



ÉCOLE NATIONALE SUPÉRIEURE DE TECHNIQUES AVANCÉES

Object Tracking in Videos

Auteurs :

Arthur RUBACK
Matheus SANO
Vanessa LOPEZ

Encadrants :
M. Antoine MANZANERA

IA323 - Deep learning based computer vision

12 mars 2024

Table des matières

1	Introduction	2
2	Mean Shift	2
2.1	Expérimentation	2
2.2	Analyse	3
3	Transformée de Hough	6
3.1	Orientation locale	6
3.2	Tracking avec la transformée de Hough	7
3.3	Amélioration de la détection avec la transformée de Hough	10
4	Deep Features	10

1 Introduction

L'objectif de ce travail est de comprendre les défis et les difficultés du suivi des objets dans les vidéos, d'expérimenter et de développer des solutions avec méthodes comme Mean Shift, Hough transform et deep features.

2 Mean Shift

2.1 Expérimentation

Le principe de la méthode **Mean Shift** consiste à prendre un échantillon de pixel illustré comme $C1_o$ dans la figure 1 et ce point est considéré comme la valeur moyenne pour ce pixel, puis il place une fenêtre $C1$ autour de cette valeur moyenne d'une certaine taille pour calculer le centroïde de la fenêtre en fonction de la distribution à l'intérieur. Lorsque ce point est trouvé ($C1_r$ dans la figure), la valeur moyenne devient ce centroïde et ce processus est répété jusqu'à ce que la zone de densité maximale de pixels $C2$ soit atteinte [1]. En d'autres termes, cette méthode recherche, fenêtre par fenêtre, la zone la plus dense pour atteindre la zone de densité maximale de pixels.

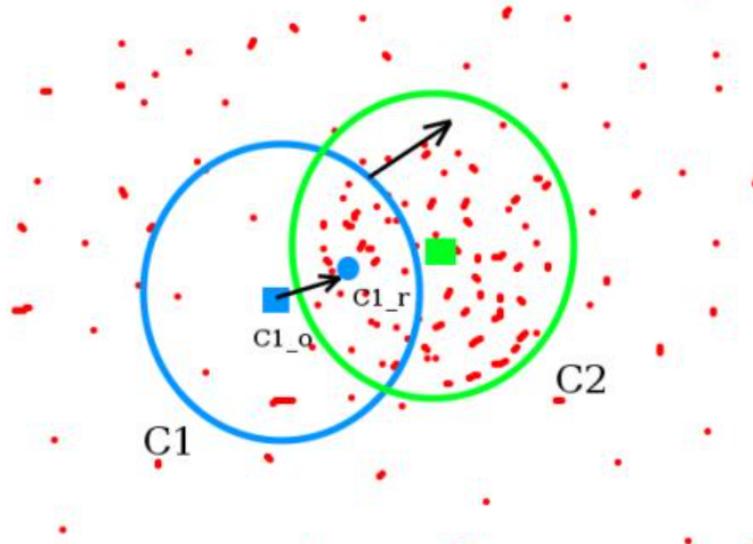


FIGURE 1 – Representation de 'Mean Shift'. Source: Docs Opencv

Dans l'algorithme proposé dans le code *Tracking_MeanShift.py*, une fenêtre est définie avec l'objet à suivre, qui identifie la zone de distribution maximale de pixels et qui sera la zone de départ pour l'exécution de la méthode "Mean Shift". Avant d'appliquer cette méthode, cette sélection de fenêtre traverse un changement de modèle de couleur et un filtrage pour ignorer certaines tonalités indésirables. Ensuite, elle calcule une "Backprojection" qui mesure la probabilité que les pixels correspondent à l'objet souhaité en prenant les histogrammes des couleurs, qui fournissent plus d'informations sur l'objet que les histogrammes

des niveaux de gris. Grâce à ces probabilités dans chaque pixel, il est possible d'observer la distribution de l'image et de localiser l'objet à suivre dans la zone la plus dense afin d'appliquer la méthode décrite ci-dessus.

Cette méthode permet facilement, avec quelques commandes simples, de suivre un objet dans une vidéo sur la base des probabilités de localisation de l'objet dans les images, avec la caractéristique d'effectuer les calculs en profitant de la différence de couleur, qui est un bon indice pour le suivi d'un objet. Cependant, cet algorithme présente certaines limites lors de la génération des probabilités de chaque pixel, car comme le montre la figure 2, cet algorithme fonctionne bien dans les vidéos lentes avec une différenciation claire entre l'image et son environnement (a), mais présente des difficultés lors du suivi d'objets en présence de changements de lumière (b), d'autres objets qui perturbent la détection de l'objet souhaité (c) et de changements soudains de la scène (d). Il est important de noter que lors du traitement de vidéos, qui sont des changements dynamiques de l'image, la distribution de probabilité peut changer très rapidement et brouiller légèrement cette détection. Enfin, un autre aspect restrictif de cet algorithme est la taille fixe de la fenêtre, qui ne permet pas de suivre clairement un objet lorsqu'il s'éloigne ou se rapproche de la caméra.

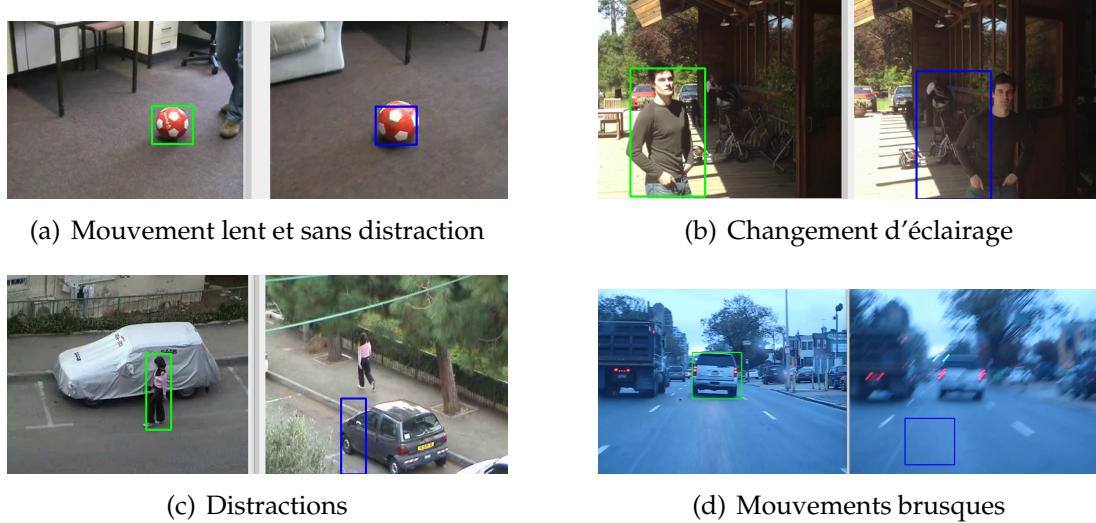


FIGURE 2 – Résultat de l'algorithme de suivi d'objets

2.2 Analyse

En approfondissant les résultats de cette méthode et le calcul pour générer la distribution des données, nous pouvons voir que la méthode recherche effectivement l'emplacement avec la plus grande densité de points de probabilité, comme le montre la partie inférieure droite des figures de la fig. 3. Dans ce cas, la densité la plus élevée est représentée par les zones les plus blanches, comme le montre plus clairement la figure 3(d) où l'homme passe d'une zone éclairée à

une zone sombre en provocant que la densité la plus élevée (points blancs) se trouve à gauche et par conséquent, la détection est légèrement décalée vers la gauche.

On peut constater que cet algorithme présente certains problèmes dans le calcul de la distribution de probabilité, puisque les probabilités deviennent maximales aux endroits où les objets ne sont pas trouvés, comme le montrent les figures 3(b) et 3(c), ce qui fait que l'algorithme tombe dans des maxima locaux en créant une distraction pour un bon suivi.

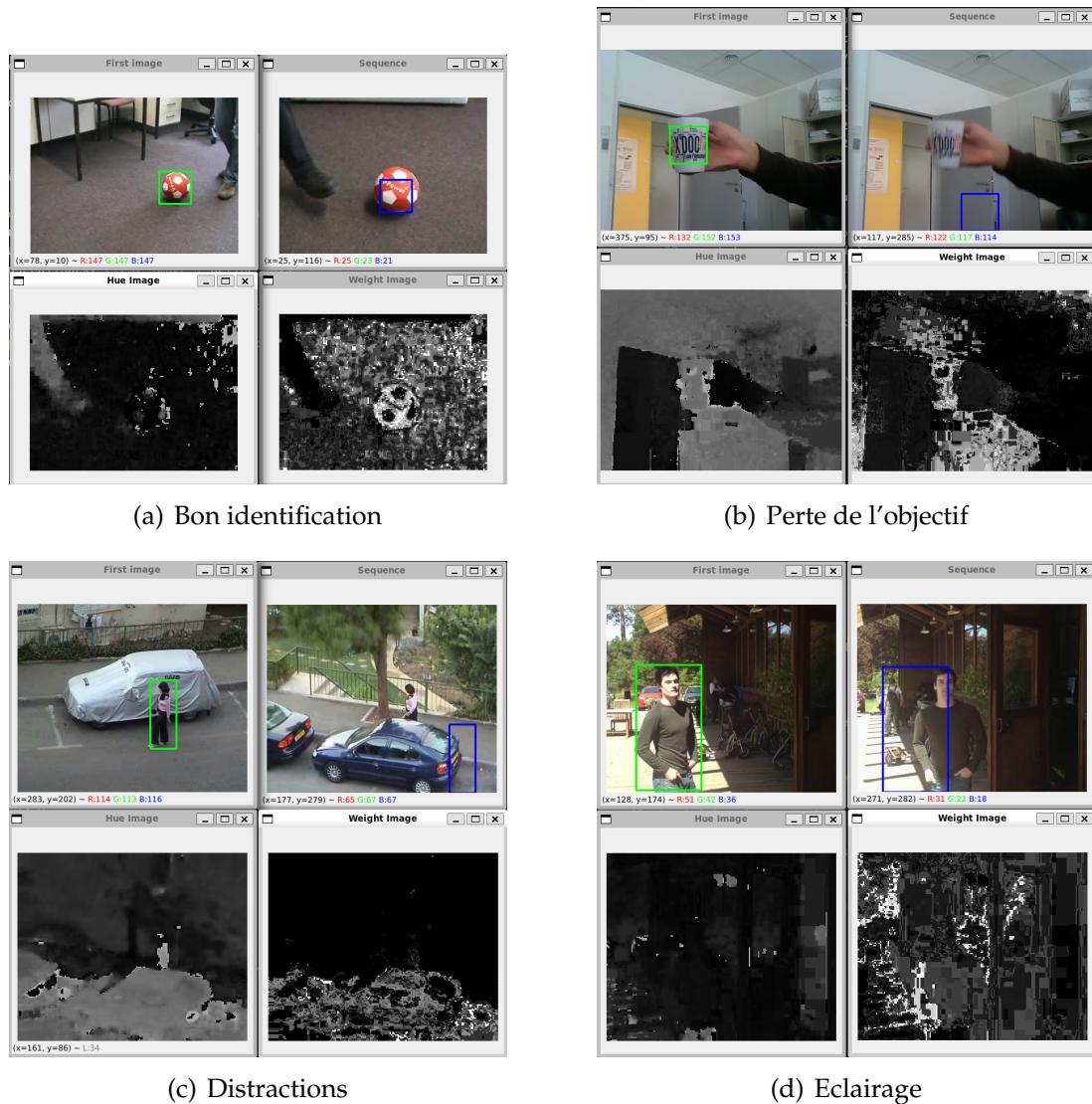


FIGURE 3 – Analyse de l'algorithme. Les images supérieures de chaque figure représentent l'initialisation de l'objet à identifier et son évolution le long de la vidéo, tandis que les images inférieures représentent la vidéo en "Hue" (conversion HSV) et la distribution de probabilité 'Blackproject'

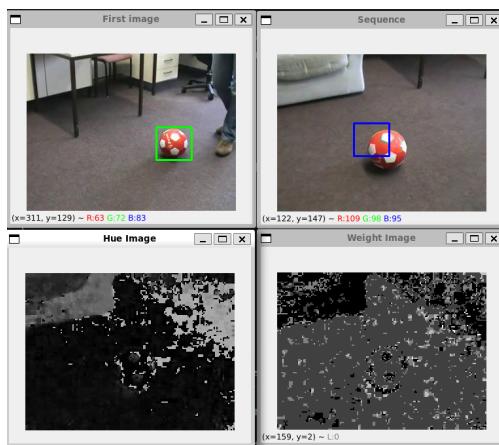
Dans le cadre de la recherche d'améliorations pour l'algorithme, nous avons essayé de changer la couche HSV pour le calcul de la rétroprojection, car visuellement, les couches de saturation et de luminosité semblaient un peu meilleures, mais lors du calcul de la rétroprojection avec ces couches, nous avons obtenu de très mauvais résultats, car ces couches mettent en évidence différents aspects de l'image qui ne nous permettent pas de bien identifier l'objet désiré.

En outre, il est décidé de mettre à jour l'histogramme de l'objet à détecter car il prend comme référence l'objet dans sa vue actuelle et une petite amélioration est évidente. Cependant, lors de la mise à jour de l'histogramme dans chaque image, lorsqu'il s'écarte un peu de l'objet à détecter, il prend comme nouvel objet à détecter l'image déplacée et peut donc facilement tomber dans le suivi d'autres choses. En d'autres termes, cette mise à jour de l'histogramme peut facilement prendre des références erronées et donc générer le suivi d'autres parties de l'image qui ne nous intéressent pas.

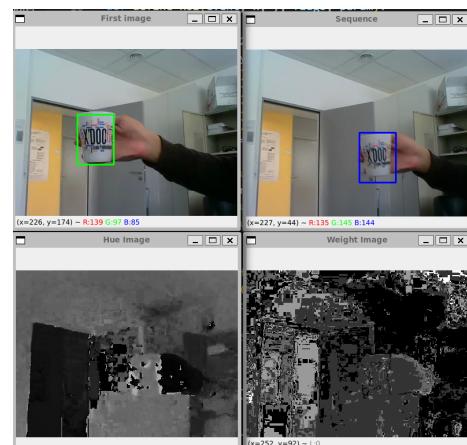
Pour résoudre le problème de la mise à jour des histogrammes erronés, il a été décidé d'augmenter la similarité entre l'histogramme mis à jour $h_{x_0}(u)$ et l'histogramme de référence initial $h_R(u)$ en maximisant l'indice de Bathchacharyya, qui est obtenu en maximisant son deuxième terme indépendant :

$$w_i = \sqrt{\frac{h_R(u)}{h_{x_0}(u)}} \quad (1)$$

Cette solution permet de suivre correctement l'objet souhaité mais contient beaucoup de bruit, c'est-à-dire que la détection se déplace beaucoup autour de l'objet comme le montre la figure 4(a) mais ne perd pas complètement la localisation de l'objet comme cela aurait été le cas sans cette maximisation de l'indice de Bathchacharyya.



(a) Bruit de détection



(b) Bon identification

FIGURE 4 – Mise à jour de l'histogramme et maximisation de l'indice de similarité Bathchacharyya

3 Transformée de Hough

3.1 Orientation locale

Pour le calcul de la transformée de Hough, l'orientation et la magnitude du gradient dans chaque pixel sont obtenues avec la fonction gradient de la bibliothèque numpy correspondant aux dérivées de l'image par rapport à chaque dimension.

```

def grad_orientation(image):
    gradx, grady = np.gradient(image)
    # Gradient orientation ( Index of the vote )
    orientation = np.arctan2(grady, gradx)
    return orientation

5      def grad_norme(image):
        gradx, grady = np.gradient(image)
        # Gradient norm
        norme = np.sqrt(gradx**2 + grady**2)
        return norme

10     def grad_threshold(image, seuil):
        norme = grad_norme(image)
        orientation = grad_orientation(image)
        index = np.where(norme < seuil)
        # Selection of the voting pixels
        norme[index] = 0
        return norme, orientation
15

```

Le gradient ∇ permet de connaître le changement dans les différentes directions (∇_x, ∇_y) de chaque pixel où l'application de l'équation 2 (gauche), indique la direction du changement d'intensité le plus important à un pixel particulier que nous appellerons *gradient orientation* et l'application de l'équation 2 (droit), permet de connaître l'intensité du plus grand changement que nous appellerons *gradient norm* qui sont illustrées à la fig. 5.

$$\tanh \nabla = \tanh \frac{\nabla y}{\nabla x} \quad ||\nabla|| = \sqrt{\nabla x^2 + \nabla y^2} \quad (2)$$

Enfin, les changements inférieurs à 20 dans le **gradient norm** pour l'image sélectionnée (cette valeur dépend de chaque image) sont éliminés pour générer un masque qui permet d'écartier les zones sans autant de changements, comme le montre en rouge dans la fig. 5 (droit).

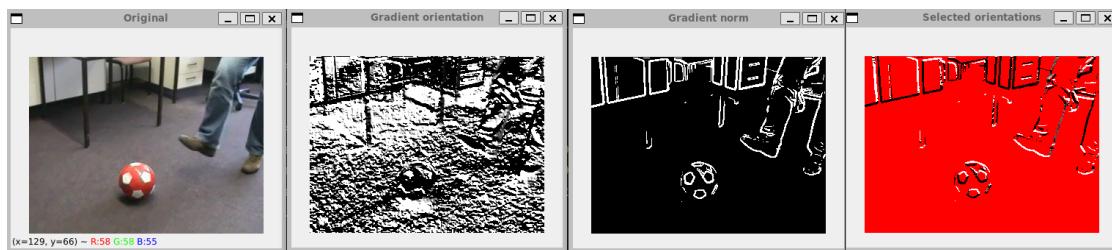


FIGURE 5 – Calcul de l'indice du vote et sélection des pixels de vote

3.2 Tracking avec la transformée de Hough

Le *Tracking* sur la base de la transformée de Hough peut être effectué en réalisant un modèle indexé sur l'orientation des pixels significatifs de l'objet initial à suivre (R-table), qui associe la position relative de chaque pixel au gradient d'orientation pour calculer la transformée de Hough associée à l'ensemble de l'image et suivre la valeur maximale de cette transformée qui représente l'objet dans l'image.

Pour établir *R-table*, on prend le centre de l'image Ω et les pixels significatifs M , qui sont indexés par la orientation de gradient i , comme le montre la fig. 6. Ensuite, on construit *R-table* en ajoutant le vecteur de déplacement $\vec{M}\Omega$ sur la ligne d'index i .

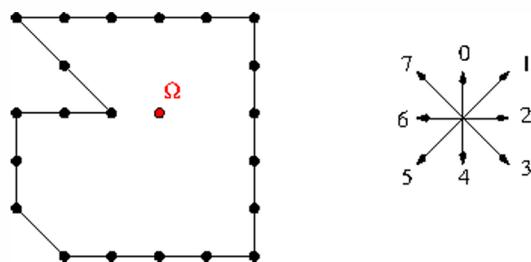


FIGURE 6 – Construction R-table

En d'autres termes, cette table constituera un dictionnaire des orientations des pixels significatifs (contour) et des positions relatives des pixels par rapport au centre de l'objet.

Ensuite, pour chaque image de la vidéo, la transformée de Hough est calculée et renvoie une carte de vote où la localisation la plus votée est la position la plus probable du centre de l'objet à suivre.

```

def hough_transform(r_table, norme, orientation):
    # Create a empty matrix
    vote_map = np.zeros_like(orientation)
    orientation = orientation*90//math.pi
    5
    for (i, j), value in np.ndenumerate(norme):
        if value and orientation[i, j] in r_table:
            for pos in r_table[orientation[i, j]]:
                if 0 < i + pos[1] < orientation.shape[0] and
                   0 < j + pos[0] < orientation.shape[1]:
                    # Increment the accumulator
                    10
                    vote_map[int(i + pos[1]), int(j + pos[0])] += 1

    return vote_map

```

Après avoir construit le modèle et calculé la transformée de Hough associée à chaque image. On va suivre la localisation la plus probable du centre de l'objet avec la valeur maximale de la carte de vote générée par la transformée de Hough, comme indiqué ci-dessous.

```

#Tracking
vote_map = hough_transform(r_table, norme, orientation)

Mpx, Mpy = np.unravel_index(vote_map.argmax(), vote_map.
    shape)
    5
r = Mpx - (h // 2)
c = Mpy - (w // 2)

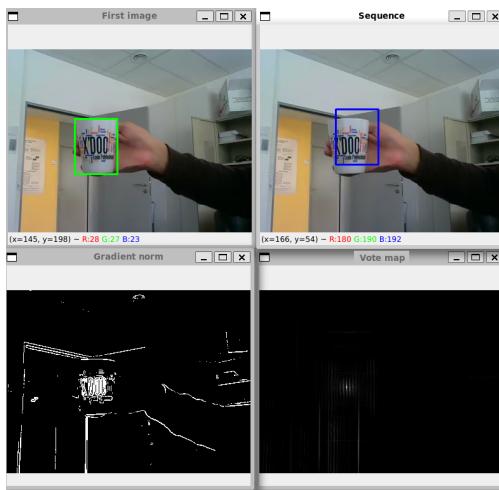
frame_tracked = cv2.rectangle(frame, (r, c), (r + h, c + w),
    (255, 0, 0), 2)

```

L'analyse des résultats permet de noter que l'algorithme obtient une bonne transformation de Hough lors de la première itération, étant donné que l'emplacement de la cible est la meilleure position identifiée dans la carte illustrée dans la partie inférieure droite, comme l'illustre la fig. 7 (a). Ensuite, l'algorithme montre que le calcul de la carte de vote pour localiser l'objet est parfois clairement identifié mais mal suivi, comme dans la fig. 7(b), ou que la détection de la carte de vote est floue ou mauvaise, comme le montrent les figures 7(c) et 7(d), respectivement.

On peut constater qu'en général, la cible est visuellement identifiée sur la carte de vote, mais lorsque la cible obtient le maximum de votes pour générer le suivi, elle prend une mauvaise position et parfois même ne parvient pas à localiser l'objet sur la carte de vote. Cela peut être dû à l'identification d'autres éléments sur la carte de vote, comme le montrent les images où l'on voit clairement le cadre de la porte ou l'objet manquant.

En conclusion, cet algorithme détecte en général correctement la position de l'objet grâce à la caractéristique de forme dans l'utilisation des gradients pour son identification, cependant il présente de nombreux problèmes dans le changement de taille de l'objet par l'approche ou la distance de la caméra et l'illumination est encore un facteur défavorable pour trouver les pixels significatifs et mettre en évidence les contours.



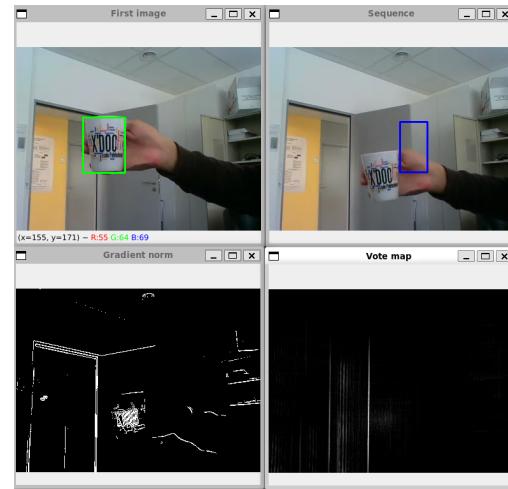
(a) Premier itération



(b) Bonne détection, mauvais suivi



(c) Détection diffuse, mauvais suivi



(d) Détection et suivi insuffisants

FIGURE 7 – Analyse du calculé de la transformée de Hough pour la suivre d'un objet

3.3 Amélioration de la détection avec la transformée de Hough

Une stratégie possible pour améliorer cet algorithme pour certains des sauts de détection qu'il génère fréquemment consiste à enregistrer la position de détection dans chaque image et à limiter la prédiction suivante de la localisation de l'objet à une position proche (à l'intérieur d'un seuil de distance). Pour ce faire, on peut enregistrer le centre de l'objet et rechercher la valeur maximale du vote générée par la transformée de Hough dans la zone définie par le seuil, en tenant compte de la taille de l'objet à détecter pour avoir une référence de la distance à laquelle le centre de l'objet peut être éloigné lors de l'itération suivante.

Une autre amélioration qui pourrait être apportée à cette méthode est l'utilisation de la forme et de la couleur pour la localisation de la cible à l'aide de la transformée de Hough et de la méthode de *Mean Shift*, respectivement. La méthode *Mean Shift* peut être utilisée en complément ou en parallèle de la méthode de la transformée de Hough. En d'autres termes, la méthode du *Mean Shift* peut utiliser le résultat de la transformation de Hough comme carte de possibilité de localisation de la cible au lieu d'utiliser la rétroprojection, ce qui serait un complément qui donnerait plus de robustesse à l'algorithme. D'autre part, les deux méthodes pourraient être mises en œuvre en parallèle pour détecter l'objet dans l'image suivante et, enfin, calculer l'importance de chaque méthode dans la réponse finale en fonction de la proximité du centre de la détection prédite par rapport au centre identifié précédemment. En d'autres termes, La distance par rapport au centre de la prédiction faite par le *Mean Shift* et par la transformée de Hough est calculée, le centre le plus proche du centre de l'image précédente sera prioritaire (minimiser les déplacements de la zone estimée). Le command CV2.ADDWEIGHTED sera mise en œuvre en pondérant la réponse par une valeur alpha α qui attribue un poids plus important à l'algorithme qui donne une prédiction plus proche de l'estimation réelle.

4 Deep Features

Pour améliorer les méthodologies précédentes, l'utilisation d'un réseau neuronal pré-entraîné nous permet d'extraire des features des images vidéo. En intégrant ces features dans nos approches existantes, telles que le Mean Shift ou l'algorithme de Hough, nous améliorons les performances du tracking.

Dans un premier temps, nous avons opté pour ResNet50, un réseau neuronal convolutif (CNN) bien établi et pré-entraîné sur de grands datasets. Dans l'architecture de ResNet, les couches supérieures capturent des features abstraites telles que les formes, tandis que les couches inférieures se concentrent sur des détails plus fins tels que les textures. Ainsi, en choisissant une couche intermédiaire, nous pourrions capturer à la fois des informations spatiales et des caractéristiques sémantiquement significatives.

```
resnet_model = models.resnet50(pretrained=True)
resnet_model.eval()
```

Il convient de noter qu'avant d'être traitées par ResNet, les images doivent être redimensionnées pour s'assurer que toutes les images ont les mêmes dimensions d'entrée, comme l'exige ResNet. En outre, les images sont également normalisées pour s'assurer que les valeurs se situent dans une intervalle appropriée pour le traitement par ResNet, afin de garantir que ResNet produise des résultats cohérents.

```
#Define transformations
preprocess = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=
        [0.229, 0.224, 0.225])
])

#Get the desired intermediate layer (last layer of the third
#block)
desired_layer = resnet_model.layer3[-1]
```

En outre, les couches de ResNet produisent de nombreux canaux de features. Nous nous concentrerons donc uniquement sur les canaux contenant des informations pertinentes pour le suivi des objets. Des techniques telles que l'analyse en composantes principales (ACP) permettent d'identifier les features pertinentes.

Ensuite, au lieu d'utiliser des histogrammes dans la méthode du décalage moyen, nous pouvons construire la fonction de densité de probabilité avec les caractéristiques extraites. Cette approche affinée nous permet d'exploiter l'algorithme de décalage moyen pour estimer précisément l'emplacement de l'objet, améliorant ainsi la précision et la robustesse du tracking.

Dans l'algorithme de Hough, à son tour, nous pouvons appliquer les features pour détecter et localiser les objets d'intérêt dans les images. L'algorithme de Hough fonctionne généralement sur des images avec détection des contours ou des représentations binaires, nous devons donc convertir les features en représentations binaires. Pour ce faire, nous pouvons seuiller les features afin d'obtenir une image binaire, où chaque pixel est classé comme faisant partie soit d'un objet, soit d'un arrière-plan.

Références

- [1] G. BRADSKI : The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
Accessed : March 6, 2024.