

ÉCOLE CENTRALE DE LILLE

PROJET D'INTÉGRATION

Suivi d'un grimpeur par caméra dome

RAPPORT DE PDI
25 mars 2025



CHEVALIER Romain

Tuteur : BOURDEAUD'HUY T.



Table des matières

1 Contexte	2
1.1 Objectifs	2
2 Architecture du système	2
2.1 Architecture matérielle	2
2.2 Architecture logicielle	3
3 Développement de l'application	4
3.1 Application système	4
3.1.1 Programme principal	4
3.1.2 Écriture mémoire	7
3.1.3 Détection de mouvement	9
3.1.4 Capture vidéo	12
3.2 Application web	15
3.2.1 Page web index (figure 5a)	15
3.2.2 Programme CGI : <code>video.cpp</code>	16
3.2.3 Programme CGI : <code>action.c</code>	17
3.2.4 Page web de visualisation des vidéos (figure 5b)	18
3.3 Problèmes rencontrés	18
3.3.1 Détection de mouvement	18
3.3.2 Enregistrement vidéo	19
3.3.3 Problèmes liés au matériel	20
3.3.4 Problèmes web	20
4 Tests et résultats	20
5 Perspectives d'amélioration	20
6 Conclusion	21
Annexes	23
A Github	23
B Driver vidéo	23
C Notice utilisation	24



1 Contexte

Afin d'améliorer les services offerts aux utilisateurs de la salle d'escalade *Sac à Pof* situé à Mons-en-Barœul (59), on envisage de mettre en place un système de suivi automatisé des grimpeurs à l'aide de caméras IP dômes motorisées. Ces caméras permettront de suivre les mouvements des grimpeurs sur les différentes voies en ajustant automatiquement le cadrage et le zoom en fonction de leur position. Le service sera accessible via une application web hébergée sur un serveur local. Cette solution vise à offrir un support d'analyse post-session, afin de permettre aux grimpeurs de comprendre leurs forces et faiblesses pour s'améliorer.

Afin de réaliser ce projet, nous sommes en possession de deux caméras IP dômes motorisées de la marque *AXIS* modèle *Q6304-E*¹, d'un routeur, d'une tablette Android et d'un raspberry pi 4.

1.1 Objectifs

Les objectifs de ce projet sont multiples et résumés dans la liste suivante :

- Prendre en main les caméras IP dômes motorisées et les intégrer dans un réseau local.
- Développer une interface web permettant de piloter les caméras, l'enregistrement de
- Avoir un retour visuel de la caméra en temps réel sur l'interface web.
- Développer une méthode de communication entre l'application web et les programmes de contrôle des caméras.
- Développer un algorithme de suivi de grimpeur en temps réel.
- Développer une méthode permettant d'enregistrer sur un stockage une vidéo.
- Avoir une application modulable permettant de modifier les caméras et les paramètres de détection.

2 Architecture du système

2.1 Architecture matérielle

Le système est composé de plusieurs éléments qui interagissent entre eux, la figure 1 montre l'architecture du système. Le système repose sur un **Raspberry Pi 4**, qui joue un rôle central en assurant la communication entre les différentes composantes. Il est connecté au réseau local via un **routeur**, qui agit comme un serveur DHCP, attribuant dynamiquement les adresses IP aux appareils du système.

Les **caméras Dome IP** sont accessibles via HTTP et envoient leur flux vidéo vers le Raspberry Pi lorsqu'elles sont activées. Ces caméras disposent de positions prédéfinies correspondant aux différentes voies d'escalade, mais peuvent également être orientées dynamiquement par l'utilisateur via la **tablette** fournie ou tout autre appareil connecté au réseau local.

Le Raspberry Pi exécute un serveur d'application web, ce serveur permet :

- de piloter les caméras (changement de position, orientation manuelle),
- de gérer les enregistrements vidéo,

1. Voir [fiche technique](#) pour plus d'informations



- de stocker dans une base de données les vidéos enregistrées ainsi que les positions des caméras définies par l'utilisateur.

L'enregistrement des vidéos est déclenché manuellement depuis l'interface de la tablette et les fichiers sont conservés jusqu'à suppression par l'utilisateur.

L'architecture ainsi mise en place assure une gestion centralisée et accessible du système, tout en permettant un contrôle précis et flexible des caméras et des enregistrements.

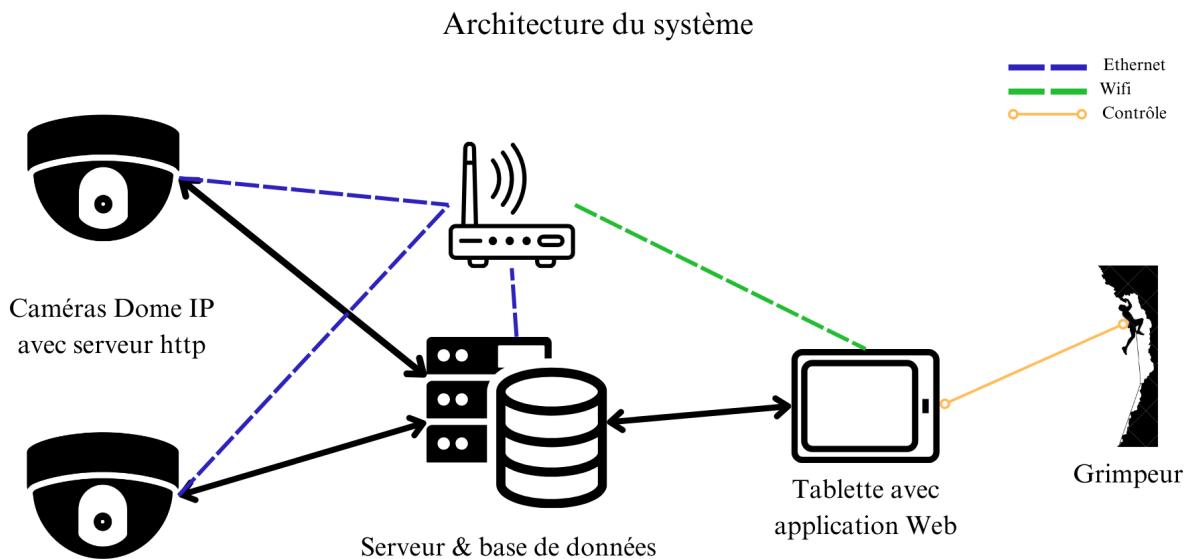


FIGURE 1 – Architecture du système

2.2 Architecture logicielle

L'architecture logicielle du système est représentée sur la figure 2. Le raspberry Pi divise ses tâches en deux parties distinctes : le serveur web et la partie gestion de l'application développée. En fonction des actions effectuées sur la page web, le serveur apache exécute deux scripts CGI (Common Gateway Interface) différents, le premier script, nommé **video.cgi** codé en **c++**, démarre au chargement de la page permet d'afficher le flux vidéo de la caméra sur la page web en lisant les images brutes dans la mémoire partagée et en les encodant en **JPEG**. Le second (**action.cgi** développé en **c**) permet de transmettre les ordres de déplacement de la caméra ou lancer un enregistrement vidéo via l'écriture de cet ordre dans un segment de mémoire partagé puis de réveillé le programme principal en lui envoyant un signal.

De l'autre côté, le programme principal (**main** programmé en **c**) est en charge de la gestion des caméras, de la détection de mouvement et de l'enregistrement des vidéos. Au démarrage, il crée un processus fils (**écriture mémoire**) qui est en charge de récupérer le flux vidéo du serveur HTTP de la caméra et de le stocker en image brute dans une mémoire partagée. Le programme principal est ensuite en attente d'un signal pour effectuer une action, comme le déplacement de la caméra, l'enregistrement de position ou encore le lancement d'un enregistrement qui va créer un processus fils (**enregistrement vidéo**) qui encode le flux video en MP4 afin de le stocker sur le disque dur et lance



la détection du grimpeur pour suivre ces mouvements à l'aide d'un second fils (**détection**). Ces deux processus récupèrent les images en lisant la mémoire partagée et les traitent pour effectuer leur tâche. Enfin, le programme de détection peut directement agir sur la position de la caméra en lui envoyant une requête HTTP.

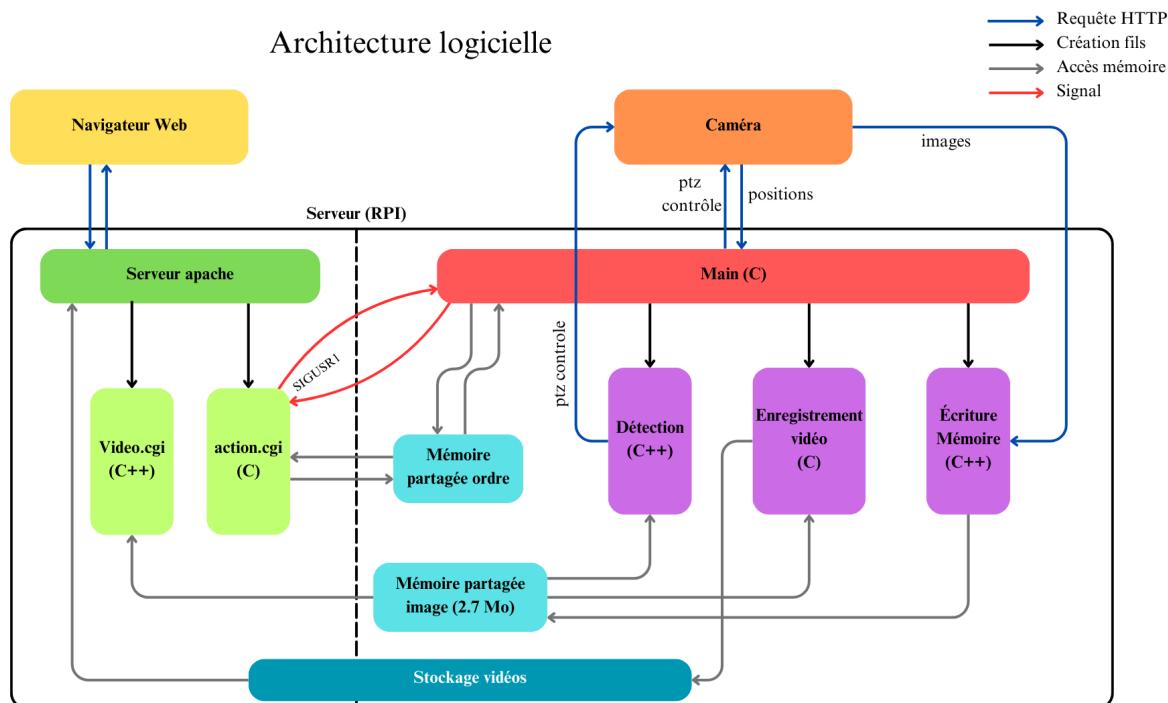


FIGURE 2 – Architecture logicielle

3 Développement de l'application

Dans cette partie, nous allons détailler les différentes étapes de développement de l'application pour chaque brique du système présent dans l'architecture logicielle en figure 2. Tout le code source est disponible sur le dépôt [GitHub](#).

3.1 Application système

La partie système de l'application est composée de plusieurs fichiers et programmes qui interagissent pour gérer les caméras, les enregistrements vidéo et la communication avec l'interface web.

3.1.1 Programme principal

Le fichier `main.c` constitue le cœur du système de gestion des caméras et de la communication inter-processus. Il est responsable de l'initialisation des caméras, de la gestion des enregistrements vidéo, de la détection des mouvements, et de l'interaction avec l'interface CGI via des segments de mémoire partagée et des sémaphores.

Initialisation du programme Lors de son démarrage, le programme effectue plusieurs étapes d'initialisation :



- **Gestionnaire de fin d'exécution** : La fonction `atexit(bye)` garantit que toutes les ressources (mémoire partagée, sémaphores, processus enfants) sont correctement libérées à la fin du programme.
- **Initialisation des sémaphores** : La fonction `initSemaphores()` crée des sémaphores nommés pour synchroniser les processus et gérer l'accès à la mémoire partagée, voir le paragraphe 3.1.2 pour leur fonctionnement.
- **Initialisation de la mémoire partagée** : La fonction `initSegmentMemoire()` configure deux segments de mémoire partagée : un pour les ordres envoyés par l'interface CGI et un autre pour stocker les images brutes capturées par les caméras.
- **Initialisation des caméras** : La fonction `initCamera()` charge les paramètres des caméras depuis un fichier JSON et configure les caméras actives. Afin que le système puisse s'adapter à l'orientation de la caméra (portrait, paysage, portrait renversé ou paysage renversé), la fonction `setParamCamera` vient définir les paramètre de la caméra pour que le système puisse s'adapter automatiquement. En effet, on definit le mouvement d'inclinaison et le mouvement panoramique en fonction de l'orientation (par exemple, pour le mode paysage un déplacement vers le haut correspond à une augmentation de l'angle d'inclinaison et le mode portrait renverse correspond à une diminution de l'angle panoramique).

Gestion des signaux Le programme installe un gestionnaire de signaux (`signalHandler`) pour répondre aux événements suivants :

- **SIGINT** : Interruption manuelle (Ctrl+C) pour arrêter le programme proprement.
- **SIGCHLD** : Gestion des processus enfants terminés pour éviter les processus zombies.
- **SIGUSR1** : Signal utilisé pour indiquer qu'un nouvel ordre a été reçu via la mémoire partagée.

Boucle principale Le programme entre dans une boucle infinie en appelant `pause()`, ce qui le met en attente d'un signal. Lorsqu'un signal est reçu, le gestionnaire de signaux (`signalHandler`) est exécuté pour traiter l'événement.

Gestion des ordres Lorsqu'un signal `SIGUSR1` est reçu, la fonction `gestionOrdres()` est appelée pour lire et exécuter les ordres envoyés par l'interface CGI. Les types d'ordres gérés incluent :

- **Déplacement de la caméra** : Commandes de mouvement (`move`) ou de zoom (`zoom`). Le programme reçoit une direction (haut, bas, gauche, droite) et une valeur de précision pour ajuster l'angle de déplacement (entre 0 et 30 degrés). Il execute ensuite la fonction `requetePTZ` qui crée une requête HTTP avec l'aide de la librairie `curl` pour envoyer l'ordre de déplacement à la caméra.
- **Capture vidéo** : Commandes de capture (`reco`) pour démarrer ou arrêter un processus de capture vidéo. Cet ordre execute la fonction `enregistrerVideo` qui s'assure que le stockage disponible est suffisant, puis crée le nom de fichier en fonction du nom et prénom du grimpeur, de la voie et de la date et heure de l'enregistrement. Enfin, il crée un processus fils pour démarrer l'enregistrement. Voir section 3.1.4 pour le détail de fonctionnement de la capture vidéo à l'aide du programme `ffmpeg`.



- **Détection de mouvement** : Commandes de détection (`detc`) pour analyser les images et suivre les mouvements. Cet ordre fait appel à la fonction `processusDetection` qui crée un processus fils pour effectuer la détection de mouvement. Voir section 3.1.3 pour le détail de fonctionnement de la détection de mouvement.
- **Enregistrement vidéo** : Commandes d'enregistrement (`enrg`) pour démarrer ou arrêter un processus d'enregistrement vidéo. Ce ordre vient lancer un processus de capture vidéo et de détection de mouvement.
- **Gestion des positions prédefinies** : Commandes (`rout`) pour ajouter, supprimer ou afficher des positions prédefinies, voir le paragraphe suivant pour plus d'informations.
- **Changement de caméra active** : Commandes (`cam`) pour basculer entre les caméras configurées.

Gestion des positions prédefinies Le fichier `positions.c` est responsable de la gestion des positions des caméras PTZ et des voies associées. Il permet d'ajouter, de supprimer et d'afficher des positions prédefinies en interagissant avec un fichier JSON pour stocker les données. Les fonctions principales de ce fichier sont `addRoute`, `removeRoute` et `showRoute`.

Ajout d'une route : `addRoute` La fonction `addRoute` permet d'ajouter une nouvelle position PTZ associée à une voie dans le fichier JSON. Voici les étapes principales de son fonctionnement :

1. **Récupération de la position actuelle de la caméra** : La fonction `recupererPosition` est appelée pour obtenir les valeurs actuelles de `pan`, `tilt` et `zoom` de la caméra active via une requête HTTP. Pour cela, une URL est construite avec les informations de la caméra et envoyée à l'aide de la bibliothèque `libcurl` permettant d'afficher sur une page HTML les données de la caméra. Enfin, les données retournées par la caméra sont analysées pour extraire les valeurs de `pan`, `tilt` et `zoom`.
2. **Ajout de la position dans le fichier JSON** : La fonction `addPositionFile` est appelée pour insérer les valeurs récupérées dans le fichier JSON. Si le fichier JSON n'existe pas, il est créé. Une nouvelle entrée est ajoutée sous la forme d'un tableau contenant les valeurs de `pan`, `tilt`, `zoom` et l'identifiant de la caméra.

Suppression d'une route : `removeRoute` La fonction `removeRoute` permet de supprimer une position PTZ associée à une voie dans le fichier JSON. Voici les étapes principales de son fonctionnement :

1. **Chargement du fichier JSON** : Le fichier JSON contenant les positions PTZ est chargé en mémoire à l'aide de la bibliothèque `jansson`.
2. **Suppression de la clé associée à la voie** : La clé correspondant à la voie spécifiée est supprimée du fichier JSON à l'aide de la fonction `json_object_del`.
3. **Sauvegarde du fichier JSON modifié** : Le fichier JSON est mis à jour et sauvegardé sur le disque.



Affichage d'une route : `showRoute` La fonction `showRoute` permet d'afficher les informations d'une position PTZ associée à une voie et de déplacer la caméra vers cette position. Voici les étapes principales de son fonctionnement :

1. **Chargement du fichier JSON** : Le fichier JSON contenant les positions PTZ est chargé en mémoire à l'aide de la bibliothèque `jansson`.
2. **Récupération des données de la voie** : Les données associées à la voie spécifiée sont extraites du fichier JSON. Ces données incluent les valeurs de `pan`, `tilt`, `zoom` et l'identifiant de la caméra. Si la voie n'existe pas ou si les données sont invalides, un message d'erreur est affiché, et la fonction retourne `-1`.
3. **Déplacement de la caméra** : Si la caméra active n'est pas celle spécifiée dans les données, la fonction `setActiveCamera` est appelée pour activer la caméra correspondante. La fonction `allerPosition` est appelée pour envoyer une requête HTTP à la caméra afin de la déplacer vers la position spécifiée.

Libération des ressources Lorsque le programme se termine, la fonction `bye()` est appelée pour libérer toutes les ressources :

- Les processus enfants encore actifs sont arrêtés.
- Les segments de mémoire partagée sont détachés et supprimés.
- Les sémaphores sont fermés et supprimés.

3.1.2 Écriture mémoire

Le fichier `ecritureMemoire.cpp` est un processus dédié à la capture des images vidéo depuis une source donnée (on exécute le programme avec comme argument l'url de vidéo) et à leur écriture dans un segment de mémoire partagée. Ce processus est essentiel pour permettre la communication inter-processus entre les différents modules du système.

Initialisation du programme Lors de son démarrage, le programme effectue plusieurs étapes d'initialisation :

- **Gestionnaire de signaux** : Le programme installe un gestionnaire de signaux (`signalHandler`) pour gérer les interruptions (`SIGINT`) et les arrêts (`SIGTERM`) de manière propre.
- **Ouverture des sémaphores** : Les sémaphores nommés sont ouverts pour synchroniser l'accès à la mémoire partagée entre les processus. Voir la figure 3 pour comprendre l'enchaînement des différents sémaphores et voir ci-dessous pour leur description :
 - `SEM_READERS` : Compte les lecteurs actifs.
 - `SEM_WRITER` : Contrôle l'accès en écriture à la mémoire partagée.
 - `SEM_NEW_FRAME` : Indique qu'une nouvelle image est disponible.
 - `SEM_ACTIVE_READERS` : Compte les lecteurs actifs qui attendent une nouvelle image.
 - `SEM_MUTEX` : Protège l'accès aux compteurs de sémaphores.
- **Ouverture de la mémoire partagée** : Le segment de mémoire partagée est ouvert et mappé en mémoire virtuelle pour permettre l'écriture des images capturées.

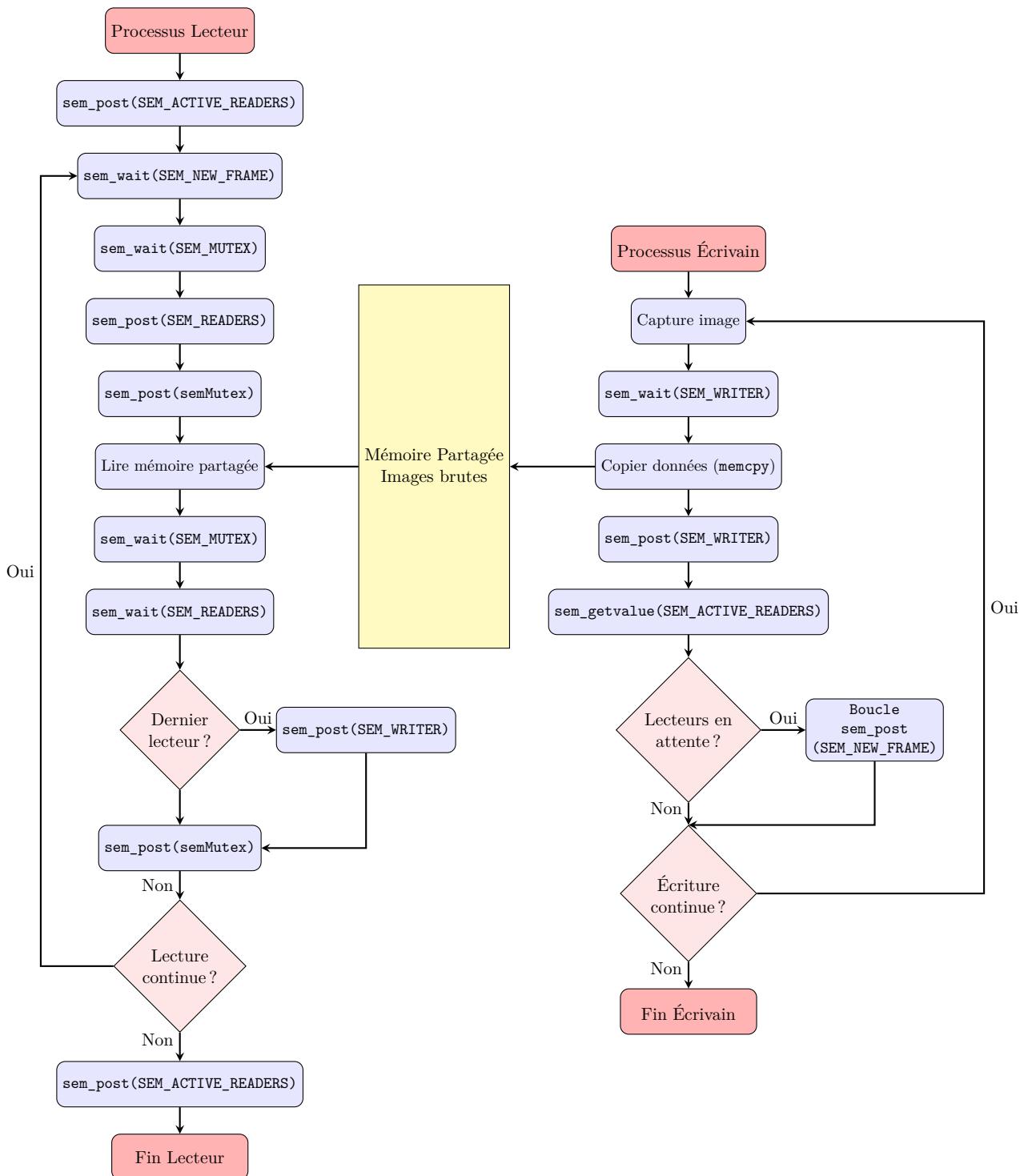


FIGURE 3 – Synchronisation des lecteurs et écrivains avec sémaphores

Capture vidéo Le programme utilise OpenCV pour capturer les images vidéo depuis la source spécifiée en argument :

- **Initialisation de la capture vidéo** : La capture est ouverte avec `cv::VideoCapture` en sur l'URL de l'argument passé au programme. Les propriétés de la capture sont configurées pour être sur les même paramètres que la source vidéo.



- **Vérification de la capture** : Si la capture ne peut pas être ouverte, le programme affiche une erreur et se termine.

Boucle principale Le programme entre dans une boucle principale pour capturer les images et les écrire dans la mémoire partagée :

- **Capture d'une image** : Une image est capturée à l'aide de `cap.read(frame)`. Si la capture échoue, le programme affiche une erreur et quitte la boucle.
- **Écriture dans la mémoire partagée** : Le sémaphore `SEM_WRITER` est utilisé pour bloquer l'accès en écriture pendant que l'image est copiée dans la mémoire partagée. Les données de l'image sont copiées avec `std::memcpy`.
- **Notification des lecteurs** : Le programme utilise le sémaphore `SEM_NEW_FRAME` pour notifier tous les lecteurs actifs (`SEM_ACTIVE_READERS`) qu'une nouvelle image est disponible.

Libération des ressources Lorsque le programme se termine (par un signal ou une erreur), il libère toutes les ressources utilisées :

- **Libération de la mémoire partagée** : Le segment de mémoire partagée est détaché (`munmap`) et fermé (`close`).
- **Fermeture des sémaphores** : Tous les sémaphores ouverts sont fermés avec `sem_close`.
- **Libération de la capture vidéo** : La capture vidéo est libérée avec `cap.release`.

Gestion des signaux Le gestionnaire de signaux (`signalHandler`) permet de gérer les interruptions et les arrêts du programme :

- **SIGINT** : Interrompt immédiatement le programme.
- **SIGTERM** : Arrête proprement le programme en mettant fin à la boucle principale.

3.1.3 Détection de mouvement

Le fichier `detection.cpp` est un programme dédié à la détection de mouvements dans un flux vidéo et au contrôle des caméras PTZ en fonction des mouvements détectés. Il utilise OpenCV pour le traitement des images, des sémaphores pour la synchronisation entre processus, et la mémoire partagée pour accéder aux images brutes.

Initialisation du programme Lors de son démarrage, le programme effectue plusieurs étapes d'initialisation :

- **Gestionnaire de signaux :**
 - Le programme installe un gestionnaire de signaux (`signalHandler`) pour gérer les interruptions (`SIGINT`) et les arrêts (`SIGTERM`) de manière propre.
 - En cas de réception d'un signal, le programme termine la détection en cours et libère les ressources.
- **Chargement des paramètres de la caméra active :**



- Les paramètres de la caméra active (IP, orientation, dimensions, commandes PTZ, etc.) sont chargés depuis un fichier JSON, que le programme principal met à jour à chaque changement de caméra.
- Récupération du zoom actuel de la caméra pour ajuster les commandes PTZ en fonction du niveau de zoom, via une requête HTTP.
- **Chargement des paramètres de détection** : Les paramètres de détection (seuils verticaux et horizontaux, coefficients de flou gaussien, etc.) sont également chargés depuis un fichier JSON, nommé **PATH_PARAM_DETECTION**. Cela permet de modifier facilement les paramètres de détection sans recompiler le programme. Le détail des paramètres dont une description est présentée ci contre :
 - **framesBetweenReferences** : Nombre d'images entre deux actualisations de l'image de référence utilisée pour la détection de mouvement.
 - **verticalThreshold** : Seuil vertical utilisé pour détecter les mouvements dans la zone de détection. Définit la fraction de la hauteur de la zone de détection où un mouvement est considéré comme un déplacement vertical (haut seulement).
 - **horizontalThreshold** : Seuil horizontal utilisé pour détecter les mouvements dans la zone de détection. Définit la fraction de la largeur de la zone de détection où un mouvement est considéré comme un déplacement horizontal (gauche ou droite).
 - **coefAverageMovingFilter** : Coefficient du filtre de moyenne glissante appliqué au barycentre des mouvements détectés.
 - **coefGaussianBlur** : Taille du noyau utilisé pour appliquer un flou gaussien aux images.
 - **heightResizeRatio** et **widthResizeRatio** : Facteurs de redimensionnement pour réduire la taille des images avant le traitement.
 - **nbMoveBeforeChangeDetectionArea** : Nombre de mouvements verticaux nécessaires avant de modifier la position de la zone de détection.
 - **cropRatioDetectionAreaLandscape** et **cropRatioDetectionAreaPortrait** : Ratios utilisés pour définir la taille et la position de la zone de détection en fonction de l'orientation de la caméra (paysage ou portrait). Les zones de detection diffèrent en fonction de l'orientation de l'image pour assurer une detection optimale.
 - **numberFrameBetweenMove** : Nombre minimum d'images entre deux mouvements consécutifs de la caméra. Empêche la caméra de se déplacer trop fréquemment, ce qui pourrait entraîner des oscillations.
 - **numberFrameWithoutMove** : Nombre d'images sans mouvement détecté avant d'arrêter automatiquement la détection.
 - **increasePTZ** : Facteur d'augmentation utilisé pour ajuster les commandes PTZ en fonction du zoom actuel de la caméra. Permet de compenser les mouvements en fonction du niveau de zoom pour maintenir une précision constante.
- **Ouverture des sémaphores et de la mémoire partagée** :
 - Les sémaphores nécessaires à la synchronisation entre processus sont ouverts.
 - Le segment de mémoire partagée contenant les images brutes est ouvert et mappé en mémoire virtuelle.



Boucle principale Le programme entre dans une boucle principale pour analyser les images et détecter les mouvements :

- **Lecture des images depuis la mémoire partagée** : Les images brutes sont lues depuis la mémoire partagée en utilisant les sémaphores pour garantir une synchronisation correcte.
- **Prétraitement des images** :
 - Les images sont redimensionnées, leur taille est divisé par deux pour accélérer le traitement.
 - Ensuite l'image est converties en niveaux de gris
 - Puis, on applique une égalisation d'histogramme à l'image en niveaux de gris. Afin d'améliorer le contraste de l'image en redistribuant les intensités des pixels.
 - Enfin, un flou gaussien est appliqué pour réduire le bruit dus à de faible mouvement, comme par exemple la corde qui bouge. Voir les images en noir et blanc sur la figure 4.
- **Détection des mouvements** :
 - Une image de référence est utilisée pour calculer la différence absolue avec l'image actuelle. L'image de référence est actualisée toutes les 5 images, ou dès lors que la caméra est déplacée.
 - Un seuil binaire est appliqué pour identifier les zones de mouvement. L'image est donc devenu une image binaire avec des pixels à 0 pour les zones sans mouvement et à 255 pour les zones avec mouvement.
 - Une zone de détection est définie pour limiter l'analyse à une partie spécifique de l'image. Au démarrage, la zone de détection est sur le haut de l'image, voir le rectangle vert figure 4a. Après que la caméra se soit déplacée vers le haut, la zone de détection est déplacée vers le bas, voir figure 4b. Après, la zone de détection reste fixe pour tout le reste du suivi du grimpeur, voir figure 4c.

Il est nécessaire de changer la zone de détection car en bas d'une voie, l'algorithme de détection ne sait pas faire la distinction entre le grimpeur et l'assureur. Donc on regarde que le haut de l'écran au début avant de recentrer la zone de détection sur le centre de l'image.

 - Les moments de la zone de détection sont calculés pour déterminer le barycentre du mouvement. On applique un filtre de moyenne glissante de coefficient 0.4 pour lisser les mouvements.
- **Analyse des mouvements** :
 - Si le barycentre se trouve dans une zone spécifique, un mouvement est détecté :
 - * **Mouvement vertical** : Si le barycentre est dans la partie supérieure la caméra est déplacée vers le haut, voir lignes rouges horizontales sur les figures 4. La caméra est déplacée vers le haut si le barycentre se trouve entre la ligne rouge et la ligne verte du haut de l'image.
 - * **Mouvement horizontal** : Si le barycentre est dans la partie gauche ou droite de la zone de détection, la caméra est déplacée vers la gauche ou la droite. voir lignes rouges verticales sur les figures 4.
 - Les commandes PTZ sont envoyées à la caméra via des requêtes HTTP, comme pour le programme principal.
- **Arrêt de la détection** : Si aucun mouvement n'est détecté pendant un certain nombre de images, la détection s'arrête automatiquement.

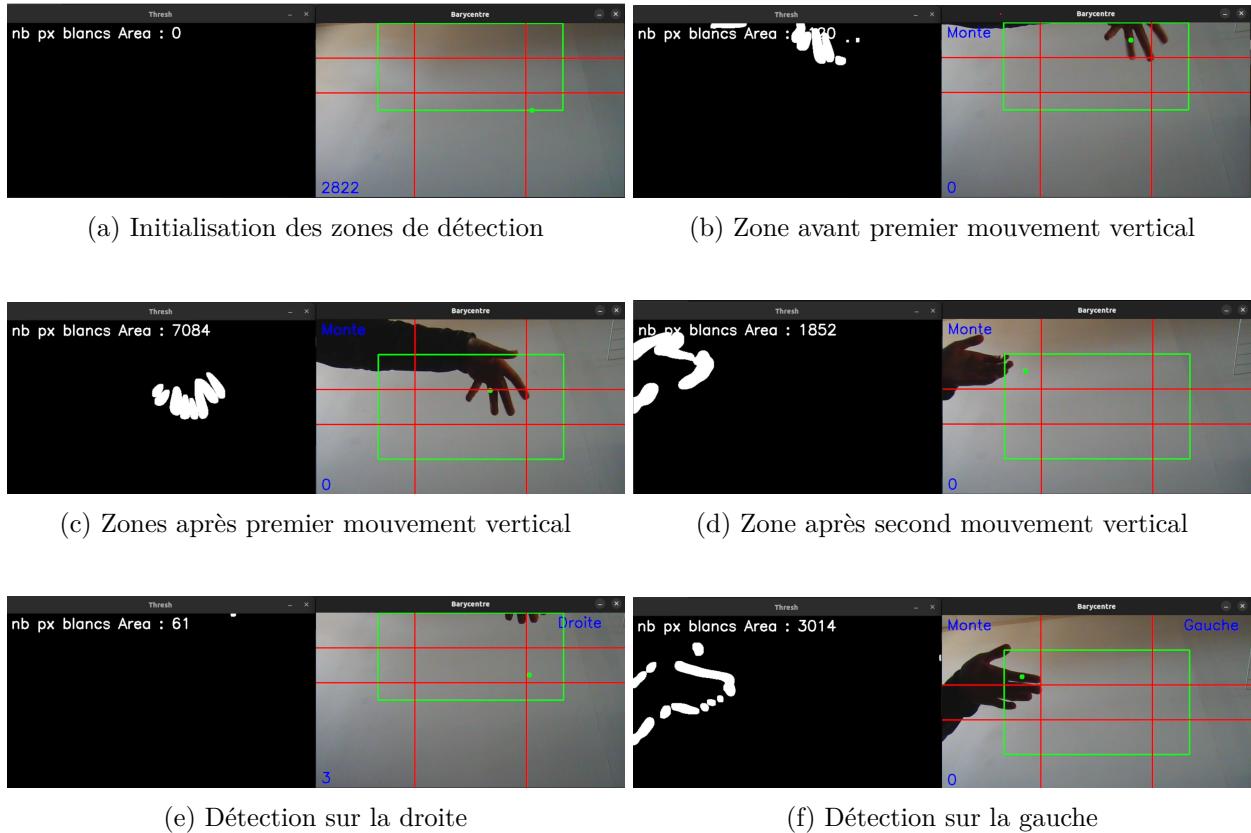


FIGURE 4 – Zones de détection de mouvement

Libération des ressources À la fin du programme, toutes les ressources utilisées sont libérées :

- **Fermeture des sémaphores** : Tous les sémaphores ouverts sont fermés avec `sem_close`.
- **Libération de la mémoire partagée** : Le segment de mémoire partagée est détaché (`munmap`) et fermé (`close`).
- **Fermeture des fenêtres OpenCV** : Toutes les fenêtres créées pour l'affichage des résultats sont fermées avec `cv::destroyAllWindows`.

Gestion des signaux Le gestionnaire de signaux (`signalHandler`) permet de gérer les interruptions et les arrêts du programme :

- **SIGTERM** : Arrête proprement le programme en mettant fin à la boucle principale.
- **SIGINT** : Interrompt immédiatement le programme.

3.1.4 Capture vidéo

Le fichier `enregistrementVideo.c` est un programme dédié à l'enregistrement des flux vidéo provenant de la mémoire partagée. Il utilise des sémaphores pour la synchronisation entre processus, la mémoire partagée pour l'accès aux images brutes, et FFmpeg² pour encoder les vidéos dans un format

2. FFmpeg est une suite logicielle open-source puissante et polyvalente utilisée pour manipuler des fichiers multimédias (audio et vidéo). Elle permet de réaliser des tâches telles que la conversion de formats, l'encodage, le décodage, le montage, le streaming, et bien plus encore. Pour plus d'information voir [la documentation en ligne](#)



compressé. Ce programme est conçu pour fonctionner de manière autonome et s'arrêter proprement en cas de signal d'interruption.

Initialisation du programme Lors de son démarrage, le programme effectue plusieurs étapes d'initialisation :

- **Gestionnaire de signaux :**

- Le programme installe un gestionnaire de signaux (`signalHandler`) pour gérer les interruptions (`SIGINT`) et les arrêts (`SIGTERM`) de manière propre.
- En cas de réception d'un signal, le programme termine l'enregistrement en cours et libère les ressources.

- **Ouverture des sémaphores :**

- Les sémaphores nommés (`SEM_READERS`, `SEM_WRITER`, `SEM_NEW_FRAME`, etc.) sont ouverts pour synchroniser l'accès à la mémoire partagée entre les processus.
- Le compteur de lecteurs actifs (`SEM_ACTIVE_READERS`) est incrémenté pour indiquer qu'un nouveau lecteur est actif.

- **Ouverture de la mémoire partagée :**

- Le segment de mémoire partagée contenant les images brutes est ouvert et mappé en mémoire virtuelle pour permettre la lecture des images.

- **Préparation de la commande FFmpeg** : Afin d'optimiser l'utilisation de FFmpeg, on utilise sur le raspberry l'accélération matériel pour l'encodage vidéo. Pour cela, on utilise l'encoder `v4l2m2m` qui permet d'utiliser le codec H.264 pour l'encodage vidéo. La commande FFmpeg est construite pour encoder les images brutes en vidéo compressée.

- `-f rawvideo` : Spécifie que l'entrée est une vidéo brute.
- `fflags +discardcorrupt+genpts` : Ignore les images corrompues et génère des horodatages.
- `-pixel_format bgr24` : Définit le format des pixels en BGR 24 bits.
- `-s 1280x720` : Définit la résolution d'entrée.
- `-r 25` : Définit le débit d'images d'entrée (25 FPS).
- `-framerate 25` : Force le débit d'image sortant (25 FPS).
- `-c:v h264_v4l2m2m` : Utilise le codec H.264 pour l'encodage et l'accélération matérielle sur le rpi. Sur une architecture x86, on peut utiliser `-c:v libx264` pour l'encodage.
- `-pix_fmt yuv420p` : Définit le format des pixels en YUV 4 :2 :0. Ce format est optimisé pour lire les vidéos sur navigateur web.
- `-b:v 5M` : Définit le débit binaire de sortie à 5 Mbit/s (Meilleur débit pour la 720p).
- `-an` : Désactive l'audio.



Boucle principale Le programme entre dans une boucle principale pour capturer les images et les écrire dans le pipe standard de FFmpeg :

- **Attente d'une nouvelle image :**
 - Le programme attend qu'une nouvelle image soit disponible en utilisant le sémaphore SEM_NEW_FRAME.
- **Synchronisation avec les sémaphores :**
 - Le sémaphore SEM_WRITER est utilisé pour bloquer l'accès en écriture pendant que l'image est lue.
 - Le sémaphore SEM_READERS est utilisé pour indiquer qu'un nouveau lecteur est en cours de lecture.
- **Écriture dans le pipe FFmpeg :**
 - L'image brute est lue depuis la mémoire partagée et écrite dans le pipe standard de FFmpeg à l'aide de la fonction `fwrite`. L'image est encodé dans une vidéo temporaire en attendant la fin de l'enregistrement.

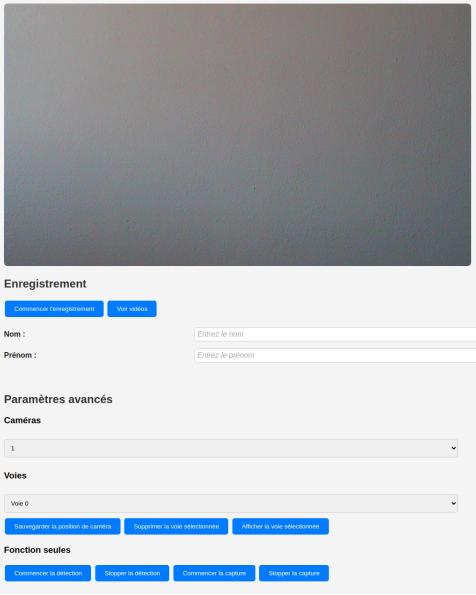
Découpage de la vidéo Une fois l'enregistrement terminé, la fonction `découperVideo` est appelée pour découper la vidéo et ne conserver que la partie intéressante :

- **Lecture des images de début et de fin :**
 - Les images de début et de fin sont lues depuis un fichier (PATH_FRAMES), qui sont écrites à la fin du processus de détection de mouvement.
 - Si le fichier n'existe pas, des valeurs par défaut sont utilisées.
- **Ajustement des images :**
 - Les images sont ajustées en fonction des paramètres définis, comme le nombre de images avant le premier mouvement ou après l'absence de mouvement afin de garder une partie de la video juste avant le premier mouvement de la caméra.
- **Exécution de la commande FFmpeg :**
 - Une commande FFmpeg est construite pour découper la vidéo en fonction des images ajustées et si besoin la réorienter dans le bon sens en fonction de l'orientation de la caméra.
 - La vidéo temporaire (VIDEO_TEMP) est utilisée comme entrée, et la vidéo finale est sauvegardée avec un nom personnalisé construit par le programme principal.

Libération des ressources À la fin du programme, toutes les ressources utilisées sont libérées :

- **Fermeture des sémaphores** : Tous les sémaphores ouverts sont fermés avec `sem_close`.
- **Libération de la mémoire partagée** : Le segment de mémoire partagée est détaché (`munmap`) et fermé (`close`).
- **Fermeture du pipe FFmpeg** Le pipe standard vers FFmpeg est fermé avec `pclose`.





Enregistrement

[Commencer l'enregistrement](#) [Voir les vidéos](#)

Nom : Entrer le nom

Prénom : Entrer le prénom

Paramètres avancés

Caméras

1

Voies

Voie 0

[Sauvegarder la position de caméra](#) [Supprimer la voie sélectionnée](#) [Afficher la voie sélectionnée](#)

Fonction seules

[Commencer la détection](#) [Stopper la détection](#) [Commencer la capture](#) [Stopper la capture](#)

Liste des vidéos

Réchercher par nom, prénom, voie, date ou heures...

Nom	Prénom	Voie	Date	Heure	Lien
essai	1	1	09/01/2025	21h19	Voir Visionner Supprimer
2	2	1	09/01/2025	21h43	Voir Visionner Supprimer
2	2	1	09/01/2025	21h48	Voir Visionner Supprimer
2	2	2	09/01/2025	21h48	Voir Visionner Supprimer
2	2	1	09/01/2025	21h50	Voir Visionner Supprimer
2	3	1	09/01/2025	21h54	Voir Visionner Supprimer
essai1	a	1	06/02/2025	20h39	Voir Visionner Supprimer
test	envers	1	06/02/2025	21h08	Voir Visionner Supprimer
essaieInvers	5	1	06/02/2025	21h55	Voir Visionner Supprimer
essaieInvers	5	1	06/02/2025	22h00	Voir Visionner Supprimer

(a) Index de l'application
(b) Page web de visualisation des vidéos

FIGURE 5 – Pages web de l'application

Gestion des signaux Le gestionnaire de signaux (`signalHandler`) permet de gérer les interruptions et les arrêts du programme :

- **SIGTERM** : Arrête proprement le programme en mettant fin à la boucle principale.
- **SIGINT** : Interrompt immédiatement le programme.

3.2 Application web

Le serveur web est composée de deux pages web voir figure 5, la première permet de visualiser le flux vidéo de la caméra, de la piloter et de piloter l'application. La seconde permet de visualiser les vidéos enregistrées et de les télécharger, le code de ces pages est écrit en HTML, CSS et JAVASCRIPT peut-être consulté sur la page [GitHub](#).

3.2.1 Page web index (figure 5a)

Au chargement de cette page, une fonction `javascript` permet de charger les voies disponibles dans la base de données et de les ajouter dans une liste déroulante pour permettre à l'utilisateur de sélectionner la voie qu'il souhaite visualiser. Une seconde fonction `js` charge les caméras disponibles et les ajoute dans une seconde liste déroulante. Puis le script `video.cgi` est exécuté pour afficher le flux vidéo de la caméra sélectionnée, voir la sous section 3.2.2 pour plus d'information.

Sur la gauche de cette page, on retrouve tout les boutons de contrôle de la caméra (haut, bas, gauche et droite) associés à un curseur pour la précision de l'angle afin de permettre un déplacement plus précis. On retrouve un second curseur pour gérer le zoom de la caméra entre 1x et 18x. La pression des boutons créent une requête **AJAX** pour exécuter le script `action.cgi`, voir la sous section 3.2.3 pour plus d'information.

En dessous de ces curseurs, il y a une liste déroulante pour choisir la voie que l'on souhaite observer. À la selection d'une voie on vient changer de caméra si besoin et la positionner au bon endroit.

En dessous de la video, on retrouve un bouton pour enregistrer une vidéo avec deux champ pour entre le nom et prenom du grimpeur, un exemple de requête **AJAX** est présenté ci-contre :



`http://localhost/cgi-bin/action.cgi?enrg=on&nom=Chevalier&prenom=Romain&voie=0`. On récupère le nom et la voie afin de produire un nom de fichier unique et pouvoir le récupérer facilement dans la page de visualisation des vidéos.

Paramètres avancés En bas de page, on trouve les options pour configurer une nouvelle voie. Il y a une liste déroulante pour choisir la caméra, puis une seconde liste déroulante pour choisir une voie que l'on souhaite supprimer ou afficher sur l'appui du bouton adéquat. Il y a également un bouton pour sauvegarder une nouvelle position de la caméra dans la base de données en affichant une fenêtre popup pour demander le numéro de voie. En addition, on peut également lancer ou stop un enregistrement seul sans suivi du grimpeur et à l'inverse, on peut lancer seulement et stopper la détection du grimpeur sans enregistrement.

3.2.2 Programme CGI : video.cpp

Le fichier `video.cpp` compilé en tant que programme CGI permet de diffuser le flux vidéo en continu au format MJPEG (Motion JPEG) via HTTP. Il lit les images depuis le segment de mémoire partagée, les encode au format JPEG, et les envoie au client sous forme de flux MJPEG. Le programme utilise des sémaphores pour la synchronisation entre processus et gère les signaux pour une terminaison propre.

Initialisation du programme Lors de son démarrage, le programme effectue plusieurs étapes d'initialisation :

- **Chargement des paramètres de la caméra** Les dimensions (`width, height`) et l'orientation (`orientation`) de l'image sont récupérées depuis un fichier JSON (`PATH_CAMERA_ACTIVE`). Ce fichier est mis à jour par le programme principal à chaque changement de caméra.
- **Installation des gestionnaires de signaux** :
- **Ouverture des sémaphores et de la mémoire partagée** :
- **Préparation du flux HTTP** : Le programme envoie un en-tête HTTP indiquant que le contenu est un flux MJPEG (`Content-Type: multipart/x-mixed-replace`). Cela permet de diffuser les images en continu sans interruption.

Boucle principale Le programme entre dans une boucle principale pour diffuser les images en continu :

- **Attente d'une nouvelle image** : Le programme attend qu'une nouvelle image soit disponible en utilisant le sémaphore `SEM_NEW_FRAME`.
- **Lecture de l'image depuis la mémoire partagée** : L'image brute est lue depuis la mémoire partagée et stockée dans une matrice OpenCV (`cv::Mat`).
- **Pré-traitement de l'image** :
 - Si nécessaire, l'image est tournée en fonction de l'orientation de la caméra (`cv::rotate`).
 - L'image est convertie du format *BGR* (*blue green red*) au format *textit{RGB}* (*red green blue*) à l'aide (`cv::cvtColor`) car le JPEG nécessite un profil colorimétrique en RGB.
- **Encodage en JPEG** : L'image est encodée au format JPEG à l'aide de la fonction `encode_jpeg`, avec la librairie `libjpeg`. La qualité de l'encodage est fixée à 75 pour un bon compromis entre qualité et taille.



- **Envoi de l'image au client :** L'image encodée est envoyée au client avec un en-tête indiquant la taille et le type de contenu (`Content-Type: image/jpeg`). Il envoie les images au navigateur en écrivant les en-têtes appropriés et les images encodées sur la sortie standard.

Libération des ressources À la fin du programme, toutes les ressources utilisées sont libérées :

- **Mémoire partagée** : Le segment de mémoire partagée est détaché (`munmap`) et fermé (`close`).
- **Sémaphores** : Tous les sémaphores ouverts sont fermés avec `sem_close`.
- **Tampon JPEG** : Le tampon mémoire utilisé pour stocker les images encodées est libéré.

3.2.3 Programme CGI : `action.c`

Le fichier `action.c` est un programme CGI utilisé pour gérer les interactions entre le serveur web Apache et le programme principal. Il reçoit les requêtes des utilisateurs via l'interface web, les traite, et communique avec le programme principal à l'aide de la mémoire partagée et des signaux.

Initialisation du programme Lors de son démarrage, le programme effectue plusieurs étapes d'initialisation :

- **Gestionnaire de fin d'exécution** : La fonction `atexit(bye)` garantit que toutes les ressources (mémoire partagée, sémaphores) sont correctement libérées à la fin du programme.
- **Installation des gestionnaires de signaux** :
 - **SIGUSR1** : Utilisé pour recevoir une notification du programme principal indiquant que l'ordre a été traité.
 - **SIGALRM** : Utilisé pour gérer un délai d'attente (timeout) si aucune réponse n'est reçue dans un délai de 5 secondes.

Traitement des requêtes Le programme traite les requêtes CGI comme suit :

- **Lecture de la requête** :
 - Les paramètres de la requête sont récupérés depuis de la variable `QUERY_STRING`. Cette variable d'environnement contient la chaîne de requête envoyée par le client dans l'URL après le point d'interrogation (?).
 - Si aucun paramètre n'est reçu, le programme retourne une erreur HTTP.
- **Écriture dans la mémoire partagée** :
 - Le programme ouvre un segment de mémoire partagée nommé `SHM_ORDRE`.
 - Il écrit l'ordre reçu dans ce segment, en incluant le PID du processus CGI et les paramètres de la requête. Afin que le programme puisse répondre à ce processus en retournant un signal à la fin de l'exécution de l'ordre.
- **Envoi d'un signal au programme principal** :
 - Le programme principal est identifié à l'aide de son PID, lu depuis un fichier (`PATH_FPID`).
 - Un signal `SIGUSR1` est envoyé au programme principal pour notifier l'arrivée d'un nouvel ordre.



- **Attente de la réponse :**

- Le programme attend une réponse du programme principal en utilisant la fonction `pause()` avec le signal `SIGUSR1`.
- Si aucune réponse n'est reçue dans un délai de 5 secondes, un signal `SIGALRM` est déclenché pour retourner un message d'erreur HTTP.

Retour HTTP Une fois la réponse reçue, le programme retourne un message HTTP au client en fonction du statut lu dans la mémoire partagée :

- **Succès (`status == 0`)** : Retourne un message JSON indiquant que l'action a été exécutée avec succès.
- **Erreur disque (`status == -2`)** : Retourne un message JSON indiquant que l'espace disque est insuffisant, code spécial pour le script de capture vidéo.
- **Erreur générale** : Retourne un message JSON indiquant qu'une erreur est survenue.

Libération des ressources À la fin du programme, toutes les ressources utilisées sont libérées :

- **Mémoire partagée** : Le segment de mémoire partagée est détaché (`munmap`) et fermé (`close`).
- **Gestion des signaux** : Les gestionnaires de signaux sont désinstallés automatiquement à la fin du programme.

3.2.4 Page web de visualisation des vidéos (figure 5b)

Cette page permet de visualiser les vidéos enregistrées, de les télécharger et de les supprimer. Elles sont répertoriées dans un tableau avec le nom du grimpeur, la voie, la date et l'heure de l'enregistrement. On peut, de plus, rechercher une vidéo particulière à l'aide de la barre de recherche en haut de page. Avec les fonctions du navigateur, on peut lire, mettre en pause, accélérer, ralentir ou télécharger la vidéo. Enfin, on peut voir la vidéo en haut de page en cliquant sur `voir`, la visionner dans un nouvel onglet en cliquant sur `visionner` ou la supprimer en cliquant sur `supprimer`.

3.3 Problèmes rencontrés

Lors du développement des différentes étapes de l'application, plusieurs problèmes ont été rencontrés. Les deux principaux problèmes sont liés à la détection de mouvement et à l'enregistrement vidéo.

3.3.1 Détection de mouvement

La détection de mouvement est une étape cruciale pour le suivi du grimpeur. En première approche, j'avais décidé d'utiliser la fonction de détection de personne avec OpenCV (HOG descriptor³) pour détecter le grimpeur. Mais cette méthode n'était pas très performante et perdait régulièrement la détection du grimpeur.

Avec Mr Bourdeaud'hui, nous avons fait l'hypothèse que la caméra n'a pas besoin de descendre et qu'un grimpeur ira toujours vers le haut ce qui a facilité la détection des grimpeurs. Nous avons donc décidé de changer de méthode et de passer à une détection de mouvement plus simple basée sur la différence d'images. Cette méthode est plus rapide et plus fiable, mais elle est sensible aux changements

3. Voir la [documentation en ligne](#) pour plus d'information



de luminosité, aux ombres et aux petits mouvements parasites. Pour essayer de contourner ce problème, j'ai ajouté un filtre de moyenne glissante pour lisser les mouvements et un flou gaussien afin de réduire les fausses détections. J'ai aussi ajouté une condition sur le nombre de pixel minimum avec un mouvement pour détecter un barycentre.

Un autre problème fut le départ du grimpeur, en effet en bas de la voie, il est difficile de distinguer le grimpeur de l'assureur. Pour résoudre ce problème, j'ai décidé de déplacer la zone de détection vers le haut de l'image pour ne regarder que le grimpeur. Et je la recentre une fois que la caméra s'est déplacée vers le haut, l'assureur n'est donc plus sur l'image.

Au départ, j'avais mis une valeur fixe pour l'angle de déplacement de la caméra, mais cela ne fonctionnait pas bien pour les différentes voies. Car en fonction du coefficient de zoom la caméra se déplaçait trop ou pas assez. J'ai donc décidé de mettre une valeur d'angle de déplacement en fonction du zoom actuel de la caméra (celui ci doit être bien défini lors de l'ajout d'une nouvelle voie). Cela a permis d'obtenir un déplacement plus précis et mieux adapté à la distance entre la voie et la caméra.

3.3.2 Enregistrement vidéo

Dans un premier temps, j'avais choisi d'utiliser un encodage en H.264 avec l'encodeur `libx264` de FFmpeg. Cette méthode fonctionnait très bien sur une architecture matérielle puissante (ordinateur portable avec un processeur Intel Core i7 6 cœurs 2.6 GHz) Cependant, lorsque j'ai porté le programme sur un raspberry pi 4 (micro-processeur Cortex-A72 disposant de 4 cœurs à 1,5 GHz 64 bits avec 4Go de RAM), j'ai rencontré des problèmes de performances et de latence lors de l'encodage des vidéos. En effet, l'encodage en H.264 est très gourmand en ressources CPU, ce qui entraînait des ralentissements et des décalages dans le flux vidéo. Le programme d'enregistrement vidéo consommait jusqu'à 400 % de CPU pour sortir une vidéo à 5 FPS.

Pour résoudre ce problème, après quelques recherches, j'ai découvert que le rpi possédait un encodeur matériel, j'ai décidé d'utiliser cet encodeur. Anciennement, l'encodeur vidéo était un encodeur `h264_omx` mais sur le raspberry que je possède il a été déprécié et remplacé par l'encodeur `v4l2m2m`. J'ai mis beaucoup de temps à comprendre ce problème et comment utiliser l'encodeur `v4l2m2m` et comment installer FFmpeg avec cet encodeur. Voir l'annexe B pour plus d'information sur le driver vidéo utilisé.

Après avoir compris comment utiliser cet encodeur, j'ai modifié le programme d'enregistrement vidéo pour utiliser l'encodeur uniquement sur le raspberry pi, c'est pour cela que j'ai laissé une option de compilation `H264` pour pouvoir compiler le programme sur un ordinateur x86 avec la librairie `libx264`.

Une fois ce problème résolu, j'ai rencontré un autre problème lié à l'encodage des vidéos qui n'était pas en mode paysage. En effet, dès lors que la vidéo était retournée, les pixels s'échangeaient de place et la vidéo était illisible. Lorsque j'ai développé cette fonctionnalité sur PC, la rotation de l'image se faisait très simplement avec la librairie `libx264` avec l'argument `-transpose` mais sur le rpi avec l'encodeur `v4l2m2m` cette option ne fonctionnait pas. J'ai du encoder la vidéo en mode paysage une première fois puis la ré-encoder avec la bonne orientation. Cette solution n'est pas optimale car chaque programme qui utilise la vidéo doit retourner l'image avant utilisation alors que le programme qui écrit l'image dans la mémoire pourrait le faire directement. Au final, j'ai profité de ce ré-encodage pour découper la vidéo en fonction des images de début et de fin pour ne garder que la partie intéressante de la vidéo.

Cependant, un problème persiste : bien que le nombre d'images par seconde en sortie soit forcé à 25 FPS, FFmpeg produit parfois un débit d'images dans un intervalle de 25 ± 5 fps, ce qui peut entraîner un effet de ralenti ou d'accélération de la vidéo.



3.3.3 Problèmes liés au matériel

Lors d'un essai en salle d'escalade avec un routeur peu performant, j'ai été tributaire de la connexion wifi et le programme ne recevait que très peu d'image par seconde de la caméra. Auparavant, chaque programme (video.cgi, detection.cgi et enregistrementVideo) récupérait directement le flux vidéo via HTTP généré par la caméra. Ainsi le réseau voyait passer trois fois les mêmes informations et saturait. J'ai donc décidé de récupérer le flux vidéo une seule fois et de le partager entre les différents programmes. Cela a permis de réduire la charge réseau et d'améliorer les performances de l'application.

3.3.4 Problèmes web

Lors de l'envoi de requêtes AJAX pour piloter la caméra, j'ai rencontré des problèmes de latence et de synchronisation. En effet, les requêtes étaient envoyées de manière asynchrone, ce qui entraînait des problèmes de synchronisation entre les différentes actions. Pour résoudre ce problème, j'ai ajouté des fonctions de callback pour attendre les réponses des requêtes avant d'envoyer la suivante. J'ai aussi ajouté un système de timeout pour éviter les blocages en cas de non-réponse du serveur.

Cependant, le problème n'est pas résolu. Depuis un pc, l'ordre s'exécute en 100 ms dans toutes les directions mais sur le raspberry pi, l'ordre vers le haut s'exécute aussi en moins de 100ms mais les autres ordres attend le signal SIGALRM fixé à 500ms car l'ordre est bien traité avant ces 500ms.

Voir cette [vidéo](#) pour voir le problème en action. Dans un premier temps, on fait le test sur le raspberry et dans un second sur un pc. On voit que sur le pc, les ordres sont exécutés en moins de 100ms alors que sur le raspberry, les ordres sont exécutés en 500ms sur la droite de l'écran dans l'inspecteur réseau.

4 Tests et résultats

Je suis allé plusieurs fois dans la salle d'escalade afin de tester le programme en conditions réelles. J'ai pu tester la détection de mouvement, le suivi du grimpeur, l'enregistrement vidéo et la visualisation des vidéos. J'ai aussi pu affiner les paramètres de détection. On trouve en figure 6 un exemple de détection du grimpeur en mode paysage et en mode portrait. Au final, lors de l'essai final à Mons le programme fonctionnait bien. J'ai pu enregistrer des vidéos de grimpeurs, les visualiser et les télécharger ainsi que paramétriser les voies. Pour voir un exemple de détection voir cette [vidéo](#).

5 Perspectives d'amélioration

L'application fonctionne correctement, mais il reste des améliorations à apporter pour la rendre plus robuste et plus performante. Voici quelques pistes d'amélioration pour le futur :

- Amélioration de l'interface Web, avoir une interface plus intuitive et plus ergonomique.
- Reformater le code et tout traduire en anglais
- Sur l'écran du Raspberry Pi, afficher un meilleur status de l'exécution de l'application.
- Utiliser les boutons du Raspberry Pi pour relancer l'application et l'écran tactile.
- Ajouter un système de notification pour les erreurs.
- Mieux gérer le stockage des vidéos.
- Serveur accessible depuis internet pour voir les vidéos (chiffrement des données).



FIGURE 6 – Test et installation

- Ajouter un système de détection de chute.
- Ajouter la possibilité d'enregistrer avec les deux caméras en même temps, voir si un changement de matériel est nécessaire.

6 Conclusion

Ce projet a permis de développer une application complète et fonctionnelle pour le suivi automatisé des grimpeurs dans une salle d'escalade à l'aide de caméras motorisées. En combinant des technologies variées telles que la détection de mouvement, l'encodage vidéo, et une interface web intuitive, nous



avons pu répondre aux objectifs initiaux tout en surmontant plusieurs défis techniques.

Les tests réalisés en conditions réelles ont démontré la robustesse et l'efficacité du système, bien que des améliorations soient encore possibles, notamment en termes d'ergonomie, de gestion des ressources, et d'optimisation des performances. Les perspectives d'amélioration identifiées ouvrent la voie à des évolutions futures pour rendre l'application encore plus performante et adaptée aux besoins des utilisateurs.

En conclusion, ce projet offre une solution innovante et pratique pour l'analyse et le suivi des performances des grimpeurs.



Annexes

A Github

Le lien du dépôt GitHub contenant le code source de l'application est le suivant [page GitHub](#).

B Driver vidéo

```

1   pi@raspberrypi:~ $ v4l2-ctl --device=/dev/video11 --all
2   Driver Info:
3       Driver name      : bcm2835-codec
4       Card type       : bcm2835-codec-encode
5       Bus info        : platform:bcm2835-codec
6       Driver version   : 6.6.74
7       Capabilities    : 0x84204000
8           Video Memory-to-Memory Multiplanar
9           Streaming
10          Extended Pix Format
11          Device Capabilities
12         Device Caps     : 0x04204000
13         Video Memory-to-Memory Multiplanar
14         Streaming
15         Extended Pix Format
16     Media Driver Info:
17         Driver name      : bcm2835-codec
18         Model            : bcm2835-codec
19         Serial           : 0000
20         Bus info         : platform:bcm2835-codec
21         Media version    : 6.6.74
22         Hardware revision: 0x00000001 (1)
23         Driver version   : 6.6.74
24     Interface Info:
25         ID                : 0x0300001a
26         Type              : V4L Video
27     Entity Info:
28         ID                : 0x0000000f (15)
29         Name              : bcm2835-codec-encode-source
30         Function          : V4L2 I/O
31         Pad 0x01000010    : 0: Source
32             Link 0x02000016: to remote pad 0x1000012 of entity 'bcm2835-codec-
33             encode-proc' (Video Encoder): Data, Enabled, Immutable
34         Priority: 2
35     Format Video Capture Multiplanar:
36         Width/Height      : 32/32
37         Pixel Format     : 'H264' (H.264)
38         Field             : None
39         Number of planes : 1
40         Flags             :
41         Colorspace        : Rec. 709
42         Transfer Function : Default
43         YCbCr/HSV Encoding: Default
44         Quantization      : Default
45         Plane 0           :
46             Bytes per Line : 0

```



```

46          Size Image      : 524288
47 Format Video Output Multiplanar:
48     Width/Height       : 32/32
49     Pixel Format       : 'YU12' (Planar YUV 4:2:0)
50     Field              : None
51     Number of planes   : 1
52     Flags               :
53     Colorspace         : Rec. 709
54     Transfer Function   : Default
55     YCbCr/HSV Encoding: Default
56     Quantization       : Default
57     Plane 0             :
58           Bytes per Line: 64
59           Size Image    : 3072
60

```

Listing 1 – Propriétés du driver vidéo utilisé pour encoder le flux vidéo

C Notice utilisation

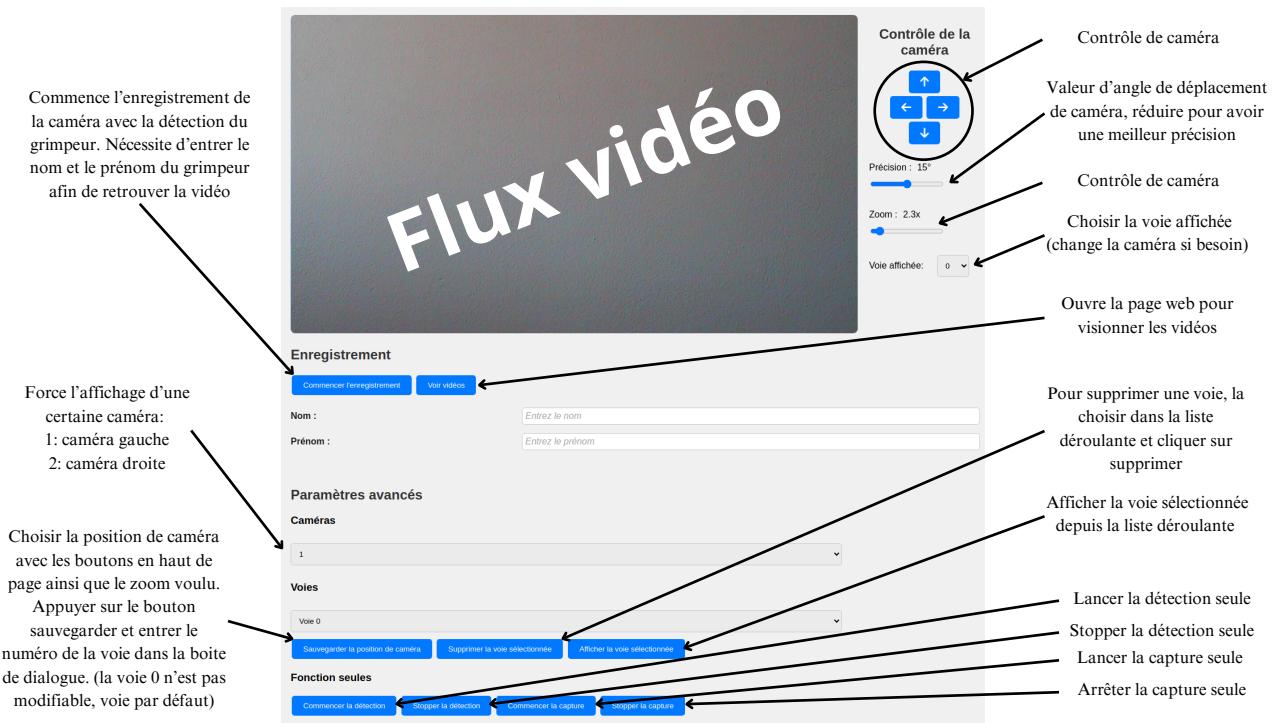


FIGURE 7 – Notice d'utilisation