

ROB313 TP4 : Analyse vidéo et Tracking

Zheyi SHEN & Dajing GU

Décembre 2020

1 Introduction

L'objectif de ce TP est de comprendre les enjeux et difficultés du suivi (tracking) d'objets dans les séquences vidéo, et d'expérimenter et programmer les solutions fondées sur les algorithmes de Mean Shift et de Transformées de Hough Généralisées. L'algorithme de Mean Shift utilise les caractéristiques de couleur de l'objet et l'algorithme de la Transformées de Hough utilise les caractéristiques liées à la forme de l'objet à suivre.

2 Mean Shift

Q1: Avantages et limitations de *Mean Shift*

Dans cette partie, on rappelle le principe de la méthode Mean Shift et expérimente le suivi avec des différentes séquence. Ensuite, on analyse des avantages et des limitations de l'algorithme en illustrant ces expériences.

La structure du code *Tracking_MeanShift.py* suivi les étapes ci-dessous :

- Choisir la séquence et sélectionner manuellement la région d'intérêt (RoI) sur sa première image.
- Convertir RoI de l'image de RGB en HSV et se concentrer sur la partie importante en utilisant un *mask*.
- Créer un histogramme de la composante de teinte H de la région d'intérêt et faire la normalisation.
- Faire des itérations pour chaque image dans la séquence :
 - Convertir les images en HSV et utiliser la fonction `cv2.calcBackProject()` pour calculer la probabilité que le pixel fasse partie de l'objet que l'on choisit à suivre. Le résultat obtenu est la densité marginale f_H .
 - Utiliser la fonction `cv2.meanShift()` pour appliquer la méthode de Mean Shift à la densité de probabilité obtenue et mettre à jour la fenêtre de suivi.

Mean Shift est un algorithme itératif utilisé pour estimer le mode de distribution et déplacer le point initial vers la moyenne locale dans chaque itération. Finalement, il trouve la position avec la densité la plus grande. Considérez que l'on a un ensemble de points, ici c'est une distribution de pixels comme une rétro-projection d'histogramme de composante H . Chaque fois l'algorithme reçoit une trame, le point départ est le centre de la fenêtre obtenue dans la trame précédente. En considérant la densité de pixel, on peut trouver le barycentre réel de la fenêtre et on déplace le point départ vers ce barycentre. La fenêtre se bouge aussi pour que le nouveau centre corresponde au barycentre. L'itération s'arrête quand la fenêtre ne bouge plus. Finalement, la fenêtre se trouve sur la position avec la densité maximale.

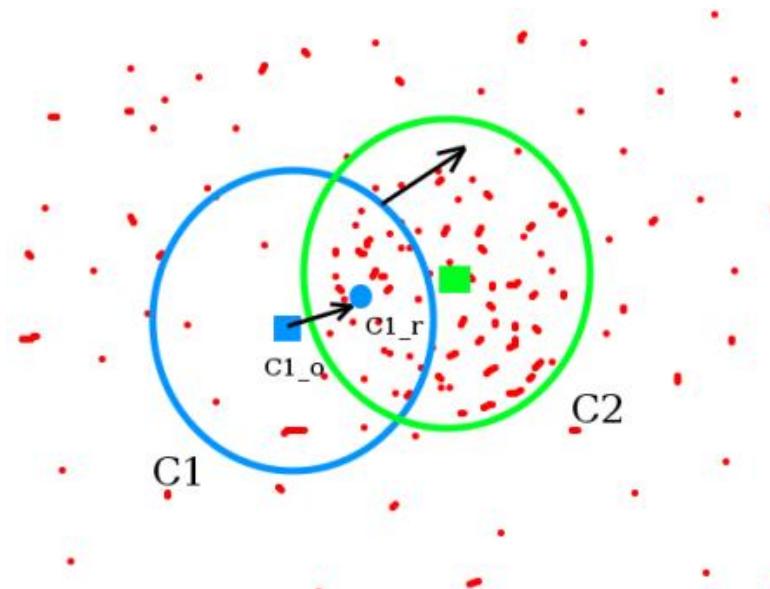


Figure 2.1: Schéma de l'algorithme Mean Shift (Source : OpenCV-Python Tutorials)

Après avoir fait des expériences du code *Tracking_MeanShift.py* avec différentes séquences, on analyse ses avantages et ses limitations. Dans les figures suivantes, le rectangle vert est la fenêtre choisie par nous et les rectangles bleus sont les fenêtres calculées par l'algorithme.

Avantage :

- La computation de l'algorithme n'est pas lourd, donc le suivi en temps réel peut être réalisé.
- Le Mean Shift est appliqué sur la rétro-protection du histogramme H , donc lorsque l'objet est tournée, déformée, l'algorithme peut bien la suivre. La figure 2.2 représente un exemple des expériences.



Figure 2.2: Un exemple du suivi

On peut observer que dans la vidéo, la balle roule constamment, mais l'algorithme suit bien cet objet.

Limitations :

- Pendant le processus de suivi, la taille de la fenêtre reste la même. Lorsque l'échelle de l'objet change, le suivi peut-être échoue.
- Lorsque la vitesse de l'objet est très grande, la performance de suivi n'est pas bonne.
- La limitation la plus importante est que l'algorithme Mean Shift utilise seulement les caractéristiques de couleur de l'objet et n'utilise aucune caractéristique liée à la forme. Cela en résulte que la performance de l'algorithme sont très sensibles à la couleur de l'objet.

Si la teinte de l'objet d'intérêt est très différente de celle de l'arrière-plan, l'algorithme fonctionnera bien. La figure 2.3 montre un exemple. Quand on choisit seulement la tasse comme l'objet d'intérêt, il est possible que l'histogramme de RoI se confond avec l'arrière-plan. En revanche, si on choisit la main, l'algorithme peut bien la suivre comme la teinte de la peau est différente avec la porte derrière.

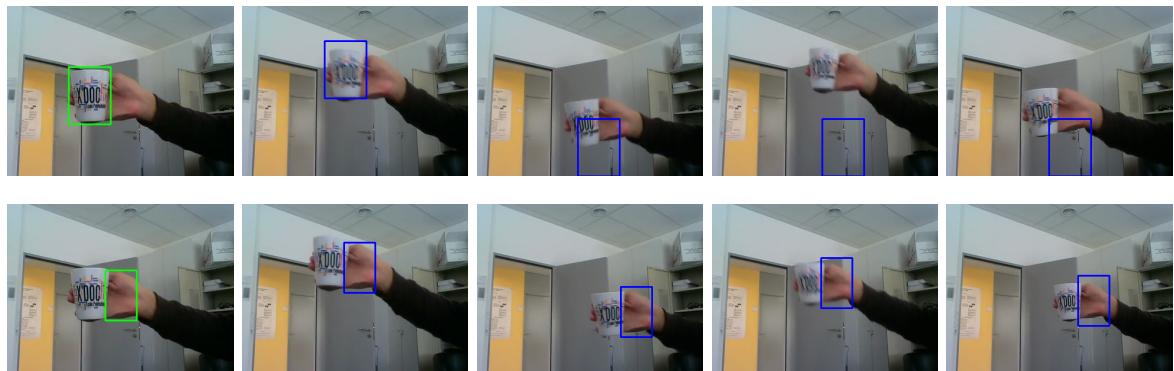


Figure 2.3: L'influence du choix de RoI

La figure 2.4 montre un autre exemple, dans ce cas, on peut voir que l'algorithme n'est pas capable de suivre la voiture blanche car sa teinte est similaire avec la route et le ciel. Cependant, si on choisit le camion noir à côté, le résultat de suivi est assez bon même si la séquence est très tremblante et la vitesse de l'objet change parfois soudainement.

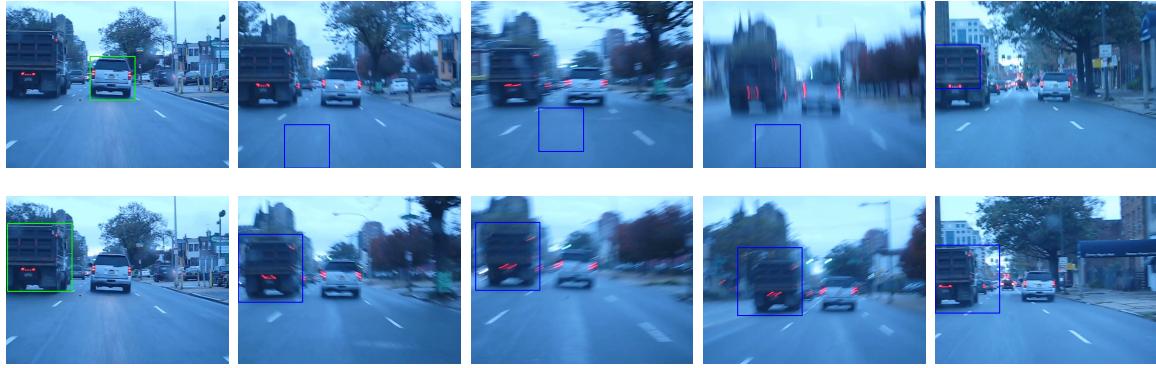


Figure 2.4: L'influence de la teinte de RoI

De plus, le suivi sera affecté par la lumière et l'ombre comme dans la figure 2.5. Lorsque le garçon a fait des allers-retours entre le soleil et l'ombre, l'algorithme n'a pas réussi à suivre son visage.



Figure 2.5: L'influence de la lumière

- Quand le arrière-plan de la séquence est très compliqué et il y a plusieurs objets similaires, il est difficile d'avoir une bonne performance. Comme un exemple du match de basket représenté dans la figure 2.6.



Figure 2.6: L'arrière-plan compliqué et des objets similaires

- Les performances de l'algorithme sont très sensibles au choix de la région d'intérêt. Dans la même situation, différentes rectangles initiales apporteront des résultats différents. Compare la figure 2.2 et la figure 2.7, si on sélectionne une ROI un peu plus grande, la précision de l'algorithme réduit.



Figure 2.7: La sensibilité sur la sélection de RoI

Q2 : Stratégie d'amélioration

Afin d'analyser plus finement le résultat, on a affiché la séquence des poids à partir de la rétro-projection R_H de l'histogramme de teinte, qui est obtenue à l'aide de `cv2.calcBackProject()`.

La rétro-projection peut être utilisée pour la segmentation de l'image, qui nous aide de trouver la région d'intérêt. Chaque valeur de pixel dans l'image de sortie représente la probabilité que le point correspondant à la région d'intérêt (RoI). Plus la valeur de pixel est élevée, plus le point est possible d'appartenir à l'objet.

Binarisation de la rétro-projection

Comme nous avons déjà mentionné dans Q1, le choix de RoI est vraiment important et il est capable d'influencer le résultat de la détection.

On pose la vidéo *VOT-Ball.mp4* : figure 2.8 comme un exemple. On ne choisit pas toute la balle comme RoI mais seulement une partie de la balle. La figure à gauche est la figure réelle avec la balle entouré par le rectangle. La balle est l'objet dont on veut suivre la piste. La figure à droite est la représentation de la rétro-projection, dans laquelle la partie blanche est supposée plus similaire à l'objet.



(a) Figure de trame-tracking (b) Figure de rétro-projection

Figure 2.8: Illustration de deux figures

Même si la piste de l'objet (la balle) dans la figure 2.8(a) est bien suivi par l'algorithme et les pixels correspondants sont blancs, il y a beaucoup d'autres pixels qui sont aussi blancs. Cela est possible d'influencer le suivi de la piste, comme présenté dans la figure 2.9. Le bruit a beaucoup influencé l'identification de l'objet, qui cause un mauvais suivi.



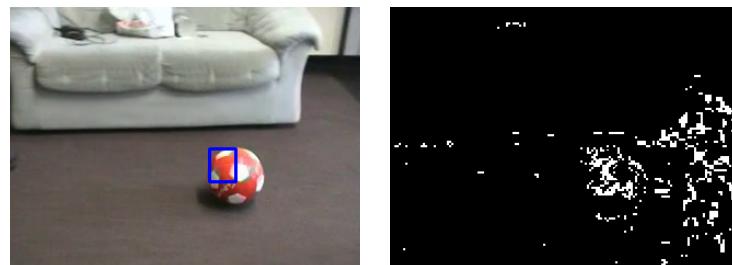
(a) Figure de trame-tracking (b) Figure de rétro-projection

Figure 2.9: Illustration des problèmes

Afin de résoudre ce problème, la fonction `cv2.threshold()` est utilisé pour obtenir la binarisation de la figure de rétro-projection. Selon la figure 2.10, les bruits blancs dans la figure de rétro-projection ont été réduits, qui nous permet de bien suivre la piste de la balle.

```
1 # Binarization of the Backprojection
2 ret, dst = cv2.threshold(dst, 127, 255, cv2.THRESH_BINARY)
```

On peut en déduire que la binarisation de la rétro-projection peut aider à atténuer le problème causé par le mauvais choix de ROI.

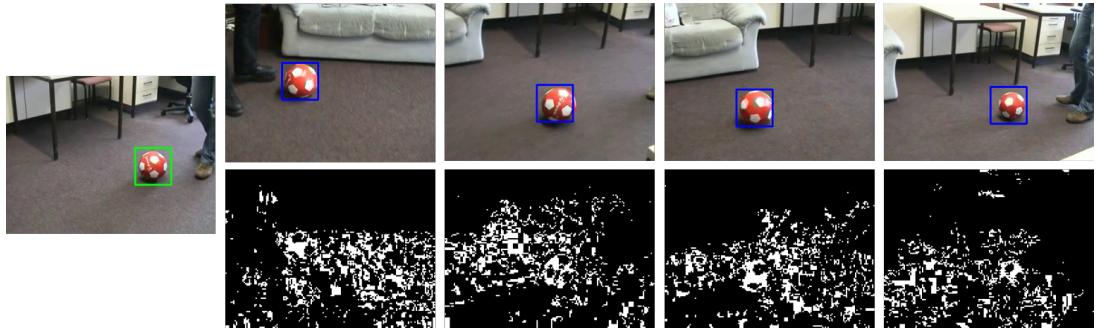


(a) Figure de trame-tracking (b) Figure de rétro-projection

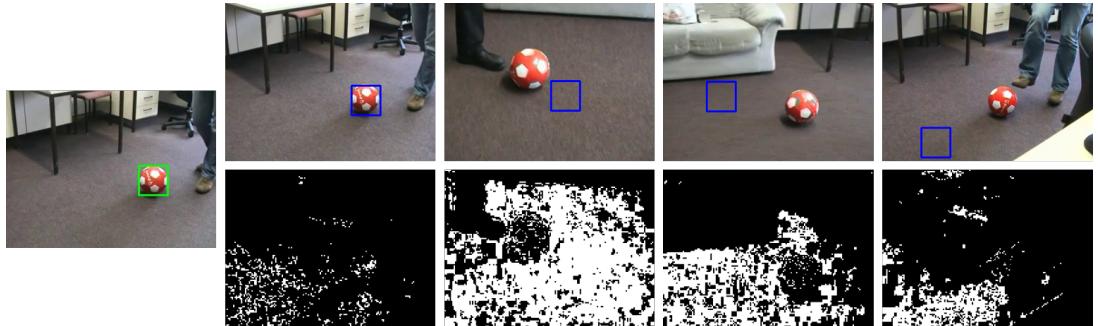
Figure 2.10: Amélioration de l'algorithme

Mise à jour de l'histogramme de ROI

Une autre idée est de mettre à jour pour chaque trame de la vidéo l'histogramme de composante teinte de la région d'intérêt. Cependant, la performance du nouveau code n'est pas satisfaisante. La figure 2.11 représente une comparaison entre le cas sans mise à jour de l'histogramme et le cas avec mise à jour.



(a) Sans mise à jour de l'histogramme



(b) Avec mise à jour de l'histogramme

Figure 2.11: Amélioration de l'algorithme

En effet, en mettant à jour l'histogramme de RoI pour chaque trame, les erreurs de calcul sont superposées. Une fois il y a une mauvaise détection de l'objet, l'algorithme ne peut plus nous rendre le résultat correcte. Comme présenté dans la figure 2.11, il y a plus de pixels blancs après la mise à jour l'histogramme, en plus, la position de balle est mal détectée et les pixels associés sont noirs dans la figure de rétro-projection. Cela influence beaucoup l'effet de la détection.

3 Transformée de Hough

La transformation de Hough se sert seulement à détecter des formes déjà bien définies. La transformation de Hough généralisée (GHT), basée la transformation de Hough, ne nécessite pas d'expression analytique de la forme à détecter. Elle peut détecter n'importe quelle forme donnée.

Dans cette section, on va implémenter la transformée de Hough. Dans Q3, on va définir l'index de la Transformée de Hough en utilisant la norme de gradient. Seulement les pixels dont la norme de gradient est petites sont choisis. Ensuite, dans Q4, un modèle l'objet sous la forme de R-Table va être construit et la transformée de Hough de chaque trame du vidéo va être calculée. Le suivi correspondant à la valeur maximale va aussi être betenu.

Q3 : Calcul de l'orientation du gradient

L'objectif de cette question est de définir l'index de la Transformée de Hough (l'orientation), ainsi que l'ensemble des pixels votants.

Dans cette section, on a calculé à chaque trame, l'orientation locale, i.e. l'argument du gradient des pixels de l'image et le module du gradient. A l'aide de ROB 317, on fait tout d'abord la conversion d'image RGB en image de niveaux de gris. Ensuite on calcule la norme euclidienne du gradient :

$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2} = \sqrt{(I * h_x)^2 + (I * h_y)^2}$$

et son orientation :

$$\arg \nabla I = \arctan \frac{I_y}{I_x}$$

avec

$$h_x = \frac{\partial I}{\partial x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

et

$$h_y = \frac{\partial I}{\partial y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Cela peut aussi être fait par `cv2.Sobel()`.

En choisissant le seuil sur le module du gradient pour l'orientation n'est pas significative **60**, on a réussi à afficher la séquence des orientations, avec les pixels de l'orientation petite en rouge, qui est présenté dans la figure [3.1](#). L'index des pixels de l'orientation significative va être utilisé comme l'index de la Transformée de Hough dans la sous-section suivante.

Ici le code associé:

```
1 def get_index_hough(norme, orientation_clone, th_min):
2     index = np.where(norme < th_min)
3     orientation_clone[0][index] = 255 # red
4     orientation_clone[1][index] = 0
5     orientation_clone[2][index] = 0
6     return orientation_clone, index
7
8 def get_index_hough2(norme, th_min):
9     norme_hough = norme.copy()
10    norme_hough[norme_hough[:, :] < th_min] = 0
11    return norme_hough
```

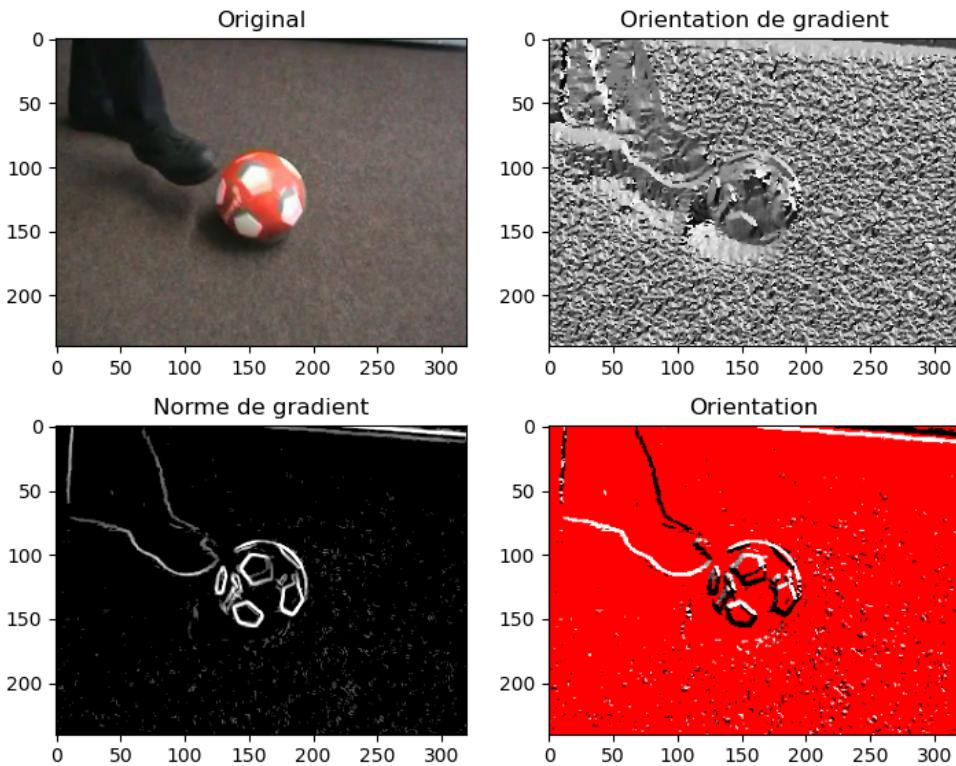


Figure 3.1: Calcul de l'index de vote (orientation du gradient), avec sélection des pixels votants (norme du gradient).

Q4 : Calcul de la transformée de Hough

Construction de R-Table

Dans cette sous-section, un modèle de l'objet sous la forme de R-Table va être construit. La transformée de Hough de chaque trame du vidéo et le suivi correspondant à la valeur maximale vont aussi être calculés.

Dans la transformée de Hough, on va construire un R-Table car l'expression analytique de la forme à détecter n'est pas disponible. L'illustration de R-Table est présentée dans la figure 3.2.

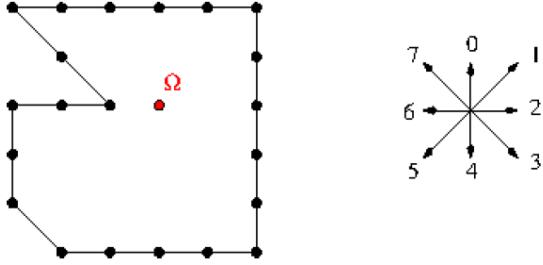


Figure 3.2: Illustration de la construction de R-Table

Notant Ω le centre de l'objet (dans notre cas le *RoI*) et M les points sur cet objet. Chaque point M est indexé par une caractéristique géométrique i , et le vecteur déplacement $\overrightarrow{M\Omega}$ est ajouté dans chaque ligne d'index i .

Dans notre cas, c'est l'orientation de gradient qui est utilisée comme la caractéristique, et le R-Table est construit en fonction de tous les pixels du contouor de l'objet. On a aussi essayé d'autres caractéristiques comme l'orientation de la ligne entre M et Ω , mais ça prend trop de temps pour construire le R-Table.

Ici le code associé:

```

1 def build_r_table(img):
2     """
3         Build the R-table from the given shape image
4     """
5     img_center = [int(img.shape[0] / 2), int(img.shape[1] / 2)]
6     edges = img
7     gradient = gradient_orientation2(edges) * 180 // np.pi
8     r_table = defaultdict(list)
9     for (i, j), value in np.ndenumerate(edges):
10         if value:
11             r_table[gradient[i, j]].append((img_center[0] - i, ...
12                                         img_center[1] - j))
13
14     return r_table

```

Calcul de la transformée de Hough

Après la construction de R-Table, on va maintenant calculer la transformée de Hough, dont les étapes sont présentés ci-dessous:

- Initialiser le map de vote $H(M) = 0$ pour chaque pixel M de l'image
- Pour chaque pixel M de l'image:
 - Calculer l'index $\lambda(M)$ de caractéristique auquel il appartient.
 - Pour tous les pixels de l'objet t_j tels que $\lambda(t_j) = \lambda(M)$, incrémenter dans le map de vote $H(M + vector(t_j)) += 1$.
- Donner la position la plus probable du centre de l'objet: $arg \max_x H(x)$.

Ici le code associé:

```

1
2 def matchTable(img, table):
3     """
4         matches the reference table with the given input image
5     """
6     acc = np.zeros((img.shape[0]+int(img.shape[0]*0.2), ...
7                     img.shape[1]+int(img.shape[0]*0.2)))
8     # acc array requires some extra space
9     gradient = gradient_orientation(img) * 180 // np.pi
10    for i in range(img.shape[0]):
11        for j in range(img.shape[1]):
12            if img[i, j] != 0: # boundary point
13                theta = gradient[i, j]
14                vectors = table[theta]
15                for vector in vectors:
16                    acc[vector[0]+i, vector[1]+j] += 1
17
18
19 def findMax(acc):
20     ridx, cidx = np.unravel_index(acc.argmax(), acc.shape)
21     return [acc[ridx, cidx], ridx, cidx]

```

Résultat et commentaire

Comme nous avons déjà analysé dans Q1, la performance de l'algorithme Mean Shift est très sensible à la couleur de l'objet puisqu'il utilise seulement les caractéristiques de couleur de l'objet et n'utilise aucune caractéristique liée à la forme.

En revanche, l'algorithme de la transformée de Hough utilise la caractéristique liée à la forme, qui peut nous aider à résoudre les problèmes causés par la couleur.

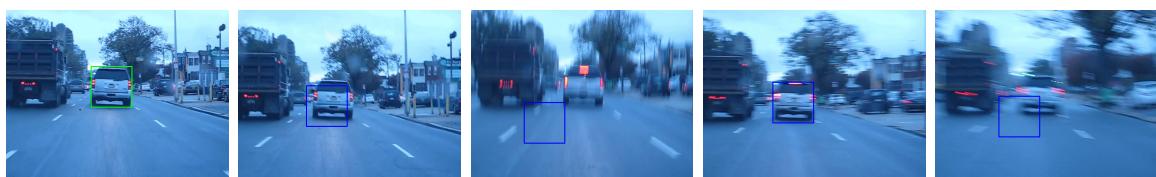


Figure 3.3: Un exemple de suivi d'un voiture

En comparant avec la figure 2.4 où la voiture blanche est mal détectée à cause de la similarité de sa couleur avec la couleur d'arrière plan, on trouve que cette fois la voiture blanche dont la forme est assez spéciale est bien identifiée dans la figure 3.3, qui prouve l'effet de la transformée de Hough.



Figure 3.4: Un exemple de suivi d'une tasse

De la même raison, la tasse blanche dans la figure 3.4 est aussi mieux identifiée grâce à la forme de la tasse. Mais le phénomène de décalage existe encore. Quelque fois la tasse est mal-identifiée et on pense que cela provient de la déformation de la tasse : changement de taille de la tasse.



Figure 3.5: Un exemple de suivi d'une personne sous la lumière

Selon la figure 3.5, on trouve que l'algorithme de la transformée de Hough ne peut pas résoudre le problème causé par la lumière et l'ombre, parce que quand l'homme se déplace entre la lumière et l'ombre, le gradient de l'objet d'intérêt (son visage) change aussi.

De plus, la méthode de transformation de Hough est beaucoup impactée par la rotation de l'objet comme montré dans la figure 3.6. C'est parce que quand l'objet roule ou déforme, le gradient associé change beaucoup, les pixels d'intérêt changent aussi. Donc, si on utilise le R-table de ROI initiale, l'algorithme ne peut pas trouver la nouvelle position.

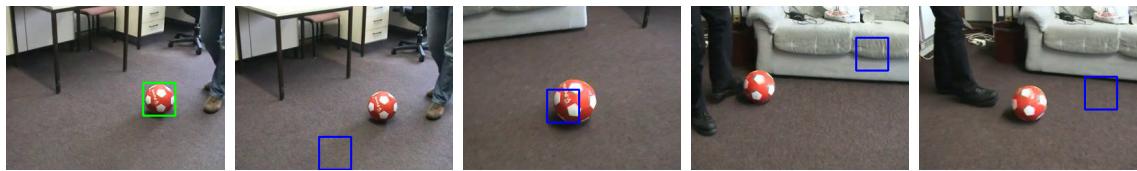


Figure 3.6: Un exemple de suivi d'une balle

Conclusion L'algorithme de la transformée de Hough utilise la caractéristique liée à la forme, qui peut nous aider à résoudre les problèmes de détection causés par la couleur. Mais la déformation de l'objet et l'influence de la lumière sont encore deux problèmes restant à résoudre.

Q5 : Amélioration des algorithmes

Remplacement du calcul du maximum par *Mean Shift*

Dans la question 4, on utilise le pixel dont le vote est le plus grand comme le centre de l'objet. Dans cette question, on remplace le calcul du maximum par l'application du Mean Shift.

```
1 # Meanshift sur la transformee de Hough
2 ret, track_window = cv2.meanShift(acc, track_window, term_crit)
```

L'accumulation de vote est utilisée comme la densité de probabilité dans l'algorithme de Mean Shift. La figure 3.7 et 3.8 représentent deux exemples de résultats de suivi.



Figure 3.7: Un exemple de suivi d'un voiture

En comparant avec la figure 3.3, le résultat de l'algorithme avec remplacement est meilleur. La position de la fenêtre est plus précise et le suivi est plus stable. Le décalage existe quand la caméra tremble soudainement, mais l'algorithme peut bien retrouver la position de la voiture.



Figure 3.8: Un exemple de suivi d'une tasse

Cependant, le suivi de la tasse n'est pas assez satisfaisant. Il est facile de considérer l'affiche ou la porte comme l'objet à suivre. Peut-être la raison est que le contour de la tasse et celui de l'affiche ou de la porte sont tous des lignes droites.

On a aussi fait des tests sur d'autres séquence, mais les résultats ne présentent pas beaucoup de différences que les résultats précédents

Conclusion Le remplacement du calcul de maximum par l'application du Mean Shift peut rendre le suivi plus stable et améliorer un peu la performance de l'algorithme, mais il ne peut pas résoudre le problème causé par la lumière et l'ombre et par la déformation de l'objet.

Traitement des déformations de l'objet

Pour traiter des déformations de l'objet, on propose de mettre à jour la région d'intérêt pour chaque itération comme dans la Q2. On calcule la R-table pour la nouvelle RoI et l'utilise pour calculer l'accumulation de votes. La déformation de l'objet entre deux trames consécutives ne sera pas particulièrement important, donc en théorie, il est plus raisonnable d'utiliser la R-table calculée à partir du RoI de la trame précédente pour prédire la nouvelle position de l'objet.

Cependant, il y a aussi une limitation similaire avec la Q2, une fois l'algorithme trouve une fenêtre fausse, il est difficile de retrouver la position réelle de l'objet car les erreurs sont superposées. Comme dans la figure 3.9, le rectangle suit la tasse au début, mais il reste sur la même position une fois il se trouve autour l'affiche.



Figure 3.9: Un exemple de suivi d'une tasse