Documentation du Code Python

Acquisition, affichage, et Encodage/enregistrement d'images
BAYER24 en Temps Réel d'un appareil genicam

4 février 2025

Table des matières

1 Bibliothèques Utilisées

Le code s'appuie sur plusieurs bibliothèques, chacune ayant un rôle spécifique :

— **time**
Fournit des fonctions pour gérer le temps, les délais et mesurer des durées (exemple : calculer le FPS, temporiser les essais de connexion).

— **numpy**
Utilisée pour la manipulation de tableaux multidimensionnels et les calculs numériques (création, transformation et manipulation d'images).
— **ctypes**
Permet l'accès à la mémoire (notamment pour récupérer les données du buffer de la caméra via un pointeur) et interagir avec du code en C.
— **cv2 (OpenCV)**
Bibliothèque de traitement d'images et de vision par ordinateur. Elle est utilisée pour le redimensionnement, le filtrage, la conversion entre espaces colorimétriques et l'application d'algorithmes comme CLAHE.
— **arena_api.system**
Interface pour la détection, la connexion et la configuration des appareils de capture (caméras) compatibles avec l'API Arena.
— **numba**
Accélère l'exécution de certaines fonctions en compilant du code Python en code machine (via `@njit`). La parallélisation est obtenue grâce à `prange`.
— **subprocess**
Permet de lancer des processus externes. Ici, il est utilisé pour lancer FFmpeg en vue d'encoder les images sauvegardées en une vidéo.
— **os**
Fournit des fonctionnalités pour interagir avec le système d'exploitation, notamment pour gérer les fichiers et les dossiers (création et suppression du dossier temporaire).
— **threading**
Permet d'exécuter plusieurs tâches en parallèle via des threads. Dans ce code, il est utilisé pour lancer simultanément la sauvegarde des images et l'encodage vidéo.
— **queue**
Offre une file d'attente thread-safe afin de communiquer entre différents threads (par exemple, pour transmettre les images capturées au thread de sauvegarde).
— **signal**
Permet de gérer l'envoi et la réception de signaux (ici, pour transmettre un signal d'arrêt à FFmpeg).

2 Détail des Fonctions

Dans cette section, chaque fonction du code est expliquée en détail.

2.1 `create_device_automatically()`

But : Se connecter automatiquement au premier appareil détecté.
Description :
- La fonction effectue plusieurs tentatives (jusqu'à 6 essais) pour détecter un appareil via l'appel à `system.create_device()`.
- Si aucun appareil n'est détecté lors d'un essai, le programme attend 10 secondes avant de réessayer.
- Dès qu'un appareil est trouvé, le premier de la liste est retourné.
- Si aucun appareil n'est détecté après le nombre maximal d'essais, une exception est levée.

2.2 `setup(device)`

But : Configurer les paramètres de la caméra.
Description :
- La fonction récupère la *nodemap* du dispositif pour accéder aux paramètres de configuration.
- Elle définit les valeurs pour `Width`, `Height`, `OffsetX`, `OffsetY` et `PixelFormat`.
- Le format choisi est `BayerRG24` pour la capture en 24 bits.
- Elle calcule la taille attendue du buffer (largeur $\times$ hauteur $\times$ 3 octets) et renvoie la largeur, la hauteur et la taille.

2.3 `construct_bayer_image(buffer, padded_buffer, buffer_array, height, width)`

But : Reconstruire une image Bayer brute à partir des données du buffer.
Description :
- Accède aux données du buffer via `ctypes` en utilisant l'adresse mémoire de `buffer.pdata.contents`.
- Convertit ces données en un tableau NumPy de dimensions `(height, width, 3)` via `np.ctypeslib.as_array`.
- Recopie les données dans un `padded_buffer` de dimensions `(height, width, 4)` (le dernier canal servant d'espace de padding).
- Interprète le `padded_buffer` comme une image 24 bits (via un view en `uint32`) et retourne l'image Bayer.

2.4 `apply_gamma_lut_to_image(b_channel, g_channel, r_channel, gamma_lut, rgb_image_8bit, height, width)`

But : Appliquer une correction gamma à l'image à l'aide d'une Look-Up Table (LUT).

Description :

— Fonction compilée par Numba (`@njit`) pour accélérer le traitement.

— Pour chaque pixel (parcours parallélisé avec `prange`), la fonction récupère la valeur corrigée dans la LUT pour chacun des canaux bleu, vert et rouge.

— Les valeurs corrigées sont placées dans l'image `rgb_image_8bit` qui sera affichée ou sauvegardée.

2.5 `demosaic_bayer_image(bayer_image, r_channel, g_channel, b_channel, kernel_r_b, kernel_g)`

But : Réaliser le dématriçage (demosaicing) de l'image Bayer pour obtenir les canaux couleur séparés.

Description :

— Initialise les tableaux `r_channel`, `g_channel` et `b_channel` en les remplissant de zéros.

— Répartit les pixels de l'image Bayer selon la disposition typique RGGB :

  — Les pixels situés aux positions `[0::2, 0::2]` sont affectés au canal Rouge.

  — Les pixels situés aux positions `[0::2, 1::2]` et `[1::2, 0::2]` sont affectés au canal Vert.

  — Les pixels situés aux positions `[1::2, 1::2]` sont affectés au canal Bleu.

— Applique ensuite des filtres 2D (via `cv2.filter2D`) sur chaque canal pour interpoler et affiner les valeurs :

  — `kernel_r_b` est utilisé pour les canaux Rouge et Bleu.

  — `kernel_g` est utilisé pour le canal Vert.

2.6 `apply_clahe_to_lab_channels(rgb_image)`

But : Appliquer l'algorithme CLAHE pour améliorer le contraste de l'image.

Description :

— Convertit l'image RGB en espace de couleur LAB à l'aide de `cv2.cvtColor`.

— Sépare l'image en ses trois canaux : L (luminance), A et B.

— Applique la méthode CLAHE (Contrast Limited Adaptive Histogram Equalization) sur le canal L pour améliorer le contraste de façon locale sans amplifier le bruit.

— Recombine les canaux pour former une image LAB et reconvertit l'image en RGB.

2.7 `ensure_temp_dir()`

But : Vérifier et créer le dossier temporaire pour stocker les images capturées.

Description :

— Vérifie si le dossier défini par la variable `TEMP_DIR` existe.
— S'il n'existe pas, la fonction crée le dossier.

2.8 `clean_temp_dir()`

But : Nettoyer le dossier temporaire après encodage en supprimant toutes les images et le dossier lui-même.

Description :

— Parcourt le dossier temporaire et supprime chaque fichier qu'il contient.
— Supprime ensuite le dossier temporaire.

2.9 `save_images_from_queue()`

But : Sauvegarder les images depuis la file d'attente dans un thread séparé.

Description :

— La fonction récupère en boucle des tuples `(frame, frame_count)` depuis la file d'attente `image_queue`.
— Chaque image est sauvegardée au format PNG dans le dossier temporaire avec un nom formaté (exemple : `frame_0001.png`).
— La boucle continue tant que le flag `stop_saving` n'est pas activé ou que la file d'attente n'est pas vide.

2.10 `encode_video_with_ffmpeg_parallel(output_file, fps, stop_event)`

But : Lancer FFmpeg dans un thread séparé pour encoder les images en une vidéo en temps réel.

Description :

— Construit la commande FFmpeg pour lire les fichiers images sauvegardés (nommés selon le pattern `frame_%04d.png`) et créer une vidéo avec le codec FFV1.
— Le processus FFmpeg est lancé via `subprocess.Popen`.
— La fonction surveille régulièrement le flag `stop_event`. Une fois cet événement activé (par exemple lors de l'appui sur Échap), un signal SIGINT est envoyé à FFmpeg pour arrêter l'encodage proprement.
— Attend la terminaison du processus FFmpeg et affiche un message de succès.

2.11 `mouse_callback(event, x, y, flags, param)`

But : Gérer les interactions de la souris sur la fenêtre d'affichage pour activer ou désactiver les corrections d'image.

Description :

- La fonction est enregistrée comme callback pour la fenêtre OpenCV.
- Elle détecte si l'utilisateur clique dans l'une des zones correspondant aux boutons :
  - Zone du bouton CLAHE (coordonnées entre 10 et 110 en abscisse, et 10 et 50 en ordonnée).
  - Zone du bouton Gamma (coordonnées entre 120 et 220 en abscisse, et 10 et 50 en ordonnée).
- En cas de clic, le flag correspondant (`use_clahe` ou `use_gamma`) est inversé, permettant ainsi d'activer ou de désactiver la correction.

2.12 `stream_and_save_images_parallel()`

But : Capturer, traiter et afficher les images en temps réel tout en les envoyant pour sauvegarde et encodage.

Description :

- **Connexion et configuration :**
  Se connecte automatiquement à la caméra via `create_device_automatically()` et configure ses paramètres via `setup()`.
- **Préparation du traitement :**
  Crée des tableaux NumPy pour stocker l'image brute, l'image dématriçée et l'image finale. Prépare également les noyaux de filtrage pour le dématriçage.
- **Traitement de l'image :**
  Dans une boucle de capture :
  1. Récupère un buffer de la caméra.
  2. Reconstruit l'image Bayer brute avec `construct_bayer_image()`.
  3. Effectue le dématriçage avec `demosaic_bayer_image()` pour obtenir les canaux R, G, B.
  4. Applique, selon le flag, la correction gamma (`apply_gamma_lut_to_image()`) ou extrait directement les bits significatifs.
  5. Si activé, applique le CLAHE via `apply_clahe_to_lab_channels()`.
- **Affichage et sauvegarde :**
  Redimensionne l'image pour l'affichage et dessine des boutons interactifs pour contrôler CLAHE et Gamma. L'image traitée est affichée dans une fenêtre OpenCV et est également envoyée dans la file d'attente (`image_queue`) pour la sauvegarde.
- **Gestion du flux :**
  La fonction calcule le FPS par paquet de 50 images et gère la réini-

tialisation des compteurs. La capture s'arrête si l'utilisateur appuie sur la touche Échap ou en cas d'interruption.

— **Nettoyage :**
À la fin de la capture, les buffers sont réenfilés et la capture est proprement arrêtée. Un signal est envoyé pour arrêter l'encodage vidéo.

2.13 `main()`

But : Orchestrer l'ensemble de l'application.
Description :

1. Démarre le thread de sauvegarde d'images via `save_images_from_queue()`.
2. Lance en parallèle le thread d'encodage vidéo avec FFmpeg en appelant `encode_video_with_ffmpeg_parallel()`.
3. Démarre le streaming et le traitement des images avec `stream_and_save_images_parallel()`.
4. Une fois la capture terminée (par appui sur Échap ou interruption), signale l'arrêt aux threads, attend leur terminaison et nettoie le dossier temporaire via `clean_temp_dir()`.