EXP NO: 8	IMPLEMENT OPTICAL FLOW COMPUTATION ALGORITHM

AIM:

To implement and compare optical flow algorithms: Lucas-Kanade (sparse), Farneback (dense), and Horn-Schunck (global).

THEORY:

Optical flow assumes brightness constancy: a pixel's intensity remains constant between two nearby frames, and that motion is small and continuous. The optical flow field is a 2D vector field (u,v) that describes motion at each image location.

Common algorithms:

• Lucas-Kanade (sparse): Solves for flow in small neighborhoods for a set of feature points (goodFeaturesToTrack). Efficient and robust for tracking sparse keypoints.

• Farneback (dense): Estimates flow for every pixel using polynomial expansion. Produces dense flow maps.

• Horn–Schunck (global): Formulates optical flow as a global energy minimization problem combining brightness constancy and smoothness constraints (less used in real-time but useful conceptually)

REQUIREMENTS:

Software: Python 3.x
Libraries: OpenCV, NumPy, Matplotlib

Install command:
pip install opencv-python numpy matplotlib

GENERAL ALGORITHM:

Step 1: Start

Step 2: Read video or sequence of frames (cv2.VideoCapture or a folder of images).

Step 3: Convert frames to grayscale using cv2.cvtColor().

Step 4: Choose optical flow method (Lucas-Kanade, Farneback, Horn-Schunck).

Step 5: For Lucas-Kanade: detect feature points in the first frame using cv2.goodFeaturesToTrack(), then track with cv2.calcOpticalFlowPyrLK().

Step 6: For Farneback: compute dense flow with cv2.calcOpticalFlowFarneback() between consecutive frames.

Step 7: (Optional) For Horn-Schunck: implement iterative solver or use existing implementations from external libraries.

Step 8: Visualize flow: draw vectors on the image for sparse flow or convert dense flow to HSV color map for visualization.

Step 9: Evaluate qualitatively (visual) and quantitatively if ground-truth flow is available (e.g., endpoint error).

Step 10: End

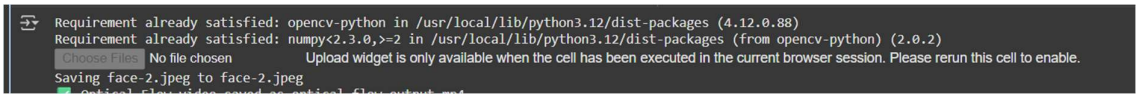
CODE:

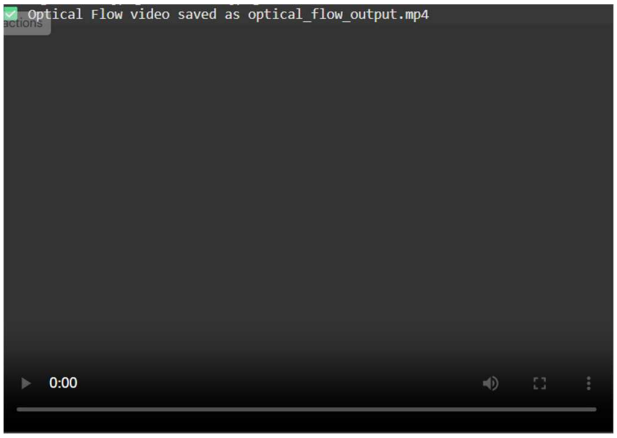
```
# ✅ Step 1: Install and import libraries
!pip install opencv-python

import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from IPython.display import HTML
from base64 import b64encode
from google.colab import files

# ✅ Step 2: Upload video
uploaded = files.upload()  # Upload 'video.mp4' here
video_path = list(uploaded.keys())[0]

# ✅ Step 3: Read video
cap = cv2.VideoCapture(video_path)
ret, frame1 = cap.read()
if not ret:
    print("Error: Unable to read video.")
    cap.release()
    exit()

prvs = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[..., 1] = 255

# ✅ Video writer (H.264 encoding might work better in Colab)
```

```
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('optical_flow_output.mp4', fourcc, 20.0, (frame1.shape[1], frame1.shape[0]))

frame_count = 0

while True:
    ret, frame2 = cap.read()
    if not ret:
        break

    next_frame = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)

    flow = cv2.calcOpticalFlowFarneback(prvs, next_frame, None,
                                        0.5, 3, 15, 3, 5, 1.2, 0)
    mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
    hsv[..., 0] = ang * 180 / np.pi / 2
    hsv[..., 2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)
    bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
    out.write(bgr)

    if frame_count % 20 == 0:
        print(f"Processing frame {frame_count}")
        cv2_imshow(bgr)

    frame_count += 1
    prvs = next_frame

# ✅ Release everything
cap.release()
out.release()
cv2.destroyAllWindows()
```

```
print("✅ Optical Flow video saved as optical_flow_output.mp4")


# ✅ Display video inline (force reload)
from IPython.display import display

video_file = 'optical_flow_output.mp4'
mp4 = open(video_file, 'rb').read()
data_url = f"data:video/mp4;base64,{b64encode(mp4).decode()}"
display(HTML(f"""
<video width="600" height="400" controls autoplay loop>
  <source src="{data_url}" type="video/mp4">
  Your browser does not support the video tag.
</video>
"""))
```

OUTPUT:





RESULT:

The optical flow computation successfully estimated the motion between consecutive frames, showing the direction and magnitude of movement in the scene. The resulting flow vectors visually represented how objects in the image shifted over time.