

TP N°2 : Traitement d'images



ROB4 - Polytech Sorbonne

2023–2024

TEIXEIRA Pierre | LE LAY Louis

SOMMAIRE

SOMMAIRE.....	2
INTRODUCTION.....	2
PARTIE 1 - Détection des voies.....	3
a. Détection de la première frame.....	3
b. Conversion en espace de couleur HSV.....	3
c. Explication du cas de la conversion en nuances de gris.....	4
d. Filtrage des couleurs et détection des bords à l'aide de l'algorithme de Canny.	4
e. Création des lignes sur les bordures.....	6
PARTIE 2 : Suivi des véhicules.....	7
a. Détection des mouvements.....	7
b. Conversion en nuances de gris et application d'un flou gaussien.....	8
c. Opérations morphologiques et création des contours.....	8
d. Création des rectangles englobant.....	9
PARTIE 3 : Compter les véhicules.....	11
a. Création de 4 zone d'intérêt.....	11
b. Rotation de l'image et extraction de la zone tournée.....	11
c. Analyse de la région tournée.....	11

INTRODUCTION

Dans le cadre de notre formation en informatique, et plus particulièrement dans le domaine de l'analyse d'images, nous avons dû appliquer les compétences acquises en programmation C++, à l'aide d'OpenCV, pour traiter et analyser une séquence vidéo de trafic routier. Ce rapport détaille notre approche et nos réalisations au cours du TP2. Le travail a été divisé en trois parties principales, chacune visant à explorer différents aspects et défis de l'analyse vidéo dans des conditions réelles.

PARTIE 1 - Détection des voies

1. Compléter le code fourni pour détecter automatiquement les deux voies rapides.
2. Justifier vos choix et commenter les résultats obtenus.
3. Dessiner les bordures dans la vidéo.

Cette section présente notre démarche pour détecter automatiquement les deux voies rapides à partir des vidéos fournies. Nous avons complété le code initial en utilisant diverses techniques de traitement d'image pour isoler et marquer les voies de circulation. Les méthodes choisies ainsi que notre logique de sélection sont justifiées, et nous discutons de l'efficacité de notre approche à travers l'analyse des résultats obtenus, y compris la manière dont nous avons dessiné les bordures des voies dans la vidéo.

a. Détection de la première frame

La première chose que l'on a fait a été de sélectionner dans notre vidéo un moment sans véhicules. Quand on regarde la vidéo, on s'aperçoit qu'après un court temps où la vidéo est noire, les premières frames qui suivent sont vides de véhicules. Donc dans la boucle nous avons créé une condition qui fait que l'on va traiter chaque frame jusqu'à ce que ce soit une frame en couleur et que l'on garde ensuite le traitement de celle-ci jusqu'à la fin.

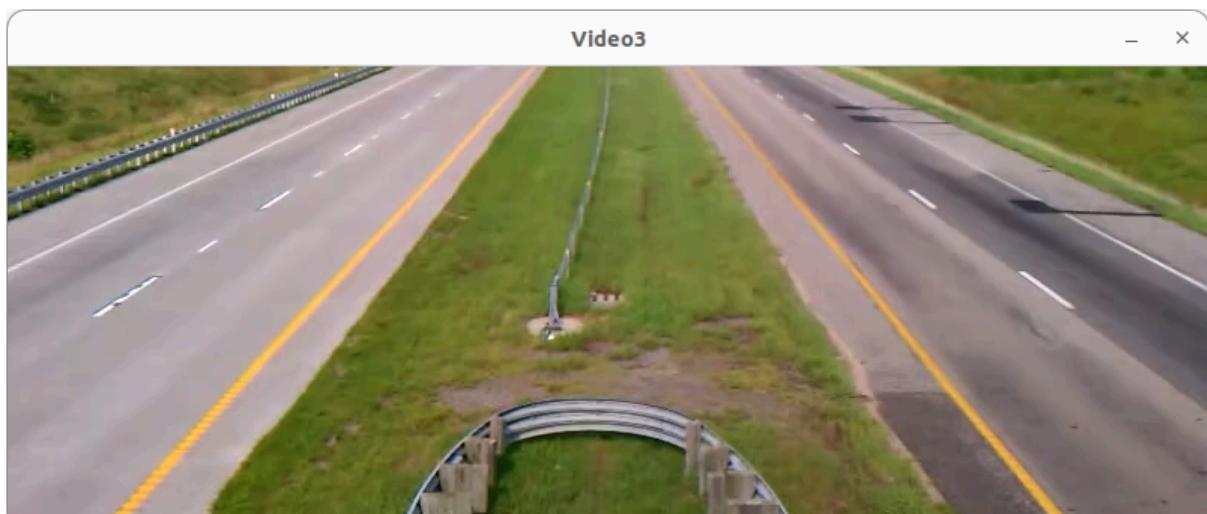


Fig. 1.1 : Frame sans véhicule

b. Conversion en espace de couleur HSV

Pour le traitement, après quelques essais d'idées, nous avons opter pour une conversion en espace de couleur HSV. Cette conversion est courante dans le traitement d'image car elle permet de séparer la teinte de la luminosité, rendant le traitement de couleur plus intuitif et moins sensible aux variations de lumière.

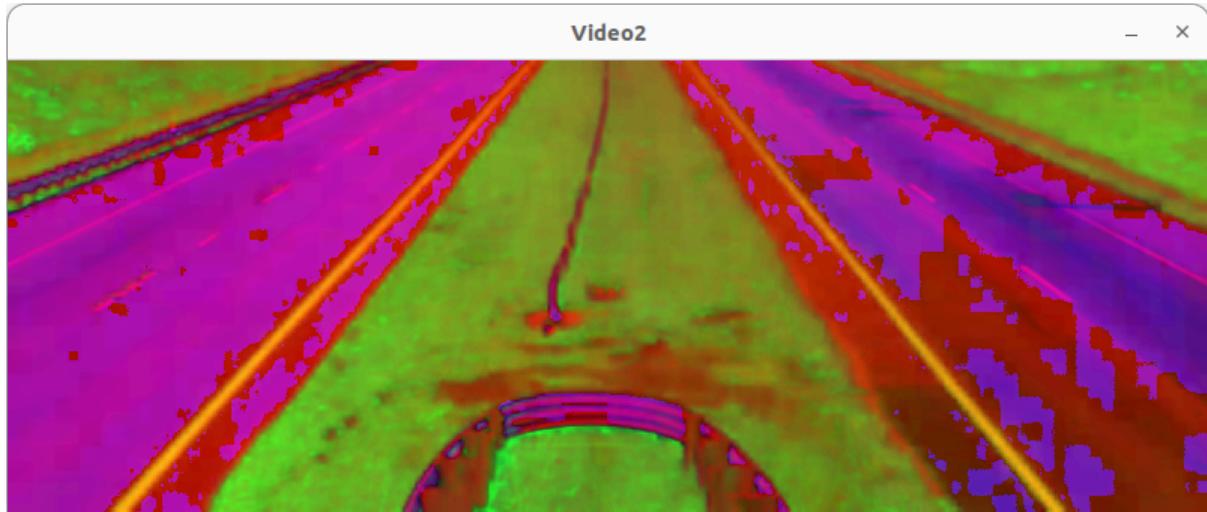


Fig. 1.2 : Frame après la conversion en espace de couleur HSV

c. Explication du cas de la conversion en nuances de gris

Ce choix est judicieux pour la robustesse face à une lumière variable et la facilité de segmentation des couleurs, nous avions d'abord opté pour une méthode utilisant la conversion en nuances de gris mais les lignes créées étaient trop approximatives et c'est donc pour ça que l'on a préféré partir sur du HSV. En effet, cela ne nous plaisait pas de voir que d'un côté les lignes qui délimitent la voie étaient entre la pelouse et la route mais que de l'autre elles étaient par rapport au marquage au sol.

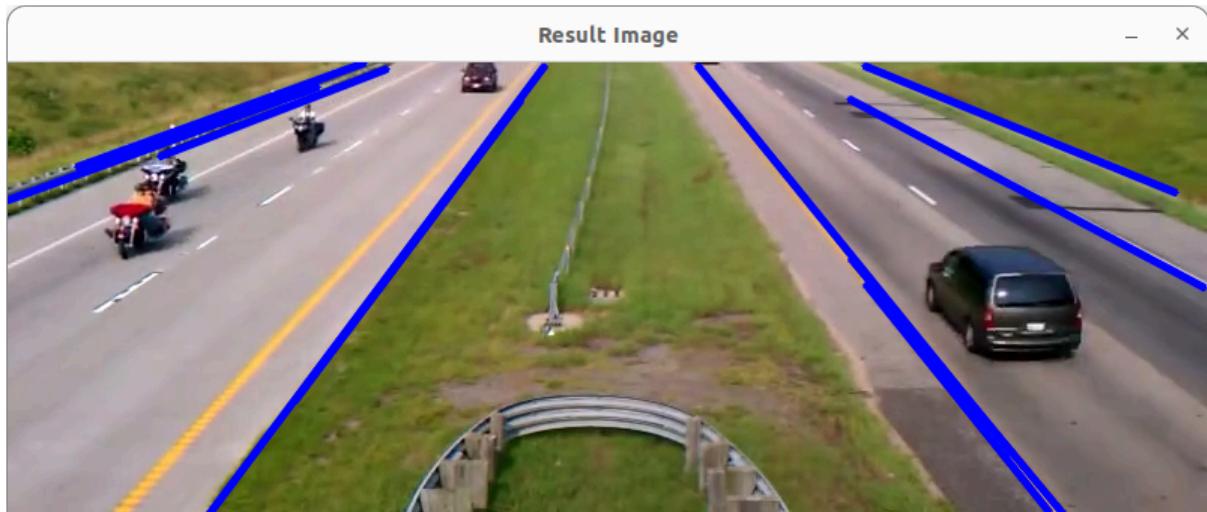


Fig. 1.3 : Frame avec la bordures dans le cas de la conversion en nuances de gris

d. Filtrage des couleurs et détection des bords à l'aide de l'algorithme de Canny

Avec le HSV, l'idée a donc été d'utiliser le vert de la pelouse qui est autour de la route. On utilise donc le filtrage par couleur, à l'aide de `cv::inRange` avec des seuils HSV prédéfinis, le code filtre ainsi les couleurs qui ne correspondent pas à la plage spécifiée (visant à détecter une plage de couleurs spécifique). Pour déterminer les seuils qui seraient le plus intéressant, on a créé un programme intermédiaire pour voir les réglages de seuils avec un slider. Cela nous a permis de rapidement trouver les seuils nécessaires sans faire un long

TP2 : Traitement d'images

enchaînement de `make` et `./Tp`. Une fois ceux-ci trouvés, on détecte ensuite les bords dans l'image filtrée à l'aide de l'algorithme de Canny. Ces bords vont nous servir à identifier les contours et les formes, dans notre cas les lignes de voie.

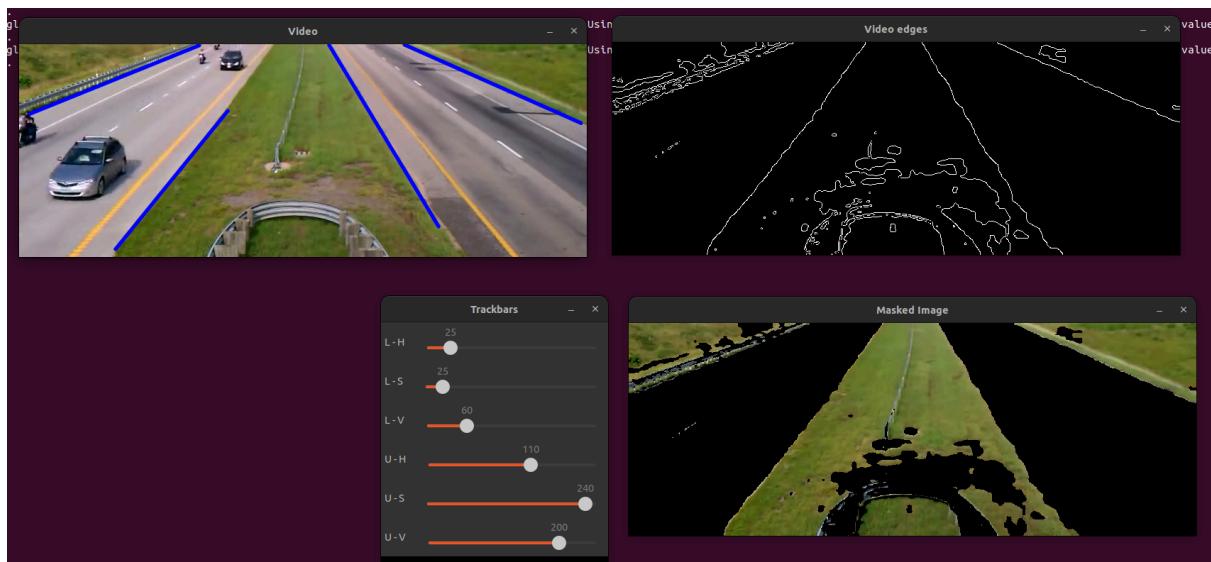


Fig. 1.4 : Capture d'écran du résultat du programme intermédiaire avec les différents sliders correspondant pour chacun à un seuil HSV



Fig. 1.5 : Frame avec le filtrage par couleur avec des seuils HSV prédéfinis

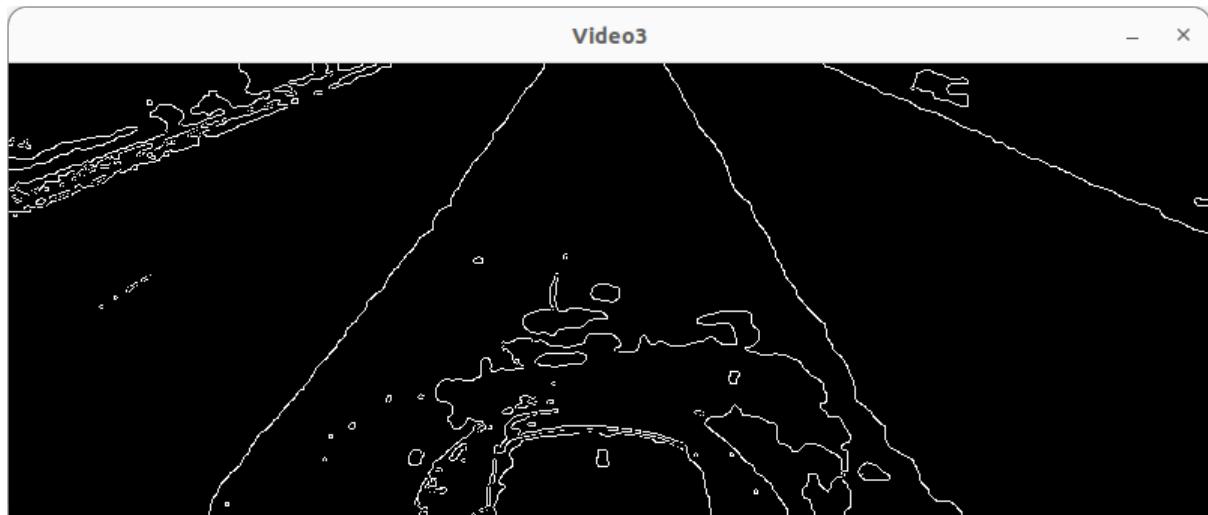


Fig. 1.6 : Frame avec les bords détectés par l'algorithme de Canny dans le cas de la conversion HSV

e. Création des lignes sur les bordures

Après avoir détecté les bords, on applique la transformation de Hough pour les lignes afin d'identifier les segments de ligne dans l'image. Cela permet de détecter les lignes de voies même lorsqu'elles sont discontinues ou partiellement visibles. Puis à l'aide de la fonction que l'on a créé, `drawInfiniteLine`, qui nous sert à étendre et dessiner les lignes détectées sur toute la largeur de l'image, ce qui était très pratique dans le cas de voies. Et comme on peut le voir, cette fois les lignes sont bien placées des deux côtés.



Fig. 1.7 : Frame avec les bordures dans le cas de la conversion HSV

PARTIE 2 : Suivi des véhicules

1. En utilisant les techniques données dans le cours, effectuer le suivi des véhicules au cours de la vidéo.
2. Justifier vos choix et commenter les résultats obtenus.
3. Dessiner un carré autour des véhicules détectés.

Ici, nous expliquons comment nous avons appliqué les techniques de suivi de véhicules présentées durant le cours pour identifier et suivre les déplacements des véhicules à travers les différentes séquences vidéo. Nous justifions nos choix d'algorithmes et de paramètres, tout en analysant les défis rencontrés lors de cette phase. Les résultats, illustrés par le dessin de carrés autour des véhicules détectés, sont commentés en détail.



Fig. 2.1 : Frame avec des véhicules

a. Détection des mouvements

Cette fois, on convertit l'image de l'espace de couleur BGR en des nuances de gris. Pour détecter le mouvement, on reprend l'image que l'on avait mis de côté dans la partie 1. En effet, le fait d'avoir une image sans véhicules, va nous permettre de mettre en lumière les changements entre l'image actuelle et l'image sans. On effectue donc une soustraction des deux images puis une binarisation, ce qui nous permet de garder que les objets en mouvement. Cette méthode est couramment utilisée pour mettre en évidence les opérations morphologiques, néanmoins elle pourrait poser un souci dans le cas où il y aurait des arbres en mouvement avec le vent dans l'arrière-plan.



Fig. 2.2 : Frame après détection des mouvements

b. Conversion en nuances de gris et application d'un flou gaussien

On y applique ensuite un flou gaussien pour réduire le bruit de l'image et les détails. Ce qui permet d'améliorer les performances des étapes de détection suivantes.



Fig. 2.3 : Frame après application d'un flou gaussien

c. Opérations morphologiques et création des contours

On effectue ensuite des opérations morphologiques (érosion et dilatation) pour mettre en évidence les objets en mouvements. On détecte les contours avec `findContours` pour les objets en mouvements détectés. Ce qui permet d'identifier les éléments individuels dans l'image qui seront dans notre cas des véhicules. Néanmoins, dans le cas où il y aurait des objets non véhiculaires en mouvement (piétons, arbres, ...), ils seraient eux aussi identifiés comme des véhicules. Il faudrait rendre donc l'analyse de la forme plus sophistiquée.



Fig. 2.4 : Frame après les opérations morphologiques

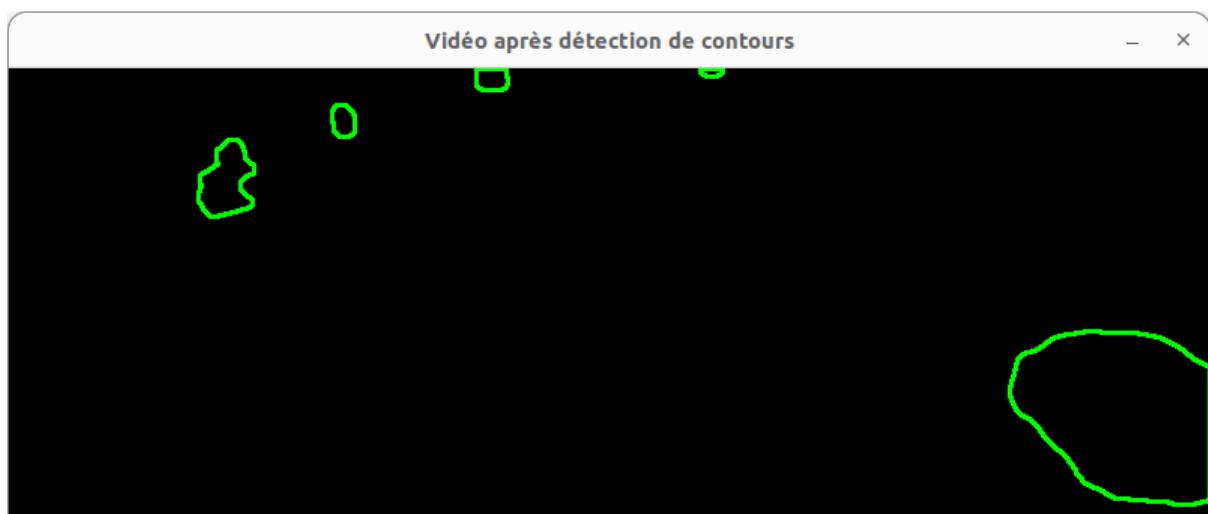


Fig. 2.5 : Frame après la détection des contours

d. Création des rectangles englobant

Enfin pour chaque contour détecté qui passe un certain seuil de surface, un rectangle englobant est dessiné autour de lui. Cela représente visuellement le véhicule détecté. Cela présente l'intérêt d'être assez simple de mettre en évidence les véhicules en mouvement. Un problème se pose néanmoins, cela peut manquer de précision. En effet, dans le cas où deux véhicules se retrouvent un peu trop proches, Les véhicules vont devenir une seule et même surface ce qui va faire que le rectangle les englobe tous les deux malgré que ce soit deux véhicules différents.



Fig. 2.6 : Frame avec les rectangles englobant

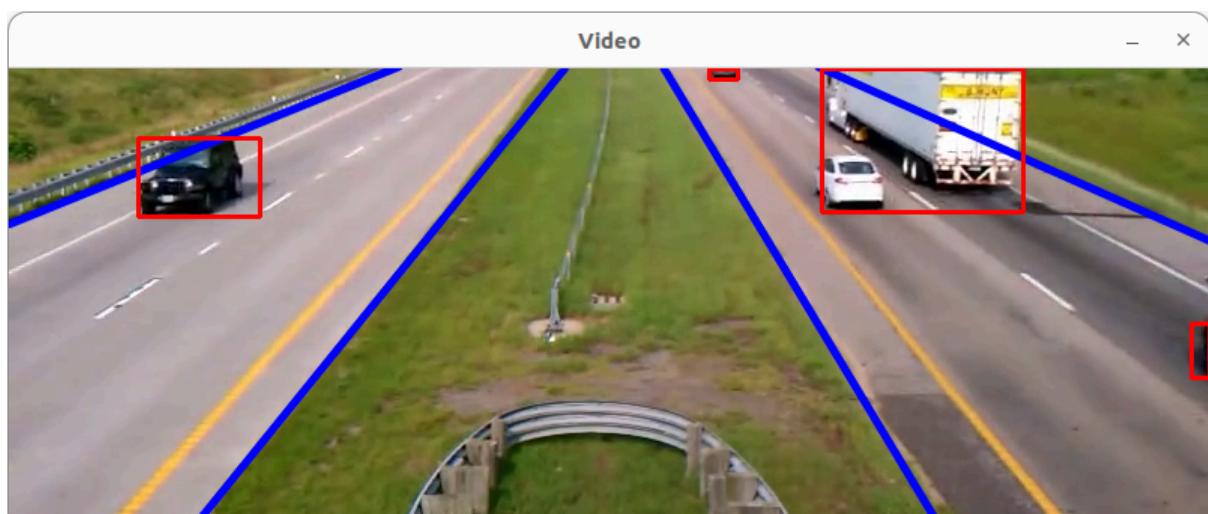


Fig. 2.7 : Frame avec les rectangles englobant dans le cas où le rectangle prend deux véhicules pour un seul

PARTIE 3 : Compter les véhicules

1. Après les avoir détectés, compter le nombre de véhicules qui passent dans chaque direction.
2. Quels sont les problèmes rencontrés ? Commentez les solutions trouvées.

Enfin, cette partie s'attache à quantifier le flux de véhicules en comptant le nombre de ceux passant dans chaque direction. Nous exposons les problématiques rencontrées, notamment en termes de détection et de suivi, et décrivons les solutions que nous avons implémentées pour y répondre. L'impact de ces solutions sur l'exactitude du comptage est également discuté.

a. Création de 4 zone d'intérêt

On appelle la fonction zone 4 fois, qui est une fonction que l'on a codée. Elle commence par créer une zone d'intérêt, qui correspond à un rectangle incliné placée sur la route de façon à détecter de la manière la plus optimale. Il fallait faire attention à où les placer pour être sûr que tous les véhicules étaient bien détectés. Notamment le moment où les deux motos sont collées l'une à l'autre.



Fig. 3.1 : Image avec les rectangles inclinées

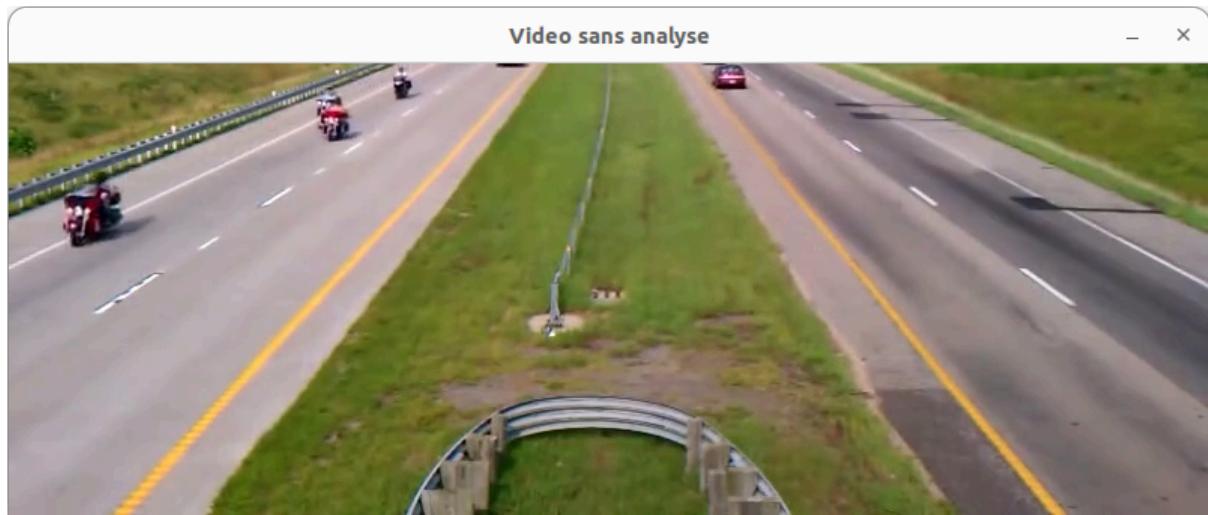


Fig. 3.2 : Frame avec les deux motos collés

b. Rotation de l'image et extraction de la zone tournée

La fonction calcule ensuite une matrice de rotation M avec `getRotationMatrix2D`, utilisant le centre et l'angle du `RotatedRect` ainsi qu'un facteur de mise à l'échelle de 1.0 (pas de mise à l'échelle). Ensuite, `warpAffine` applique cette matrice de rotation à l'image d'entrée `img`, résultant en une nouvelle image `rotated` où la région spécifiée a été tournée. `getRectSubPix` extrait la région rectangulaire spécifiée (après rotation) de l'image inclinée, en utilisant la taille et le centre de `rRect`. Cette région est stockée dans `cropped`.



Fig. 3.3 : Image extraite de la zone d'intérêt après avoir été tournée

c. Analyse de la région tournée

La fonction évalue ensuite le nombre de pixels non nuls dans la région `cropped` et compare cette quantité à des seuils définis par les paramètres `up` et `down`, multipliés par l'aire de la région `cropped`. Si le nombre de pixels non nuls est supérieur au seuil haut (`up * area`) et que `pixImg` est égal à 0, `compt` est incrémenté et `pixImg` est mis à 1. Si le nombre de pixels non nuls est inférieur au seuil bas (`down * area`), `pixImg` est remis à 0. Cela permet de détecter si la région tournée contient suffisamment de caractéristiques (par exemple, des pixels non nuls) pour satisfaire certains critères. Le but étant de connaître la présence ou non d'un véhicule sur la zone d'intérêt en fonction du pourcentage de pixels de remplissage. Et va donc ensuite incrémenter un compteur. On utilise donc ces 4 comptages différents et on les utilise pour connaître le nombre de voitures passant à gauche et le nombre de voitures passant à droite.

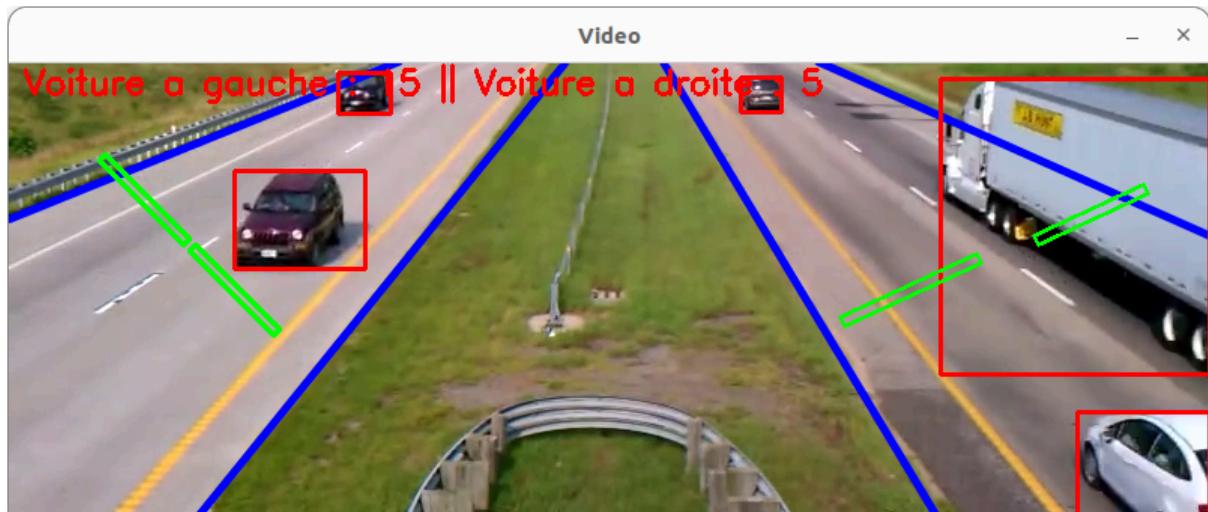


Fig. 3.4 : Frame pendant le comptage

Et c'est comme ça qu'on arrive à un comptage final de **26 voitures** avec **17 voitures à gauche et 9 voitures à droite**.