Lycée Dhuoda

Table des matières

PARTIE COMMUNE

Lycée Dhuoda

Répartition des tâches et cahier des charges

Ce projet a pour but d'enrichir un système de chronométrage déjà existant appartenant à l'association Sud-Chrono. Sur ce système doit être ajouter une prise de temps intermédiaires avec des photos/vidéos et un suivi en temps réel pour les coureurs équipés d'un smartphone fonctionnant sous Android. Ces nouvelles données générées devront êtres compatibles avec les logiciels de chronométrage fournies et déjà utilisé par l'association (WinIpico et GmCAP). L'ensemble des éléments de la course d'un coureur devra lui être envoyé par mail de manière automatique à la fin de celle-ci. Les informations devront aussi être disponible sur une page Web interfaçable avec le site de l'association. Les objectifs à atteindre sont détaillés ci-dessous :

- Elève 1 : Quentin SIRUGUET

Réaliser un système de prise de temps intermédiaires qui devra :

- Déclencher une prise de vue du coureur lors d'un passage (d'un ou plusieurs coureurs).
- Effectuer une reconnaissance du numéro du dossard du ou des coureurs passant devant la prise de temps.
- Mettre en place un horodatage du coureur ainsi que la mise en forme de celui-ci, qui devra être compatible avec le système de chronométrage.
- Faire une sauvegarde locale des photos et des vidéos du passage des coureurs.

- Elève 2 : Tony WULLEPIT

Mise en place d'une application de suivi Android qui devra :

- Enregistrer la position GPS d'un coureur.
- Enregistrer en local le parcours horodaté.
- Envoyer le parcours au système de gestion de la base de données (SGBD).
- Donner au coureur et au serveur les informations sur le déroulement de la course (classement actuel, temps, position…).
- Détecter « éventuellement » la chute d'un coureur.

- Elève 3 : Adam VU

Création d'un site Web de suivi et d'une base de données qui devront :

- Permettre le suivi d'un coureur sur une page d'affichage (site Web).
- Permettre le suivi de la course avec tous les points de passages (site Web).
- Récupérer les informations de la course pour les afficher sur le site Web (BDD).

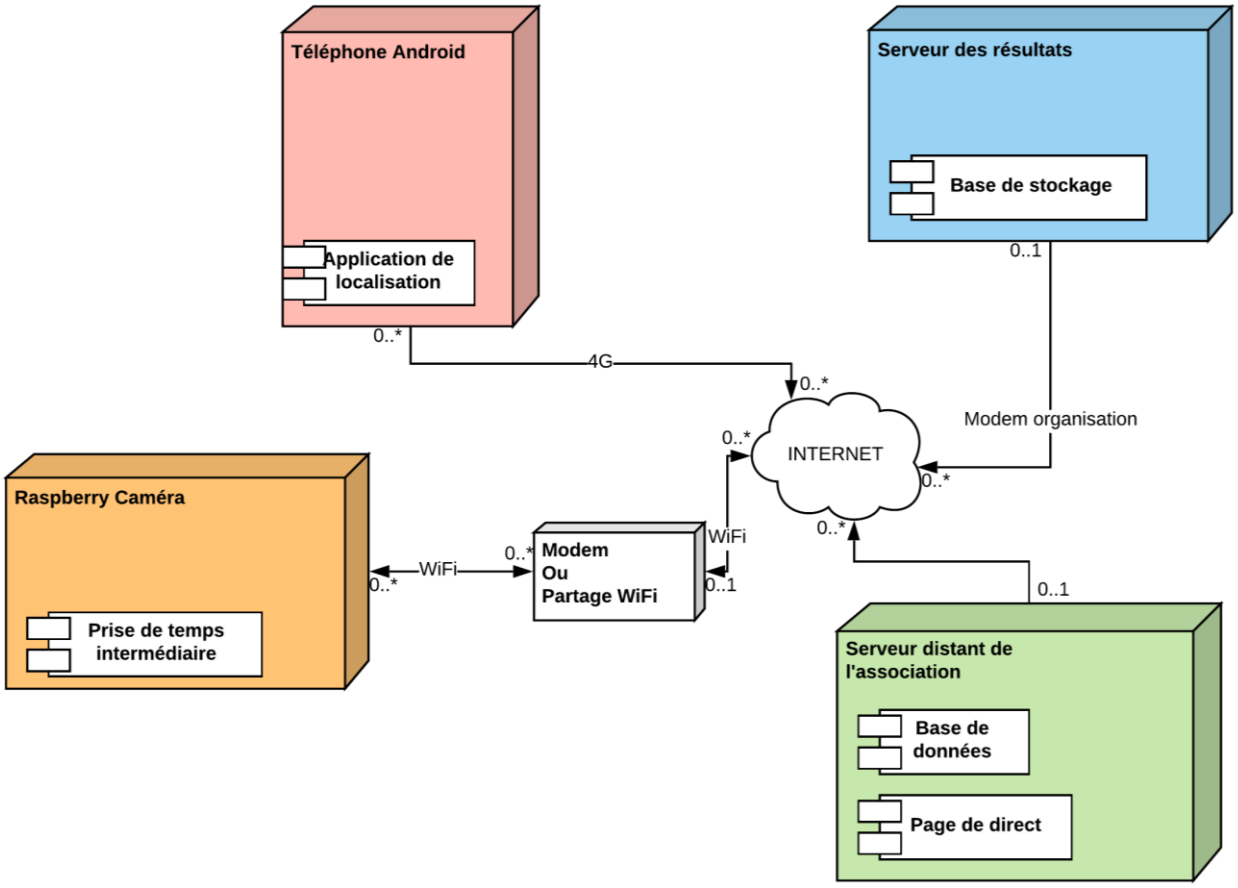
- Elève 4 : Matteo PITZOLU

Mise en place d'une communication réseau et d'un serveur de résultat qui devront assurer :

- La communication entre tous les points de prise de temps intermédiaires.
- Récupérer les informations des temps intermédiaires sur les systèmes distants (point de prise de temps et application de suivi) ainsi que centraliser ces informations.
- Récupérer les vidéos sur les systèmes distants (point de prise de temps).
- Création des fichiers utilisables par le système de chronométrage ainsi qu'une génération de mails automatique contenant tous les résultats d'un coureur et de la course.

Analyse UML

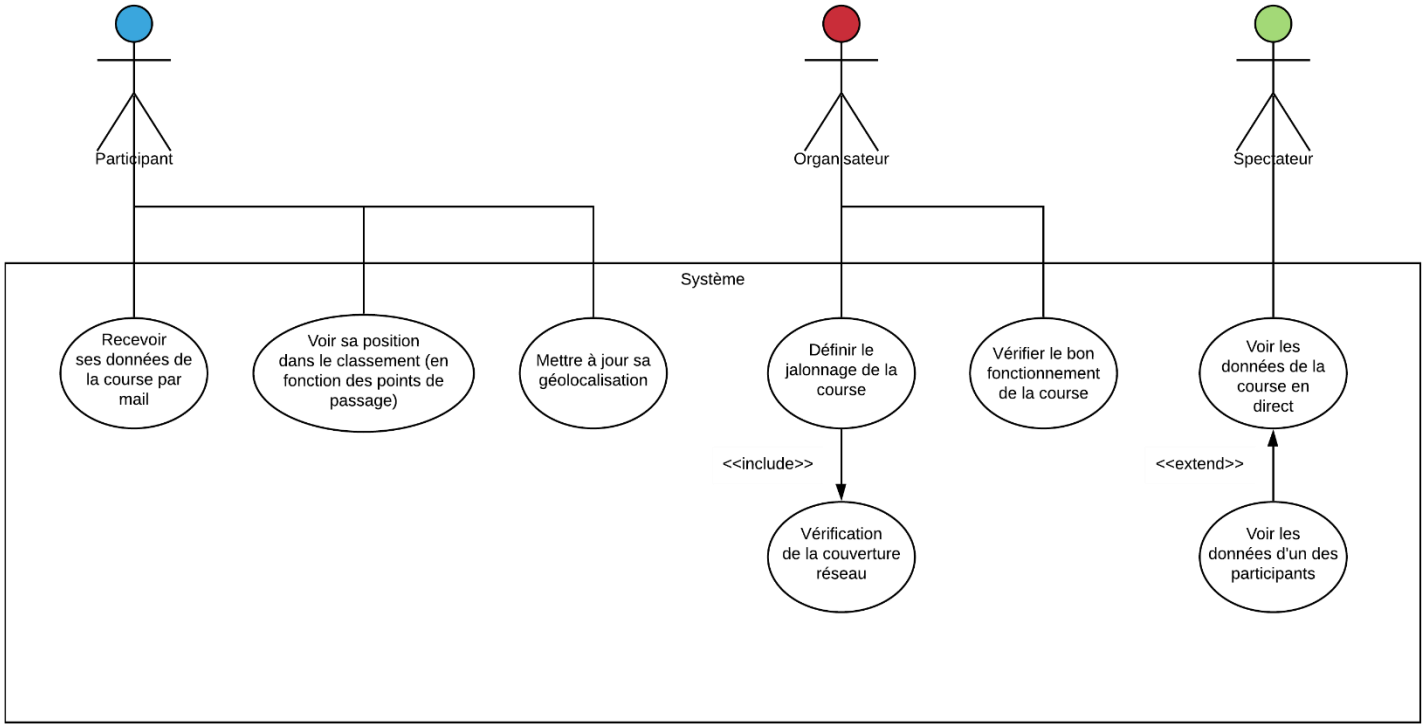
- Diagramme de déploiement



Le diagramme de déploiement permet de représenter l'infrastructure physique du système mise en place pour développer la solution demandée. Il permet aussi de réaliser que la totalité du projet est relié par Internet. Ainsi, chaque partie de celui-ci, est mise évidence par un code couleur correspond à chaque élève :

Couleur	Elève
Jaune	Quentin SIRUGUET
Rouge	Tony WULLEPIT
Vert	Adam VU
Bleu	Matteo PITZOLU

- Diagramme de cas d'utilisation



Le diagramme de cas d'utilisation permet d'obtenir une vision globale du comportement fonctionnel du système. Ainsi, on y retrouve les différentes actions réalisables par les utilisateurs du système mis en place.

Il est ici composé de trois acteurs différents, chacun ayant accès à différentes fonctions.

- Le participant est l'acteur muni de l'application sur son smartphone Android.
- L'organisateur (ou administrateur) est l'acteur qui s'assure du bon fonctionnement de la course.
- Le spectateur est l'acteur qui n'a aucun impact sur la course (si l'on ne prend pas en compte les supporters qui ont bien évidement un impact moral) mais qui peut tout de même assister à celle-ci et voir les résultats.

Plan qualité logiciel

- Outils de développement utilisés

Outils	Utilisation	Fonction
Microsoft Word 2019	Documentation	Editeur de texte
Visual Studio 2019	Développement	IDE C++ / C
Android Studio	Développement	IDE Java
Magic Draw	Modélisation UML	Modélisation de diagramme
Putty	Contrôle à distance Raspberry Pi	Client SSH
Analyse SI	Création de base de données	Modélisation BDD
Win Ipico	Système de chronométrage	Edition de fichier spécifique
GmCAP	Système de chronométrage	Edition de fichier spécifique
Trello	Gestion et Communication	Gestion de projet
Lucid Chart	Modélisation UML	Modélisation de diagramme
Raspbian Buster	Système d'exploitation	O.S de la Raspberry Pi

- Matériels utilisés

Matériels	Utilisation
Cartes Raspberry Pi 3	Gestion de la prise de vue
Point d'accès Wi-Fi Routeur	Simulation d'un partage de connexion
Clé USB	Support de stockage
PiCAM	Caméra pour prise de vue

PARTIE INDIVIDUELLE

Description de la partie personnelle

Ma partie concerne la prise de temps intermédiaires.

Je dois d'abord rendre facile l'identification de la caméra par un ID présent dans un fichier à la racine de la carte SD (« /piCamNumberFile.txt »).

Je dois ensuite, au passage d'un ou plusieurs coureurs, enregistrer une vidéo de dix secondes dont cinq secondes avant le passage.

Une fois la vidéo prise, je dois récupérer l'image au centre de la vidéo (qui correspond donc à la détection d'un mouvement) pour détecter un ou plusieurs dossards. Si un dossard est détecté, je dois enregistrer avec un nom précis (CAMERAID-jj-mm-aaaa-hh-mm-ss.avi) la vidéo dans un dossier correspondant au numéro du dossard.

Pour finir, je dois avertir le serveur de résultats qu'une vidéo est disponible en lui envoyant une trame précise (<D>DOSSARD-CAMERAID-jj:mm:aaaa_hh:mm:ss<F>).

Ressources

Certaines ressources sont disponibles sur le GitHub suivant :

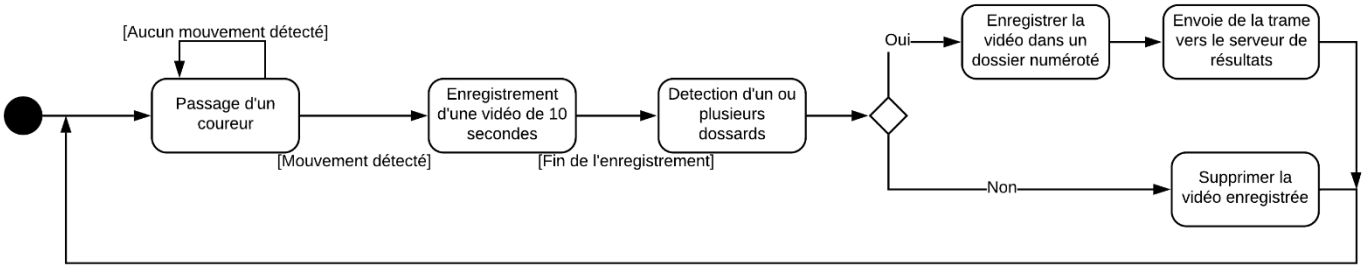
https://github.com/ghyfodia/Trail

Ressources présentes :

- Codes Sources
- Programme « PriseDeVue.out » (WIP)
- Ce document

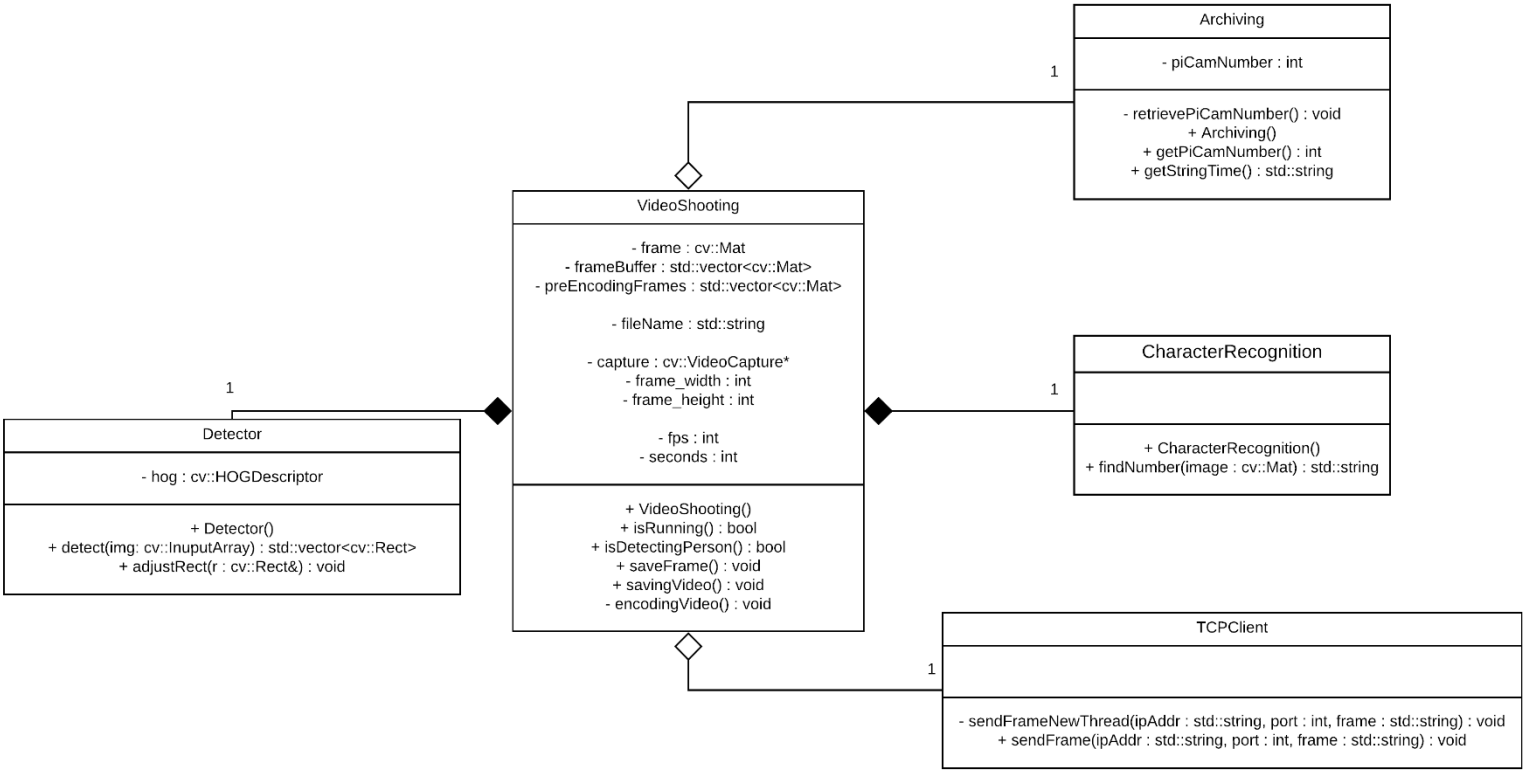
Analyse UML

- Diagramme d'état



Le diagramme d'état ci-dessus permet d'observer les différents états dans lesquels le système est présent dépendant du moment. Ainsi, cette représentation permet de comprendre la façon dont le système est développé.

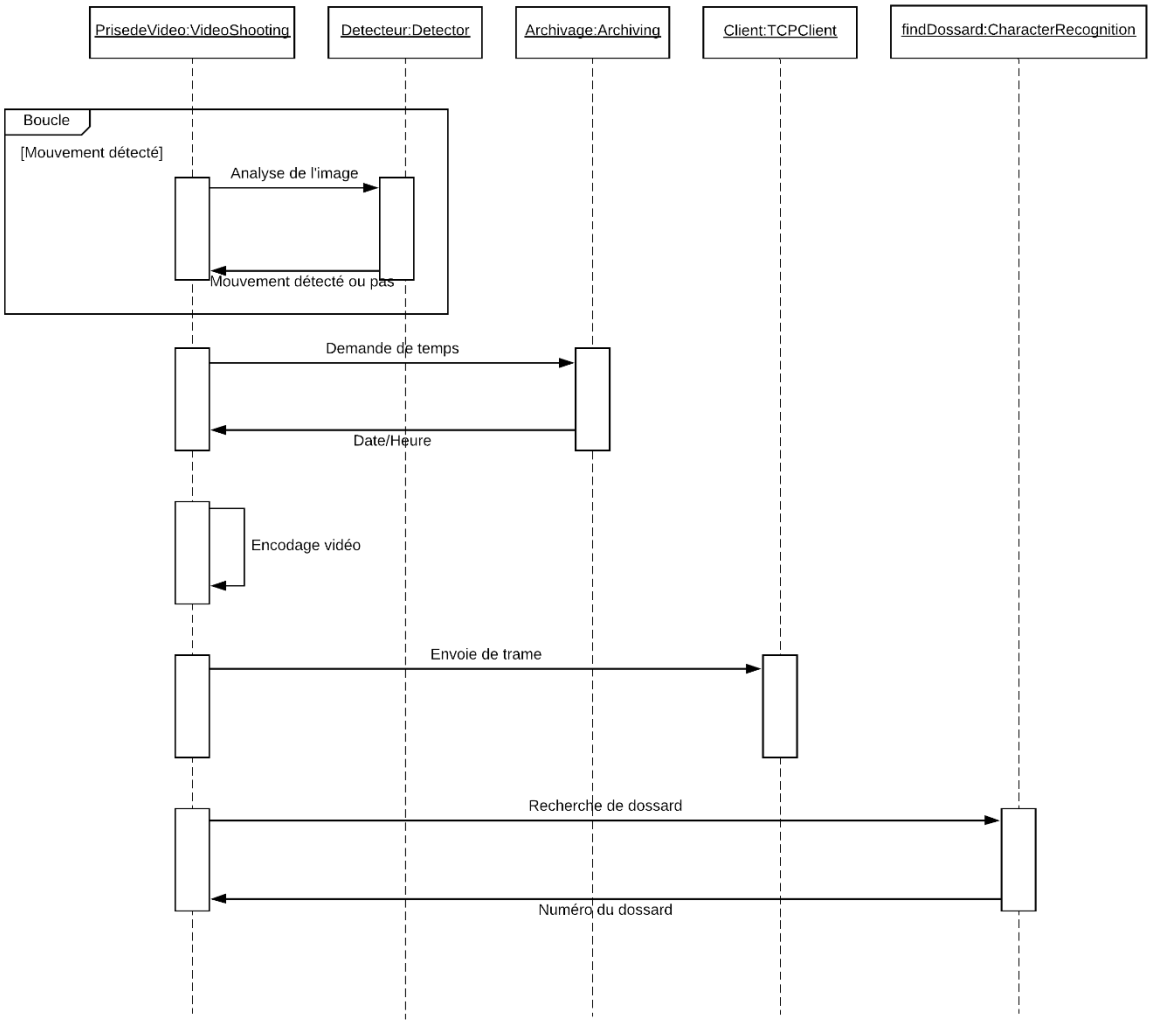
- Diagramme de classe



Le diagramme de classe ci-dessus permet de présenter les différentes classes et leurs relations présentes dans le code source.

On retrouve au centre la classe principale qui s'occupe de la gestion du système dans son ensemble.

- Diagramme de séquence



Le diagramme de séquence ci-dessus permet de représenter l'intégralité des interactions entre les différents acteurs du système.

INSTALLER OPENCV 4.1.0 SUR RASPBERRY PI

- Dans le terminal :

```
wget https://github.com/sol-prog/raspberry-pi-opencv/releases/download/opencv4rpi2.1/opencv-4.1.0-armhf.tar.bz2
```

- Récupérer l'archive, puis l'extraire :

```
tar xvf opencv-4.1.0-armhf.tar.bz2
```

- Déplacer le dossier vers /opt :

```
sudo mv opencv-4.1.0 /opt
```

- Supprimer l'archive :

```
rm opencv-4.1.0-armhf.tar.bz2
```

- Installer toutes les librairies nécessaires :

```
sudo apt install libtiff-dev zlib1g-dev
sudo apt install libjpeg-dev libpng-dev
sudo apt install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt install python-numpy python3-numpy
sudo apt install git
sudo apt install libgtk-3-dev libcanberra-gtk3-dev
```

- Ajouter OpenCV au system library path :

```
cd ~
echo 'export LD_LIBRARY_PATH=/opt/opencv-4.1.0/lib:$LD_LIBRARY_PATH' >> .bashrc
. .bashrc
```

- Créer les liens symboliques pour python :

```
sudo ln -s /opt/opencv-4.1.0/lib/python2.7/dist-packages/cv2 /usr/lib/python2.7/dist-packages/cv2
sudo ln -s /opt/opencv-4.1.0/lib/python3.7/dist-packages/cv2 /usr/lib/python3/dist-packages/cv2
```

- Récupérer la configuration d'OpenCV sur Github :

```
git clone https://gist.github.com/sol-prog/ed38347487295808 1985de733eaf352d opencv_cpp_compile_settings
cd opencv_cpp_compile_settings/
sudo cp opencv.pc /usr/lib/arm-linux-gnueabihf/pkgconfig
cd ~
rm -rf opencv_cpp_compile_settings/
```

Pour lancer le programme par défaut, une dernière étape est nécessaire.

- Télécharger le programme « PriseDeVue.out » sur la RaspberryPi.
- Ouvrir le rc.local :

```
sudo nano /etc/rc.local
```

- Ajouter à la fin (avant le « exit 0 ») :

```
sudo /home/pi/PriseDeVue.out
```

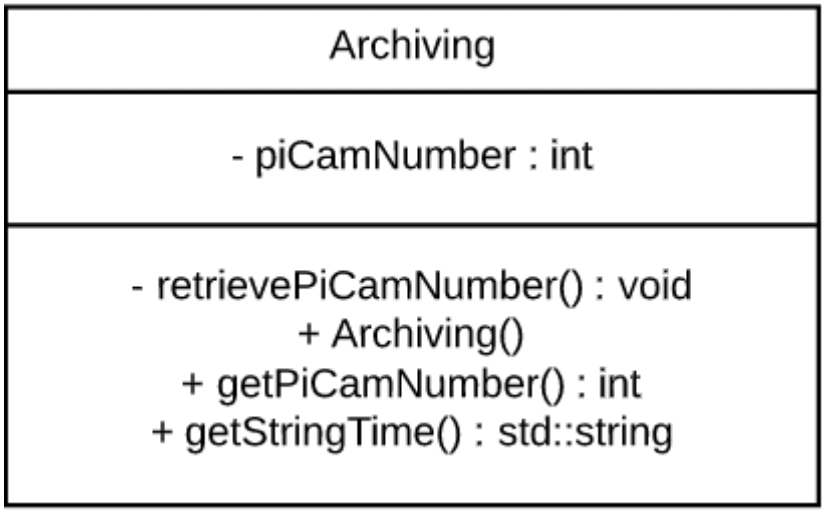
De cette manière, le programme se lancera automatiquement à chaque redémarrage de la Raspberry Pi.

Il est aussi important d'ajouter un fichier texte à la racine de la carte SD pour y ajouter l'ID de la caméra :

```
sudo nano /piCamNumberFile.txt
```

Et ajouter un nombre (seulement un nombre) qui correspondra donc à l'ID de la Raspberry Pi.

- Récupération de l'heure et d'autres informations.



La classe « **Archiving** » permet de récupérer l'heure et la date, ainsi que le numéro de la Raspberry Pi.

En effet, chacune des Raspberry Pi doit être identifiable par un numéro lors de l'envoie de trame vers le serveur de résultat. Ainsi, lorsque la classe « **Archiving** » instancie un objet, elle appelle la méthode « **retrievePiCamNumber()** » qui permet de récupérer dans le fichier « piCamNumberFile.txt » disponible à la racine de la Raspberry Pi, le numéro de celle-ci.

La méthode « **getPiCamNumber()** » permet donc de récupérer ce numéro.

Il est nécessaire pour envoyer la trame et pour enregistrer la vidéo, de connaître l'heure où le coureur est passé. La méthode « **getStringTime()** » permet de récupérer cette heure facilement sous forme de std::string.

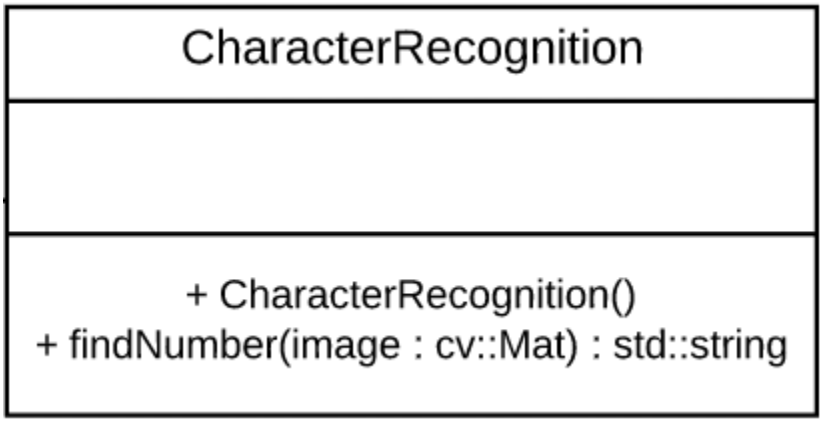
- Envoie d'une trame contenant les informations de la vidéo

TCPClient
- sendFrameNewThread(ipAddr : std::string, port : int, frame : std::string) : void + sendFrame(ipAddr : std::string, port : int, frame : std::string) : void

La classe « **TCPClient** » permet d'avoir un léger client TCP permettant ainsi l'envoie de trame. Celle-ci possède deux méthodes, dont « **sendFrame()** » qui permet seulement de lancer la seconde sur un nouveau thread non bloquant. La lancer sur un thread non bloquant permet de continuer à détecter le passage des coureurs.

La méthode « **sendFrameNewThread()** » permet donc d'envoyer au serveur de résultats qu'une nouvelle vidéo est disponible (si un dossard est détecté) dans le dossier indiqué (« frame : std::string »).

- Détection du dossard

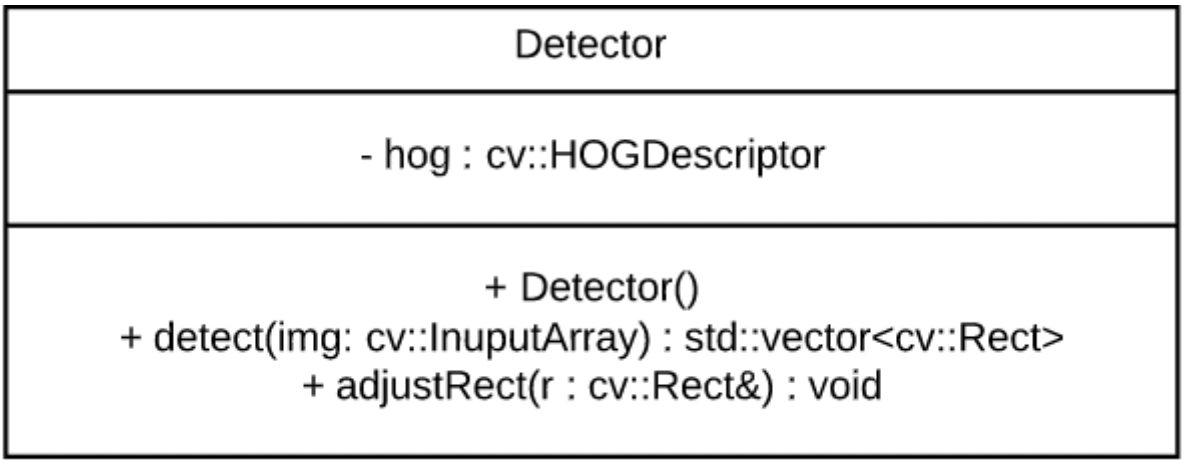


La classe « **CharacterRecognition** » est celle qui va tenter de détecter sur une image, un ou plusieurs numéros de dossard.

Son constructeur permet d'initialiser différents objets permettant la détection de celui-ci comme le « Tesseract ocr » par exemple.

La méthode « **findNumber()** » est celle qui détecte si un dossard est présent sur l'image. Elle renvoie sous forme de texte, soit le numéro du dossard si elle l'a trouvé, soit le texte « null » (cela permet de bloquer l'envoie d'une trame vers le serveur de résultats et de supprimer la vidéo pour ne pas encombrer l'espace de stockage).

- Détection d'un mouvement



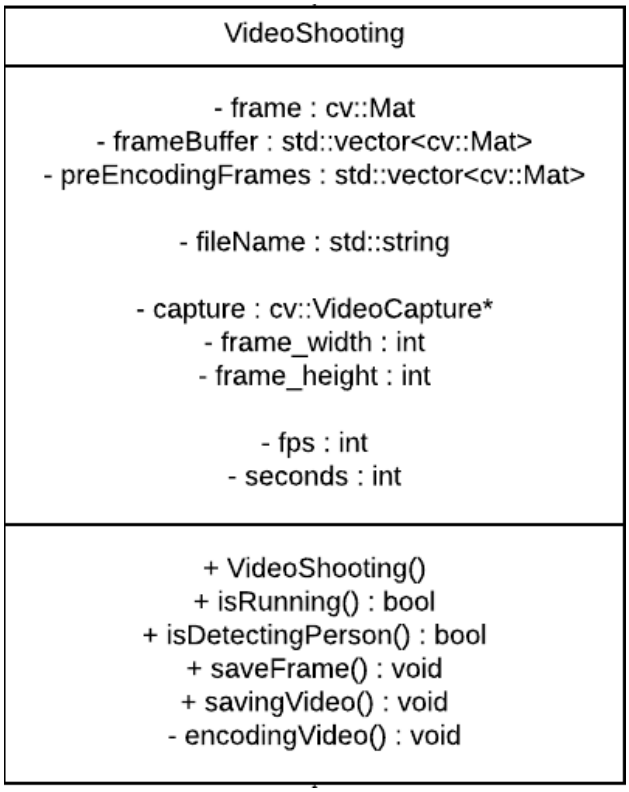
La classe « **Detector** » permet à l'aide d'un histogramme de gradient orienté (HOG) de détecter le passage d'un coureur.

Son constructeur est seulement présent pour initialiser le HOG.

La méthode « **detect()** » permet de détecter sur l'image un ou plusieurs coureurs et de renvoyer des rectangles entourant les objets. En sachant la taille du vector, on connaît le nombre de personnes (ou autre parfois par manque de précision) détecter.

Ainsi, la méthode « **adjustRect()** » permet de revoir ces rectangles pour les rendre plus précis.

- Gestion de la prise vidéo



La classe « **VideoShooting** » est au centre du programme. Elle gère tout l'aspect de l'encodage vidéo ainsi que des appels des autres composantes du programme.

Pour fonctionner correctement un buffer d'image a été inclus dans cette classe. Il permet, quand une vidéo de 10 secondes est enregistrée, d'avoir 5 secondes avant le passage du coureur et 5 secondes après.

Le constructeur de la classe permet d'initialiser la prise de photo et de remplir le buffer.

La méthode « **isRunning()** » retourne un booléen sur l'état de la piCam (si elle à planté, le programme s'arrête).

La méthode « **saveFrame()** » permet de remplir le buffer d'image et donc de supprimer la première image du vector pour en ajouter une nouvelle à la fin, cela permet d'avoir toujours l'image la plus « vieille » en première position et la plus récente en dernière position.

La méthode « **isDetectingPerson()** » permet, grâce à la classe « **Detector** », de détecter si une personne est présente. Elle renvoie un booléen en fonction d'une présence ou non.

La méthode « **savingVideo()** » est appelé dans le cas où une personne est détecté. Elle gère la création d'un nouveau thread bloquant utilisant la méthode « **encodingVideo()** ».

La méthode « **encodingVideo()** » est celle qui va créer un fichier vidéo grâce aux images du buffer. C'est aussi elle qui fait appel aux classes « **Archiving** » (pour créer le nom du fichier), « **TCPClient** » (pour envoyer la trame) et « **CharacterRecognition** » (pour détecter un dossard).

Planification et Réalisation des tests

Le code couleur est le suivant :

- Vert – Fonctionnel
- Orange – Non Fonctionnel et à Corrigé
- Rouge – A testé

- **Pour tester le système, j'ai fait le choix de tester fonctionnalité par fonctionnalité, en commençant par « l'archivage », c'est-à-dire la classe qui s'occupe de récupérer l'identifiant de la Raspberry Pi ainsi que la date et l'heure :**
  - Il n'y a aucune protection sur la lecture du fichier « piCamNumberFile.txt ». Cela signifie que si un autre caractère qu'un nombre est rentré, l'ID peut potentiellement avoir un aspect non attendu.
  - En revanche, si le fichier est illisible ou inexistant, l'ID vaudra simplement 0.
  - Si un nombre est rentré, il est correctement récupéré.
  - L'heure est correctement récupérée.
  - Elle peut en revanche être déréglée si le routeur est mal configuré (ne dépend donc pas du programme).
- **Pour tester la fonctionnalité d'envoie de trame, il faut tester la classe « TCPClient » :**
  - L'envoie de la trame est correctement réalisé si le serveur de réception est fonctionnel.
  - En revanche, si le serveur est indisponible, il arrive que le programme s'éteigne soudainement.
- **La fonctionnalité de détection d'une personne est légèrement plus compliquée à réaliser que les autres fonctionnalités testées ci-dessus, les résultats finaux en sont donc impactés :**
  - La manière la plus simple pour tester le taux de précision de cette fonctionnalité est de la mettre à l'épreuve en testant sur un grand nombre d'images différentes.
  - Des nombreux essaies réaliser en cours, j'en retient qu'environs 40 à 50% du temps, il arrive à correctement détecter la personne.
- **De la même manière, la fonctionnalité de détection du dossard ne semble pas vraiment précise.**
  - Il n'est pour le moment possible que détecter un seul dossard sur une image.
  - Le taux de précision doit être légèrement plus élevé que la détection d'une personne.
- **Pour finir, la classe « VideoShooting » encodant les vidéos semble rencontré beaucoup moins de problème que les autres :**
  - La vidéo est correctement encodée à chaque fois qu'il est nécessaire de le faire.
  - Les images ne sont pas toujours reçues de la caméra, cela créer donc parfois des accélérations de la vidéo.

Critique de la solution

La solution comporte actuellement de nombreux défauts, mais sont parfois plutôt simple à corriger.

Pour le problème de programme qui s'éteint quand le serveur est indisponible, il suffit simplement d'ajouter la gestion des exceptions (« try/catch » en C++) qui permettra probablement d'empêcher ce « crash ».

De la même manière, il est simple de corriger l'accélération de la vidéo. Quand la caméra piCAM n'a pas réussie à récupérer une image, la fonction d'OpenCV nous prévient. Pour corriger le problème il suffit donc que quand cela intervient, on duplique la dernière image du buffer. Cela réduit la fluidité de la vidéo mais évite l'accélération (c'est mieux !).

Pour la détection d'une personne et d'un dossard, il me semble qu'il n'y a qu'un moyen simple pour augmenter artificiellement la réussite de celle-ci. Il suffit de multiplier le nombre d'image tester. Donc au lieu de ne prendre que l'image du milieu pour tenter de trouver le dossard, on en prend tant qu'un dossard n'a pas été trouvé (avec une limite pour éviter de chercher dans toute les images de la vidéo).