L3 - *« Computer Vision »* *L. Mascarilla,*	**TP n°3 - Rétroprojection de l'histogramme CamShift**	UNIVERSITÉ La Rochelle

L'objectif de ce TP est d'expérimenter la **rétroprojection de l'histogramme** vu en cours pour détecter des objets dans des videos sur la caméra JeVois.

- **Dans un premier temps,** nous allons mettre au point les programmes sur l**a machine locale**. La caméra **jevois** ne servira alors que de webcam classique.

- **Dans un second temps**, nous porterons ces programmes sur la jevois qui réalisera les calculs de façon autonome. Dans ce cas, la JeVois sera totalement autonome.

Vous utiliserez l'IDE de votre choix (Spyder, Pycharm, VSCode) et le même interpréteur Python que durant le TP précédent.

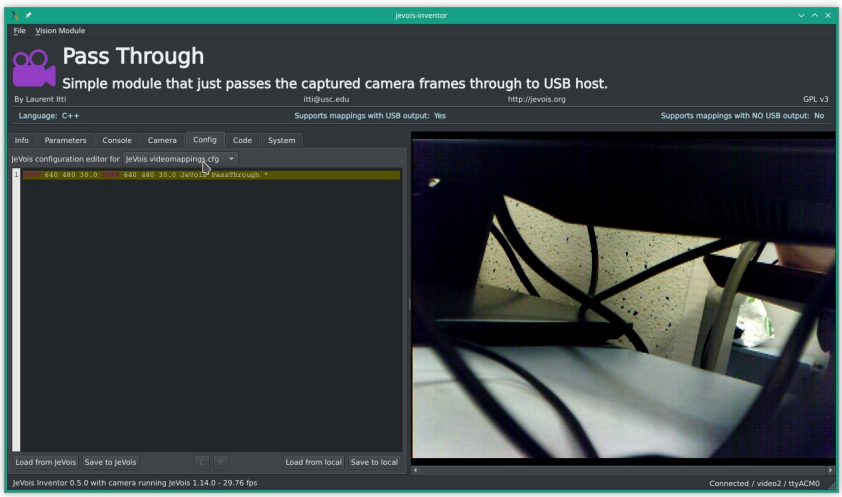
Partie 1 : Calculs sur la machine hôte (Windows,Mac, Linux)

1) **Initialisation :**

Deux possibilités s'offrent à vous ici : soit vous utilisez une webcam (celle de votre portable par exemple) soit vous effectuez les manipulations qui suivent pour utiliser la JeVois comme webcam.

Pour utiliser la **jevois comme webcam**, vous allez procéder de la façon suivante :

- lancez **jevois-inventor, puis selectionnez l'onglet videomappings.cfg**



Avant toute modification, vous effectuerez une sauvegarde de ce fichier en copiant son contenu dans un fichier **videomappings.cfg** sur votre ordinateur personnel.

Quand cela est fait, remplacez, **sur la jevois,** le contenu de ce fichier par l'unique ligne :

YUYV 640 480 30.0 YUYV 640 480 30.0 JeVois PassThrough *

Sauvegardez par **Ctrl+S** puis redémarrez la caméra. Elle doit se comporter comme une simple caméra USB.

Arrêtez **jevois-inventor** avant de passer à la suite.

2) Utilisation sur OpenCV (côté ordinateur)

Pour tester le bon fonctionnement de votre configuration, exécutez le code python **skel_opencv.py.** Normalement, le flux de la JeVois (ou de la webcam) doit s'afficher. Toutefois, le numéro de caméra (variable **cam** dans le code python) peut être différent selon votre configuration : la seule solution est d'essayer successivement avec les valeurs 0,1,2,3...

Pour quitter **appuyez sur la touche Echap.**

Normalement, vous devez avoir le numéro de caméra (cam). Notez-le.

3) Exécutez le code python **skel_camshift.py.** Si vous avez **reporté le numéro de caméra** dans le code, le flux de la jevois doit s'afficher.

En cliquant sur l'image vous pourrez sélectionner 4 points pour définir la région d'intérêt initiale : une fois que ces points sont sélectionnés, le programme affiche les coordonnées de ces 4 points et s'arrête sur une erreur. C'est normal : il vous faut compléter le code ! **Notez ces coordonnées pour la suite du TP (Partie 2).**

Complétez le code pour calculer l'**histogramme de teinte** sur cette roi sur l'image convertie en HSV.

Fonctions à utiliser :

cv2.cvtColor : pour convertir l'image courante (frame) en HSV

cv2.calcHist([roi], [0], None, [16], [0, 180]) : calculer l'histogramme dans la roi en HSV

cv2.normalize : pour ramener les valeurs dans [0, 255]

Vous regarderez et interpréterez les paramètres des fonctions sur la page de documentation d'opencv :

https://docs.opencv.org/4.7.0/

4) **Rétroprojection :** Complétez le code pour **afficher l'image de rétroprojection**.

Fonctions à utiliser :

cv2.cvtColor : toujours pour convertir l'image courante en HSV

cv2.calcBackProject([hsv], [0], roiHist, [0, 180], 1) : calcule l'image de retroprojection en utilisant histogramme obtenu précédemment sur la roi et l'image courante en HSV

5) **Amélioration :** Vous pourrez essayer d'améliorer le résultat avec une ouverture morphologique que nous verrons en cours.

Fonctions à utiliser :

disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))

backProj = cv2.morphologyEx(backProj, cv2.MORPH_OPEN, disc)

backProj = cv2.morphologyEx(backProj, cv2.MORPH_CLOSE, disc)

Vous pouvez changer la taille de l'élément structurant (dans getStructuringElement) pour améliorer la qualité du résultat. Toutefois, plus cette taille est importante plus le code sera lent !

6) Améliorez le résultat du programme précédent en remplaçant le calcul de l'histogramme de la teinte par celui de **la**

teinte et de la saturation :

roiHist = cv2.calcHist([roi], [0,1], None, [16, 16], [0, 180, 0, 256])

puis pour la rétroprojection :

backProj = cv2.calcBackProject([hsv], [0,1], roiHist, [0,180,0,256],1)

7) **Suivi :** Mettez en place un tracker (sur l'image de rétroprojection) par **Meanshift**

ret, roiBox = cv2.meanShift(backProj, roiBox, term_criteria)

Normalement, vous devriez suivre le visage dans la video avec une boite englobante **(ROI=Region Of Interest)** dont les côtés restent parallèles aux bords de l'image : les coordonnées de cette boite sont contenues dans roiBox et l'affichage se fait par :

x, y, w, h = roiBox **# observez le format**

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

8) La variante de **Camshift** présente dans opencv permet de changer l'orientation de la boite englobante pour mieux suivre l'objet d'intérêt.

Remplacez le **meanshift** par la fonction **camshift** d' openCV (voir la documentation)

ret, roiBox = cv2.CamShift(dst, roiBox, term_criteria)

pts = cv2.boxPoints(ret)
pts = np.int0(pts)
cv2.polylines(frame, [pts], True, (0, 255, 0), 2)

En utilisant les équations décrites dans la partie 3, vous pourrez tracer les axes calculés par votre algorithme et les comparer à la roi calculée par camshift. Pour l'instant, vous allez tester votre code sur la JeVois.

Avant de poursuivre la séance, restaurez la configuration de la JeVois en remettant le fichier videomappings.cfg de départ si vous l'avez modifié.

Partie 2: Portage sur la JeVois

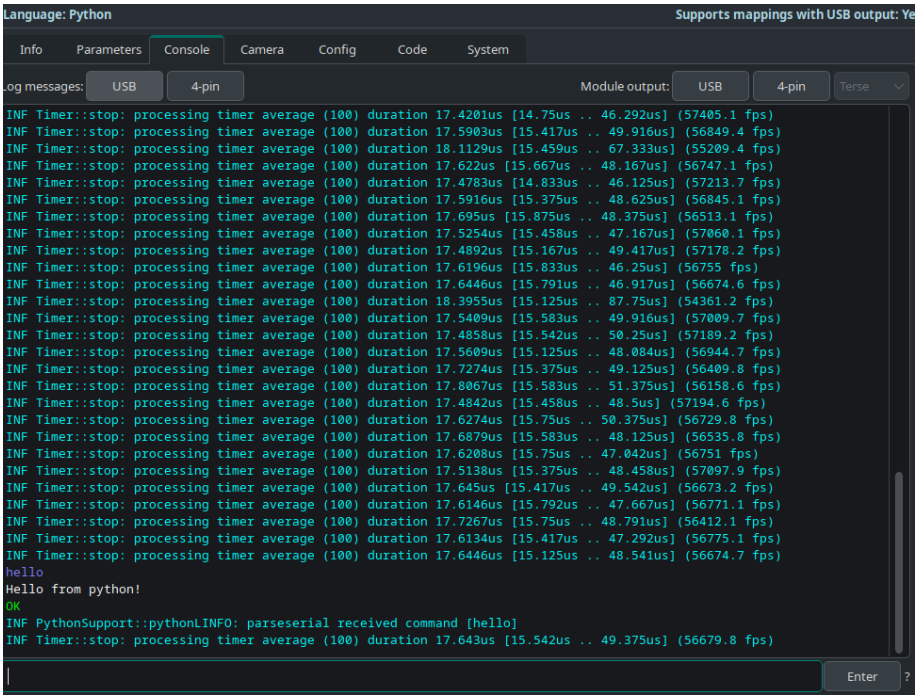
En suivant la même approche que dans le TP précédent, portez votre code sur la Jevois.

Même si vous visualisez le résultat de votre programme sur JeVois Inventor, il n'y a pas d'interactions possibles avec la souris : pas de clic pour sélectionner une région ou lancer une fonction. Pour sélectionner le visage à suivre, vous devez donc trouver des alternatives :

- Fixer une région d'intérêt fixe sur l'image et déclencher le suivi au bout d'un temps fixé. N'oubliez pas que la méthode process est appelée à chaque frame, il est donc facile de connaître le temps d'exécution de votre code depuis

son lancement en comptant les frames et en multipliant par le frame rate.

- Une autre possibilité est de communiquer avec la JeVois par la liaison série. Pour ce TP nous utiliserons la liaison par USB. Il suffit de déclarer la méthode **parseSerial** qui reçoit une chaine de caractères entrée dans le prompt de la console :



Dans l'exemple ci-dessus, le code est simplement :

```
    def parseSerial(self, str):
        jevois.LINFO("parseserial received command [{}]".format(str))
        if str == "hello":
            return self.hello()
        return "ERR: Unsupported command"

    def hello(self):
        return "Hello from python!"
```

Vous pouvez donc définir une commande (par exemple « start ») qui lance le suivi lorsque vous l'entrez sur la console, mais vous pouvez être plus imaginatifs : parSerial reçoit une chaine de caractères que vous pouvez « parser » pour extraire les instructions que vous voulez (« reset », « roi x1 y1 x2 y2 » etc.)

Testez.

Partie 3 : Tester la sélection par la « teinte chair »

Comme nous l'avons vu en cours, une méthode simple pour sélectionner les zones contenant le visage est d'utiliser des seuils sur les images en HSV et en YcrCb.

1) A partir de ces deux images, le seuillage appliqué (voir le cours pour les seuils) conduit à un masque binaire. Pour appliquer ces seuils, vous pouvez utiliser les fonctions de numpy ou de python, mais OpenCV propose une fonction très efficace qui renvoie un masque à partir d'une image couleur (RGB,HSV, YcrCb…) et des seuils hauts et bas sur chaque canal :

cv2.inRange : voir https://docs.opencv.org/4.7.0/da/d97/tutorial_threshold_inRange.html

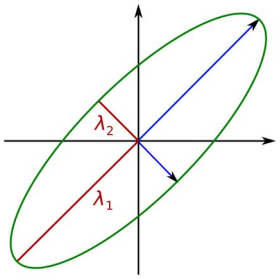
Le masque peut ensuite être appliqué sur l'image par la ligne suivante :

img = cv2.bitwise_and(img,img,mask=mask)

2) Améliorez votre code camshift en utilisant ce masque.

Partie 4: Bonus - Comprendre le camshift

Le principe est de calculer la matrice de covariance de l'image rétro-projetée (ou de sa version binarisée) et de calculer ses vecteurs propres et valeurs propres qui donnent l'orientation et l'allongement du nuage de points.



Le centre (x_c,y_c) de l'image est donné par : $x_c=M_{10}/M_{00}$ et $yc=M_{01}/M_{00}$ où

$$M_{00}=\sum_x \sum_y I(x,y) \qquad M_{10}=\sum_x \sum_y xI(x,y)$$
$$M_{01}=\sum_x \sum_y yI(x,y)$$

Pour trouver l'angle $\boldsymbol{\theta}$ par rapport à l'horizontale, la longueur **l** et la largeur **w** de l'ellipse on effectue les calculs suivants :

$a = M_{20} / M_{00} - x_c^2$

$b = 2 * (M_{11} / M_{00} - x_c .y_c$

$c = M_{02}/ M_{00}- y_c .y_c$

$d = b^2 + (a - c)^2$

$$l=\sqrt{\frac{(a+b)+\sqrt{d}}{2}}$$

$$w=\sqrt{\frac{(a+b)-\sqrt{d}}{2}}$$

$$\theta=\frac{1}{2}\arctan\left(\frac{2\left(\frac{M_{11}}{M_{00}}-x_c y_c\right)}{\left(\frac{M_{20}}{M_{00}}-x_c^2\right)-\left(\frac{M_{02}}{M_{00}}-y_c^2\right)}\right)$$ si le dénominateur est nul, $\theta = \pi/2$

où :

$$M_{11}=\sum_x \sum_y x\,yI(x,y) \qquad M_{20}=\sum_x \sum_y x^2 I(x,y) \qquad M_{02}=\sum_x \sum_y y^2 I(x,y)$$

Pour tester ce principe vous allez créer un programme orientation.py (pour l'instant indépendant du précédent) qui va :

- créer une image noire

- y rajouter une ellipse (pleine) en blanc avec cv2.ellipse

- puis calculer l'angle et les dimensions de la boite englobante avec les formules précédentes.

Les moments $M_{00,}$ M_{01} etc sont calculés par la fonction cv2.moments.

Des exemples d'utilisation sont disponibles ici :

https://www.learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/

Le tracé des axes pourra être fait avec le code suivant :

```
c = math.cos(theta)
s = math.sin(theta)
cv2.line( frame, (int(xc - c * w), int(yc - s * w)), (int(xc + c * w), int(yc + s * w)), (255, 255, 0),2,)
cv2.line( frame, (int(xc + s * l), int(yc - c * l)),  (int(xc - s * l), int(yc + c * l)), (255, 255, 0), 2,)
```

Vous devriez vérifier que les axes s'alignent sur les axes de l'ellipse.

Modifiez votre code utilisant le camshift sur la jevois pour tracer ellipse et axes.

Bonus : Vous pouvez aligner l'image sur les axes de l'ellipse (qui devient donc verticale). Pour cela, il vous faut :

- translater le centre $(x_c y_c)$ au centre de l'image.

- effectuer une rotation de θ

Vous trouverez les outils nécessaires sur cette page :

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html

Cette technique peut-être utilisée pour stabiliser une image. Par exemple, un objet en rotation peut donner l'impression de rester fixe. Il faut toutefois « bien » gérer la rotation, ce qui peut demander un peu de réflexion…