

RAPPORT TECHNIQUE

TRAITEMENTS DE DONNÉES POUR LE GÉO-POSITIONNEMENT DU TGV IRIS 320



Auteurs:
Eric KHELLA
Baptiste HUBERT
Antoine GATARD

Projet CODEVSI 41
Version 1.0
27/05/2024

Professeur encadrant:
Patrice PAJUSCO

Résumé

Ce rapport présente les différents travaux réalisés dans le cadre du projet mené par le département Micro-Onde d'IMT Atlantique qui participe à l'établissement de modèles de canaux de transmission pour le déploiement du futur standard de télécommunication ferroviaire, le FRMCS (Futur Railway Mobile Communication System). Dans ce but, le TGV IRIS 320 a permis d'effectuer certaines mesures de propagation, ainsi le géo-positionnement de l'antenne de mesure devient primordial pour mener à bien ce projet.

L'objectif du projet a été la réalisation d'un programme permettant l'analyse de fichiers vidéo enregistrés grâce au TGV IRIS 320. Cette analyse comporte deux phases principales, la reconnaissance d'image afin de recueillir les données visuellement présentes sur les vidéos, ainsi qu'une datation précise de ces données pour connaître la position du TGV, la vitesse du TGV, l'instant auquel la mesure est réalisée et la frame associée.

Le projet, sous la supervision de M. PAJUSCO, vise donc à synchroniser et à analyser ces heures de vidéos sous différents angles, en les coordonnant dans une perspective spatio-temporelle.

Table des matières

Résumé.....	1
Introduction.....	3
1. Conception du programme.....	4
1.1 Analyse des besoins et objectifs.....	4
1.2 Choix des technologies et outils.....	5
1.3 Fonctionnement global du programme.....	6
2. Développement et implémentation.....	8
2.1 Implémentation concernant le traitement vidéo.....	8
2.2 Détection des états de la LED et Interpolation.....	10
2.3 Exploitation des fichiers XML.....	12
3. Validation et évaluation.....	13
3.1 Éléments de validation concernant les cas nominaux d'utilisation.....	13
3.2 Évaluation de la conformité des résultats aux contraintes.....	14
Conclusion.....	16
Bibliographie.....	17
Glossaire.....	18
Annexes.....	19
Planning prévisionnel.....	19
Planning réel.....	19
Analyse des écarts des plannings.....	20
Rédacteurs/Relecteurs.....	20

Introduction

Les conducteurs de train utilisent la radio pour rester en contact avec les régulateurs du trafic ferroviaire et pour envoyer ou recevoir des alertes radio si nécessaire. Elle est également utilisée pour transmettre des informations numériques entre la cabine de conduite et les équipements au sol. Aujourd'hui, cette radio fonctionne grâce à la technologie GSM-R (2G). A l'avenir, le FRMCS (5G) devrait prendre le relais du GSM-R [1].

Pour permettre cette transition vers un nouveau système de communication il est nécessaire d'avoir une connaissance précise de la position de l'antenne de mesure placée au-dessus du TGV IRIS 320 à chaque instant. Le projet consiste ainsi à affiner cette localisation à une échelle extrêmement précise en étant capable de dater les données recueillies avec une précision de l'ordre de la milliseconde. L'ambition est de parvenir à déterminer la position exacte du TGV à l'aide d'une interpolation avancée des données issues des vidéos et des capteurs, en utilisant les technologies de programmation Python et XML.

Dans ce rapport, nous nous intéresserons d'abord à la conception du programme. Nous passerons ensuite à la phase de développement et d'implémentation, en présentant les différents composants du programme . Enfin, nous procéderons à la validation et à l'évaluation des performances du programme , en discutant des résultats obtenus et des perspectives d'amélioration.

Rédacteur(s) : Antoine Gatard / Relecteur(s) : Eric Khella

1. Conception du programme

1.1 Analyse des besoins et objectifs

Ce programme de restitution des données doit générer un fichier (3 formes sont possibles : CSV, liste ou dictionnaire) pour chaque vidéo, incluant le numéro de la frame, l'heure, la date, la position et la vitesse du train. Ce fichier doit comporter ces différentes données pour chaque frame de la vidéo. Le temps de génération de ce fichier doit être inférieur ou égal au temps de la vidéo. Cette limitation en temps de traitement s'explique par le fait que le client souhaite ne pas accumuler des vidéos à traiter, ainsi en respectant cette limite la vidéo enregistrée un jour n peut être traitée entièrement au jour n+1.

Pour pouvoir recueillir les différentes données visibles sur la vidéo de la caméra arrière plusieurs méthodes sont utilisées. Pour les fichiers vidéo, le programme doit extraire de chaque image les données de la vitesse, de la borne kilométrique, de l'heure, ainsi que de la date. Quant aux fichiers XML, l'un d'eux est visible sur la figure 1, le programme doit permettre d'extraire les noms des fichiers vidéos à analyser (qui varient en fonction des besoins de l'utilisateur), et il est également possible de récupérer la date, l'heure et la durée de l'enregistrement. Il est aussi possible de récupérer le numéro de la ligne et le nom du repère correspondant à la position du train, pour que l'utilisateur situe bien sur quelle voie se situe le TGV IRIS 320. De plus, afin que le programme puisse récupérer de manière fiable les informations de chaque image, il faut que la police soit supérieure à 10 pt et que la résolution soit supérieure à 300 dpi (dots per inch) - d'après la documentation de la librairie Pytesseract [2].

```
325    1002" PK_fin="110710" Video="M2_231018_1529_216000_V1_110002.avi" Date="23/10/18 15:12" Durree="0:14" Raison="EndVoie" Camera="M2" />
326    1002" PK_fin="110710" Video="M1_231018_1329_216000_V1_110002.avi" Date="23/10/18 13:29" Durree="0:44" Raison="EndVoie" Camera="M1" />
327    PK_fin="10002" Video="M1_231018_1553_216000_V1_1.avi" Date="23/10/18 15:53" Durree="3:33" Raison="SplitDistance" Camera="M1" />
328    PK_fin="10002" Video="M2_231018_1553_216000_V1_6.avi" Date="23/10/18 15:53" Durree="3:33" Raison="SplitDistance" Camera="M2" />
329    107" PK_fin="20001" Video="M2_231018_1557_216000_V1_10007.avi" Date="23/10/18 15:57" Durree="2:59" Raison="SplitDistance" Camera="M2" />
330    102" PK_fin="20001" Video="M1_231018_1557_216000_V1_10002.avi" Date="23/10/18 15:57" Durree="2:59" Raison="SplitDistance" Camera="M1" />
331    101" PK_fin="30001" Video="M2_231018_1600_216000_V1_20001.avi" Date="23/10/18 16:00" Durree="2:27" Raison="SplitDistance" Camera="M2" />
332    106" PK_fin="30001" Video="M1_231018_1600_216000_V1_20006.avi" Date="23/10/18 16:00" Durree="2:27" Raison="SplitDistance" Camera="M1" />
333    106" PK_fin="40003" Video="M2_231018_1602_216000_V1_30006.avi" Date="23/10/18 16:02" Durree="2:27" Raison="SplitDistance" Camera="M2" />
334    106" PK_fin="40003" Video="M1_231018_1602_216000_V1_30006.avi" Date="23/10/18 16:02" Durree="2:27" Raison="SplitDistance" Camera="M1" />
335    108" PK_fin="50004" Video="M2_231018_1604_216000_V1_40008.avi" Date="23/10/18 16:04" Durree="2:0" Raison="SplitDistance" Camera="M2" />
336    108" PK_fin="50004" Video="M1_231018_1604_216000_V1_40008.avi" Date="23/10/18 16:04" Durree="2:0" Raison="SplitDistance" Camera="M1" />
337    109" PK_fin="60003" Video="M2_231018_1606_216000_V1_50009.avi" Date="23/10/18 16:06" Durree="2:1" Raison="SplitDistance" Camera="M2" />
338    109" PK_fin="60003" Video="M1_231018_1606_216000_V1_50009.avi" Date="23/10/18 16:06" Durree="2:1" Raison="SplitDistance" Camera="M1" />
339    108" PK_fin="70004" Video="M2_231018_1608_216000_V1_60008.avi" Date="23/10/18 16:08" Durree="2:0" Raison="SplitDistance" Camera="M2" />
340    108" PK_fin="70004" Video="M1_231018_1608_216000_V1_60008.avi" Date="23/10/18 16:08" Durree="2:0" Raison="SplitDistance" Camera="M1" />
341    109" PK_fin="80002" Video="M2_231018_1610_216000_V1_70009.avi" Date="23/10/18 16:10" Durree="2:0" Raison="SplitDistance" Camera="M2" />
342    109" PK_fin="80002" Video="M1_231018_1610_216000_V1_70009.avi" Date="23/10/18 16:10" Durree="2:0" Raison="SplitDistance" Camera="M1" />
343    107" PK_fin="90000" Video="M2_231018_1612_216000_V1_80007.avi" Date="23/10/18 16:12" Durree="2:0" Raison="SplitDistance" Camera="M2" />
344    107" PK_fin="90000" Video="M1_231018_1612_216000_V1_80007.avi" Date="23/10/18 16:12" Durree="2:0" Raison="SplitDistance" Camera="M1" />
345    103" PK_fin="10003" Video="M1_231018_1614_216000_V1_90003.avi" Date="23/10/18 16:14" Durree="2:0" Raison="SplitDistance" Camera="M1" />
346    103" PK_fin="10003" Video="M2_231018_1614_216000_V1_90003.avi" Date="23/10/18 16:14" Durree="2:0" Raison="SplitDistance" Camera="M2" />
347    1004" PK_fin="110001" Video="M2_231018_1616_216000_V1_100004.avi" Date="23/10/18 16:16" Durree="3:33" Raison="SplitDistance" Camera="M2" />
```

Figure 1: Extrait du fichier XML

Pour générer ce fichier, on dispose pour chaque trajet effectué par le TGV de 3 vidéos avec différents points de vue visibles sur les figures 2 et 3, deux caméras à

l'avant et à l'arrière du train, ainsi qu'une caméra périphérique au-dessus du train. La caméra située sur le toit à un intérêt tout particulier : en effet, elle filme une LED rouge qui s'allume de manière très précise au début de chaque seconde. C'est grâce à cet allumage régulier qu'il est possible de définir une échelle des temps plus précise que celle proposée par le lecteur vidéo. On peut également voir que cette vidéo affiche l'heure (qui semble être précise à la milliseconde), mais il est très difficile d'analyser les chiffres correspondant aux millisecondes ou aux secondes (car on a des changements qui perturbent l'affichage, et un framerate trop faible). Le programme doit donc être capable de mettre en relation ces 3 vidéos pour pouvoir générer le fichier attendu.



Figure 2: Point de vue de la caméra à vision périphérique



Figure 3: Point de vue de la caméra arrière

Voici la démarche que le programme doit suivre pour associer les données à un instant précis avec une précision jusqu'à la milliseconde. Le programme utilise la détection de la luminescence de la LED rouge, cette LED fonctionne comme une horloge de précision et est utilisée comme référence temporelle. Toutefois, étant donné la précision de la caméra utilisée pour cette LED (60 frames/sec), une méthode plus précise pour déterminer le temps de début de la seconde (à la milliseconde près) est nécessaire. Pour cela, une interpolation sur la luminescence de la LED est réalisée. En suivant l'évolution de la luminescence sur les différentes frames (à partir du moment où elle s'allume faiblement, jusqu'à sa luminosité maximale) il est possible de déterminer à quel instant de la seconde la frame a lieu avec une précision de l'ordre de la milliseconde.

De même, nous réalisons une interpolation sur les deux autres fichiers vidéos de la caméra avant et de la caméra arrière, cette fois en récupérant l'heure sur l'affichage directement (voir figure 3). Ensuite, nous synchronisons ces 3 vidéos pour qu'à un instant donné, nous sortions 3 images, qui correspondent aux 3 frames à ce même instant sur les 3 fichiers vidéos (caméra avant, caméra arrière, caméra à vision périphérique).

1.2 Choix des technologies et outils

Pour la réalisation de l'algorithme d'analyse vidéo, le langage de programmation retenu est Python, possédant des bibliothèques plutôt facile d'accès mais aussi assez complètes pour ce projet.

Quatre modules ont été principalement utilisés pour le développement de l'algorithme. NumPy [3], une bibliothèque essentielle pour la manipulation efficace des tableaux de données, a permis d'accélérer significativement les opérations sur les matrices et les vecteurs grâce à la vectorisation. Cette capacité de NumPy à effectuer des opérations en parallèle sur des ensembles de données volumineux réduit considérablement les temps de calcul, ce qui est crucial pour le traitement en temps réel des vidéos.

Le module Open-CV (CV2) [4], un module primordial qui nous est utile pour le traitement vidéo. Nous utilisons principalement ces fonctions pour ouvrir et traiter les fichiers avec les méthodes `cv2.VideoCapture`, pour récupérer les données de notre vidéo (FPS, hauteur, largeur) avec les méthodes `cv2.CAP_PROP_FPS`, `cv2.CAP_PROP_FRAME_WIDTH`; mais aussi pour appliquer des filtres, passer les images en niveaux de gris, etc.

Pour la reconnaissance optique de caractères (OCR), Pytesseract, une bibliothèque agissant comme un wrapper pour Tesseract-OCR, a été intégrée. Cette technologie est essentielle pour extraire les informations textuelles des images vidéo, permettant ainsi de recueillir des données additionnelles pour l'interpolation et la localisation précise du TGV. Certaines configurations permettent d'améliorer la précision de cet outil, par exemple, on peut lui préciser que l'on souhaite détecter certains caractères, et d'autres non; permettant alors d'optimiser la reconnaissance d'une heure, une date, ou simplement une suite de chiffres. Après avoir essayé un autre modèle, EasyOCR, celui-ci s'avère être plus efficace et rapide.

Le module concurrent.futures [5] permet de gérer des threads pour exécuter des tâches de manière asynchrone. Facilitant ainsi l'utilisation du parallélisme dans le programme, il nous permet de réaliser plus d'opérations en un temps donné, ce qui nous permet de gagner un temps important compte tenu de la contrainte qui nous est imposée.

L'environnement de développement intégré utilisé pour ce projet est Visual Studio Code (VSCode). VSCode a été choisi pour sa flexibilité et la possibilité d'utiliser Git ce qui permet de faciliter la collaboration à plusieurs sur un même programme.

Le module CSV a également été utilisé, en plus faible proportion, pour la gestion des données en tableau. Il permet de lire et d'écrire facilement des fichiers CSV, couramment utilisés pour le stockage et l'échange de données structurées.

Cette fonctionnalité est par exemple utile pour importer les données des capteurs et exporter les résultats de l'analyse sur un fichier Excel.

Rédacteur(s) : Antoine Gatard / Relecteur(s) : Baptiste Hubert

1.3 Fonctionnement global du programme

Le programme se compose de plusieurs fichiers Python qui fonctionnent de manière coordonnée pour permettre d'aboutir au résultat souhaité. Parmi ces fichiers, certains sont des outils qui nous amènent à une meilleure précision dans nos décisions (par exemple des aperçus de ce qu'on analyse, faire du débogage), certains regroupent les différents paramètres (qui ont été amenés à être modifiés à plusieurs reprises), les différents modules, les différents messages d'erreur et d'informations. Et enfin, les autres fichiers ont eux un côté largement plus pratique, qui permet l'analyse, la récupération et l'organisation de données.

Traitement du Fichier XML

L'objectif du code qui traite ce fichier est de retourner les noms des fichiers vidéo que l'utilisateur souhaite analyser. Plusieurs possibilités sont codées : premièrement, lorsque l'utilisateur indique une heure (du format HH:mm:ss, qui est vérifié avec une fonction annexe) et qu'il souhaite analyser uniquement les fichiers qui contiennent cet instant. L'utilisateur a également la possibilité d'analyser tous les fichiers d'une journée, ou d'analyser tous les fichiers compris entre une heure A et une heure B précisées par l'utilisateur. Les fichiers récupérés sont au format .avi, un format que OpenCV2 sait traiter.

Ouverture de la Vidéo

Une fois le fichier vidéo sélectionné, le programme ouvre le fichier vidéo en utilisant OpenCV. La méthode `open_video` vérifie si la vidéo s'est correctement ouverte lors de l'exécution du code. Si l'ouverture échoue, un message d'erreur approprié est affiché. L'étape suivante consiste à récupérer les informations de base de la vidéo, telles que la fréquence d'images (FPS), les dimensions des frames et le nombre total de frames, en utilisant la méthode `get_video_info` (qui contient des fonctions déjà pré-établies dans la bibliothèque OpenCV).

Traitement des frames

Le cœur du programme réside dans le traitement des frames de la vidéo. Chaque frame est lue individuellement par la méthode `read_frame`. Pour chaque frame, le programme extrait les zones d'intérêt spécifiées pour chaque donnée (la vitesse, les marqueurs kilométriques, le temps, la date) à l'aide de la méthode

`extract_data_from_frame` (les paramètres de ces segmentations sont regroupés dans un fichier `segmentation_settings`). Ayant une contrainte temporelle assez importante pour le traitement, nous ne pouvons pas nous permettre d'analyser, à chaque frame, toutes les données présentes sur l'image, sachant que certaines d'entre elles sont amenées à peu bouger. Nous faisons alors la différence terme à terme de 2 frames consécutives (différence possible car l'image est représentée sous forme d'un tableau numpy), si la différence est relativement faible, c'est que les pixels ont peu changé (ou pas... on essaie d'intégrer de la robustesse au bruit), et donc nous n'avons pas besoin d'analyser à nouveau cette zone. Nous répétons cette technique pour toutes les frames d'une vidéo, pour toutes les zones d'intérêts, ce qui optimise le processus.

Extraction et sauvegarde des données

Les données extraites de chaque frame sont sauvegardées dans le format spécifié par l'utilisateur (CSV, dictionnaire ou liste) à l'aide de la méthode `save_data`. Cette étape garantit que toutes les informations pertinentes sont conservées pour une analyse ultérieure. Une barre de progression est affichée pendant le traitement pour informer l'utilisateur de l'avancement du processus.

Nettoyage et affichage des résultats

Une fois toutes les frames traitées, le programme exécute une étape de nettoyage pour libérer les ressources utilisées, exécuter la fermeture des fichiers et libérer objets de capture vidéo. Enfin, les résultats et les temps d'exécution sont affichés à l'utilisateur à l'aide de la méthode `display_results`. Les temps d'exécution des différentes étapes sont également visualisés sous forme de graphique circulaire pour fournir une vue d'ensemble des performances du programme et s'assurer que la contrainte concernant le temps d'exécution du programme est respectée. Ce graphique nous est utile également pour savoir où nous pouvons gagner du temps d'exécution.

Interpolation temporelle

Pour permettre l'accès à des frames spécifiques en fonction du temps, le programme inclut une fonction d'interpolation temporelle. Cette fonction utilise une relation linéaire entre le temps vidéo et le temps LED, déterminée par une régression linéaire, facilitant ainsi l'analyse et l'extraction des informations souhaitées par l'utilisateur.

La conception du programme ayant été détaillée avec ses objectifs, besoins, et choix technologiques, l'accent sera maintenant mis sur le développement et l'implémentation. En particulier, l'efficacité des différents codes Python permettant de récupérer les images souhaitées, comme `VideoConstructor`, `VideoDataAnalyzer`,

`VideoFrameDisplay`, sera examinée, ainsi que les méthodes de traitement vidéo, de détection et d'interpolation des états de la LED. Les méthodes de traitement asynchrone nécessaires au respect des contraintes temporelles et de précision des données, seront également abordées.

Rédacteur(s) : Eric Khella / Relecteur(s) : Baptiste Hubert

2. Développement et implémentation

2.1 Implémentation concernant le traitement vidéo

Pour pouvoir facilement gérer le traitement des vidéos, toutes les méthodes nécessaires à la génération du fichier CSV ont été réparties dans ces différents fichiers : `VideoConstructor`, `VideoDataAnalyzer`, `VideoDirectoryTool`, `VideoFrameDisplay`.

Fonctionnement de `VideoConstructor`

`VideoConstructor` implémente la classe `VideoProcessor`, qui s'organise autour de 5 fonctionnalités principales. Différentes variables sont définies pour stocker des informations telles que le chemin de la vidéo, l'objet de capture vidéo et l'identifiant de frame.

Puis viennent les fonctions utilitaires qui vont faciliter le code par la suite, telles que `convert_ms_to_time_format` qui est utilisée pour convertir des millisecondes en format de temps, ou `rewrite_marker_format` qui permet de transformer l'affichage de la borne kilométrique affichée (de la forme XXX+mmm, X étant les kilomètres et m, les mètres) en format de distance approprié (notamment pour le traitement ensuite) et `progress_bar` qui affiche une barre de progression pendant le traitement de la vidéo, montrant le pourcentage d'avancement et le temps estimé restant.

On trouve ensuite le wrapper, qui agit sur d'autres méthodes, et qui permet de récupérer le temps pris chaque étape de notre traitement vidéo. La récupération de ces différents temps n'a pas d'intérêt pour l'utilisateur, mais il est intéressant pour nous de savoir dans quelle direction s'orienter pour gagner du temps d'exécution.

La classe `VideoProcessor` utilise plusieurs méthodes pour accomplir son objectif de traitement vidéo. La méthode `open_video` ouvre la vidéo à traiter, tandis que `get_video_info` récupère des informations caractéristiques de la vidéo, telles que la résolution et le nombre total de trames. La méthode `read_frame` lit chaque image de la vidéo successivement. Pour optimiser le traitement, `detect_change` détecte les changements significatifs entre deux images successives. Si un changement est détecté, la méthode `extract_data_from_frame` extrait les données spécifiques de l'image, comme la vitesse, le temps et les marqueurs kilométriques. Cette extraction

de données est facilitée par `get_text`, qui utilise la reconnaissance optique de caractères pour extraire le texte des zones d'intérêt des images. Les données extraites sont sauvegardées à l'aide de la méthode `save_data`, qui les enregistre dans un fichier au format CSV, liste ou dictionnaire. Enfin, la méthode `process_video` coordonne l'ensemble du processus de traitement de la vidéo en appelant les autres méthodes appropriées

Fonctionnement de la classe VideoProcessor

L'organisation de la classe `VideoProcessor` et l'utilité de ses différentes méthodes ont été présentées. L'attention sera maintenant portée sur le fonctionnement global de la classe ainsi que sur les interactions entre ses méthodes.

Dans le processus de traitement vidéo, la méthode `detect_change` joue un rôle important en identifiant les changements significatifs entre deux images successives. Cette méthode prend en entrée deux zones d'intérêt de l'image actuelle et de l'image précédente, puis calcule la différence entre ces deux zones en niveaux de gris. Cette différence est ensuite passée en noir et blanc facilitant ainsi la détection des zones où un changement notable s'est produit. Si la zone de changement dépasse un seuil donné, cela indique qu'une modification significative s'est produite entre les deux images, et la méthode renvoie `True` pour signaler ce changement.

Une fois qu'un changement est détecté entre deux images successives, la méthode `extract_data_from_frame` intervient pour extraire les données spécifiques de l'image actuelle. Cette méthode reçoit l'image en entrée et commence par découper l'image en zones d'intérêt, telles que la zone de vitesse, la zone de marqueur kilométrique, et la zone de temps. Ces zones sont ensuite analysées pour extraire les données pertinentes à l'aide de la reconnaissance optique de caractères (OCR) fournie par Pytesseract.

L'étape clé ici est que la méthode `extract_data_from_frame` utilise la méthode `detect_change` pour déterminer si les données dans une zone spécifique de l'image ont changé depuis la dernière image. Si aucun changement significatif n'est détecté dans une zone donnée, les données précédemment extraites de cette zone sont réutilisées plutôt que de relancer le processus d'extraction. Cela permet d'optimiser les performances en évitant de répéter inutilement les opérations de traitement d'image et d'OCR sur des zones qui n'ont pas changé.

Cette approche qui utilise le principe de la mémoïsation [6] contribue à accélérer le processus d'extraction de données en exploitant la cohérence temporelle entre les images successives. En détectant les changements et en n'extrayant que les nouvelles données lorsque cela est nécessaire, le module parvient à traiter efficacement les vidéos de trains tout en minimisant les ressources requises et en optimisant les performances globales du système. La figure 4

ci-dessous est un diagramme de séquence qui illustre les opérations effectuées par VideoProcessor sur les zones à analyser.

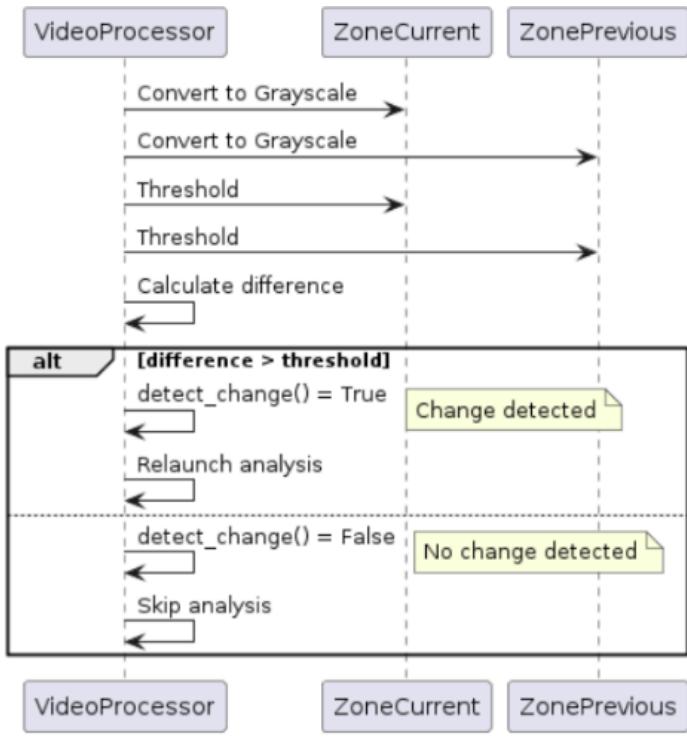


Figure 4: Diagramme de séquence illustrant les deux scénarios possibles selon le booléen retourné par *detect_change*

Y'a plus de MorphologyEx, c'était un test de preprocessing pour voir si ça améliorait tesseract

Rédacteur(s) : Eric Khella et Baptiste Hubert / Relecteur(s) : Antoine Gatard

2.2 Détection des états de la LED et Interpolation

Détection des états de la LED

Afin de recueillir les états de la LED, le module *LED_on_multiprocess* s'organise autour de 2 étapes principales. La première correspond au calcul du masque circulaire qui est appliqué sur les différentes frames de la vidéo. Ce masque est calculé à l'aide de numpy. Les coordonnées de chaque pixel de l'image sont générées à l'aide de la fonction ogrid du module Numpy. Ensuite, la distance au centre du cercle est calculée pour chaque pixel à l'aide de la formule de distance euclidienne . Enfin, un masque binaire est créé en comparant la distance euclidienne

calculée avec le rayon du cercle. Les pixels qui sont à l'intérieur du cercle ont la valeur True, sinon False.

Une fois la zone d'intérêt définie grâce au masque la méthode *process_frame* est utilisée pour recueillir les niveaux moyens de couleur RGB de la led à chaque frame de la vidéo. Ce traitement peut rapidement s'avérer coûteux en temps. C'est pour cela que la méthode *process_frame* est exécutée à l'aide de la méthode ThreadPoolExecutor du module concurrent.futures. ThreadPoolExecutor permet de paralléliser les opérations et méthodes appliquées aux frames pendant leur analyse permettant ainsi de gagner en temps d'exécution. L'utilisation de ThreadPoolExecutor pour le traitement des trames de vidéo permet une amélioration significative des performances, réduisant le temps d'exécution à environ 10% de la durée totale de la vidéo. En comparaison, traiter les trames de manière séquentielle, sans ThreadPoolExecutor, augmente le temps d'exécution à environ 80% de la durée totale de la vidéo. Ce temps gagné est non négligeable au vu de la contrainte concernant le temps d'exécution du programme global. Les instants où la LED est allumée sont facilement repérables sur la Figure 5 ci-dessous, ils correspondent aux pics visibles à intervalle de temps régulier.

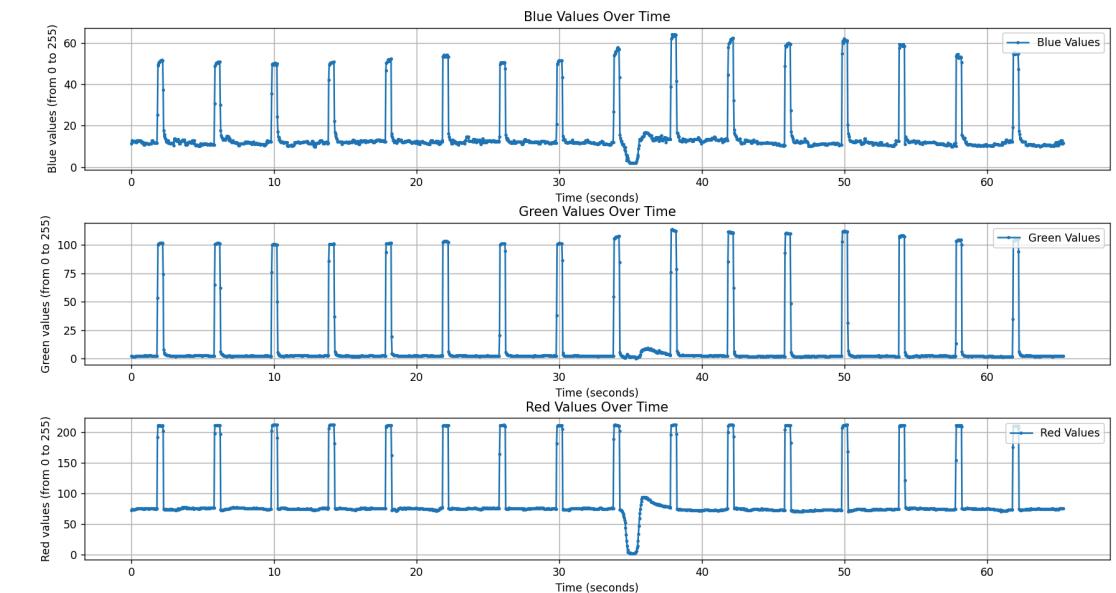


Figure 5: Evolution des niveaux moyens de luminescence de la led en fonction du temps

Exploitation des états de la LED et interpolation

Une fois l'évolution des niveaux moyens de couleur de la LED connus, il reste à exploiter cette évolution en déterminant précisément l'instant auquel la LED passe de l'état éteint à l'état allumé. Afin de déterminer uniquement les fronts montants qui correspondent à l'allumage de la LED, la différence entre chaque point successif

dans le tableau contenant les niveaux de rouge est calculée pour obtenir les changements entre les valeurs successives. Ainsi les fronts montants correspondent à une différence d'environ +120 (en terme de niveau de rouge) alors que les fronts descendants correspondent à une différence de -120. Les instants qui correspondent à un front montant sont alors facilement identifiables. Ensuite une régression linéaire est effectuée sur les points où les fronts montants ont lieu comme l'illustre la figure 6.

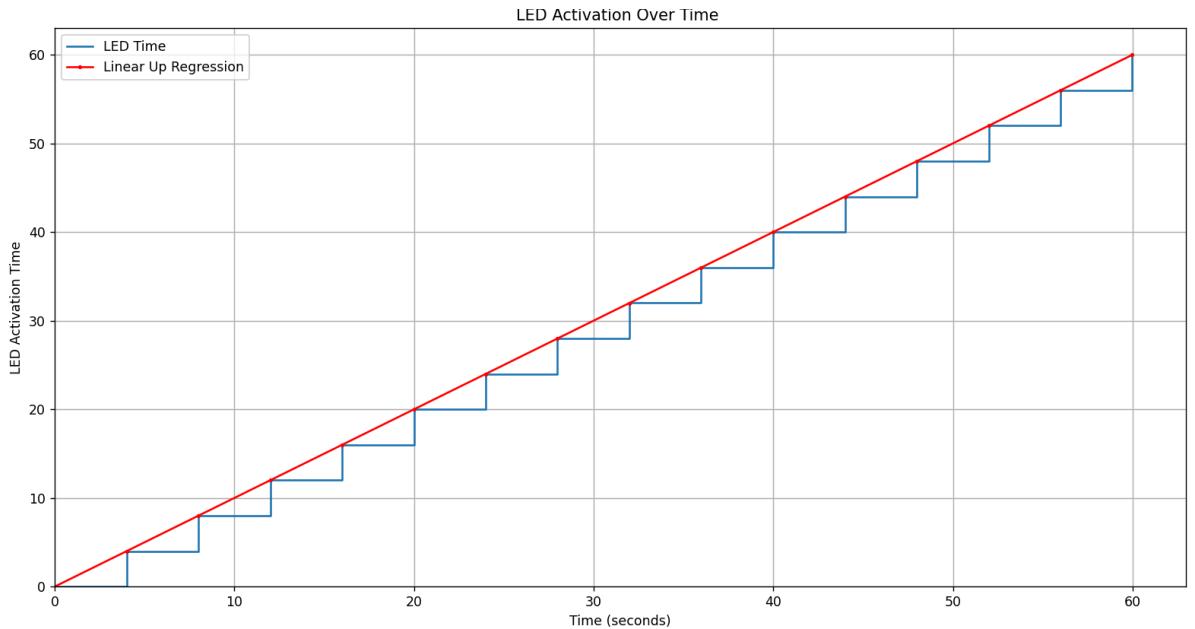


Figure 6: Régression linéaire effectuée à partir de la détection de l'allumage de la LED

Avec la connaissance des paramètres (coefficients directeur, ordonnée à l'origine) cette droite permet de définir une nouvelle échelle temporelle basée sur la LED, respectant ainsi la contrainte de précision qui exige une précision de l'ordre de la milliseconde.

Rédacteur(s) : Eric Khella et Baptiste Hubert / Relecteur(s) : Antoine Gatard

2.3 Exploitation des fichiers XML

Le fichier XML qui nous est donné regroupe tous les enregistrements vidéos réalisés des motrices 1 et 2 du TGV IRIS 320 (les vidéos avec la LED rouge qui nous permet d'être précis dans le géopositionnement sont autre part). Dans ce fichier XML, voir figure 7, à chaque "feuille" de ce fichier, nous avons plusieurs informations à notre disposition : les points kilométriques de début et de fin lors de

l'enregistrement (Pk_debut et PK_fin), le nom du fichier vidéo en .avi (qu'on extrait afin d'analyser la vidéo), la date, la durée, ainsi que la raison de l'arrêt de l'enregistrement, et enfin le nom de la caméra (M1 pour motrice 1 et M2 pour motrice 2).

```
<?xml version="1.0" encoding="UTF-8"?>
<Voie Repere="V?????">
  <Sections>
    <Section PK_debut="249106" PK_fin="249304" Video="M1_231018_1208_272000_V_249106.avi" Date="23/10/18 12:08" Duree="1:54" Raison="EndVoie" Camera="M1"/>
    <Section PK_debut="249106" PK_fin="249304" Video="M2_231018_1208_272000_V_249106.avi" Date="23/10/18 12:08" Duree="1:54" Raison="EndVoie" Camera="M2"/>
  </Sections>
</Voie>
</Voies>
```

Figure 7: Extrait du fichier XML avec les caméras M1 et M2 (avant et arrière) du TGV IRIS 320

L'objectif dans l'utilisation de ce fichier est de permettre d'analyser une portion uniquement de tous les enregistrements et du géo-positionnement du TGV, par exemple lorsque l'utilisateur souhaite n'avoir le géo-positionnement du TGV qu'à un certain moment. Dans les faits, l'utilisateur indique la date et l'heure où il souhaite avoir la position (s'il souhaite le positionnement à 12 h/09 min/33,467sec alors il indique 12:09), puis les fichiers correspondant sont extraits et analysé par un autre algorithme pour obtenir la précision souhaitée.

D'autre part, si l'utilisateur souhaite analyser toute la journée, il n'est évidemment pas obligé de le faire minute par minute, le code lui donne la possibilité de choisir entre analyser tous les fichiers d'une journée, analyser tous les fichiers entre les heures A et B (données par l'utilisateur), ou d'analyser que 3 fichiers vidéos (M1, M2 et celui de la LED) pour obtenir un instant précis.

Après avoir détaillé l'implémentation des méthodes de traitement vidéo dans la classe VideoProcessor, ainsi que la détection des états de la LED et l'exploitation des fichiers XML, nous allons maintenant évaluer les performances et la précision du système. Cette section de validation portera sur les cas nominaux d'utilisation et vérifiera si le programme répond aux exigences de temps d'exécution et de précision définies.

Rédacteur(s) : Baptiste Hubert / Relecteur(s) : Eric Khella

3. Validation et évaluation

3.1 Éléments de validation concernant les cas nominaux d'utilisation

Dans un premier temps, le cahier des charges de ce projet imposait un premier cas d'utilisation. Le programme devait être capable de générer pour une vidéo de motrice en entrée, un fichier contenant toutes les informations (date,heure, vitesse, borne kilométrique).

```

1 Frame,Speed,Date,Time,Km marker
2 0,205,18/10/2023,16:00:15,019+906
3 1,205,18/10/2023,16:00:15,019+911
4 2,205,18/10/2023,16:00:15,019+911
5 3,205,18/10/2023,16:00:15,019+916
6 4,205,18/10/2023,16:00:15,019+921
7 5,205,18/10/2023,16:00:15,019+921
8 6,205,18/10/2023,16:00:16,019+926
9 7,205,18/10/2023,16:00:16,019+931
10 8,205,18/10/2023,16:00:16,019+931
11 9,206,18/10/2023,16:00:16,019+936
12 10,206,18/10/2023,16:00:16,019+941
13 11,206,18/10/2023,16:00:16,019+941
14 12,206,18/10/2023,16:00:16,019+946
15 13,206,18/10/2023,16:00:16,019+951
16 14,206,18/10/2023,16:00:16,019+951
17 15,206,18/10/2023,16:00:16,019+956
18 16,206,18/10/2023,16:00:16,019+961
19 17,206,18/10/2023,16:00:16,019+961
20 18,206,18/10/2023,16:00:16,019+966
21 19,206,18/10/2023,16:00:16,019+971
22 20,206,18/10/2023,16:00:16,019+971
23 21,206,18/10/2023,16:00:16,019+976
24 22,206,18/10/2023,16:00:17,019+981
25 23,206,18/10/2023,16:00:17,019+981
26 24,206,18/10/2023,16:00:17,019+986

```

Figure 8 : Extrait du fichier CSV contenant toutes les informations de la vidéo motrice arrière passée en entrée

Ensuite, ce code offre la possibilité d'exporter les données sous la forme d'une liste ou d'un dictionnaire Python, le tout écrit dans un fichier .txt contenu dans le dossier de départ.

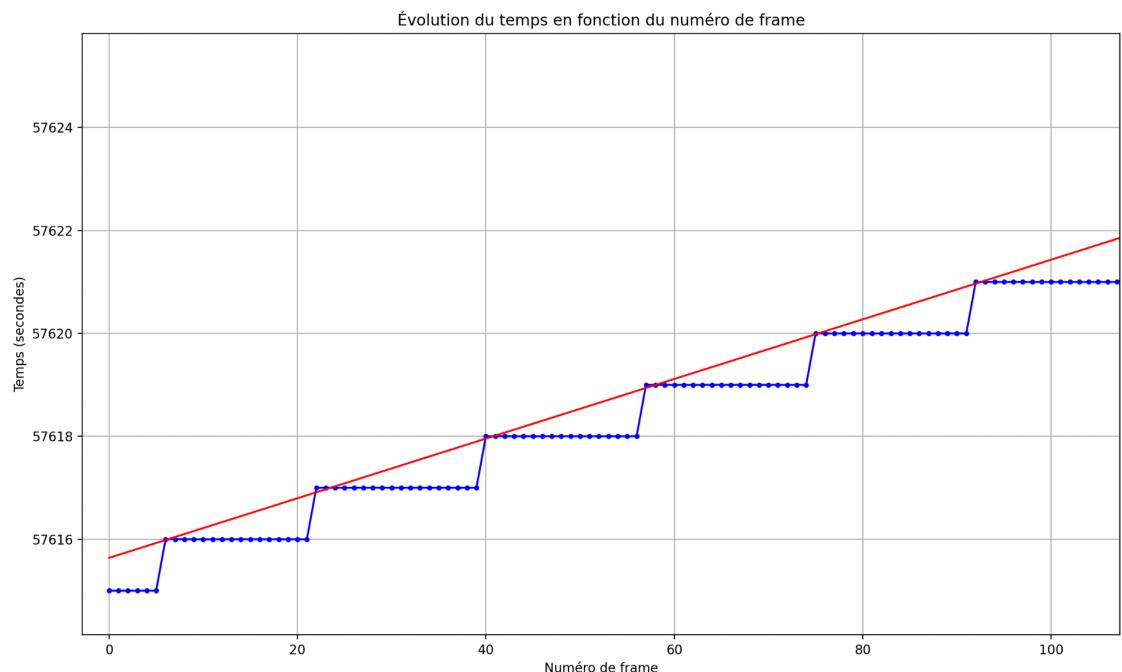
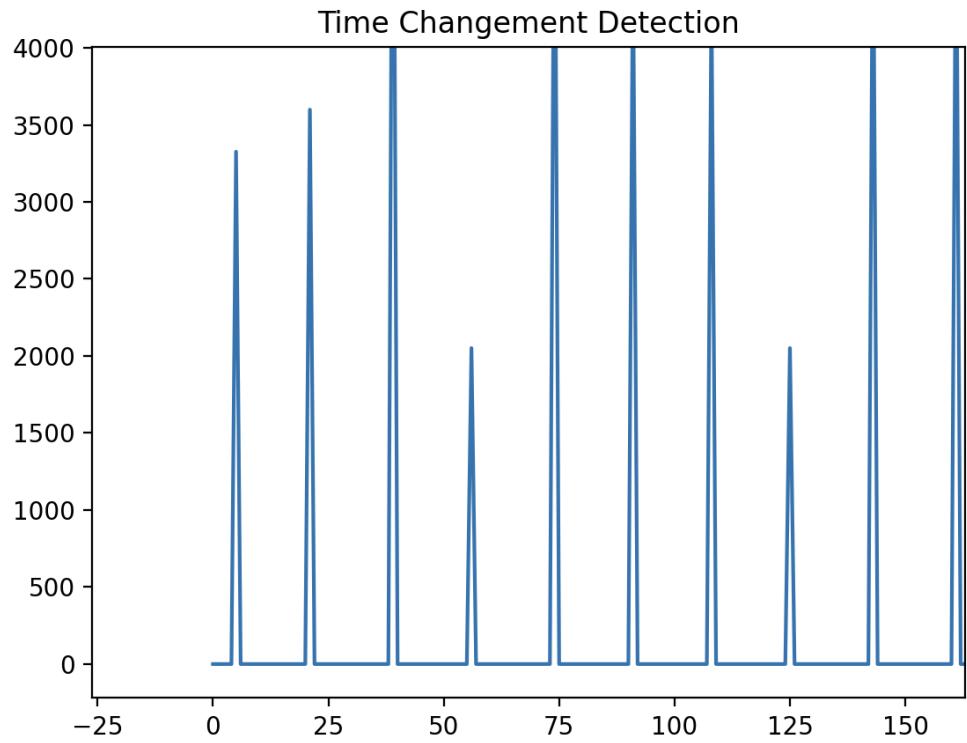
```

1 {0: {'speed': '205', 'marker': '019+906', 'time': '16:00:15', 'date': '18/10/2023'}}
2 {1: {'speed': '205', 'marker': '019+911', 'time': '16:00:15', 'date': '18/10/2023'}}
3 {2: {'speed': '205', 'marker': '019+911', 'time': '16:00:15', 'date': '18/10/2023'}}
4 {3: {'speed': '205', 'marker': '019+916', 'time': '16:00:15', 'date': '18/10/2023'}}
5 {4: {'speed': '205', 'marker': '019+921', 'time': '16:00:15', 'date': '18/10/2023'}}
6 {5: {'speed': '205', 'marker': '019+921', 'time': '16:00:15', 'date': '18/10/2023'}}
7 {6: {'speed': '205', 'marker': '019+926', 'time': '16:00:16', 'date': '18/10/2023'}}
8 {7: {'speed': '205', 'marker': '019+931', 'time': '16:00:16', 'date': '18/10/2023'}}
9 {8: {'speed': '205', 'marker': '019+931', 'time': '16:00:16', 'date': '18/10/2023'}}
10 {9: {'speed': '206', 'marker': '019+936', 'time': '16:00:16', 'date': '18/10/2023'}}

```

Figure 9 : Extrait du même fichier, en format .txt enregistré sous format de dictionnaire Python

On vérifie qu'il n'y a aucune aberration en traçant les différentes valeurs obtenues en fonction du temps; mais aussi en analysant les valeurs de modifications de pixels des zones étudiées.



Figures 10 & 11 : Ces figures permettent de retrouver les FPS de la caméra utilisée (17.25 en l'occurrence); pareillement, la valeur des secondes change selon le même ratio

D'autre part, le code devait offrir la possibilité suivante :
Le client doit pouvoir mettre en entrée une heure précise à la milliseconde près; et un dossier se crée sur son bureau; contenant alors les frames correspondantes aux différents points d'enregistrement.

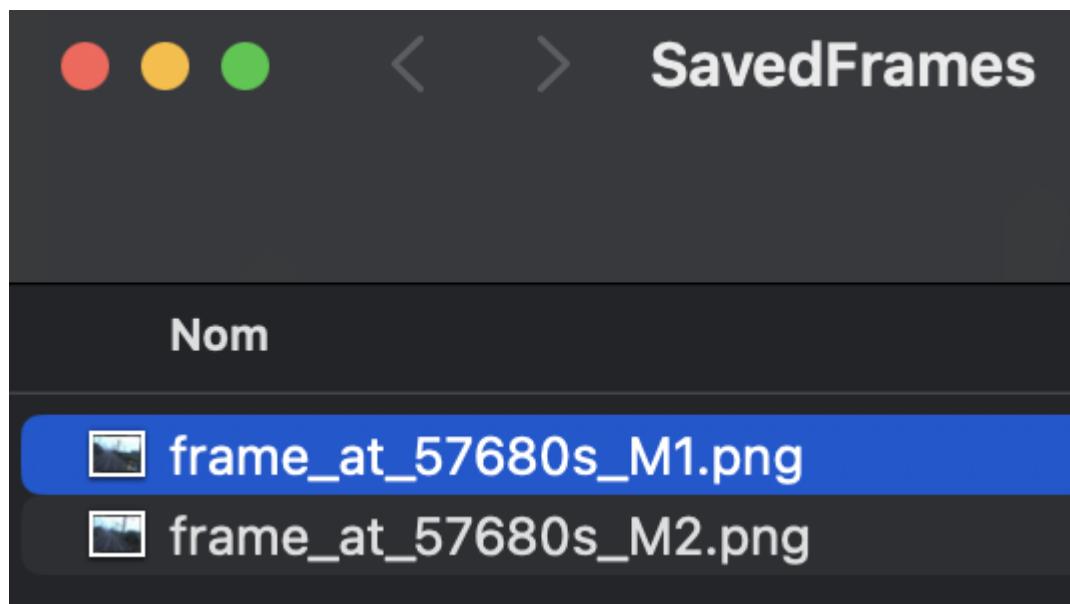


Figure 12 : Capture d'écran du dossier créé sur le bureau après avoir exécuté le code avec pour entrée 57680 sec

Rédacteur(s) : Antoine Gatard / Relecteur(s) : Baptiste Hubert

3.2 Évaluation de la conformité des résultats aux contraintes

Temps d'exécution du programme

Pour déterminer le comportement du programme vis à vis de la contrainte concernant son temps d'exécution, ce dernier a été testé sur une vidéo fournie par la caméra arrière du train.

La figure 8 ci-dessous représente la distribution du temps d'exécution du programme selon les différentes tâches à exécuter. On constate que la grande majorité du temps d'exécution correspond à la reconnaissance optique des caractères de la borne kilométrique (extraction_marker). En effet, le changement de valeur de la borne kilométrique est bien plus récurrent et rapide que celui de la vitesse ou de l'heure. Cela réduit le temps gagné grâce au principe de mémoïsation et explique le déséquilibre dans la répartition de la figure 8.

Malgré cette difficulté, la mémoïsation appliquée sur les zones d'extraction de la vitesse et de l'heure permet de gagner un temps suffisant pour que le programme puisse s'exécuter en 103.6% de durée de vidéo. On peut donc considérer que la contrainte en temps d'exécution est satisfaite

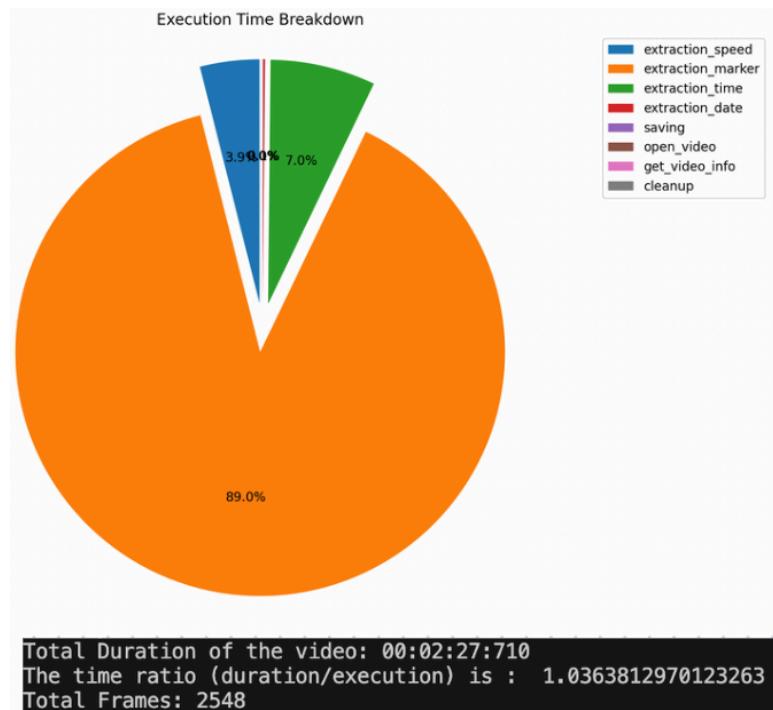


Figure 8 : Diagramme circulaire de la distribution du temps d'exécution du programme

Précision de l'échelle temporelle basée sur la régression linéaire de la LED

Pour évaluer la précision de la régression linéaire obtenue, la fonction de conversion `convert_video_time_to_led_time` a été testée à l'aide de la vidéo fournie par la caméra à vision périphérique. Trois valeurs temporelles sont comparées à l'aide de la figure 9 ci-dessous. Les valeurs de la deuxième colonne correspondent aux instants réels obtenus en récupérant les frames de la vidéo. La troisième colonne présente les valeurs obtenues grâce à la fonction de conversion. Enfin, la dernière colonne contient les valeurs obtenues via une fonction utilisant les fps de la vidéo. Comme on peut le voir figure 2, l'instant donné par la caméra est du type heure:minute:seconde.décimale, pour plus de clarté seules les secondes ont été retranscrites dans le tableau.

n° de la frame	instant réel (vu sur la caméra à vision périphérique)	convert_video_time_to_led_time	fonction naïve utilisant les fps
20	42.692	42.671	42.686
100	43.357	43.351	43.338

300	45.019	45.014	45.000
500	46.691	46.686	46.674
700	48.357	48.353	48.341
900	50.000	49.997	50.009
1100	51.690	51.688	51.677
1300	53.357	53.355	53.344
1500	55.021	55.020	55.012
1700	56.693	56.693	56.680
1900	58.357	58.358	58.347

Figure 9 : Tableau comparatif des performances entre la fonction de conversion utilisée et une fonction naïve

Écart moyen avec la fonction de conversion : 2,9 ms

Écart moyen avec la fonction naïve : 13,8 ms

L'écart moyen de 13,8 ms commis par la fonction naïve démontre la nécessité d'utiliser une méthode plus précise pour satisfaire la contrainte en précision qui exige une erreur de l'ordre de la milliseconde. Notre fonction de conversion commet, sur l'échantillon de valeur recueilli, un écart moyen de 2,9 ms. Cet écart est dans l'ordre de grandeur attendu et est bien inférieur à l'intervalle de temps compris entre deux frames de la vidéo qui est ici d'approximativement 57ms. Ainsi, en se basant uniquement sur l'écart moyen, la fonction de conversion permet de répondre à l'exigence de la contrainte de précision.

Mais on remarque que l'erreur commise par la fonction de conversion varie selon la position de la frame dans la vidéo. En effet, plus la frame est proche du début de la vidéo plus l'erreur est importante. Elle atteint 6 ms pour la 20ème frame et se rapproche donc d'un ordre de grandeur assez important au regard de l'exigence qu'impose la contrainte. Ce comportement s'explique par les paramètres de la droite obtenue par régression linéaire. L'ordonnée à l'origine de cette droite est de -0.005 s et son coefficient directeur vaut 1.00046. Lorsqu'on cherche à dater une frame en début de vidéo, l'influence du coefficient directeur est minimisée. On obtient alors une erreur au niveau de la datation très proche de l'ordonnée à l'origine.

Cependant les valeurs responsables de ce comportement ont été obtenues avec une régression linéaire utilisant 16 points (voir figure 6). On peut s'attendre à ce que l'augmentation du nombre de points permette d'obtenir des paramètres de droite plus précis qui pourraient permettre de réduire l'erreur commise sur la datation

des frames. En résumé, on peut considérer que la fonction de conversion permet de satisfaire la contrainte de précision.

Rédacteur(s) : Eric Khella et Antoine Gatard / Relecteur(s) : Baptiste Hubert

Conclusion

Rédacteur(s) : Antoine Gatard / Relecteur(s) : Eric Khella

Bibliographie

[1] FRMCS or the Future Railway Mobile Communication System, disponible sous:

<https://www.sncf-reseau.com/fr/a/frmcs-or-future-railway-mobile-communication-system>

[2] Documentation de la librairie Pytesseract, disponible sous:

<https://pypi.org/project/pytesseract/>

[3] Introduction à l'utilisation de Numpy,disponible sous:

<https://datascientest.com/numpy>

[4] Documentation du module Open-CV (Open Source Computer Vision), disponible sous:

<https://docs.opencv.org/4.x/>

[5] Documentation du module concurrent.futures,disponible sous:

<https://docs.python.org/3/library/concurrent.futures.html#>

[6] Présentation du principe de mémoïsation,disponible sous:

<https://developpement-informatique.com/article/562/methodes-de-la-programmation-dynamique---memoisation-et-tabulation>

Glossaire

CSV (Comma-Separated Values) : Format de fichier texte utilisé pour stocker des données tabulaires, où chaque ligne correspond à un enregistrement et chaque valeur est séparée par une virgule.

FPS / Framerate (Frames Per Second) : Nombre d'images affichées par seconde dans une séquence vidéo. Plus le FPS est élevé, plus le mouvement dans la vidéo est fluide.

FRMCS (Futur Railway Mobile Communication System) : Le futur standard de communication mobile ferroviaire, basé sur la technologie 5G, destiné à remplacer le système GSM-R actuel.

Git : Système de contrôle de version distribué utilisé pour suivre les modifications apportées aux fichiers et coordonner le travail entre plusieurs personnes.

GSM-R (Global System for Mobile communications - Railways) : Système de communication sans fil utilisé dans les réseaux ferroviaires pour les communications vocales et de données entre les trains et les centres de contrôle.

Interpolation : Méthode mathématique utilisée pour estimer les valeurs entre deux points connus. Dans le contexte de ce projet, elle permet de déterminer avec précision les temps intermédiaires à partir de la séquence d'allumage de la LED.

LED (Light Emitting Diode) : Diode électroluminescente, une source de lumière émise par un matériau semi-conducteur lorsqu'un courant électrique la traverse.

Mémoïsation : Technique d'optimisation qui consiste à stocker les résultats des appels de fonction coûteux en temps de calcul et à réutiliser ces résultats lorsque les mêmes appels sont effectués avec les mêmes paramètres.

NumPy (Numerical Python) : Bibliothèque Python utilisée pour le calcul scientifique. Elle offre un support pour les tableaux multidimensionnels, des fonctions mathématiques, logiques et des algorithmes d'algèbre linéaire.

OCR (Optical Character Recognition) : Reconnaissance optique de caractères, une technologie permettant de convertir différents types de documents, tels que des documents scannés ou des images de texte, en données modifiables et consultables par une machine.

OpenCV (Open Source Computer Vision Library) : Bibliothèque de vision par ordinateur en open source, largement utilisée pour des tâches de traitement d'image et de vidéo.

Pytesseract : Bibliothèque Python qui sert de wrapper pour Tesseract-OCR, permettant l'extraction de texte à partir d'images.

Régression linéaire : Méthode statistique utilisée pour modéliser la relation entre une variable dépendante et une ou plusieurs variables indépendantes en ajustant une ligne droite (ou une courbe).

Thread:(ou fil d'exécution) Unité d'exécution distincte au sein d'un programme. Il permet l'exécution simultanée de plusieurs tâches à l'intérieur du même processus. #c'est pas mal de définir thread avec thread dans la définition

ThreadPoolExecutor : Composant du module concurrent.futures en Python, permettant de gérer efficacement des pools de threads pour exécuter des tâches de manière asynchrone. **#DONC ON DEFINIT ALGORITHME MAIS PAS POOL DE THREADS**

Vectorisation : Technique de programmation utilisée pour effectuer des opérations sur des ensembles de données de manière simultanée, plutôt que séquentielle, pour améliorer les performances de calcul.

VSCode (Visual Studio Code) : Éditeur de code source développé par Microsoft pour Windows, Linux et macOS. Il inclut des fonctionnalités comme le support de débogage, le contrôle de version intégré (Git), des surbrillances syntaxiques et une personnalisation étendue grâce aux extensions.

Wrapper : objet ou fonction qui "enveloppe" un autre objet ou une autre fonction pour modifier ou étendre son comportement

XML (Extensible Markup Language) : Langage de balisage utilisé pour encoder des documents dans un format lisible à la fois par l'homme et par la machine. Utilisé pour structurer, stocker et transporter des informations.

Annexes

Planning prévisionnel

Planning réel

Analyse des écarts des plannings

Rédacteurs/Relecteurs