Labs 10

Smain Belghazi ING3

Dans un premier temps je vais installer OpenCV et importer le package cv2

In [23]:
```
%pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\smain\appdata\local\programs\pyt
hon\python312\lib\site-packages (2.2.2)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.2 -> 25.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [24]:
```
%pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\smain\appdata\local\prog
rams\python\python312\lib\site-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in c:\users\smain\appdata\local\prog
rams\python\python312\lib\site-packages (from opencv-python) (2.2.2)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.2 -> 25.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [10]:
```
import cv2

# Compute the frame differences
def frame_diff(prev_frame, cur_frame, next_frame):
    # Difference between the current frame and the next frame
    diff_frames_1 = cv2.absdiff(next_frame, cur_frame)

    # Difference between the current frame and the previous frame
    diff_frames_2 = cv2.absdiff(cur_frame, prev_frame)

    return cv2.bitwise_and(diff_frames_1, diff_frames_2)

# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    return gray

if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the scaling factor for the images
```

```
    scaling_factor = 0.5

    # Grab the current frame
    prev_frame = get_frame(cap, scaling_factor)

    # Grab the next frame
    cur_frame = get_frame(cap, scaling_factor)

    # Grab the frame after that
    next_frame = get_frame(cap, scaling_factor)

    # Keep reading the frames from the webcam
    # until the user hits the 'Esc' key
    while True:
        # Display the frame difference
        cv2.imshow('Object Movement', frame_diff(prev_frame,
                cur_frame, next_frame))

        # Update the variables
        prev_frame = cur_frame
        cur_frame = next_frame

        # Grab the next frame
        next_frame = get_frame(cap, scaling_factor)

        # Check if the user hit the 'Esc' key
        key = cv2.waitKey(10)
        if key == 27:
            break

    # Close all the windows
    cv2.destroyAllWindows()
```

Le bloc de code ouvre une fenêtre qui capture les mouvements en utilisant la différence entre plusieurs frames successives. Il calcule le AND bitwise entre deux frames pour comparer pixel par pixel, et si une différence est détectée, elle est mise à jour et affichée. Cela permet de montrer les silhouettes des objets en mouvement dans la fenêtre ! [Capture d'écran](détécteur de mouv.png)

In [20]:
```
import cv2
import numpy as np

# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    return frame

if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the scaling factor for the images
```

```
    scaling_factor = 0.5

    # Keep reading the frames from the webcam
    # until the user hits the 'Esc' key
    while True:
        # Grab the current frame
        frame = get_frame(cap, scaling_factor)

        # Convert the image to HSV colorspace
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Define range of skin color in HSV
        # Définir une nouvelle plage de couleurs pour mieux correspondre à ta pe
        lower = np.array([0, 40, 50])   # Plage basse (peut-être ajustée)
        upper = np.array([20, 150, 255]) # Plage haute (ajuster selon le besoin)


        # Threshold the HSV image to get only skin color
        mask = cv2.inRange(hsv, lower, upper)

        # Bitwise-AND between the mask and original image
        img_bitwise_and = cv2.bitwise_and(frame, frame, mask=mask)

        # Run median blurring
        img_median_blurred = cv2.medianBlur(img_bitwise_and, 5)

        # Display the input and output
        cv2.imshow('Input', frame)
        cv2.imshow('Output', img_median_blurred)

        # Check if the user hit the 'Esc' key
        c = cv2.waitKey(5)
        if c == 27:
            break

    # Close all the windows
    cv2.destroyAllWindows()
```

Dans le bloc de code précédent, le programme capture les images de la webcam et détecte une couleur spécifique (comme la peau) à partir des pixels de l'image. D'abord, une plage de couleurs est définie en utilisant l'espace de couleurs HSV. Ensuite, un masque est créé pour isoler cette couleur spécifique. Sur la deuxième frame, le programme applique un AND bitwise entre l'image d'origine et le masque pour ne conserver que la couleur de la peau détectée. Cela génère deux fenêtres : une montrant l'image originale et l'autre affichant uniquement la couleur choisie (par exemple, la peau).

In [19]:
```
import cv2
import numpy as np

# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)
```

```
    return frame

if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the background subtractor object
    bg_subtractor = cv2.createBackgroundSubtractorMOG2()

    # Define the number of previous frames to use to learn.
    # This factor controls the learning rate of the algorithm.
    # The learning rate refers to the rate at which your model
    # will learn about the background. Higher value for
    # 'history' indicates a slower learning rate. You can
    # play with this parameter to see how it affects the output.
    history = 1000

    # Define the learning rate
    learning_rate = 1.0/history

    # Keep reading the frames from the webcam
    # until the user hits the 'Esc' key
    while True:
        # Grab the current frame
        frame = get_frame(cap, 0.5)

        # Compute the mask
        mask = bg_subtractor.apply(frame, learningRate=learning_rate)

        # Convert grayscale image to RGB color image
        mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

        # Display the images
        cv2.imshow('Input', frame)
        cv2.imshow('Output', mask & frame)

        # Check if the user hit the 'Esc' key
        c = cv2.waitKey(10)
        if c == 27:
            break

    # Release the video capture object
    cap.release()

    # Close all the windows
    cv2.destroyAllWindows()
```

Ce code utilise la soustraction de fond (Background Subtraction) pour détecter les objets en mouvement dans une vidéo en direct depuis la webcam. Il capture chaque image, applique un soustracteur de fond MOG2 pour créer un masque qui isole les objets en mouvement, puis affiche à la fois l'image originale et l'image filtrée. L'apprentissage du fond est contrôlé par un paramètre history et un learning_rate ajustable

In [41]:
```
import cv2
import numpy as np

# Define a class to handle object tracking related functionality
```

```
class ObjectTracker(object):
    def __init__(self, scaling_factor=0.5):
        # Initialize the video capture object
        self.cap = cv2.VideoCapture(0)

        # Capture the frame from the webcam
        _, self.frame = self.cap.read()

        # Scaling factor for the captured frame
        self.scaling_factor = scaling_factor

        # Resize the frame
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)

        # Create a window to display the frame
        cv2.namedWindow('Object Tracker')

        # Set the mouse callback function to track the mouse
        cv2.setMouseCallback('Object Tracker', self.mouse_event)

        # Initialize variable related to rectangular region selection
        self.selection = None

        # Initialize variable related to starting position
        self.drag_start = None

        # Initialize variable related to the state of tracking
        self.tracking_state = 0

    # Define a method to track the mouse events
    def mouse_event(self, event, x, y, flags, param):
        # Convert x and y coordinates into 16-bit numpy integers
        x, y = np.int16([x, y])

        # Check if a mouse button down event has occurred
        if event == cv2.EVENT_LBUTTONDOWN:
            self.drag_start = (x, y)
            self.tracking_state = 0

        # Check if the user has started selecting the region
        if self.drag_start:
            if flags & cv2.EVENT_FLAG_LBUTTON:
                # Extract the dimensions of the frame
                h, w = self.frame.shape[:2]

                # Get the initial position
                xi, yi = self.drag_start

                # Get the max and min values
                x0, y0 = np.maximum(0, np.minimum([xi, yi], [x, y]))
                x1, y1 = np.minimum([w, h], np.maximum([xi, yi], [x, y]))

                # Reset the selection variable
                self.selection = None

                # Finalize the rectangular selection
                if x1-x0 > 0 and y1-y0 > 0:
                    self.selection = (x0, y0, x1, y1)
```

```
            else:
                # If the selection is done, start tracking
                self.drag_start = None
                if self.selection is not None:
                    self.tracking_state = 1

    # Method to start tracking the object
    def start_tracking(self):
        # Iterate until the user presses the Esc key
        while True:
            # Capture the frame from webcam
            _, self.frame = self.cap.read()

            # Resize the input frame
            self.frame = cv2.resize(self.frame, None,
                    fx=self.scaling_factor, fy=self.scaling_factor,
                    interpolation=cv2.INTER_AREA)

            # Create a copy of the frame
            vis = self.frame.copy()

            # Convert the frame to HSV colorspace
            hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)

            # Create the mask based on predefined thresholds
            mask = cv2.inRange(hsv, np.array((0., 60., 32.)),
                        np.array((180., 255., 255.)))

            # Check if the user has selected the region
            if self.selection:
                # Extract the coordinates of the selected rectangle
                x0, y0, x1, y1 = self.selection

                # Extract the tracking window
                self.track_window = (x0, y0, x1-x0, y1-y0)

                # Extract the regions of interest
                hsv_roi = hsv[y0:y1, x0:x1]
                mask_roi = mask[y0:y1, x0:x1]

                # Compute the histogram of the region of
                # interest in the HSV image using the mask
                hist = cv2.calcHist( [hsv_roi], [0], mask_roi,
                        [16], [0, 180] )

                # Normalize and reshape the histogram
                cv2.normalize(hist, hist, 0, 255, cv2.NORM_MINMAX);
                self.hist = hist.reshape(-1)

                # Extract the region of interest from the frame
                vis_roi = vis[y0:y1, x0:x1]

                # Compute the image negative (for display only)
                cv2.bitwise_not(vis_roi, vis_roi)
                vis[mask == 0] = 0

            # Check if the system in the "tracking" mode
            if self.tracking_state == 1:
                # Reset the selection variable
```

```
                self.selection = None

                # Compute the histogram back projection
                hsv_backproj = cv2.calcBackProject([hsv], [0],
                        self.hist, [0, 180], 1)

                # Compute bitwise AND between histogram
                # backprojection and the mask
                hsv_backproj &= mask

                # Define termination criteria for the tracker
                term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,
                        10, 1)

                # Apply CAMShift on 'hsv_backproj'
                track_box, self.track_window = cv2.CamShift(hsv_backproj,
                        self.track_window, term_crit)

                # Draw an ellipse around the object
                cv2.ellipse(vis, track_box, (0, 255, 0), 2)

            # Show the output live video
            cv2.imshow('Object Tracker', vis)

            # Stop if the user hits the 'Esc' key
            c = cv2.waitKey(5)
            if c == 27:
                break

        # Close all the windows
        cv2.destroyAllWindows()

if __name__ == '__main__':
        # Start the tracker
    ObjectTracker().start_tracking()
```

Ce code implémente un suivi d'objet en temps réel à l'aide d'OpenCV et de l'algorithme CamShift. Il capture des images depuis la webcam, permet à l'utilisateur de sélectionner une région d'intérêt avec la souris, puis suit cet objet à travers les images suivantes.

In [40]:
```
import cv2
import numpy as np

# Define a function to track the object
def start_tracking():
    # Initialize the video capture object
    cap = cv2.VideoCapture(0)

    # Define the scaling factor for the frames
    scaling_factor = 0.5

    # Number of frames to track
    num_frames_to_track = 5

    # Skipping factor
    num_frames_jump = 2

    # Initialize variables
    tracking_paths = []
```

```
    frame_index = 0

    # Define tracking parameters
    tracking_params = dict(winSize  = (11, 11), maxLevel = 2,
            criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,
                10, 0.03))

    # Iterate until the user hits the 'Esc' key
    while True:
        # Capture the current frame
        _, frame = cap.read()

        # Resize the frame
        frame = cv2.resize(frame, None, fx=scaling_factor,
                fy=scaling_factor, interpolation=cv2.INTER_AREA)

        # Convert to grayscale
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Create a copy of the frame
        output_img = frame.copy()

        if len(tracking_paths) > 0:
            # Get images
            prev_img, current_img = prev_gray, frame_gray

            # Organize the feature points
            feature_points_0 = np.float32([tp[-1] for tp in \
                    tracking_paths]).reshape(-1, 1, 2)

            # Compute optical flow
            feature_points_1, _, _ = cv2.calcOpticalFlowPyrLK(
                    prev_img, current_img, feature_points_0,
                    None, **tracking_params)

            # Compute reverse optical flow
            feature_points_0_rev, _, _ = cv2.calcOpticalFlowPyrLK(
                    current_img, prev_img, feature_points_1,
                    None, **tracking_params)

            # Compute the difference between forward and
            # reverse optical flow
            diff_feature_points = abs(feature_points_0 - \
                    feature_points_0_rev).reshape(-1, 2).max(-1)

            # Extract the good points
            good_points = diff_feature_points < 1

            # Initialize variable
            new_tracking_paths = []

            # Iterate through all the good feature points
            for tp, (x, y), good_points_flag in zip(tracking_paths,
                        feature_points_1.reshape(-1, 2), good_points):
                # If the flag is not true, then continue
                if not good_points_flag:
                    continue

                # Append the X and Y coordinates and check if
                # its length greater than the threshold
```

```
                tp.append((x, y))
                if len(tp) > num_frames_to_track:
                    del tp[0]

                new_tracking_paths.append(tp)

                # Draw a circle around the feature points
                cv2.circle(output_img, tuple(np.int32((x, y))), 3, (0, 255, 0),


            # Update the tracking paths
            tracking_paths = new_tracking_paths

            # Draw lines
            cv2.polylines(output_img, [np.int32(tp) for tp in \
                    tracking_paths], False, (0, 150, 0))

        # Go into this 'if' condition after skipping the
        # right number of frames
        if not frame_index % num_frames_jump:
            # Create a mask and draw the circles
            mask = np.zeros_like(frame_gray)
            mask[:] = 255
            for x, y in [np.int32(tp[-1]) for tp in tracking_paths]:
                cv2.circle(mask, (x, y), 6, 0, -1)

            # Compute good features to track
            feature_points = cv2.goodFeaturesToTrack(frame_gray,
                    mask = mask, maxCorners = 500, qualityLevel = 0.3,
                    minDistance = 7, blockSize = 7)

            # Check if feature points exist. If so, append them
            # to the tracking paths
            if feature_points is not None:
                for x, y in np.float32(feature_points).reshape(-1, 2):
                    tracking_paths.append([(x, y)])

        # Update variables
        frame_index += 1
        prev_gray = frame_gray

        # Display output
        cv2.imshow('Optical Flow', output_img)

        # Check if the user hit the 'Esc' key
        c = cv2.waitKey(1)
        if c == 27:
            break

if __name__ == '__main__':
        # Start the tracker
    start_tracking()

    # Close all the windows
    cv2.destroyAllWindows()
```

Le code utilise OpenCV pour capturer une vidéo en temps réel via la webcam, Chaque image est redimensionnée et convertie en niveaux de gris. Il détecte des "good features to track" grâce à l'algorithme de Shi-Tomasi puis il applique l'algorithme d'optical flow de

Lucas-Kanade pour suivre les points détectés sur plusieurs images consécutives. Il calcule un "reverse optical flow" pour valider les points suivis (en comparant le déplacement aller-retour).

Affichage :

Les points suivis sont représentés par des cercles verts.

Des lignes vertes montrent la trajectoire des points suivis.

Pour resumé ce programme détecte des points caractéristiques dans une vidéo en direct, suit leurs mouvements grâce à Lucas-Kanade et visualise ces déplacements avec des lignes vertes.

In [20]:
```
!python -m pip install --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\smain\appdata\local\programs\python\python312\lib\site-packages (24.2)
Collecting pip
  Downloading pip-25.0-py3-none-any.whl.metadata (3.7 kB)
Downloading pip-25.0-py3-none-any.whl (1.8 MB)
   ---------------------------------------- 0.0/1.8 MB ? eta -:--:--
   ---------------------------------------- 1.8/1.8 MB 49.4 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.2
    Uninstalling pip-24.2:
      Successfully uninstalled pip-24.2
Successfully installed pip-25.0
```

```
  WARNING: The scripts pip.exe, pip3.12.exe and pip3.exe are installed in 'c:\Users\smain\AppData\Local\Programs\Python\Python312\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
```

In [36]:
```
!pip install --upgrade opencv-python
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: opencv-python in c:\users\smain\appdata\local\packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-packages\python313\site-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in c:\users\smain\appdata\local\packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-packages\python313\site-packages (from opencv-python) (2.2.2)
```

```
[notice] A new release of pip is available: 24.3.1 -> 25.0
[notice] To update, run: C:\Users\smain\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip
```

In [ ]:
```
c:\Users\smain\Downloads\haarcascade_frontalface_default.xml

"C:/Users/smain/Downloads/haarcascade_eye.xml"
```

In [51]:
```
import cv2
import numpy as np

# Load the Haar cascade file
face_cascade = cv2.CascadeClassifier('c:/Users/smain/Downloads/haarcascade_front
```

```
# Check if the cascade file has been loaded correctly
if face_cascade.empty():
        raise IOError('Unable to load the face cascade classifier xml file')

# Initialize the video capture object
cap = cv2.VideoCapture(0)

# Define the scaling factor
scaling_factor = 0.5

# Iterate until the user hits the 'Esc' key
while True:
    # Capture the current frame
    _, frame = cap.read()

    # Resize the frame
    frame = cv2.resize(frame, None,
            fx=scaling_factor, fy=scaling_factor,
            interpolation=cv2.INTER_AREA)

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Run the face detector on the grayscale image
    face_rects = face_cascade.detectMultiScale(gray, 1.3, 5)

    # Draw a rectangle around the face
    for (x,y,w,h) in face_rects:
        cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 3)

    # Display the output
    cv2.imshow('Face Detector', frame)

    # Check if the user hit the 'Esc' key
    c = cv2.waitKey(1)
    if c == 27:
        break

# Release the video capture object
cap.release()

# Close all the windows
cv2.destroyAllWindows()
```

le bloc de code précedent utilise OpenCV pour capturer une vidéo en direct via la webcam et détecter les visages en temps réel. Il charge un classificateur Haar pour la détection des visages, convertit chaque image capturée en niveaux de gris, puis applique l'algorithme de détection de visages. Lorsque des visages sont détectés, il dessine des rectangles verts autour d'eux.

Note: J'ai rencontré un problème pour lancer le code. Le fichier haarcascade_frontalface_default.xml ne se chargeait pas correctement. J'ai donc décidé de le télécharger sur GitHub, puis de mettre le lien du fichier sur mon ordinateur : c:/Users/smain/Downloads/haarcascade_frontalface_default.xml.

In [53]:

```
import cv2
import numpy as np

# Load the Haar cascade files for face and eye
face_cascade = cv2.CascadeClassifier('c:/Users/smain/Downloads/haarcascade_front
eye_cascade = cv2.CascadeClassifier("C:/Users/smain/Downloads/haarcascade_eye.xm

# Check if the face cascade file has been loaded correctly
if face_cascade.empty():
        raise IOError('Unable to load the face cascade classifier xml file')

# Check if the eye cascade file has been loaded correctly
if eye_cascade.empty():
        raise IOError('Unable to load the eye cascade classifier xml file')

# Initialize the video capture object
cap = cv2.VideoCapture(0)

# Define the scaling factor
ds_factor = 0.5

# Iterate until the user hits the 'Esc' key
while True:
    # Capture the current frame
    _, frame = cap.read()

    # Resize the frame
    frame = cv2.resize(frame, None, fx=ds_factor, fy=ds_factor, interpolation=cv

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Run the face detector on the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    # For each face that's detected, run the eye detector
    for (x,y,w,h) in faces:
        # Extract the grayscale face ROI
        roi_gray = gray[y:y+h, x:x+w]

        # Extract the color face ROI
        roi_color = frame[y:y+h, x:x+w]

        # Run the eye detector on the grayscale ROI
        eyes = eye_cascade.detectMultiScale(roi_gray)

        # Draw circles around the eyes
        for (x_eye,y_eye,w_eye,h_eye) in eyes:
            center = (int(x_eye + 0.5*w_eye), int(y_eye + 0.5*h_eye))
            radius = int(0.3 * (w_eye + h_eye))
            color = (0, 255, 0)
            thickness = 3
            cv2.circle(roi_color, center, radius, color, thickness)

    # Display the output
    cv2.imshow('Eye Detector', frame)

    # Check if the user hit the 'Esc' key
    c = cv2.waitKey(1)
```

```
    if c == 27:
        break

# Release the video capture object
cap.release()

# Close all the windows
cv2.destroyAllWindows()
```

Le bloc de code précédent utilise OpenCV pour capturer une vidéo en direct via la webcam et détecter les visages et les yeux en temps réel. Il charge les classificateurs Haar pour la détection des visages et des yeux, convertit chaque image capturée en niveaux de gris, puis applique l'algorithme de détection des visages. Une fois un visage détecté, il cherche également des yeux dans la région du visage. Lorsqu'il détecte des yeux, il dessine des cercles verts autour d'eux.

Note: J'ai rencontré un problème pour lancer le code. Le fichier haarcascade_frontalface_default.xml ne se chargeait pas correctement. J'ai donc décidé de le télécharger sur GitHub, puis de mettre le lien du fichier sur mon ordinateur : c:/Users/smain/Downloads/haarcascade_frontalface_default.xml. mais egalement pour le fichier haarcascade_eye.xml