

TP4: Analyse vidéo et Tracking

Vision pour la Robotique

ROB313

Yufei HU, Jianzhou MA et Yu WANG

February 27, 2022

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.

Le suivi des objets est une tâche importante en vision par ordinateur et est utilisé dans plusieurs domaines. Le suivi peut être défini comme le processus consistant à déterminer la position d'un objet dans une séquence d'images au fil du temps. Ce suivi est basé sur la reconnaissance d'un motif limité dans la région d'intérêt (ROI), qui est comparé aux caractéristiques structurelles des images de la séquence. Pendant la capture de la scène, le suivi peut être compliqué par des facteurs externes tels que l'intensité de la luminance, le bruit, ainsi que le comportement et la forme de l'objet à suivre.

2 Mean Shift

2.1 Q1:

Mean Shift est un algorithme de clustering non paramétrique basé sur la densité. L'idée de l'algorithme est de supposer que les ensembles de données de différents clusters sont conformes à différentes distributions de densité de probabilité et de trouver la direction la plus rapide dans laquelle la densité de tout point d'échantillonnage augmente (la signification de la direction la plus rapide est Mean Shift). Les régions à forte densité d'échantillonnage correspondent à la valeur maximale de cette distribution. Ces points d'échantillonnage convergent finalement vers les maxima de densité locaux, et les points qui convergent

vers les mêmes maxima locaux sont considérés comme des membres de la même classe de cluster.

L'algorithme est illustré dans la Figure 1 a ci-dessous.

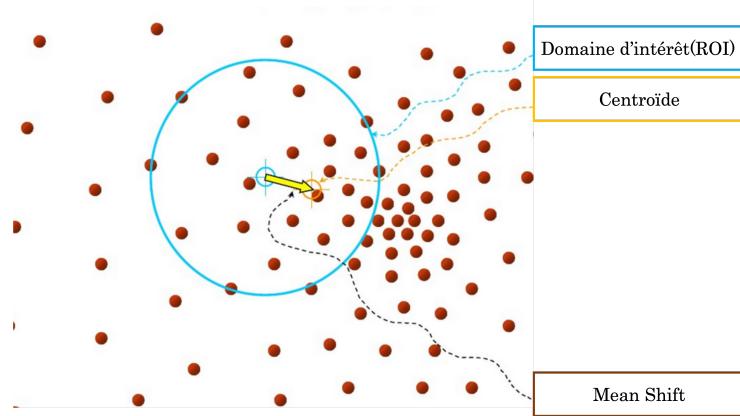


Figure 1: Schéma de principe de l'algorithme(Mean Shift)

Les étapes de l'algorithme sont les suivantes:

- Calcule la moyenne du décalage dans la zone spécifiée(ROI).
- Déplacez le centroïde de la région ROI vers le point moyen de décalage.
- Répétez le processus ci-dessus (calculez la nouvelle moyenne de décalage, déplacez-vous).
- L'algorithme s'arrête jusqu'à ce que la condition d'arrêt soit remplie.

Après l'analyse, on résume la structure du code *Tracking_MeanShift.py* comme suivant:

- Sélectionnez d'abord une vidéo à utiliser pour le test et sélectionnez manuellement la région d'intérêt (RoI) pour tracking dans la première trame de l'image.
- Convertir la région d'intérêt (RoI) de RVB en HSV et calculer l'histogramme de la zone ROI.
- Faites les itérations suivantes sur la séquence vidéo:
 - a. Convertir la trame de l'image de RVB en HSV et utiliser la fonction *cv2.calcBackProject* pour calculer la probabilité dont pixel de l'image appartient au ROI. On peut obtenir le résultat qui est la densité marginale f_H .

- b. Appliquez l'algorithme Mean Shift(*cv2.meanShift*) sur f_H pour nous aider à calculer la nouvelle centroïde et la ROI et puis on dessiner la ROI sur la nouvelle la trame de l'image.

Lorsque l'algorithme **Mean Shift** est utilisé pour le suivi de cible vidéo, l'histogramme de couleur de la cible est utilisé comme fonction de recherche et l'algorithme converge vers la position réelle de la cible en itérant en continu le vecteur meanShift. Enfin atteindre l'objectif de suivi de la cible.

Avantages:

- Le modèle d'histogramme de la fonction du noyau est adopté, qui est insensible à l'occlusion des bords, à la rotation de la cible, à la déformation et au mouvement d'arrière-plan.
- Le calcul de l'algorithme n'est pas important et le suivi en temps réel peut être réalisé lorsque la zone cible est connue.

Limitations:

- Manque de mises à jour de modèles nécessaires.
- Lorsque l'échelle cible change pendant le processus de suivi, l'algorithme de suivi peut échouer.
- Les performances de suivi souffrent lorsque la cible se déplace plus rapidement.
- La fonction d'histogramme manque légèrement dans la description de la fonction de couleur cible et manque d'informations spatiales.

Enfin, on donne un exemple d'application de **Mean Shift** à la vidéo *VOT-Ball.mp4*. L'effet expérimental est illustré dans la Figure 2 suivante. On a constaté que l'algorithme devient moins efficace lorsque la cible est suffisamment proche pour que la taille devienne plus grande et lorsque la cible se déplace trop rapidement.



Figure 2: Un exemple d'algorithme de suivi de cible basé sur **Mean Shift**

2.2 Q2:

Avec l'aide de `cv2.calcBackProject()`, on sait qu'on peut obtenir une rétro-projection de la trame de l'image qui aide l'algorithme **Mean Shift** à trouver la région d'intérêt. Chaque valeur de pixel dans l'image de la rétro-projection représente la probabilité que ce point corresponde à une région d'intérêt (RoI). Plus la valeur du pixel est élevée, plus il est probable que ce point appartienne à l'objet à suivre.

Lorsqu'on essaie de mettre à jour l'histogramme de la région ROI, on obtient les résultats suivants comme le montre la Figure 3.



Figure 3: Schéma de principe de l'algorithme(Mean Shift) avec la stratégie de mise à jour de l'histogramme modèle.

De plus, on aussi montre les résultats de la rétro-projection (obtenu par la fonction `cv2.calcBackProject`) et on constate que après avoir mis à jour l'histogramme de la région roi, on obtient de moins bons résultats. De même, à partir de la Figure 4 et de la Figure 5 ci-dessous, on constate qu'après mise à jour de l'histogramme, les caractéristiques du football sont moins évidentes.



Figure 4: Schéma de principe de l'algorithme(Mean Shift) avec la stratégie de mise à jour de l'histogramme modèle.



Figure 5: Schéma de principe de l'algorithme(Mean Shift) sans la stratégie de mise à jour de l'histogramme modèle.

On pense que la position du football n'est pas bien détectée pendant le mouvement du football, ce qui entraînera l'accumulation d'erreurs et rendra les pixels pertinents de la carte de rétro-projection noirs. Cela réduit considérablement l'effet de suivi du ballon de football.

3 Transformée de Hough

La **Transformée de Hough** (TH) est une méthode de détection de formes facilement paramétrables (lignes, cercles, ellipses) dans une image. En général, la transformation est effectuée après le pré traitement de l'image, généralement pour la détection des bords. Le concept principal de la TH est de définir une relation entre l'espace de l'image et l'espace des paramètres.

Il existe également une variante de la TH appelée la transformée de Hough généralisée (THG). La THG est une technique efficace sur le plan informatique pour détecter ou localiser des formes arbitraires dans des images binaires, à l'aide d'une table de consultation paramétrique appelée table R .

Le processus de création d'une table R se fait en définissant un point de référence $P_c = (x_c, y_c)$ (généralement le centre de la limite du modèle) et un point de contour $P_i = (x_i, y_i)$ du modèle. Pour chaque point P_i , on détermine une paire de paramètres (r_i, α_i) qui sont indexés par l'angle d'orientation θ_i de chaque point P_i , comme le montre la Figure 6.

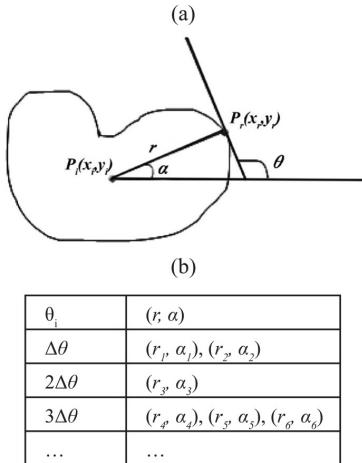


Figure 6: La R-table utilisée pour la méthode de THG [Zaid et al. \(2008\)](#)

Dans cette section, nous allons implémenter la transformée de Hough. En Q3, nous utiliserons la norme de gradient pour définir l'exposant de la transformée de Hough. Seuls les pixels avec une petite norme de gradient seront sélectionnés. Ensuite, en Q4, un modèle d'objet sous la forme d'une table R sera construit et la transformée de Hough sera calculée pour chaque image de la vidéo. Une trace correspondant à la valeur maximale sera également obtenue.

3.1 Q3

Le but de cette question est de définir les indices (directions) de la transformée

de Hough et l'ensemble des pixels votants. On va calculer l'orientation locale de chaque image, i.e. l'argument du gradient des pixels de l'image et le module du gradient. La norme euclidienne du gradient est calculée comme étant:

$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2} = \sqrt{(I * h_x)^2 + (I * h_y)^2} \quad (1)$$

et son orientation:

$$\nabla I_\theta = \arctan \frac{I_y}{I_x} \quad (2)$$

avec

$$h_x = \frac{\partial I}{\partial x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (3)$$

$$h_y = \frac{\partial I}{\partial y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (4)$$

La fonction *Sobel(src_gray, ddepth, x_ordre, y_ordre)* est utilisé pour calculer la dérivée de l'image dans la direction x et y, où:

- *src_gray* est l'image d'entrée en niveaux de gris
- *ddepth* est la profondeur de l'image de sortie
- *x_ordre* est l'ordre de la dérivée dans la direction x
- *y_ordre* est l'ordre de la dérivée dans la direction y

Ici le code associé:

```

1 def gradient_orientation(img):
2     grad_x = cv2.Sobel(img, cv2.CV_64F, 1, 0)
3     grad_y = cv2.Sobel(img, cv2.CV_64F, 0, 1)
4     orientation = np.arctan2(grad_y, grad_x)
5     orientation /= 2 * np.pi
6     orientation += 0.5
7     orientation = orientation.astype(np.float32)
8     orientation = cv2.cvtColor(orientation, cv2.COLOR_GRAY2BGR)
9     return orientation
10
11 def gradient_norme(img):
12     grad_x = cv2.Sobel(img, cv2.CV_64F, 1, 0)
13     grad_y = cv2.Sobel(img, cv2.CV_64F, 0, 1)
14     norme = np.hypot(grad_x, grad_y)
15     norme /= norme.max()
16     return norme

```

Les seuils sont définis selon le tableau 1 ci-dessous, car chaque vidéo donnée produit des résultats différents pour les pixels significatifs dans notre étude, et les seuils doivent donc être adaptés à chaque cas. Pour chaque image, les pixels non importants sont indiqués en rouge. L'index des pixels dans les directions significatives sera utilisé comme index de la transformée de Hough dans la section suivante.

Test-Videos	Seuil
Antoine_Mug.mp4	0.1
VOT-Ball.mp4	0.06
VOT-Basket.mp4	0.15
VOT-Car.mp4	0.1
VOT-Sunshade.mp4	0.05
VOT-Woman.mp4	0.08

Table 1: Valeur des seuils pour chaque vidéo fournie

Ici le code associé:

```

1 frame_clone = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
2 orientation = gradient.orientation(frame_clone)
3 norme = gradient.norme(frame_clone)
4 orientation.mask = orientation.copy()
5 orientation.mask[np.where(norme < th_min)] = [1, 0, 0] # red

```

Les images présentées dans les figures 7 à 12 montrent les résultats obtenus lors de cette étape. Comme vous pouvez le voir, chaque image montre maintenant le contour de l'objet le plus visible dans chaque image, ce qui est utile à la fois pour déterminer le modèle qui compose la table R et pour trouver l'objet à suivre.

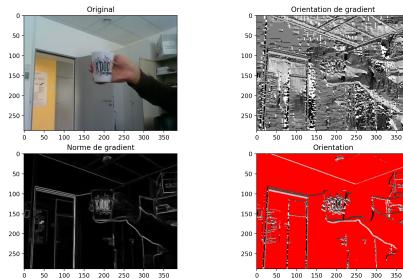


Figure 7: Résultat d'une image de la vidéo "Antoine_Mug.mp4".

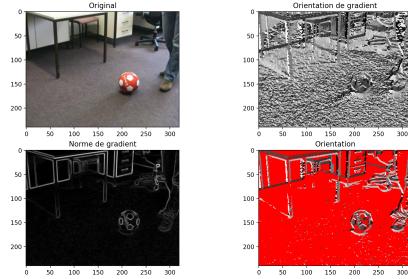


Figure 8: Résultat d'une image de la vidéo "VOT-Ball.mp4".

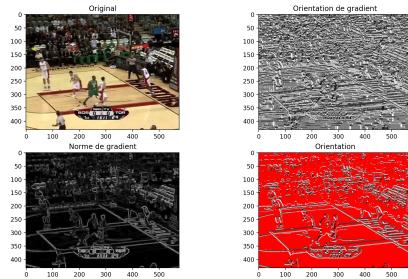


Figure 9: Résultat d'une image de la vidéo "VOT-Basket.mp4".

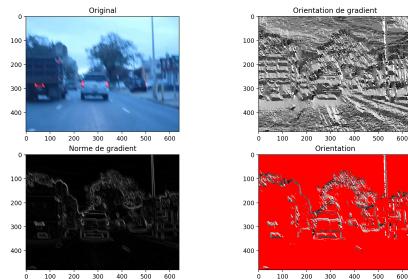


Figure 10: Résultat d'une image de la vidéo "VOT-Car.mp4".

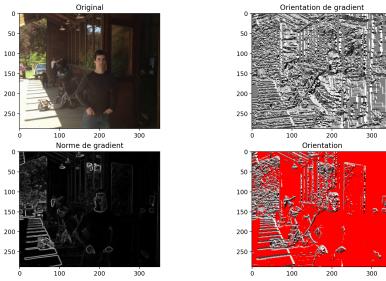


Figure 11: Résultat d'une image de la vidéo "VOT-Sunshade.mp4".

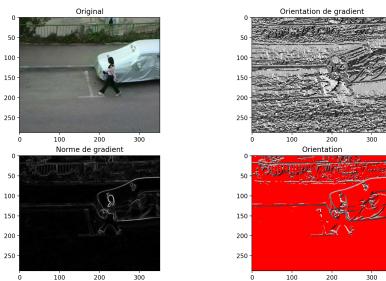


Figure 12: Résultat d'une image de la vidéo "VOT-Woman.mp4".

3.2 Q4

Dans cette partie, on va d'abord établir un R-Table basé sur l'orientation locale obtenue dans la Q3. Un dictionnaire de taille 360 sera donc établi. On va choisir le centre de ROI comme le point arbitraire pour calculer les vecteurs de R-Table. Comme précédemment, les points avec une magnitude de moins de 0.1 seront négligés. Le code associé est présenté comme suivant:

```
1 gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
2
3 #calculer orientation et magnitude
4 gX_roi = cv2.Sobel(gray_roi, cv2.CV_64F, 1, 0)
5 gY_roi = cv2.Sobel(gray_roi, cv2.CV_64F, 0, 1)
6 magnitude_roi = np.sqrt((gX_roi ** 2) + (gY_roi ** 2))
7 magnitude_roi/=magnitude_roi.max()
8 orientation_roi = np.arctan2(gY_roi, gX_roi) * 180 // np.pi
9
10 #Definir un seuil
11 lower_mag=np.array([.1])
12 higher_mag=np.array([1.])
13 mask_roi = cv2.inRange(magnitude_roi, lower_mag, higher_mag)
14
15 #etablir R-Table
16 img_center = [int(roi.shape[0] / 2), int(roi.shape[1] / 2)]
17 r_table = defaultdict(list)
18 for (i, j), value in np.ndenumerate(mask_roi):
19     if value:
20         r_table[orientation_roi[i, j]].append((img_center[0] - ...
21             i, img_center[1] - j))
```

Une fois le modèle établi, on va calculer la transformée de Hough pour chaque image de séquence et implémenter le suivi selon le principe de vote. Pour la ligne d'orientation correspondante dans le dictionnaire R-Table, on va ajouter les vecteurs associés dans l'espace de Hough. Ici, la stratégie de fiabilité n'est pas implémentée, donc tous les points ont une fiabilité de 1. Voici le code associé:

```
1 hough = np.zeros((frame.shape[0]+int(0.2*frame.shape[0]), ...
2                  frame.shape[1]+int(0.2*frame.shape[0])))
3     for (i, j), value in np.ndenumerate(mask_tracked):
4         if value:
5             vectors=r_table[orientation_tracked[i,j]]
6             for vector in vectors:
7                 hough[vector[0]+i, vector[1]+j] += 1
8             hough /= hough.max()
9
10    #Arg max
11    r_index, c_index = np.unravel_index(hough.argmax(), ...
12                                         hough.shape)
13    track_window=r_index-h//2,c_index-w//2,h,w
```

On va tester le code sur les vidéos fournies. Pour la vidéo de verre, on a les

résultats comme:

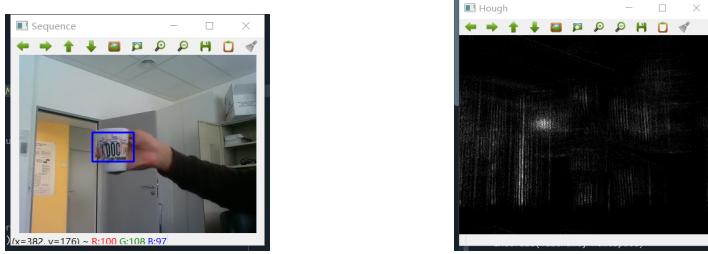


Figure 13: Détection de Hough selon Figure 14: Schéma de Hough selon la valeur maximale pour le verre valeur maximale pour le verre

On note que le contexte vote aussi dans l'espace de Hough, notamment pour la région de porte. Cela causera des problèmes quand l'image de verre devient floue, notre ROI va se localiser dans la région de porte en conséquence. Ce problème est moins grave s'il y a moins d'ambiguïté sur les orientations locales:

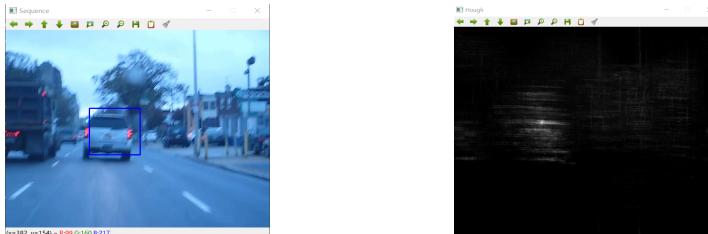


Figure 15: Détection de Hough selon Figure 16: Schéma de Hough selon la valeur maximale pour la voiture valeur maximale pour la voiture

Et plus grave pour les images compliquées:

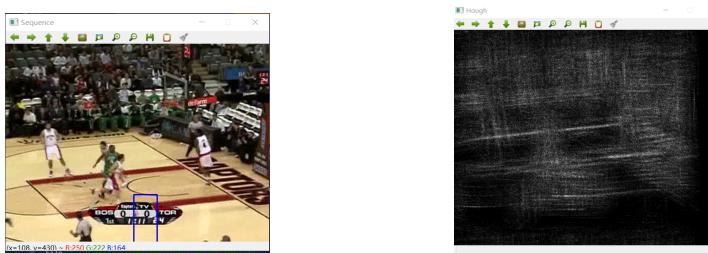


Figure 17: Détection de Hough selon Figure 18: Schéma de Hough selon la valeur maximale pour le basket valeur maximale pour le basket

3.3 Q5

Dans cette partie, on va remplacer le calcul du maximum par l'application de Mean Shift. Voici le code associé:

```
1 #Mean-shift
2         ret, track_window = cv2.meanShift(hough, track_window, ...
3                         term_crit)
4         r,c,h,w=track_window
5         frame.tracked = cv2.rectangle(frame, (r,c), (r+h,c+w), ...
6                                         (255,0,0) ,2)
```

On constate que la méthode de Mean Shift donne une meilleure performance sur les objets mobiles, par rapport au calcul de maximum. Voici quelques exemples:

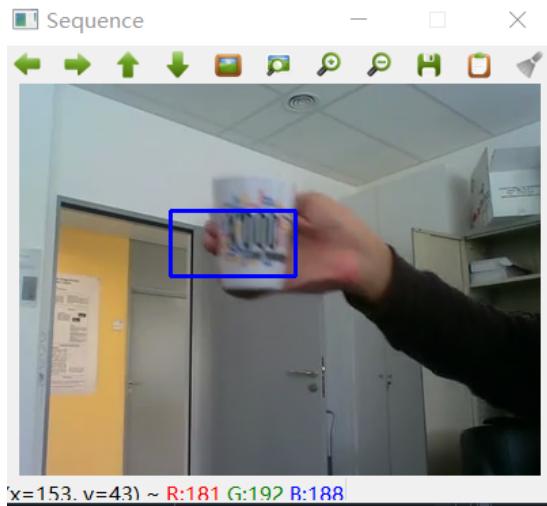


Figure 19: Détection de Hough selon Mean-Shift pour le verre

Pour réaliser une stratégie de mise à jour du modèle, il suffit de choisir la région présente comme le ROI et rétablir le R-Table. Le code associé est le suivant:

```
1 #update R-Table
2     roi=magnitude_tracked[c:c+w,r:r+h]
3     mask_roi_tracked = cv2.inRange(roi, lower_mag, higher_mag)
4     img_center = [int(roi.shape[0] / 2), int(roi.shape[1] / 2)]
5     r_table = defaultdict(list)
6     for (i, j), value in np.ndenumerate(mask_roi_tracked):
7         if value:
8             r_table[orientation_tracked[i, ...
```

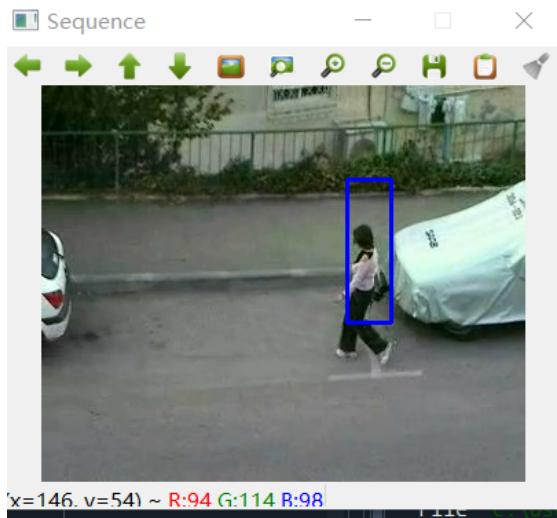


Figure 20: Détection de Hough selon Mean-Shift pour la femme



Figure 21: Détection de Hough selon Mean-Shift pour le basket

On constate que cette méthode est plus robuste pour les objets déformés à cause de mouvement:

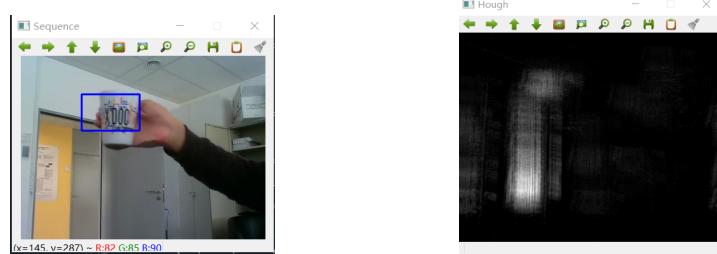


Figure 22: Détection de Hough avec la Figure 23: Schéma de Hough avec la mise à jour de modèle pour le verre mise à jour de modèle pour le verre

References

- Zaid, A. O., Hadded, I., Belhaj, W., Bouallegue, A., Abdessalem, S., and Mechmeche, R. (2008). Improved localization of coronary stents based on image enhancement. *International Journal of Biomedical Science: IJBS*, 4(3):212.