ROB317 - TD2

BRAMBILLA Davide Luigi - GOMES DA SILVA Rafael

7 Novembre 2019

Les codes se trouvent dans notre dépôt *git* accessible au lien:
https://gitlab.data-ensta.fr/brambilla/rob317.git

Nous avons écrit un code diffèrent pour chaque point du TD nommé *Qx.py* afin de pouvoir montrer les differents étapes. Les fichiers *Qx_functions.py* sont des fichiers contenant l'affichage des functions.

1 Histogramme de couleurs

Q1 - L'histogramme 2D des composantes chromatiques (u,v)

Il nous est demandé d'afficher pour chaque image de la vidéo un histogramme en deux dimensions correspondant à la probabilité jointe des composantes chromatiques (u, v) relatives au codage Yuv des images couleur.

Le codage Yuv prend en compte la perception des humaines car, au travers la possibilité d'avoir une bande réduite, permet de rendre les erreurs de transmission et les artefacts dûs à la compression moins visibles que dans le cas d'utilisation de la représentation RGB. Cette représentation définie l'espace des couleurs avec trois composantes:

- Y qui indique la *luma* (luminosité) de l'image. Sa relation avec la classique représentation RGB est que $Y = \alpha R + \beta G + \gamma B$ où les coefficients α,β e γ donnent une moyenne pondérée par la sensibilité humaine aux couleurs primaires.
- u qui est le premier composant de *chrominance* (lié à la perception du blue). Elle est calculé comme $U = \eta(B - Y)$. En particulier elle donne le contraste *Bleu-Jaune*.
- v qui est le premier composant de *chrominance* (lié à la perception du rouge). Elle est calculé comme $V = \lambda(R - Y)$. En particulier elle donne le contraste *Rouge-Cyan*.

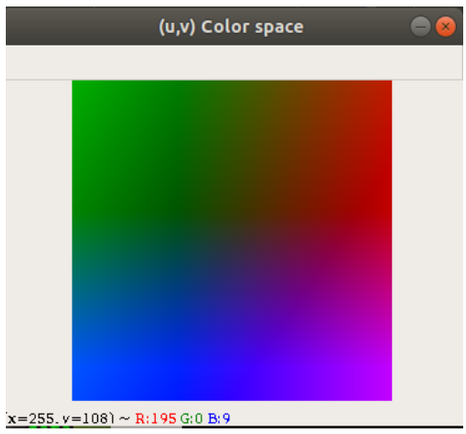


Figure 1: Espace de couleur (u,v)

Nous devrons donc convertir notre vidéo vers la représentation en Yuv. Il est en fait possible de passer de l'état des couleurs RGB à l'état Yuv en multipliant chaque pixel de l'image RGB pour la matrice suivante(qui contient les différents combinaisons des valeurs des coefficients):

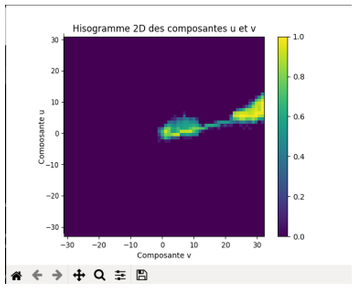
$$\begin{bmatrix} Y \\ u \\ v \end{bmatrix} = \begin{bmatrix} 0.2989 & 0.5866 & 0.1145 \\ -0.1687 & 0.3312 & 0.5 \\ 0.5 & -0.4183 & -0.0817 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Pour faire cela nous avons utilisé la fonction *img_yuv = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)*. En exécutant le code *Q1.py* il est possible d'observer l'histogramme des probabilités conjointes que nous avons obtenu et comment il évolue selon la vidéo que on est en train de considérer. Pour tester les résultats, il est possible de choisir entre une vidéo prédéfinie au travers du commande *capture = cv2.VideoCapture("Parcours du vidéo")* et la vidéo en directe de notre propre webcam au travers du commande *capture = cv2.VideoCapture(0)*.

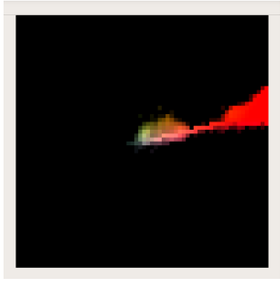
Il a été intéressant d'étudier comment l'histogramme évoluait si, à l'intérieur de notre vidéo, nous augmentons certaines composantes de couleurs. Pour mieux visualiser comment la probabilité jointe évolue nous avons fait une normalisation logarithmique des données pour donner plus d'importance aux changements dans un range des valeurs petites étant donnée que c'est dans des valeurs petites que la grand majorité des changements se passe. Nous avons aussi choisi de superposer les couleurs sur l'histogramme de façon qu'il puisse avoir la couleur correspondante à la position dans ls Figure 1.

Ici nous avons reporté l'exemple d'un frame avec une grande composante **rouge** et l'histogramme correspondante:

(a) Rouge



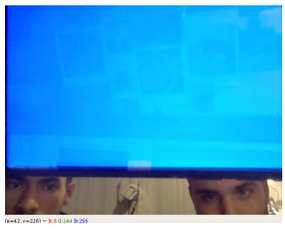
(b) Hist avec une normalisation logarithmique des valeurs



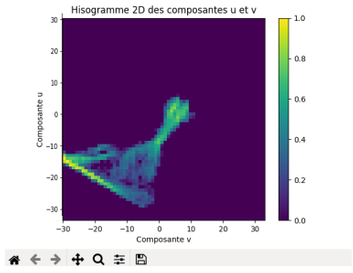
(c) Hist Coloré avec la position dans l'espace couleurs Yuv

Figure 2: Frame avec une forte composante rouge et son histogramme

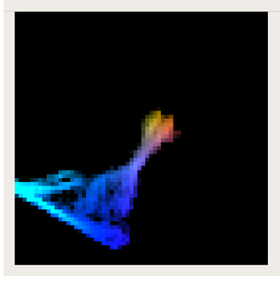
Ici nous avons reporté l'exemple d'un frame avec une grande composante **bleu** et l'histogramme correspondante:



(a) Bleu



(b) Hist avec une normalisation logarithmique des valeurs



(c) Hist Coloré avec la position dans l'espace couleurs Yuv

Figure 3: Frame avec une forte composante bleu et son histogramme

Enfin, comme dernière essaie, nous avons appliqué l'algorithme à la vidéo proposée *Extrait3-Vertigo-Dream_Scene(320p).m4v* et nous avons vu que les changements de couleurs, i.e. vers le rouge et le bleu, sont bien détectés et evidenciés.

Détection des changements de plan

Après avoir analysé les histogrammes obtenus à partir d'un vidéo, nous nous sommes demandés comment c'était possible de détecter des changements de scène. Nous nous sommes dédiés à étudier les changements dans la vidéo *Extrait1-Cosmos_Laundromat1(340p).m4v* donnée dans l'archive des vidéos prédéfinies.

Nous avons pensé de faire la corrélation entre deux différents histogrammes obtenus en deux images successives de la vidéo.
En particulier si la valeur de la corrélation est haute et ne change pas au cours de la vidéo, il est possible d'affirmer que la scène est resté la même où il n'y a pas eu un changement radical. Si par contre la valeur de la corrélation est basse et change rapidement par rapport à la valeur précédente, dans la vidéo il y a eu un brusque changement de scène.

Dans le code, nous avons utilisé la fonction *cv2.compareHist(hist_mask_old, hist_mask, cv2.HISTCMP_CORREL))* où *hist_mask_old* est l'histogramme du frame précèdent, *hist_mask* est l'histogramme du frame courant et *cv2.HISTCMP_CORREL* est la directive qui permet de calculer la corrélation entre les histogrammes.

Nous avons reporté dans la figure 4 le graphique de la corrélation des histogrammes obtenus à partir de la vidéo.

Nous pouvons observer que les zones où la valeur du graphique a une valeur constante sont le zones où l'image maintient le même fond.
Les changement dans le graphique signifient que l'image est en train de changer son fond entre un frame et le suivante.
Il est possible de suivre la construction du graphe en temps réel ensemble à la reproduction de la vidéo de façon de se rendre compte que les pics décroissants correspondent par vraie aux changements du plan dans la vidéo.

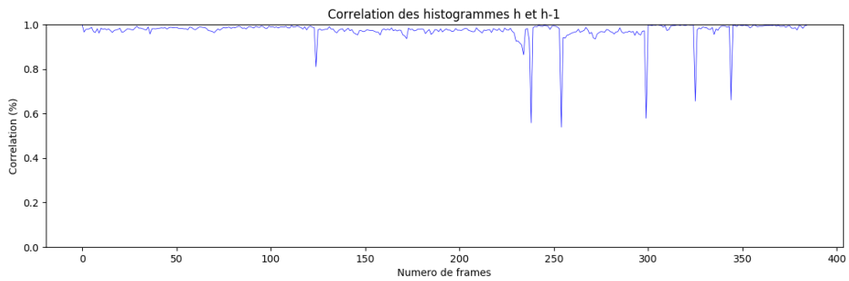
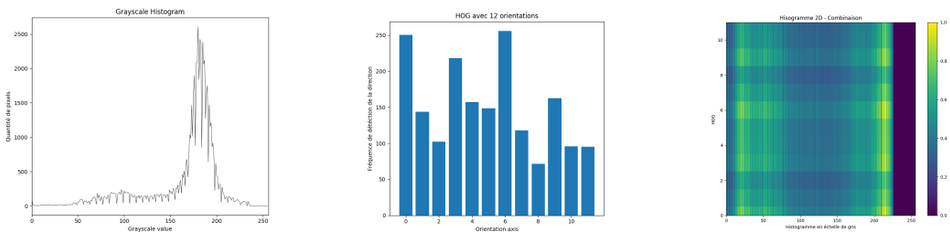


Figure 4: Comparaison au cours du temps de la corrélation entre 2 frames

Cas de Vidéos Monochromes

Pour traiter le cas d'une vidéo monochrome, nous avons initialement choisi de traduire la vidéo traitée dans le point précèdent en une échelle de gris. Pour cela, nous avons étudié une histogramme de la distribution des niveaux des gris qui reportait sur l'axe horizontal la valeur de gris entre 0 et 255 et sur l'axe vertical le numéro de pixels qui ont une même valeur pour chaque frame de la vidéo.
Nous avons obtenus des résultats satisfaisantes pour la vidéo *RGB* traduite en noir et blanc en tant que nous arrivons à voir le changement de plans en effectuant une corrélation entre l'histogramme de l'image N et lequel de l'image précédente.

En suite, nous avons considéré la vidéo noir et blanc fournie et nous avons eu des problèmes dans la détection des changements de plan. Pour cela nous avons amélioré notre algorithme en calculant trois différents histogrammes: un premier histogramme unidimensionnel qui calcule le numéro de pixels qui ont un certain niveau de gris, un deuxième qui calcul la approximation *hog* du frame courant (qui consiste en calculer l'orientation local du gradient) et le troisième bi-dimensionnel qui unifie les deux mesures. Nous avons reportés dans la figure 5 les trois histogrammes:



(a) Niveau de gris (b) Hog (c) Combinaison des deux histogrammes

Figure 5: Les trois histogrammes obtenus

Enfin, pour pouvoir détecter le changement de plan, nous avons fait encore des comparaison entre l'histogramme du frame courant et du frame précèdent. Dans le premières deux histogrammes, nous calculons l'erreur (1 - corrélation) entre les deux histogrammes et pour le troisième nous calculons l'erreur quadratique moyen entre les pixels correspondants des deux histogrammes. Nous avons obtenu:

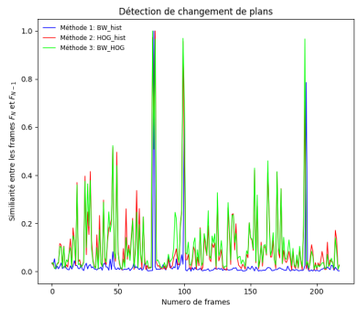


Figure 6: Comparaison des histogrammes dans le trois cas

En particulier, dans le graphique trois lignes sont reportés: une ligne **bleu** pour la comparaison des histogrammes du niveau de gris, une ligne **rouge** pour le *hog* et **vert** pour la combination des deux histogrammes.

Il est possible de voir que, les changements de plan dans la vidéo sont bien décrit par les pics positifs (car maintenant c'est l'erreur qu'indique une changement de plan) verts du graphique et c'est pour cela que nous avons choisi de faire une combinaison entre les premières deux méthodes.

2 Flot Optique et Histogramme de Vitesses

Le *Flot Optique* consiste dans l'étude du champ de mouvement apparent: à chaque pixel nous allons associer un vecteur (v_x^t, v_y^t) qui représente la *vitesse* du pixel (x, y) à l'instant t.

Q2 - L'algorithme *Dense-Optical-Flow.py*

Analyse du comportement et des arguments de la fonction de calcul du flot optique dense

La fonction qui est utilisée pour calculer le flot optique est *cv2.calcOpticalFlowFarneback(...)* qui utilise l'algorithme de Farneback.

Cette méthode nous permet de visualiser le deplacement des points dans la vidéo. En particulier, elle permet de détecter la direction dans laquelle les points du vidéo se déplacent. Pour la visualisation elle va assigner des différents couleurs selon la direction du déplacement. Dans l'image suivante nous avons reportés les differents couleurs: bleu pour un mouvement vers le bas, jaune pour un mouvement vers le haut, etc.

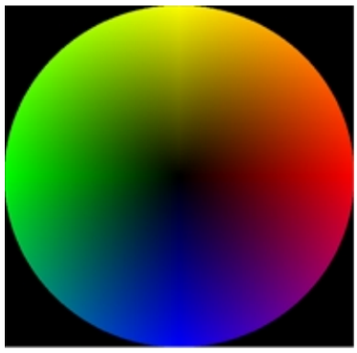


Figure 7: Couleurs utilisés pour indiquer la direction

Dans le code nous avons:

```
flow = cv2.calcOpticalFlowFarneback(prvs,
                                    nex t,
                                    None,
                                    pyr_scale = 0.5,  # Taux de reduction pyramidal
                                    levels = 3,       # Nombre de niveaux de la pyramide
                                    winsize = 15,     # Taille de fenetre de lissage des
                                        coefficients polynomiaux
                                    iterations = 3,   # Nombre d'itrations par niveau
                                    poly_n = 7,       # Taille voisinage pour approximation
                                        polynomiale
                                    poly_sigma = 1.5, # E-T Gaussienne pour calcul derives
                                    flags = 0)
```

où:

- **prvs** qui représente la première image en input.
- **next** qui représente la deuxième image en input.
- **flow** qui est mis à *None* et se réfère au Flot calculé qui a la même taille de prvs et il est de type *CV_32FC2*.
- **pyr_scale** qui doit être plus petite de 1 et spécifie l'échelle de l'image pour la construction de la pyramide pour chaque image. Dans notre cas, nous avons que *pyr_scale=0.5* qui signifie que chaque couche est 2 fois plus petite par rapport à la couche précédente.
- **levels** qui choisit le nombre de couches de la pyramide y compris l'image initiale. *levels = 3* impose que il y aura deux autre couches dans la pyramide en plus de l'image initiale.
- **winsize** qui indique la taille de la fenêtre de lissage. Une valeur grande augmente la robustesse de l'algorithme au bruit et donne plus de moyens pour la détection des mouvements rapides mais, par contre, donne un champ de mouvement flou.
- **iterations** qui représente le nombre d'itérations qui l'algorithme fait à chaque couche de la pyramide.
- **poly_n** qui indique la taille de voisinage des pixels pour trouver l'expansion polynomiale en chaque pixel. Une valeur grande rend l'algorithme plus robuste et estime l'image avec des surfaces plus doux. Par contre, il donne un champ de mouvement plus flou.

- **poly_sigma** qui représente l'écart type de la Gaussienne utilisée pour calculer les dérivés employés pour représenter l'expansion polynomiale.
- **flags** qui selon le valeur qui est lui donné utilisera une mesure différente de l'erreur. Cette variable peut prendre deux valeurs differentes: *OPTFLOW_USE_INITIAL_FLOW* qui utilise l'estimation initiale fait dans *next* où *OPTFLOW_FARNEBACK_GAUSSIAN* qui utilise une fenêtre *Gaussienne* carrée de taille *winsize*. Dans le deuxième cas, l'algorithme résulte plus lent mais plus précis dans le calcul.
  Dans notre cas elle est initialisé à zéro et elle n'est pas définie: donc nous aurons que *prev* sera copié en *next* et l'estimation initiale sera considérée.

La valeur retournée par cette fonction est sauvegardée dans la variable *flow* et il représente le flot optique de la vidéo, c'est à dire elle donne une information sur la direction vers laquelle les pixels se déplacent.

Le principe de l'algorithme

L'algorithme repose son fonctionnement sur l'approximation entre deux frames consécutifs du voisinage d'un certain pixel avec une forme polynomiale et, en suite, sur l'estimation du déplacement qui est faite en partant des coefficients de l'expansion polynomiale obtenue.

En particulier nous allons associer au voisinage x du pixel dans le première frame le polynôme:

$$f_1(x) = x^t A_1 x + b_1^t x + c_1 \tag{1}$$

et, en supposant que dans le deuxième frame il y a eu un mouvement d, on associera le polynôme suivant au voisinage du même pixel mais déplacé d'un valeur d:

$$f_2(x) = f_1(x-d) = (x-d)^t A_1 (x-d) + b_2^t (x-d) + c_2$$

en obtenant:

$$f_2(x) = (x-d)^t A_1 (x-d) + b_2^t (x-d) + c_2$$

où on devra calculer la distance d.

Nous aurons que:

$$A_2 = A_1$$
$$b_2 = b_1 - 2A_1 d$$
$$c_2 = d^t A_1 d - b_1^t d + c_1$$

et en considérant la deuxième équation:

$$2A_1 d = -(b_2 - b_1)$$

Nous pouvons obtenir finalement l'estimation de a distance d et donc du mouvement que l'image a effectué.

$$d = -\frac{A_1^{-1}(b_2 - b_1)}{2}$$

En suite, quelque approximation est effectué pour maintenir les erreurs petites et avoir toujours une bonne estimation et ne pas être trop influencé par le bruit.

L'avantage de cette méthode réside dans le fait que l'algorithme fonctionne de façon ***itérative et multi-échelle*** sur l'estimation du déplacement.
Dans notre cas, comme déjà explique on fera trois itérations pour chaque échelle et on aura une analyse à trois échelles.
L'analyse *itérative* permet d'avoir une meilleur précision dans les calculs mais elle a comme point faible le fait que, si le déplacement est trop grand, elle n'arrive pas à le capturer et donc avoir plusieurs itérations sera insignifiant.
Ce problème est résolu en faisant le calcul sur *plusieurs échelles*, avec différents résolutions, de façon que la methode puisse gérer des plus grand mouvements. Cette approche, par contre, ramène à une mineure précision et à un coût computationnel plus élevé.

Le deux approches sont utilisés au même temps pour pouvoir exploiter leur avantages et minimiser leurs défis.

Pour résumer, cet algorithme peut estimer grand vitesses pour le fait que il est fait à niveau multi-échelle et il peut aussi calculer un flot dense en considérant à chaque fois toutes les pixels de l'image car les coefficients des polynômes, pour réduire le coût computationnel, vont être calculés seulement la première fois et, en suite, réutilises.

Q3 - L'histogramme 2D des composantes (V_x, V_y)

Ici il nous est demandé, comme dans la Q1, de afficher sous la forme d'une image la probabilité jointe des composantes horizontales et verticales du flot optique.

En particulier nous avons que la fonction de *Farneback* utilisé dans le point précèdent retourne deux tableaux bi-dimensionnels différents: un qui contient les composantes du mouvement optique horizontales et

un qui contient les composantes du mouvement optique verticale.

Nous avons reporté donc les deux composantes dans un histogramme bi-dimensionnel. Ce que on a observé dans un premier moment, c'était que la variation des pixels était très peu évident dans l'histogramme: pour cela nous avons choisi de supprimer la majorité des données statiques qui ne représentaient pas un changement de façon de donner plus d'importance aux vraies changements.

Nous avons observé que, dans le cas du changement de scène, l'histogramme reporte une *explosion* vers toutes les directions: c'est à dire il est possible de observer que la figure s'étend dans toute les directions. Cela est du au fait que il y a un changement significatif dans toutes les directions et nous l'associons au changement de plan. Dans la vidéo proposée, par exemple, nous avons un exemple de cela dans le moment où la scène change: au début en ayant le visage de la personne et en suite en reportant le ciel noir. Nous avons reporté l'image de l'histogramme dans la Figure 9.

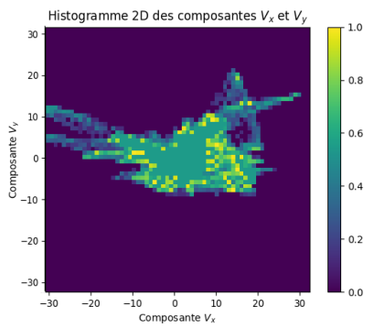


Figure 8: Histogramme qui reporte V_x et V_y

De plus, il nous est demandé de expliquer comment on pourrait exploiter l'information donnée par l'histogramme pour identifier le type de plan dans la figure actuelle.

Les différents types de plan

Il est possible d'étudier la différence entre différents types de plan en utilisant l'histogramme qui nous avons développé à la question précédente.

Ici nous avons cherché en ligne où effectué avec la caméra des vidéos qui représentent des différents mouvements de la caméra et nous avons reporté une brève explication ensemble à l'histogramme qui peut être associé à un certain type de plan.

Plan Fixe

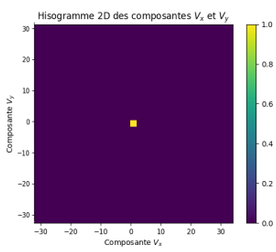


Figure 9: L'histogramme pour un plan fixe

Le *plan fixe* reporte avec lui tous les pixels statiques et donc, nous n'aurons pas de mouvement détecté par l'algorithme de calcul du flot de données. En fait le graphique que nous avons obtenu se présente con un point centrale dans la position zéro qui signifie que aucune des voisinages de pixel est en train de bouger.

Panoramique Horizontal

Le mouvement de *pan* de la caméra consiste en un déplacement des pixels en une certaine direction. Dans la figure, un mouvement de la caméra vers la gauche me donnera un histogramme avec un nuage de points orienté vers la droite car les pixels auront un mouvement opposé à lequel de la caméra.

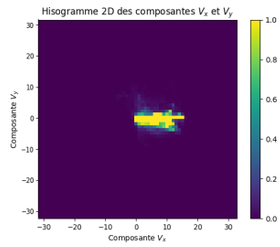


Figure 10: L'histogramme pour un mouvement de *pan*

Panoramique Vertical

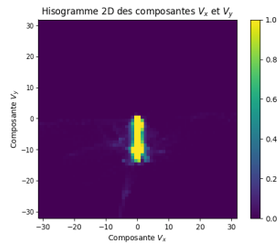


Figure 11: L'histogramme pour un mouvement de *tilt*

Le mouvement de *tilt* de la caméra consiste, aussi dans ce cas, en un déplacement des pixels dans le sense opposé. Dans ce cas, nous aurons un nuage de points verticale dans le sense opposé par rapport au mouvement de la caméra: en exemple vers le haut si le *tilt* est vers le bas.

Rotation

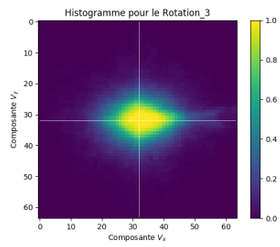


Figure 12: L'histogramme pour un mouvement de rotation

Pour un mouvement de *rotation*, l'histogramme sera caractérisé par un gros nuage de points car les pixels sont en train de se déplacer dans le sense de la rotation et donc nous aurons lesquels qui se déplacent peu (et auront un vecteur vitesse petit) car ils sont proches du centre de rotation et lesquels qui se déplacent plus (et auront un vecteur vitesse grand) pour le fait qu'il se trouvent loin du centre de rotation. En outre le deplacement sera fait vers toutes les directions et pour cela nous aurons une histogramme circulaire.

Travelling horizontal

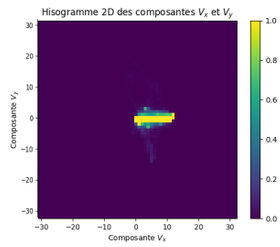


Figure 13: L'histogramme pour un mouvement de travelling horizontal

Le mouvement du caméra qui corresponde à un mouvement de *travelling horizontal* donne, dans l'histogramme, une nuage des points qui se déplace dans la direction opposée au mouvement comme dans le cas d'une panoramique horizontal.

Travelling vers l'avant

Le *travelling vers l'avant* donne un nuage des points circulaire avec un rayon moyen. Cela signifie que tous les points de l'image sont en train de être soumis à un déplacement. En particulier, les pixels de gauche se

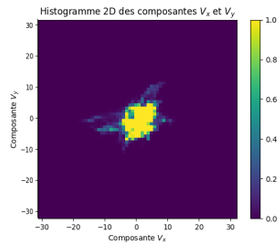


Figure 14: L'histogramme pour un mouvement de travelling vers l'avant

déplaceront vers la gauche, les pixels de droite se déplaceront encore vers la droite, les pixels de bas vers le bas etc. Cela explique la présence d'un nuage de points dans toutes les directions.

Zoom avant

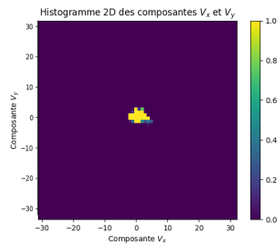


Figure 15: L'histogramme pour un mouvement de zoom avant

Le *zoom* donne un nuage de points circulaire aussi mais avec un rayon plus court. Cela est dû au fait que notre visuel est en train de se réduire vers un seul objet (su lequel on et en train de faire le zoom) et donc les déplacements seront plus petites par rapport à un mouvement de travelling.

La détection du type de plan

Pour la détection du type de plan on pourrait exploiter le module des vecteurs vitesse représentés e leur direction. En particulier, nous avons pensé de diviser l'histogramme obtenu en différents parties sur lesquelles nous ferons une somme pour pouvoir identifier vers quelle direction et avec quelle intensité l'histogramme est en train de se déplacer. Nous allons implémenter cette méthode dans la Q6 et nous discuterons les difficultés rencontrés et les limites de cette approche.

3 Découpage et Indexation

Q4 - Technique de découpage en plans

Pour effectuer la détection des différents plans, nous devrons choisir une seuil qui puisse différentier les pics qui effectivement représentent un changement de plan.

Pour faire cela nous avons pensé d'unifier les deux méthodes précédemment développées(la détection en utilisant l'histogramme qui se réfère aux champ Yuv et laquelle qui utilise le flot optique) et d'en ajouter une troisième qui est laquelle du *hog* pour pouvoir exploiter un caractéristique invariant aux changement de couleur. Cette troisième méthode a été nécessaire après avoir testé notre algorithme sur la vidéo *Vertigo* qui présente plusieurs changements de couleurs et qui donnait des problèmes surtout dans la méthode référé au champ Yuv.

De plus avoir trois méthodes différents pour détecter le changement de plan nous permet d'être plus robuste dans l'algorithme et utiliser une décision par vote majoritaire nous permet d'exploiter les points de force des trois différents méthodes. Dans le trois méthodes, nous effectuons une corrélation entre le frame courant et le frame précèdent et nous classifions un pic négatif comme changement de plan dans le cas où la valeur de la corrélation descend en dessous d'une certaine seuil σ. La valeur finale de cette comparaison sera une valeur binaire (0 dans le cas où la méthode n'a pas détecté un changement de plan et 1 dans le cas où la méthode a détecté un changement de plan) comme il est expliqué ci-dessous où $\boldsymbol{Corr_k}$ représente la corrélation entre les frames t et $t-1$.

$$\boldsymbol{Corr_k} = \begin{cases} 0, & \text{if } Corr(I, I_{t-1}) > \sigma \\ 1, & \text{if } Corr(I, I_{t-1}) \leq \sigma \end{cases}$$

En particulier nous pourrons avoir deux différents situations:

- la première où on a une connaissance des caractéristiques de la vidéo avant de l'analyser et, comme cela, nous pouvons choisir un pondération des trois méthodes pour effectuer la choix de façon d'exploiter la méthode qui s'adapte mieux aux caractéristiques de la vidéo.

  $$Changement\ de\ plan = \alpha\ Corr_{Yuv} + \beta\ Corr_{flotoptique} + \gamma\ Corr_{HOG}$$

  Dans ce cas, nous n'aurons pas besoin d'un vote majoritaire car nous pourrons choisir les coefficients pour la pondération.

- le deuxième dont on n'aura pas des connaissance à priori sur la vidéo. Dans ce cas, nous allons appliquer les trois méthodes séparément et nous allons appliquer une **vote majoritaire** pour choisir si un pic représente un changement de plan où pas. Comme déjà anticipé, cela nous permet de exploiter les avantages des trois méthodes et de réduire les défis.

Nous allons présenter ici, pour chaque méthode, leurs avantages et leurs défis par rapport à une analyse faite sur les vidéos proposées.

Corrélation des histogrammes lié à Yuv

Cette méthode fonctionne bien sur les vidéos 1 et 5 avec un seuil $\sigma = 0.84$. Dans la vidéo numéro 1, la grand majorité des plans sont détectés. Comme désavantage, cette méthode a le fait de ne pas être robuste par rapport aux changements brusques de luminosité où de vitesse (c'est possible de le voir entre le frame 1500 et 2000 du vidéo 1). Pour ce que concerne la vidéo 3, cette méthode n'est pas optimale parce que elle est très sensible aux changements de couleur et souvent, une changement brusque de couleur, donne un signal qui indique un faux changement de plan.

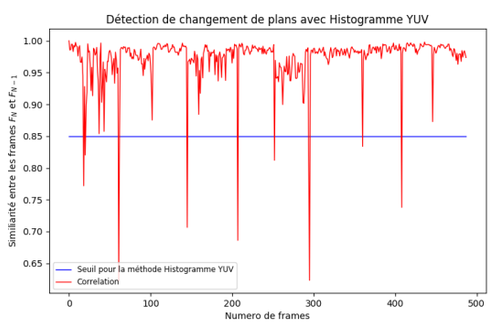


Figure 16: Corrélation des histogrammes lié à Yuv

Corrélation des histogrammes lié au *Flot Optique*

Dans cette méthode, nous allons exploiter les informations données par la vitesse à laquelle les pixels se déplacent dans l'image. Nous avons choisi une seuil de 0.2. Par rapport à la méthode précédente, nous avons que il est moins sensible aux changements brusque de la position des pixels mais, par contre, il reste le problème de la lumière e du couleur qui, à la fin de la vidéo 1 (frames 2000-2500) et dans la vidéo 3 qui présente beaucoup de changement de vitesse. De plus, quand nous avons une changement de plan fait par un *fade out*, c'est à dire quand le changement de plan est fait progressivement, nous n'arrivons pas à le bien détecter: cela est dû a fait que la transition est faite lentement et les changements ne sont pas bien mis en évidence dans l'histogramme de flot optique. Une autre défis qui on retrouve dans la vidéo 5, est le fait que quand la caméra a plusieurs mouvements de *tilt* cette méthode ne détecte pas le changement. Une autre desavantage à niveau des performances de cette methode est le fait qui prends du temps pour s'exécuter et resulte plus lente par rapport aux autres deux méthodes.
Nous avons reporté ici le graphique obtenu:

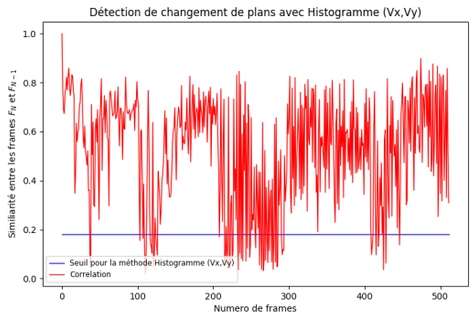


Figure 17: Corrélation des histogrammes lié au *Flot Optique*

Corrélation des histogrammes lié à *Hog*

Enfin, nous nous sommes dédiées à la recherche d'une méthode qui permettait de ne pas être trop sensible aux changements de couleurs surtout dans la vidéo 3. Nous avons choisi d'utiliser un histogramme *hog* comme on avait déjà fait dans la question 1. Cela nous permet d'exploiter sa caractéristique d'être invariant aux changements géométriques et photo-métriques. Un défis de cette méthode qui nous avons remarqué est que, dans la vidéo 3, quand beaucoup des détailles changent leur position rapidemen il y a une fausse détection du changement de plan. La seuil σ que nous avons choisi est 0.9.

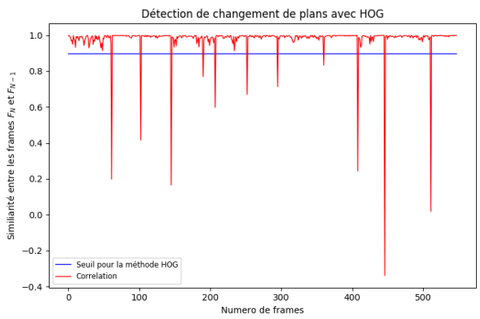


Figure 18: Corrélation des histogrammes lié à *Hog*

Conclusion

Notre algorithme permet donc de détecter les changements de plan dans les vidéos proposées et affichera sur le terminale le numéro du frame initial et final d'un certain plan. De plus, le vote majoritaire permet de avoir des performances satisfaisantes sur les vidéos disponibles.

Q5 - L'extraction d'une image-clef représentative d'un plan

Pour extraire une image représentative du plan, nous sommes partis des résultats obtenus dans la question 4 où nous avions le numéro du frame initial représentant un plan et le numéro de frame final.
Nous avons pensé deux différents méthodes pour l'extraction de l'image clef de la vidéo:

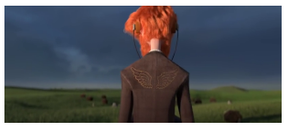
- la première qui se base sur une *comparaison* entre la *moyenne des image du plan* courant et les *images* mêmes. Nous allons calculer la corrélation entre les images et l'image moyenne qui nous permet de trouver l'image qui ressemble le plus à la moyenne et donc pouvoir choisir l'image clef représentative du

plan. Cela présente une limitation quand la scène est courte et change rapidement car nous n'aurons pas une image moyenne caractéristique et la choix de l'image clef sera compliqué.

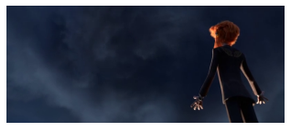
- la deuxième qui se base sur le calcul, pour chaque image, de l'histogramme de *hog*. Nous avons choisi cet histogramme pour le fait que dans la question 3 nous donnait les résultats le plus robustes et cohérents pour la détection du changement de plan avec la majorité des vidéos. Notre méthode va calculer enfin une moyenne des histogrammes *hog* qui permet de rechercher l'image qui a l'histogramme plus similaire au moyen. Avec cela nous avons extraite l'image clef de l'histogramme,

Nous nous sommes concentrés pour l'implementation de la deuxième méthode car nous avons estimé qu'il puisse avoir un résultat plus fiable par rapport au première.

Les résultats qui nous avons obtenus sont satisfaisantes car nous arrivons à extraire l'image représentative de la scène dans le cas où la scène n'est pas trop dynamique. Pour la vidéo 1, nous arrivons bien à extraire les images représentatives des plans. Par exemple nous avons extraite:



(a) Image Clef du Plan 1

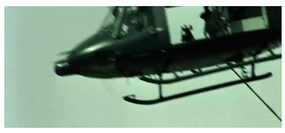


(b) Image Clef du Plan 2

(c) Image Clef du Plan 3

Figure 19: Images clef de la vidéo 1

Pour ce que concerne la vidéo 5, par exemple dans la scène initiale où le changement de plan est très dynamique, il est difficile d'associer une image clef car il n'y a pas une image qui représente la majorité de la scène. Dans la vidéo 5, nous allons obtenir les images suivantes:



(a) Image Clef du Plan 1

(b) Image Clef du Plan 2

(c) Image Clef du Plan 3

Figure 20: Images clef de la vidéo 5

Après la première scène qui change rapidement, aussi dans la vidéo 5 les résultats sont satisfaisants étant donnée que les images clef choisi représentent bien les scènes de la vidéo.

Notre algorithme s'occupe d'afficher au même temps que la vidéo évolue mais aussi de sauvegarder les images clef obtenues.

Q6 - Technique d'identification de type de plan

Pour identifier les différents type de plan, comme déjà anticipé dans la question 3, nous avons calculé l'histogramme du *Flot Optique* moyen des images qui représentent une scène. Nous aurons donc un histogramme pour chaque scene de la vidéo.

Nous avons en suite divisé ce plan en quatre parties sur lesquelles, pour chacune, nous effectuerons une somme afin de visualiser où se situent majoritairement les pixels.
En particulier, nous aurons une première somme pour la zone en haut à gauche, une pour la zone en haute

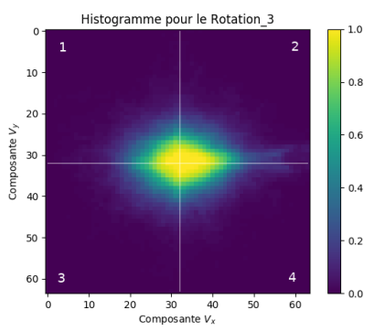


Figure 21: Histogramme découpé en quatre parties

à droite, une pour le zone en bas gauche et une dernière pour la zone en bas à droite.

Au travers d'une série de *if* et *else*, reporté dans 3 nous sommes capables de détecter les différents types de plan.
Nous sommes arrivé à avoir une distinction entre trois grand types de plan: les plans liés à un mouvement de rotation, lesquels liés à un mouvement de translation(*pan* et travelling horizontal où *tilt* et travelling vertical) et, enfin, lesquels liés à un mouvement de zoom où de travelling vers l'avant.

```
if (sum_H[1] + sum_H[3]) > seuil_H1_H3*(sum_H[0] + sum_H[2]):
   if (sum_H[1] > seuil_inner*sum_H[3]):
      print("   > Type: Travelling/ tilt vers le coin infrieur gauche (01)")
   elif (sum_H[3] > seuil_inner*sum_H[1]):
      print("   > Type: Travelling/ tilt vers le coin suprieur gauche (02)")
   else:
      print("   > Type: Travelling/ tilt vers la gauche (03)")
elif (sum_H[0] + sum_H[1]) > seuil_H0_H1*(sum_H[2] + sum_H[3]):
   if (sum_H[1] > seuil_inner*sum_H[0]):
      print("   > Type: Travelling/ tilt vers le coin infrieur gauche (04)")
   elif (sum_H[0] > seuil_inner*sum_H[1]):
      print("   > Type: Travelling/ tilt vers le coin infrieur droite (05)")
   else:
      print("   > Type: Travelling/ tilt vers le bas (06)")
elif (sum_H[0] + sum_H[2]) > seuil_H0_H2*(sum_H[1] + sum_H[3]):
   if (sum_H[0] > seuil_inner*sum_H[2]):
      print("   > Type: Travelling/ tilt vers le coin infrieur droite (07)")
   elif (sum_H[2] > seuil_inner*sum_H[0]):
      print("   > Type: Travelling/ tilt vers le coin suprieur droite (08)")
   else:
      print("   > Type: Travelling/ tilt vers la droite (09)")
elif (sum_H[2] + sum_H[3]) > seuil_H2_H3*(sum_H[0] + sum_H[1]):
   if (sum_H[2] > seuil_inner*sum_H[3]):
      print("   > Type: Travelling/ tilt vers le coin suprieur droite (10)")
   elif (sum_H[3] > seuil_inner*sum_H[2]):
      print("   > Type: Travelling/ tilt vers le coin suprieur gauche (11)")
   else:
      print("   > Type: Travelling/ tilt vers le haut (12)")
elif (sum_H[0]+sum_H[0]+sum_H[0]+sum_H[0]>seuil_zoom):
   if (compare_angMag <= seuil_rot_var):
      print("   > Type: Rotation (13)")
   else:
      print("   > Type: Zoom (14)")
else:
   print("   > Type: Plan fixe (15)")
```

Pour ce code, nous avons définie les différents seuils selon les tests effectués.

En particulier, si la caméra bougera vers la droite, nous allons avoir une histogramme avec un orientation vers la gauche et cela nous fera avoir que la somme des parties de gauche (2 et 4) sera plus grande de la somme des parties de droite(1 et 3). Si il y aura aussi un mouvement en diagonal notre algorithme permettra de distinguer entre un mouvement vers les quatre coins du champ de vue.

Nous avons des bons résultats si la vidéo se limite à avoir des mouvements horizontaux où verticaux: on arrive donc a faire une différence entre des mouvements entre le group contenant les mouvements de *panning*, *tilt*, *travelling horizontal* et le group composé par les mouvements de *rotation* et de *zoom*.

En suite nous nous sommes concentrés dans la distinction entre le *zoom* et la *rotation* qui présentent un histogramme très similaire. Leur histogramme est caractérisé par un cercle d'un rayon diffèrent selon la vitesse du zoom et de rotation.
Au début nous avons trouvé que, pour un mouvement de rotation le rayon su cercle dans l'histogramme était plus grande par rapport auquel d'un zoom et nous avons implémenté une premiere versione de l'algotithme qui utilisait cette comparaison.

En suite, en testant la robustesse de notre algorithme sur plusieurs vidéos, nous avons noté que on pouvait mieux exploiter les informations données par la magnitude et l'angle des vecteurs vitesses.
Dans nos répertoire *GitHub*, il y a un fichier excel *"Analyse"* qui reporte dans les feuil nommé *Rotation_zoom_seuil* nos études pour les valeurs de magnitude et angle dans le cas d'un rotation où d'un zoom.

Pour la détection de *zoom* et de *rotation*, nous avons combiné des informations d'orientation et magnitude obtenus à travers l'information donnée par la fonction de Farneback pour le calcul du *Flot Optique*.

Nous allons décrire ci-dessous la demarche suivie:

1. Nous avons groupé en bins les vecteurs qui ont un angle limité entre un certain intervalle défini. Dans notre cas, on a choisi de utiliser 24 bins représentant les intervales suivantes: [0°-15°, $15° - 30°$, ... $345° - 360°$].

2. Une fois défini les valeur pour les bins, on a construit deux histogrammes: un pour la quantité de vecteurs étant dedans l'intervalle concernés pour chaque bin et un autre pour la somme des magnitudes des vecteurs concernés pour l'intervalle de chaque bin. Après la construction, chaque histogramme a été normalisé, comme montrent les figures 22 et 23.

3. Pour vérifier la nature du mouvement (zoom ou rotation), les 5 bins dominantes du histogramme de orientation ont étés prises. Nous avons calculé $compare\_angMag = \sum_{i=1}^{i=5} \frac{abs(H_m - H_a)}{5}$, avec $i = n, m$

ou H_m est le vecteur de bins que appartiennent à les 5 bins plus significatifs du histogramme de magnitude, et H_a est le vecteur de bins que appartiennent à les 5 bins plus significatifs du histogramme d'orientation.

4. Une fois calculé *compare_angMag*, nous avons utilisé cette valeur comme paramètre pour distinguer entre les mouvements de *zoom* et *rotation*, vu que selon les tests réalisés avec des extraits de rotation et de zoom nous avons constaté que la valeur de *compare_angMag* est toujours inférieure à une certaine seuil(0.11) pour la rotation et supérieure à une certaine seuil pour le zoom.

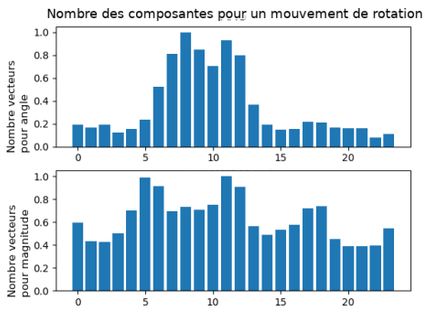


Figure 22: Histogramme de magnitude/ orientations du flot optique pour la rotation

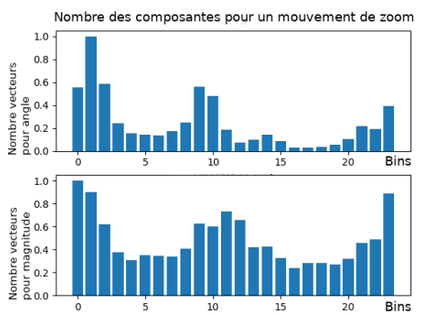


Figure 23: Histogramme de magnitude/ orientations du flot optique pour le zoom

Nous avons noté que pour le mouvement de *rotation*, cette valeur est plus grande par rapport à laquelle liée au mouvement de *zoom* et cela s'explique avec le fait que dans une *rotation* les pixels bougent par rapport à un point fixe. et donc nous aurons certaines pixels qui bougent plus car loin du centre de rotation et certaines qui bougent moins car plus proches du centre de rotation mais dans un *zoom* les pixels bougent avec la même direction et avec la même magnitude raison pour laquelle la différence entre les deux quantités donne une valeur petite.

Pistes d'amélioration

Notre algorithme a encore plusieurs pistes d'améliorations pour le fait qu'il ne permet pas de distinguer entre un mouvement de *travelling ver l'avant* et un *zoom* où encore entre un mouvement de *tilt* où *pan* et un mouvement de *travelling vertical* où *horizontal*.

Cela il nous est apparu compliqué car la forme des histogramme est très similaire entre les deux mouvements.

Nous avons pensé de pouvoir exploiter encore une fois la magnitude et la direction des vecteurs vitesse mais nous ne sommes pas arrivés à avoir un résultat satisfaisante dans ce cas. De plus, nous avons essayé de calculer la variance des histogrammes pour voir si il était une caractéristique pour pouvoir différencier les deux mouvements mais les résultats changeait de temps en temps selon la différente vidéo qu'on considérait: nous avons remarqué que les résultats changeaient selon la rapidité avec laquelle la caméra était bougé et non seulement avec le type de mouvement fait.