École Centrale De Lille

Projet d'intégration

Suivi d'un grimpeur par caméra dome

Rapport de PDI
19 mars 2025

CHEVALIER Romain

Tuteur : BOURDEAUD'HUY T.

Table des matières

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 services 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 grimpeur de comprendre leurs forces et faiblesses pour s'améliorer.

Afin de réalisé 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 sont 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 video et leur visualisation.
- 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'enregister sur un stockage une vidéo.
- Avoir une application modulable permettant de modifier les caméra 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.

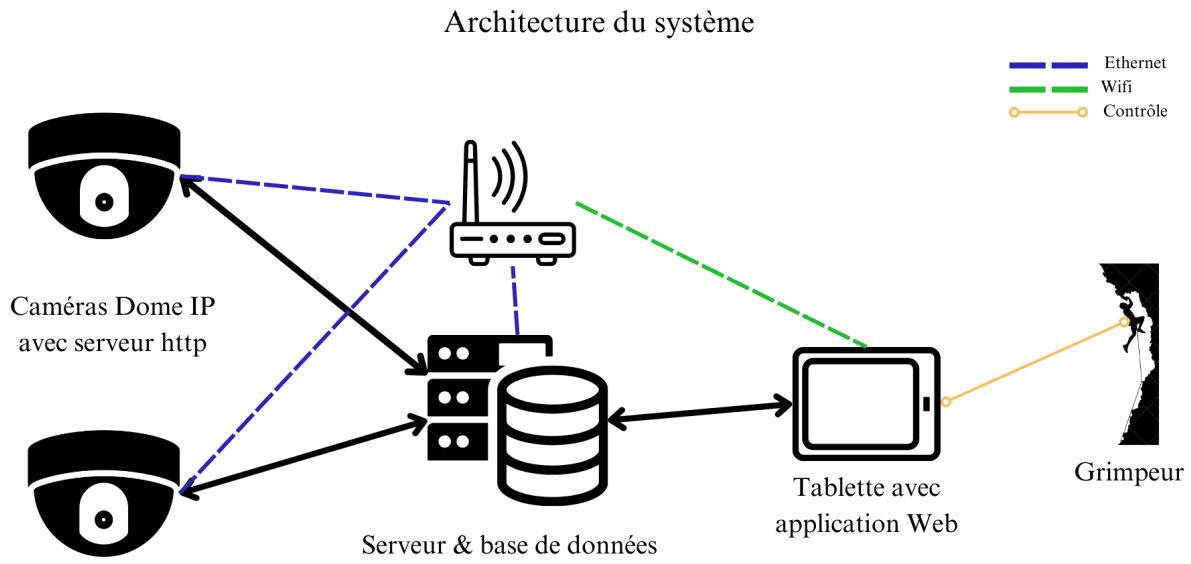


FIGURE 1 – Architecture du système

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 exécuté deux scripts `cgi` 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 crée 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.

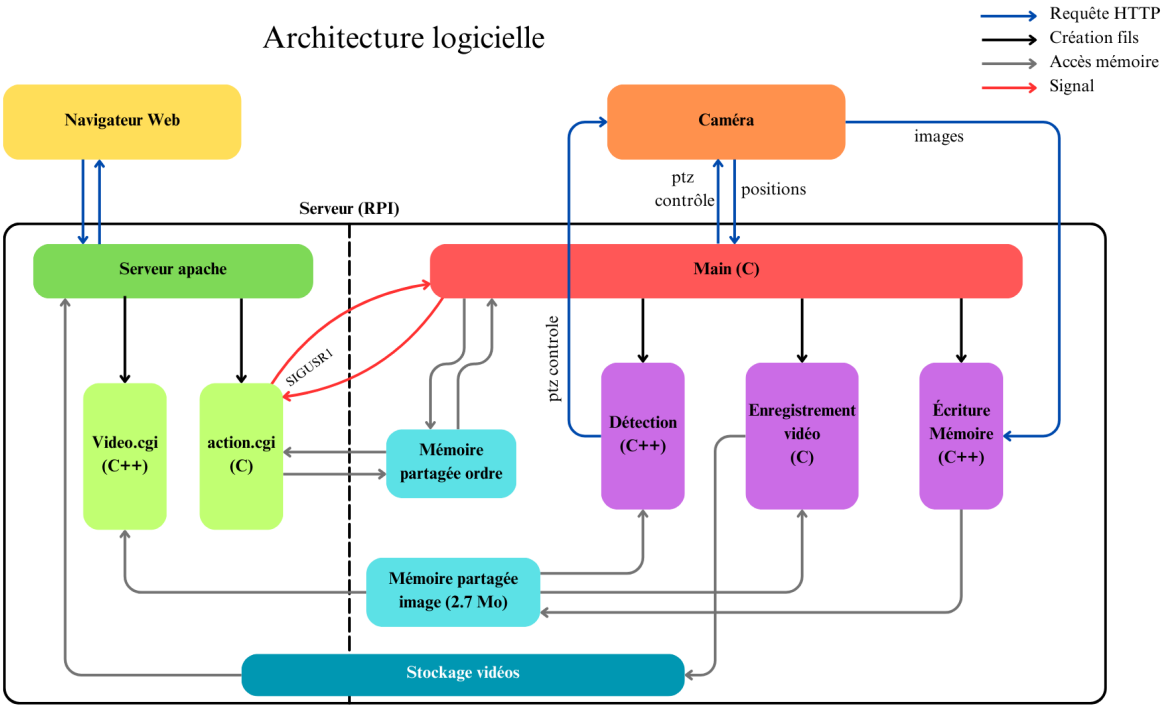


FIGURE 2 – Architecture logicielle

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.

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'orientaiton 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'orientaiton (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 a 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 prédéfinies** : Commandes (`rout`) pour ajouter, supprimer ou afficher des positions prédéfinies, voir le paragraph suivant pour plus d'informations.
- **Changement de caméra active** : Commandes (`cam`) pour basculer entre les caméras configurées.

Gestion des positions prédéfinies 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 prédéfinies 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. 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 :
  - `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.

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

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 `FFmpeg` pour encoder les vidéos dans un format 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'acceleration matérielle sur le rpi. Sur une architechture 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 de 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 `decouperVideo` est appelée pour découper la vidéo et ne conserver que la partie intéressante :

- **Lecture des frames de début et de fin** :
  - Les frames 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 frames** :
  - Les frames sont ajustées en fonction des paramètres définis, comme le nombre de frames 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 frames 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`.

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

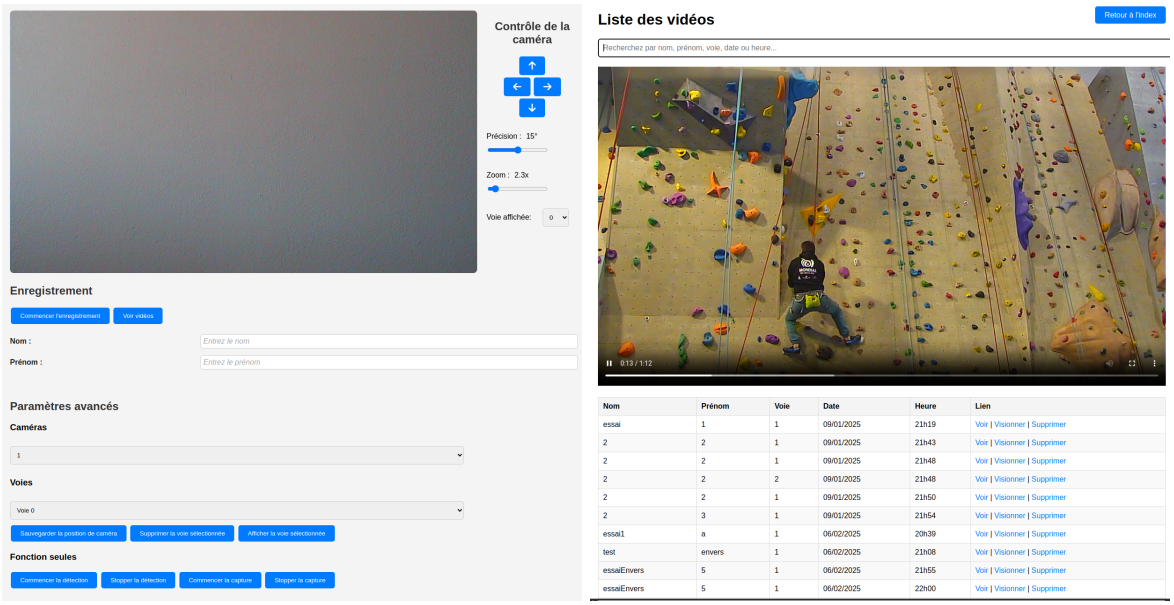
- **SIGTERM** : Arrête proprement le programme en mettant fin à la boucle principale.
- **SIGINT** : Interrompt immédiatement le programme.

3.2 Application web

Le serveur web est composée de deux pages web voir figure 3, 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 3a)

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.



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

FIGURE 3 – Pages web de l'application

Le processus lit dans la mémoire partagée les images brutes a l'aide d'`OPENCV` pour récupérer une matrice de pixel, puis le profil colorimétrique de l'image est changé car `OPENCV` travaille en *BGR (blue green red)* et le `JPEG` utilise un profil (*RGB (red green blue)*) puis est encodé en `JPEG` à l'aide de la librairie `libjpeg`. Enfin, on ajoute les entêtes `HTTP` pour que le navigateur puisse interpreter l'image et on écrit l'image encodée dans la sortie standard.

Sur la gauche de cette page, on retrouve tout les boutons de contrôle de la caméra (haut, bas, gauche et droite) associé à un curseur pour la précision de l'angle afin de premettre un déplacement plus précis. On retrouve un second curseur pour gérer le zoom de la caméra entre 1x et 18x. La pression de l'un de ses boutons créent une requête `AJAX` pour exécuter le script `action.cgi` qui va écrire dans la mémoire partagée l'action à effectuer puis réveiller le programme principal pour qu'il effectue l'action et attendre le feedback du main ou un timeout de 5 secondes avec l'intruction `SIGALRM` (il récupère le pid du main dans un fichier que ce dernier crée au démarrage). Finalement, il envoie une réponse `HTTP` pour indiquer que l'action a bien été effectuée.

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'appuie 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 Page web de visualisation des vidéos (figure 3b)

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 video particulière à l'aide de la barre de recherche en haut de page. Avec les fonction du navigateur, on peut lire, mettre en pause, accélérer, ralentir ou télécharger la vidéo. Enfin, on peut voir la video en haut de page en cliquant sur `voir`, la visionnier dans un nouvel onglet en cliquant sur `visionnier` ou la supprimer en cliquant sur `supprimer`.

3.3 Problèmes rencontrés

4 Tests et résultats

5 Perspectives d'amélioration

- Amélioration de l'interface Web, avoir une interface plus intuitive et plus ergonomique.
- Reformater le code et tout traduire en anglais
- Afficher sur l'écran du Raspberry Pi, un meilleur status de l'execution de l'application.
- Utiliser les boutons du Raspberry Pi pour relancer l'application.
- Ajouter un système de notification pour les erreurs.
- Mieux gérer le stockage des vidéos.
- Ajouter un système de détection de chute.