromorphic vision sensors. *Science*, 288(5469) :1189–1190, 2000.

[26] P. Jatesiktat, M. J. Foo, G. M. Lim, and W. T. Ang. Sdf-net : Real-time rigid object tracking using a deep signed distance network. In *International Conference on Computational Science*, pp. 28–42. Springer, 2018.

[27] D. Joseph Tan, F. Tombari, S. Ilic, and N. Navab. A versatile learning-based 3d temporal tracker : Scalable, robust, online. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 693–701, 2015.

[28] J. Kaiser, J. C. Vasquez Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, R. Dillmann, and J. M. Zöllner. Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pp. 127–134, 2016. doi : 10.1109/SIMPAR.2016.7862386

[29] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. SSD-6D : Making RGB-based 3D detection and 6D pose estimation great again. In *IEEE International Conference on Computer Vision*, 2017.

[30] H. Kim, A. Handa, R. Benosman, S.-H. Ieng, and A. J. Davison. Simultaneous mosaicing and tracking with an event camera. *J. Solid State Circ*, 43 :566–576, 2008.

[31] H. Kim, S. Leutenegger, and A. J. Davison. Real-time 3D reconstruction and 6-DOF tracking with an event camera. In *European Conference on Computer Vision*, 2016.

[32] D. P. Kingma and J. Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014.

[33] J. H. Lee, T. Delbruck, and M. Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10 :508, 2016.

[34] J. H. Lee, T. Delbruck, M. Pfeiffer, P. K. Park, C.-W. Shin, H. Ryu, and B. C. Kang. Real-time gesture interface based on event-driven processing from stereo silicon retinas. *IEEE transactions on neural networks and learning systems*, 25(12) :2250–2263, 2014.

[35] W. Li, S. Saeedi, J. McCormac, R. Clark, D. Tzoumanikas, Q. Ye, Y. Huang, R. Tang, and S. Leutenegger. Interiornet : Mega-scale multi-sensor photo-realistic indoor scenes dataset. *arXiv preprint arXiv :1809.00716*, 2018.

[36] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox. Deepim : Deep iterative matching for 6d pose estimation. In *European Conference on Computer Vision*, 2018.

[37] C.-K. Liang, L.-W. Chang, and H. H. Chen. Analysis and compensation of rolling shutter effect. *IEEE Transactions on Image Processing*, 17(8) :1323–1330, 2008.

[38] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128x128 120 db 15μs latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2) :566–576, 2008.

[39] M. Litzenberger, B. Kohn, A. N. Belbachir, N. Donath, G. Gritsch, H. Garn, C. Posch, and S. Schraml. Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor. In *2006 IEEE intelligent transportation systems conference*, pp. 653–658. IEEE, 2006.

[40] W. Maass and H. Markram. On the computational power of circuits of spiking neurons. *Journal of computer and system sciences*, 69(4) :593–616, 2004.

[41] F. Manhardt, W. Kehl, N. Navab, and F. Tombari. Deep model-based 6d pose refinement in rgb. In *European Conference on Computer Vision*, 2018.

[42] F. Manhardt, W. Kehl, N. Navab, and F. Tombari. Deep model-based 6d pose refinement in rgb. *Lecture Notes in Computer Science*, p. 833–849, 2018. doi : 10.1007/978-3-030-01264-9_49

[43] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[44] I. Marougkas, P. Koutras, N. Kardaris, G. Retsinas, G. Chalvatzaki, and P. Maragos. How to track your dragon : A multi-attentional framework for real-time rgb-d 6-dof object pose tracking, 2020.

[45] A. Mitrokhin, C. Fermüller, C. Parameshwara, and Y. Aloimonos. Event-based moving object detection and tracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.

[46] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. The event-camera dataset and simulator : Event-based data for pose estimation, visual odometry, and SLAM. *The International Journal of Robotics Research*, 36(2) :142–149, 2017.

[47] F. Mueller, D. Mehta, O. Sotnychenko, S. Sridhar, D. Casas, and C. Theobalt. Real-time hand tracking under occlusion from an egocentric RGB-D sensor. In *IEEE International Conference on Computer Vision Workshops*, 2017.

[48] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion : Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*, pp. 127–136. IEEE, 2011.

[49] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion : Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality*, 2011.

[50] A. Nguyen, T.-T. Do, D. G. Caldwell, and N. G. Tsagarakis. Real-time 6-DOF pose relocalization for event cameras with stacked spatial LSTM networks. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

[51] M. Oberweger, P. Wohlhart, and V. Lepetit. Training a feedback loop for hand pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[52] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman. Hfirst : A temporal approach to object recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10) :2028–2040, 2015.

[53] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3) :133–148, Feb. 2013.

[54] H. Rebecq, D. Gehrig, and D. Scaramuzza. ESIM : an open event camera simulator. In *Conference on Robotics Learning*, 2018.

[55] H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. 2017.

[56] H. Rebecq, T. Horstschäfer, G. Gallego, and D. Scaramuzza. Evo : A geometric approach to event-based 6-DOF parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2) :593–600, 2016.

[57] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza. Events-to-video : Bringing modern computer vision to event cameras. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[58] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza. High speed and high dynamic range video with an event camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[59] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once : Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[60] C. Reinbacher, G. Munda, and T. Pock. Real-time panoramic tracking for event cameras. In *2017 IEEE International Conference on Computational Photography (ICCP)*, pp. 1–9. IEEE, 2017.

[61] C. Scheerlinck, H. Rebecq, D. Gehrig, N. Barnes, R. Mahony, and D. Scaramuzza. Fast image reconstruction with an event camera. In *IEEE Winter Conference on Applications of Computer Vision*, 2020.

[62] T. Stoffregen, G. Gallego, T. Drummond, L. Kleeman, and D. Scaramuzza. Event-based motion segmentation by motion compensation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7244–7253, 2019.

[63] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580. IEEE, 2012.

[64] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, et al. Scalability in perception for autonomous driving : Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2446–2454, 2020.

[65] D. J. Tan and S. Ilic. Multi-forest tracker : A chameleon in tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1202–1209, 2014.

[66] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Hybrid, frame and event based visual inertial odometry for robust, autonomous navigation of quadrotors. *arXiv preprint arXiv :1709.06310*, 2017.

[67] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Ultimate SLAM ? combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2) :994–1001, 2018.

[68] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt. Event-based 3d slam with a depth-augmented dynamic vision sensor. In *IEEE International Conference on Robotics and Automation*, 2014.

[69] D. Weikersdorfer, R. Hoffmann, and J. Conradt. Simultaneous localization and mapping for event-based vision systems. In *International Conference on Computer Vision Systems*, pp. 133–142. Springer, 2013.

[70] J. Xiao, A. Owens, and A. Torralba. Sun3D : A database of big spaces reconstructed using SfM and object labels. In *IEEE International Conference on Computer Vision*, 2013.

[71] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11) :1330–1334, 2000.

[72] A. Z. Zhu, Z. Wang, K. Khant, and K. Daniilidis. Eventgan : Leveraging large scale image datasets for event cameras. *arXiv preprint arXiv :1912.01584*, 2019.

[73] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis. Ev-flownet : Self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv :1802.06898*, 2018.

[74] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 989–997, 2019.