Bureau d'étude TWIZZY

Détecteur de panneaux de signalisation

Abdelkader LAHMADI
Jean-Philippe MANGEOT
Steven LE-CAM
Maya KALLAS

MISE EN SITUATION

Les constructeurs automobiles proposent aujourd'hui des technologies dites anti-excès de vitesse permettant de lire automatiquement les panneaux de limitation de vitesse sur le bord des routes afin d'intégrer cette information sur le tableau de bord et/ou de la coupler avec les limiteurs automatiques de vitesse.

Ces systèmes ont commencé à être présents sur des véhicules haut de gamme (Ford S-MAX, Mercedes Classe S, Audi A8) et se démocratisent aujourd'hui sur un parc élargi de véhicules grand public.

*Figure 1:Le système « **Traffic Sign Assist »** de Mercedes est intégré au véhicule*

Figure 2:Caméra multifonctions permettant la détection de panneaux de signalisation (BOSCH MPC2)

Dans ces systèmes une caméra logée dans le haut du pare-brise, ou dans le rétroviseur permet de filmer l'avant du véhicule. La caméra fait partie intégrante du réseau de capteurs d'environnement et est utilisable avec les autres capteurs, tels que les capteurs radar et les capteurs à ultrasons.

Par ailleurs, dans un tout autre registre, il existe aussi des applications Android et iOS, qui fonctionnent également de manière autonome sur un smartphone attaché au pare prise du véhicule. Ces applications permettent aussi d'informer le conducteur sur la limite de vitesse en cours.

*Figure 2: L'application « **myDriveAssist** » de l'équipementier Bosch*

Tous ces systèmes sont basés sur un noyau logiciel de traitement d'images qui est capable de détecter et d'isoler les formes rondes caractéristiques des panneaux de signalisation pris en charge et même de détecter les panonceaux associés.

Le système qui fait l'objet de ce bureau d'etude est purement logiciel, il est composé d'une bibliothèque réalisant cette détection. Le noyau est codé en Java et utilise la bibliothèque Open CV 2.4.9.

Ce noyau sera d'abord testé en analysant des images fixes, puis en analysant en temps réel un flux vidéo préenregistré.

Présentation des fonctionnalités proposées par le noyau :

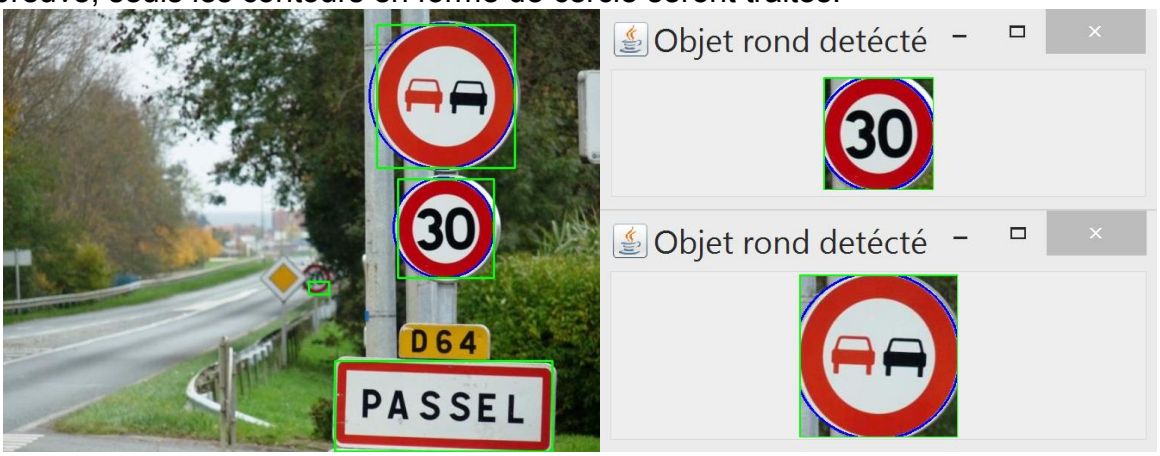
Les différentes étapes de la détection de panneaux sur une image fixe sont présentées ci-dessous.

Passage du codage de l'image de l'espace Bleu Vert Rouge (BGR) à l'espace Teinte Saturation Valeur (HSV)

Saturation des rouges

Détection de contours

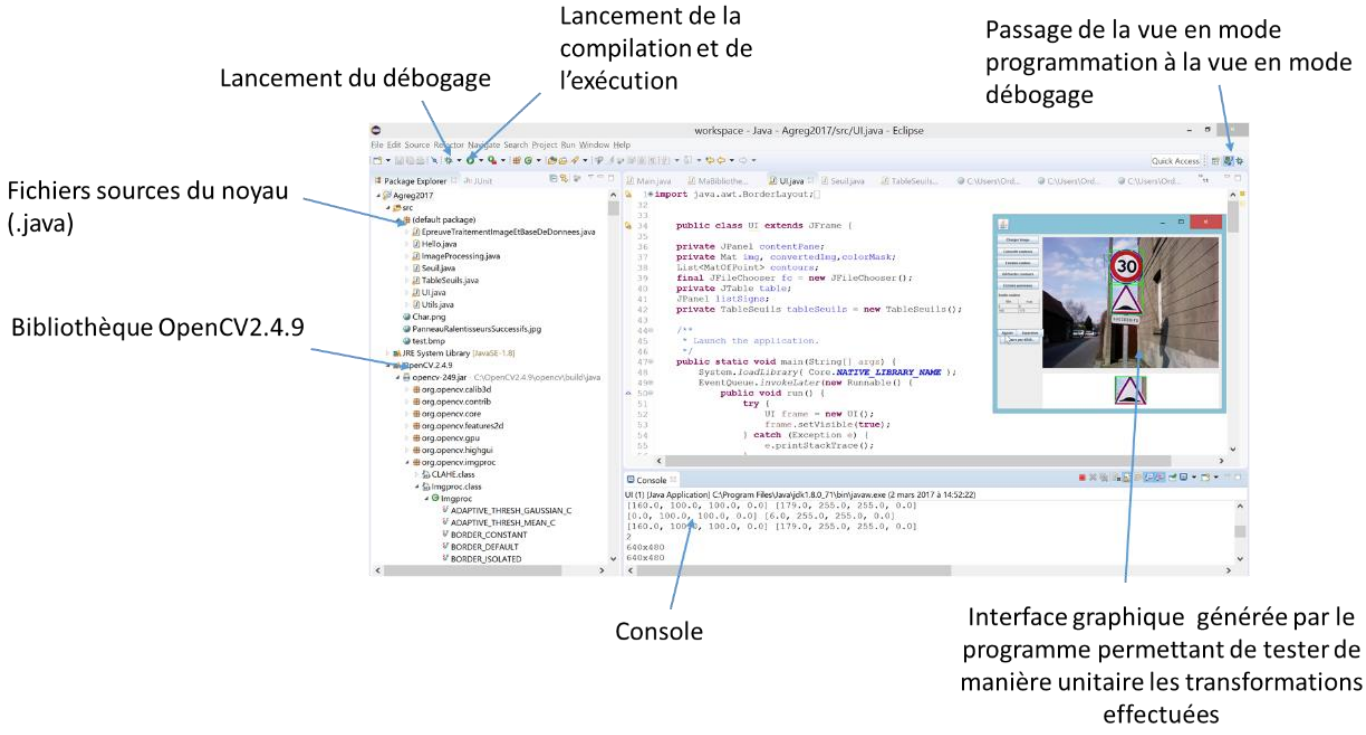
Une analyse de forme est ensuite effectuée sur les contours obtenus. Seuls les contours en forme de cercles, triangles et rectangles sont détectés. Dans cette épreuve, seuls les contours en forme de cercle seront traités.



Des images carrées enveloppant ces contours sont ensuite extraites de l'image originale. Elles seront ensuite traitées pour détecter le type de panneau. Six panneaux de références normalisés sont fournis pour faciliter cette identification.

Présentation de l'interface de développement:

L'environnement de développement du noyau de traitement d'image Eclipse Néon :



Activité 0 : (5 min)

Lire le code existant situé dans la classe « ***Activite0*** » du projet « ***PriseEnMainActivite0*** » :

- Exécuter ce programme en appuyant sur le bouton ***run*** (raccourci CTRL+F11). Ce programme montre l'objectif à atteindre à l'issue de cette épreuve, à savoir : détecter un panneau sur un flux vidéo, l'extraire de l'image et afficher dans la console la limitation de vitesse indiquée par le dernier panneau détecté.

Code source utilise une bibliothèque au format ***.jar***. Le code source de cette bibliothèque n'est pas accessible dans cette activité.

Activité 1 : (5 min)

Lire le code existant situé dans la classe « ***Activite1*** » du projet « ***PriseEnMainActivite1*** » :

- Exécuter ce programme en appuyant sur le bouton ***run***. Ce programme lit chaque pixel de l'image ***activite1.png*** et le reproduit dans la console avec une symbolique particulière :
  - le caractère « **+** » est utilisé si le pixel lu est de couleur (non blanc) ;
  - le caractère « **.** » est utilisé si le pixel lu est blanc ;
- Modifier le programme pour que la symbolique particulière soit :
  - le caractère « + » est utilisé uniquement si le pixel lu est rouge ;
  - le caractère « **.** » est utilisé pour tous les autres pixels lus;

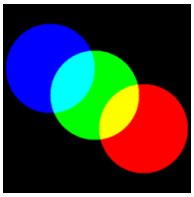
Image activite1.png : 

Activité 2 : (10 min)

Lire le code de la méthode situé dans la classe « ***Activite2*** » du projet « ***PriseEnMainActivite2***». Ce programme lit une image et la décompose en 3 canaux bleu, vert et rouge. La composante de chaque canal est ensuite affichée en niveau de gris dans 3 fenêtres graphiques java indépendantes.

- Exécuter ce programme ;
- Ce programme affiche 3 fenêtres, **modifier le programme pour qu'il n'affiche qu'une seule image**, en niveau de gris, correspondant à la composante verte de l'image originale.

Image activite2.png :

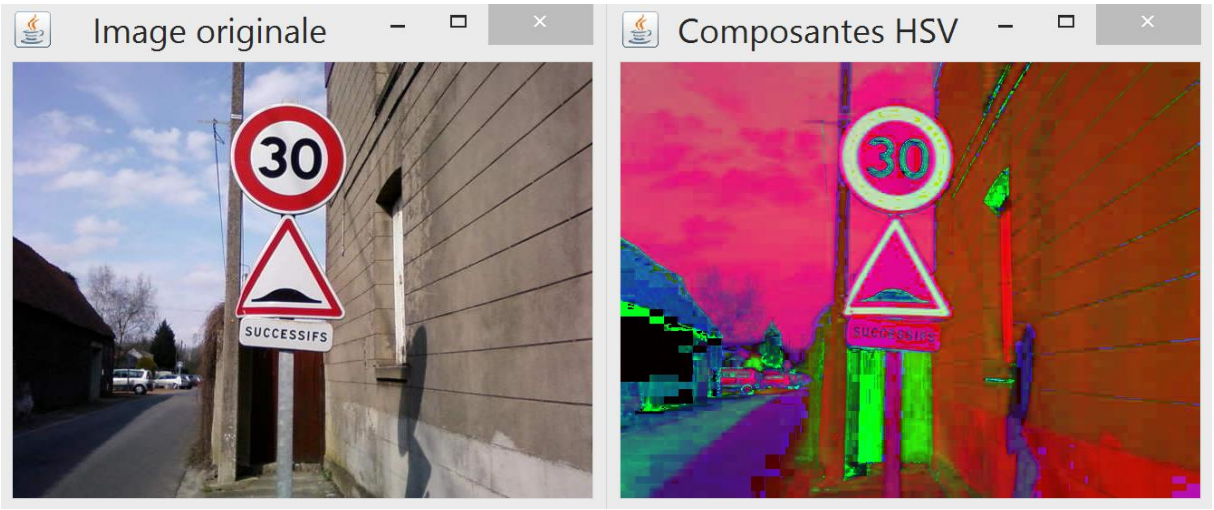


Activité 3: (10 min)

Lire le code de la méthode situé dans la classe « ***Activite3*** » du projet « ***PriseEnMainActivite***3» ainsi que les méthodes contenues la bibliothèque « ***MaBibliothequeTraitementImage*** ».

Le code fourni, n'affiche qu'une seule image. Le but de cette activité est de transformer cette image en composantes HSV (*Hue/Saturation/Value* ou *Teinte/Saturation/Valeur* en français)

- **Modifier** le code de la classe « ***Activite3*** » afin qu'il affiche 2 images : l'image originale ainsi qu'une image des composantes HSV dans 2 fenêtres java indépendantes comme ci-dessous :



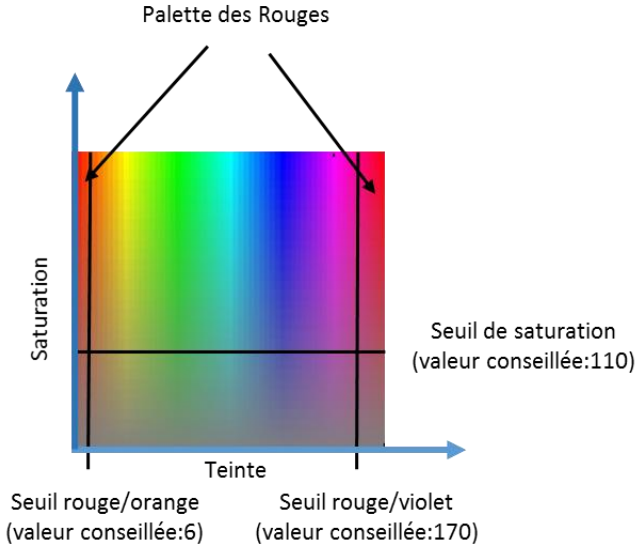
Première étude :

Objectif : Extraction des pixels rouges et création d'une image saturée

Dans cette partie, l'objectif est de créer, une image en noir et blanc qui ne conserve que les pixels rouges de l'image originale. Cette image devra être créée :

- avec des pixels blancs lorsque le pixel de l'image originale est dans la palette des rouges ;
- avec des pixels noirs lorsque le pixel de l'image originale n'est pas dans la palette des rouges.

Cette palette des rouges est encadrée par des seuils dans l'espace HSV. Comme le montre l'image ci-dessous représentant des couleurs en fonction des valeurs de teinte et de saturation, la particularité de la couleur rouge est qu'elle est présente à la fois pour des valeurs de teinte faible (inférieure à 6) et pour des valeurs de teinte élevée (supérieure à 170). Pour ne pas que le système détecte trop de gris, un seuil sur la valeur de saturation est aussi nécessaire (supérieur à 110).



A partir du squelette de code donné dans le projet ***« PremiereEtude*** », compléter la méthode

« ***Mat seuillage(Mat input, int seuilRougeOrange, int seuilRougeViolet,int seuilSaturation)***»

contenue dans la bibliothèque « ***MaBibliothequeTraitementImage.java*** » afin de réaliser cette opération sur l'image fournie « ***p1.jpg »***

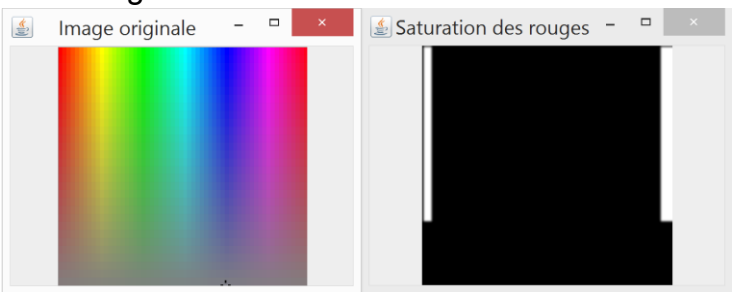
Vous pourrez choisir les méthodes de votre choix. Il pourra être utilisé toutes les fonctions proposée par la bibliothèque OpenCV dont la documentation en ligne est disponible ici :

http://docs.opencv.org/java/2.4.9/

La bibliothèque étant assez vaste, il est donné quelques indications ici:

- Créer des objets de type Scalar dans lesquels seront stockées les valeurs des seuils qui définiront les limites de la couleur rouge ;
- Utiliser des méthodes logiques matricielles du type :

  void Core.compare(Mat src1, Scalar src2, Mat dst, int cmpop)
  void Core.bitwise_or(Mat src1, Mat src2, Mat dst)
  void Core.bitwise_and(Mat src1, Mat src2, Mat dst)
- Tester le traitement sur l'image « ***temoin.png »*** présent dans le projet afin de valider l'algorithme et régler les seuils.



- Remarque : Une méthode « ***Mat seuillageExemple(Mat input, int seuilRougeViolet)***» est fournie dans la bibliothèque afin de permettre ne pas perdre de temps dans des problématique de syntaxe JAVA. C'est cette méthode à un seul seuil qui est lancée par défaut dans le squelette fourni.

Deuxième étude :

Objectif : Reconnaissance de formes et de symboles

Dans cette partie, l'objectif est de réaliser une détection de formes sur les panneaux extraits d'images fixes.

Cette partie est la suite logique de la première étude. Elle peut néanmoins être abordée sans que la première étude ait été achevée.

Données d'entrées :

Dans le projet intitulé « ***Deuxième étude*** » sont fournies 6 images de référence. : 5 images de panneaux de limitation de vitesse: 30km/h, 50km/h, 70km/h, 90km/h et 110 km/h ainsi qu'un panneau d'interdiction de doubler.

ref30.jpg

ref50.jpg

ref70.jpg

ref90.jpg

ref110.jpg


refdouble.jpg

Sont fournies également 10 images de test, numérotées de ***p1.jpg*** à ***p10.jpg***.

p1.jpg

p2.jpg


p3.jpg


p4.jpg


p5.jpg


p6.jpg


p7.JPG


p8.jpg


p9.jpg

p10.jpg

L'objectif est de cette partie est de développer une méthode qui permettra de reconnaitre un maximum de panneaux sur cet échantillon de 10 images.

Un squelette de programme est fourni. La détection des rouges, des contours ainsi que l'extraction du panneau dans l'image est déjà réalisé. Il est simplement demandé de compléter la méthode ***Similitude*** située dans la classe ***maBibliothequeTraitementImageEtendue.java*** . Son prototype est :

public static double Similitude(Mat object,String signfile)

- La matrice de type ***Mat*** appelée ***object*** est déjà fournie par le squelette de programme, elle correspond aux objets ronds et rouges détectés sur l'une des 10 images.;

- La chaine de caractère ***Signfile*** correspond aux noms des fichiers correspondant aux 6 panneaux de référence. Elle est déjà fournie par le squelette du programme.
- La variable de retour, de type double est la seule valeur à calculer. Elle doit être d'autant plus élevée que les 2 images passés en paramètre se ressemblent.

La décision finale de la valeur du panneau se base sur la comparaison du score obtenu pour le panneau détecté avec tous les panneaux de référence.

La classe ***maBibliothequeTraitementImageEtendue.java*** a été enrichie de 2 fonctions, dont le code source est visible. Elles permettent d'extraire les contours des panneaux de l'image ainsi que de détecter le type de forme géométrique.

Remarque : Pour calculer ce critère de ressemblance, aucune méthode n'est imposée (ou exclusif entre l'image obtenue et l'image de référence, méthode des moindres carrés, recherche et comparaison de points d'intérêts…).

Troisième étude :

Objectif : Intégrer du code et réaliser la détéction sur un flux vidéo.

Dans le squelette du projet « ***troisiemeEtude*** », modifier la classe « **AnalyseVideo.java** » fournie pour que le programme de telle manière à ce que le programme :

- Affiche les images des panneaux détectés dans des fenêtres java indépendantes ;
- Ecrive dans la console le nom du dernier panneau détecté (le programme ne doit pas afficher dans la console 2 fois le même panneau détecté).

2 fichiers d'entrée vidéo sont proposés : « ***video1.avi*** » et ***« video2.avi*** »

Le résultat attendu est similaire à ce qui est produit par le programme du projet « ***PriseEnMainActivite0*** »