

PROJET BUREAU D'ÉTUDE TWIZY

2019



SOMMAIRE :

- Introduction projet Twizy
- Gestion de projet
- Méthodes principales pour le traitement d'images
- Optimisation du programme
- Conclusion

EL HAIR Oussama
GARRIGUES Adrien
MANICOM Sandyla
RAZAFINDRABE Diana



Introduction projet Twizy

L'objectif du bureau d'étude Twizy est de proposer une nouvelle technologie qui se veut être utile aux utilisateurs de la route.

L'application qu'on propose permettra de détecter les panneaux de signalisation lorsque la voiture roulera à l'aide d'une caméra de type GoPro installée sur le haut du pare-brise ainsi que d'une interface ludique qui indiquera à l'utilisateur le panneau détecté. L'utilisateur pourra alors adapter son comportement face à cette nouvelle information.

Pour réaliser ce projet, nous avions à notre disposition une voiture électrique de type Twizy, une GoPro et des panneaux de signalisation.

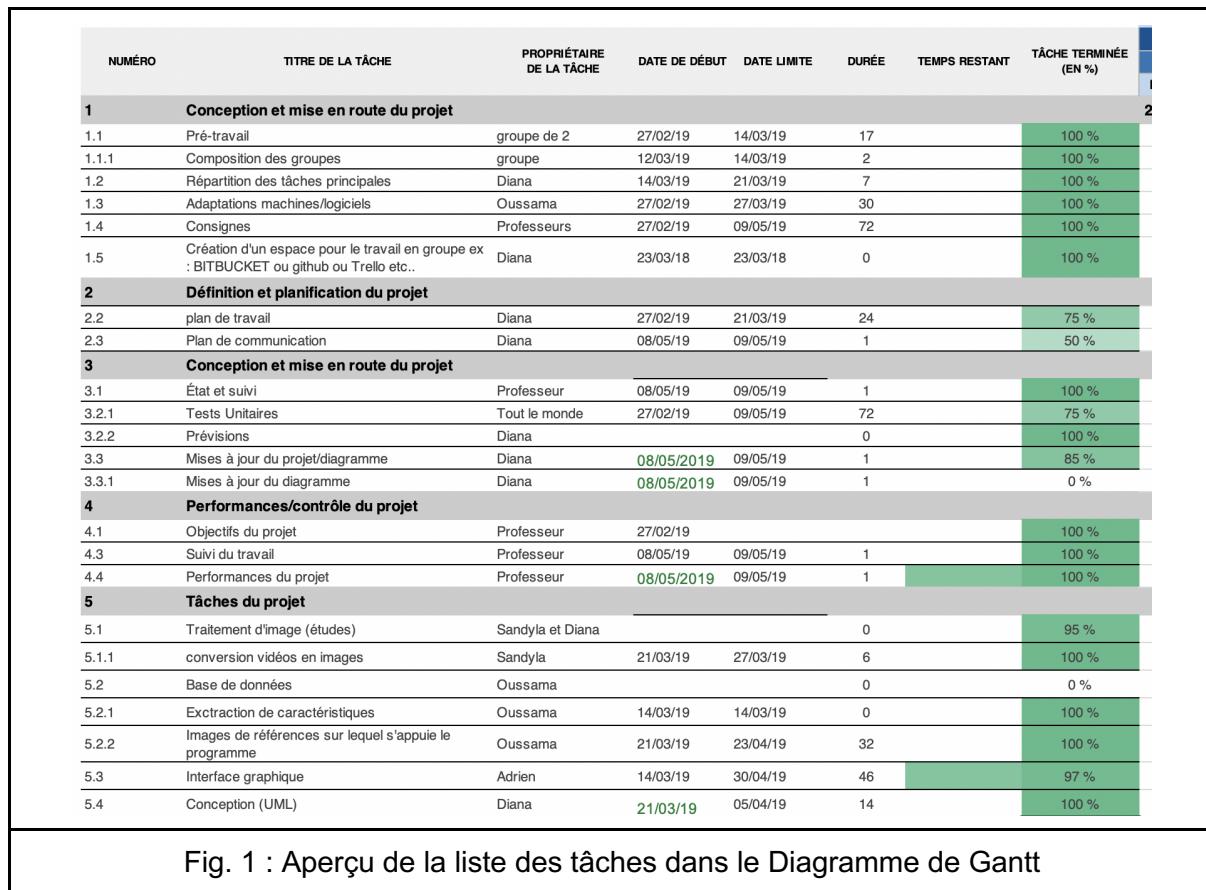
Gestion de projet

A. Déroulement du projet

Pour réaliser ce projet, nous nous sommes répartis les tâches au sein du groupe de manière homogène. Pendant les premières séances du projet Twizy, nous avons vu les différentes méthodes de traitement d'image avec Matlab puis Eclipse. C'est pourquoi, nous avons tous appris à manier la bibliothèque OpenCV à l'aide d'un tutoriel et d'exercices guidés pour faire les traitements d'image. Ces séances nous ont donc ouvert la voie pour réaliser notre projet.

Ensuite, le groupe a choisi l'attribution des différentes tâches. Nous avons créé un diagramme de Gantt (fig. 2) avec les listes de tâches, les deadlines et les personnes en charge de la tâche. Voici la répartition des tâches globales : Sandyla et Diana sont en charge de la partie traitement d'image, Oussama et Adrien se chargent de l'intégration. Adrien fait l'interface graphique, Oussama s'assure que le programme est compatible avec les vidéos, Sandyla fait le rapport et le guide d'utilisation et Diana fait la partie UML et gestion de projet.

Pour gérer le projet, nous avons mis en place un dépôt GitHub et un drive. Tout au long du projet, Diana était en charge de l'actualisation du suivi de projet. De plus, comme il s'agit d'un travail de groupe, nous avons pu nous entraider si une personne bloquait, la réalisation des tâches peut donc être considérée comme assurée par l'ensemble du groupe.



B. Problèmes rencontrés

Au cours de ce projet, nous avons également fait face à différents problèmes. En effet, c'est lors de la partie intégration qu'on a eu le plus de mal puisque suivant les versions d'openCV, notre programme ne fonctionnait pas toujours. Certains membres de l'équipe avaient travaillé sur une version d'openCV 2.4 alors que d'autres avaient travaillé sur une version d'openCV beaucoup plus récente à savoir une version 4.0.1. Et certains ordinateurs refusaient l'installation d'openCV 2.4. Après de nombreuses recherches, nous sommes enfin parvenus à résoudre le problème mais notre programme fonctionne uniquement avec la version d'OpenCV 2.4.

C. Diagramme de classes

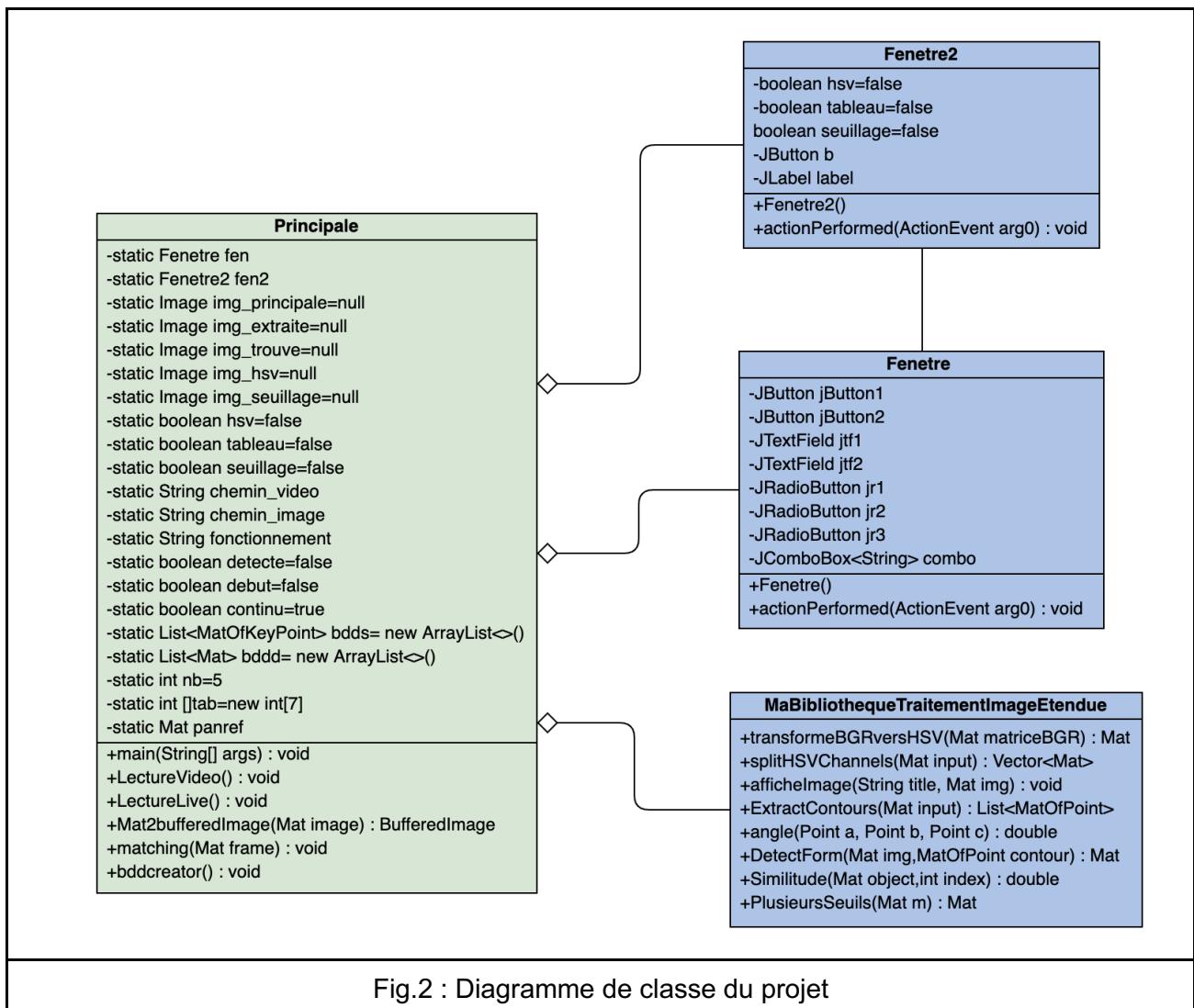


Fig.2 : Diagramme de classe du projet

Méthodes principales pour le traitement d'images

Tout le système utilisé dans ce projet est basé sur des notions de traitement d'images sous OpenCV permettant la détection de formes et l'extraction de *keypoints* et caractéristiques afin d'effectuer un matching avec des panneaux de référence. En premier lieu, ces différentes méthodes ont été testées sur des images avant d'être testées sur des flux vidéos puis ont été intégrées à une interface.

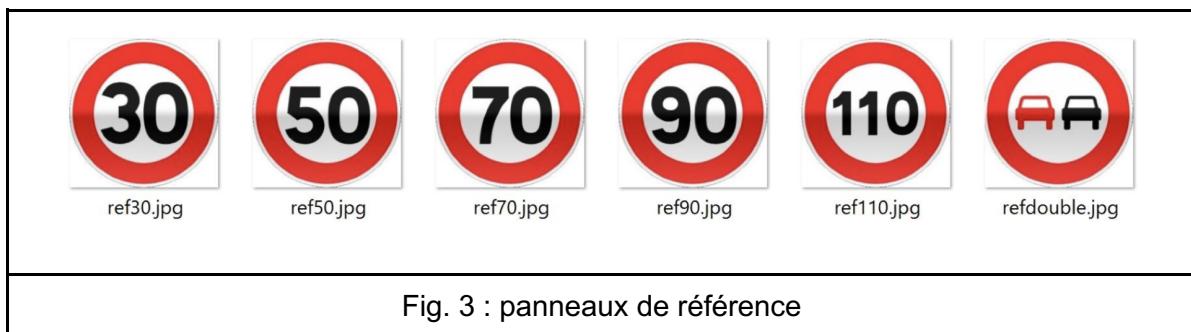
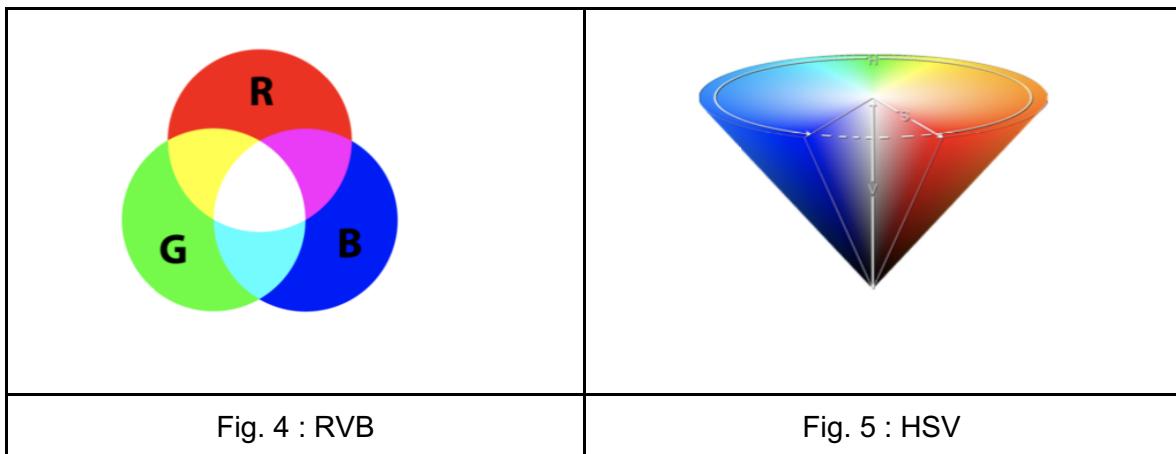
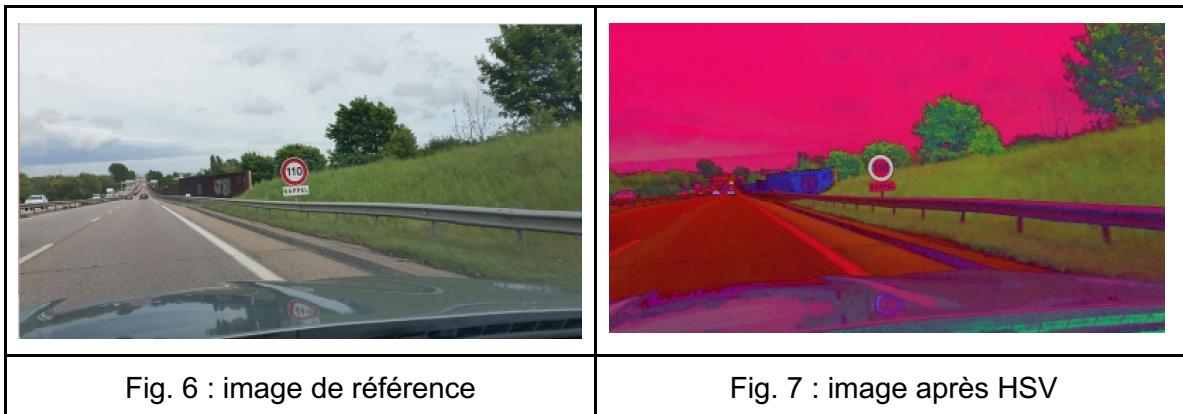


Fig. 3 : panneaux de référence

- Les couleurs

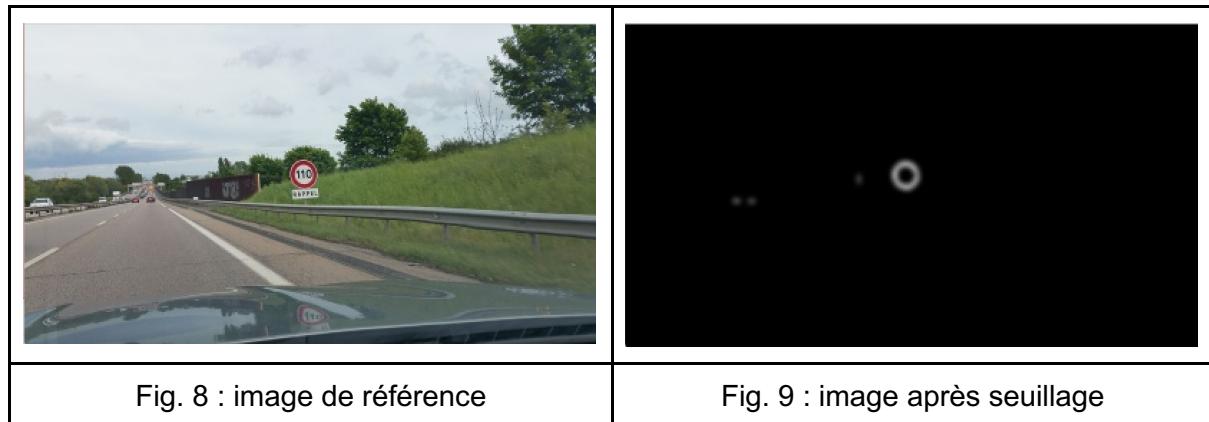
Tout d'abord, avant de faire la détection de panneaux, il faut traiter l'image qui est initialement sous format RVB au format HSV. L'image au format RVB est décomposé en 3 couleurs (rouge, bleu, vert) alors que l'image au format HSV est décomposé en 3 caractéristiques à savoir la teinte, la saturation et la valeur.





- [Le seuillage](#)

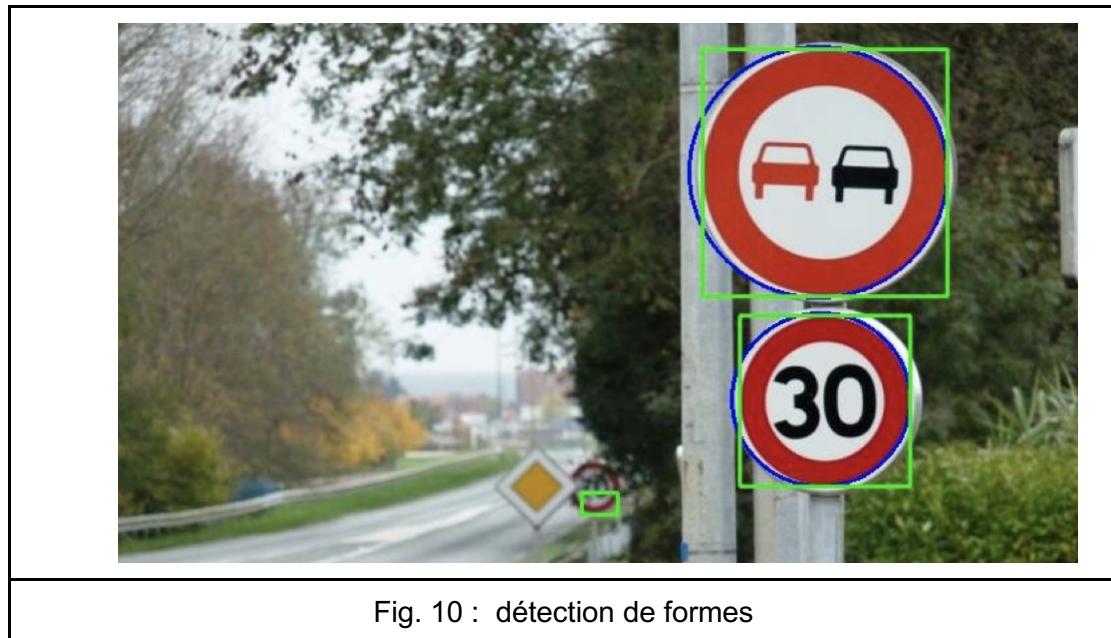
La méthode du seuillage va nous permettre de filtrer uniquement la couleur rouge présente sur l'image. On a adopté une méthode de seuils multiples pour avoir de meilleures performances et mieux détecter les panneaux. La détection de panneaux sera ainsi plus simple puisque chaque panneau présente le même contour rouge auquel on appliquera un filtre Gaussien pour des raisons de lissage.



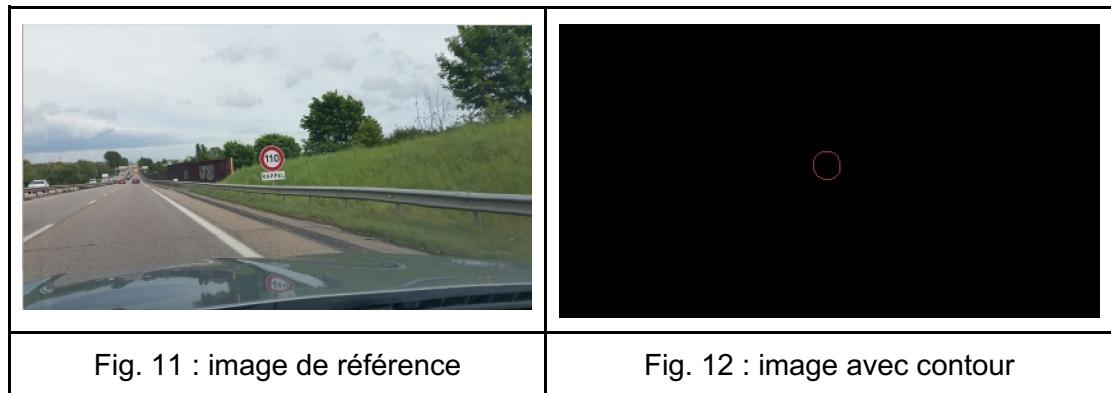
- [La méthode de détection de formes](#)

Nous considérons la forme comme un cercle que si l'air occupé par le contour détecté est supérieur à 85% de l'air occupé par le cercle parfait mais aussi si ce dernier dispose d'un certain rayon afin d'éviter de détecter des panneaux d'une assez longue distance qui peut être source d'erreur.

Par la suite on délimite notre panneau par un carré vert dessiné à l'aide des fonctions d'OpenCV ainsi qu'un cercle bleu du même rayon que le panneau détecté afin de pouvoir extraire et redimensionner l'image de notre panneau.



À partir de l'image seuillée, on extrait le contour de l'image et on obtient :



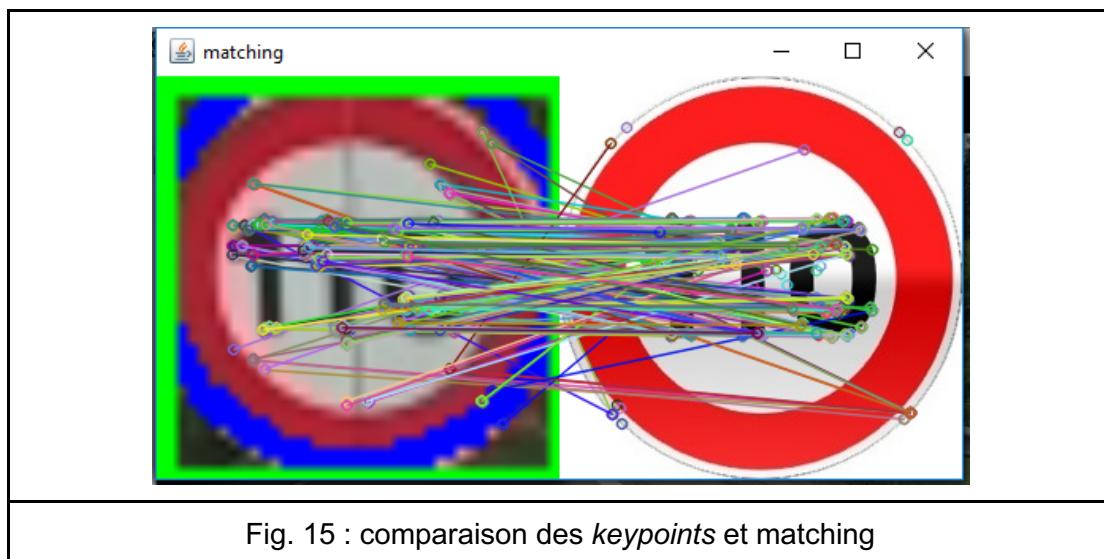
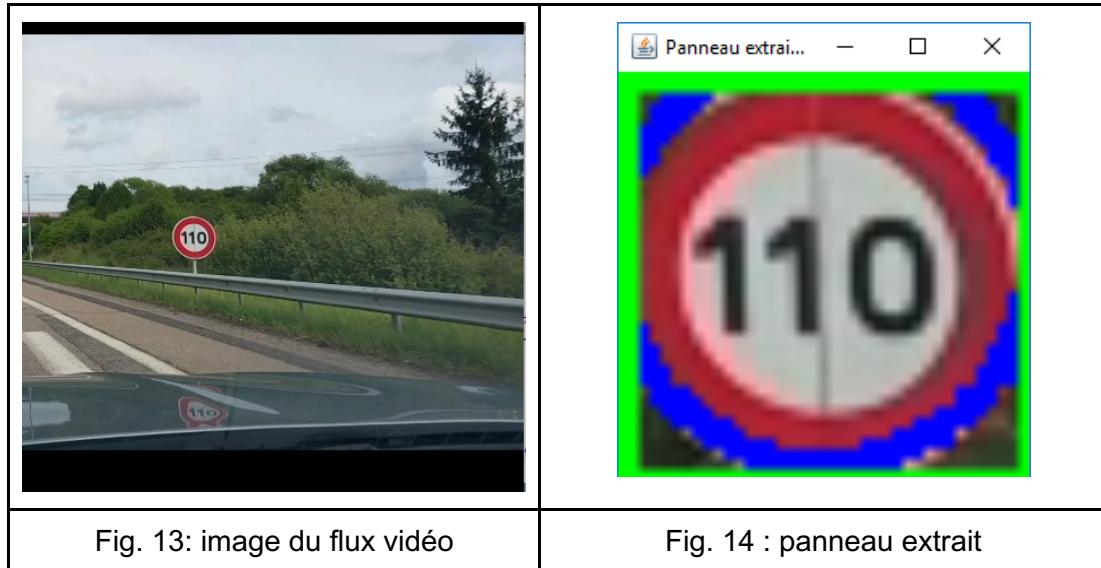
- La méthode du matching

La méthode du matching nous permet d'extraire les **keypoints** de chaque panneau de signalisation extrait selon les méthodes précédentes et aussi de comparer ces caractéristiques relevées avec celles de notre panoplie constituée de différents panneaux de référence.

Avant d'effectuer ce matching et cette comparaison des caractéristiques de notre panneau ainsi que celles du panneau de référence, on a eu besoin de redimensionner la taille de l'image du panneau extrait et de la normaliser pour qu'elle aie la même taille que l'image du panneau de référence.

Dans un souci d'optimisation on a fait le choix d'activer la méthode permettant d'effectuer le matching qu'à partir du moment où on détecte un panneau disposant d'un certain rayon permettant d'extraire une image assez nette et non pixelisée. Par la suite, et après la détection d'un panneau on lance notre méthode de matching avec les différents panneaux de références sur plusieurs *frames* tout en effectuant une moyenne afin d'avoir un résultat assez fiable et ne pas disposer de résultat faux-positif. Ainsi on dispose à la fois d'un système de

score qui nous permet de reconnaître un panneau que si il dispose d'un score assez élevé mais aussi d'un système de moyennes sur plusieurs pour déduire le bon panneau à partir de plusieurs déductions individuelles.



- Le fonctionnement global

Après avoir intégré ces différentes méthodes dans un même programme, on a été capables dans un premier temps de reconnaître et détecter des panneaux de signalisation se trouvant

sur des images fixes. L'objectif étant de reconnaître les panneaux de signalisation dans un cas réel et en temps réel, on a alors commencé à récupérer des flux vidéos pour pouvoir effectuer les premiers tests.

On a alors décomposé le flux vidéo en plusieurs *frames* (images) et on a appliqué notre programme de matching et de détection de panneaux sur chaque image.

Dans un souci d'optimisation on a adopté la démarche suivante :

on commence par extraire chaque image (*frame*) de notre vidéo et on la transforme en image HSV, par la suite on lui applique un seuillage afin de mettre en évidence notre panneau et appliquer un lissage sur l'image. On fait alors le choix d'activer la méthode permettant d'effectuer le matching qu'à partir du moment où on détecte un panneau disposant d'un certain rayon permettant d'extraire une image assez nette et non pixelisée.

Par la suite, après la détection d'un panneau, on lance notre méthode de matching avec les différents panneaux de références sur plusieurs *frames* afin d'avoir un résultat assez fiable et ne pas disposer de résultat faux-positif.

Optimisation du programme

Dans un soucis d'optimisation de notre programme, et pour permettre à notre application de reconnaître les différents panneaux sans pour autant perdre de la fluidité. On a mis en place plusieurs stratégies qui nous permettent à la fois de gagner en fluidité mais aussi de réduire

le nombre d'erreurs.

- **Distance de détection des panneaux**

Au cours de nos différents tests, on a remarqué que notre programme détectait les panneaux à une distance assez grande, ce qui avait pour cause de nous fournir des images de basse qualité et donc plus d'erreurs au niveau de l'extraction des caractéristiques et du matching avec nos panneaux de références. Pour remédier à ce problème on a choisi d'imposer à notre contour de disposer d'un certain rayon, qu'on a jugé assez suffisant pour avoir des résultats non erronés.

- **Système de moyennes**

Pour améliorer la reconnaissance des panneaux et diminuer le nombre d'erreurs, on a mis en place un tableau récupérant les scores sur chaque *frame* de la vidéo, et qui effectue une moyenne sur plusieurs *frames* pour pouvoir avoir le meilleur matching et donc le bon panneau.

- **Taille des images de références**

Puisqu'on récupère la taille des images de référence et qu'on met nos panneaux extraits à la taille de ces panneaux fournis, on perd de la qualité lors de ce redimensionnement. Pour résoudre ce problème lié à la perte de qualité de nos images extraites, on a choisi de diminuer la taille des images de références afin qu'elles soient du même ordre de grandeur des panneaux extraits.

- **Base de données**

Enfin, la majeure stratégie qui nous permet de garder un flux vidéo fluide tout en reconnaissant les différents panneaux en temps réel, est la création d'une base de données. La base de données ainsi créée au début du lancement du programme, extrait directement les différentes caractéristiques des panneaux de références et ainsi on n'a pas besoin de le faire à chaque fois qu'on lance notre méthode de matching. Ce qui nous permet d'avoir un certain gain de puissance de calcul qui est alloué à la lecture du flux vidéo et donc améliorer la fluidité.

Conclusion

Finalement, nous avons obtenu des résultats assez satisfaisants. Notre application fonctionne même si nous n'avons pas fait les tests dans différentes conditions météorologiques. Elle fonctionne néanmoins lorsque tous les paramètres sont favorables.

Il est certain que l'application que nous avons mis en place reste encore à être développée. En effet, tous les panneaux de signalisation n'ont pas été testés et toutes les situations n'ont pas été prises en considération.

Concernant la gestion du projet, nous n'avons pas eu de difficultés particulières. L'organisation du travail a été bien répartie et la communication était relativement aisée. Nous avons rencontré quelques soucis lors de l'intégration des parties dûs aux différentes versions d'openCV de chacun. Néanmoins, travailler avec cette bibliothèque fût très intéressante et nous sera utile dans notre futur métier d'ingénieur.

Le bureau d'étude Twizy nous a beaucoup apporté puisque nous avons pu entrevoir tout un pan du domaine du traitement de l'image ainsi que de nombreuses méthodes telles que le matching, la détection de contours, la détection de formes ou encore la transformation de l'image en HSV.

Lien prezi de présentation : <https://prezi.com/view/K8THKaSTrsygxKvRoAAm/>