|  |  |
| --- | --- |
| **Exercise No**: **12**  **Date : 03.04.25** | **OPTICAL FLOW USING LUCAS KANADE AND FEATURE TRACKING USING KANADE LUCAS TOMASI ALGORITHM** |

**AIM :**

To estimate Optical Flow of first two Frames of the video and the full video and Feature Tracking using Kanade Lucas Tomasi Algorithm.

**FUNCTIONS :**

**OpenCV Functions**

1. cv2.VideoCapture(filename)
   * Opens a video file for reading.
   * Returns a video capture object.
2. cap.read()
   * Reads the next video frame.
   * Returns a boolean (ret) and the frame.
3. cv2.cvtColor(image, code)
   * Converts the image color space.
   * Common code: cv2.COLOR\_BGR2GRAY.
4. cv2.goodFeaturesToTrack(image, mask, \*\*params)
   * Detects strong corners to track using Shi-Tomasi method.
   * Parameters include maxCorners, qualityLevel, minDistance, blockSize.
5. cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts, \*\*params)
   * Calculates optical flow for a sparse set of points using Lucas-Kanade method.
   * Returns the new point positions, status, and error.
6. cv2.line(image, pt1, pt2, color, thickness)
   * Draws a line between two points.
7. cv2.circle(image, center, radius, color, thickness)
   * Draws a circle at the specified location.
8. cv2.add(image1, image2)
   * Adds two images element-wise.
9. cv2.imshow(winname, image)
   * Displays an image in a window.
10. cv2.waitKey(delay)
    * Waits for a key event for a specified time (ms).
11. cv2.destroyAllWindows()
    * Closes all OpenCV windows.
12. cv2.VideoWriter(filename, fourcc, fps, frameSize)
    * Creates a video writer to save frames as a video.
13. cv2.resize(image, dsize)
    * Resizes the image to the given size.
14. cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT
    * Termination condition for iterative algorithms.

**NumPy Functions**

1. numpy.sqrt(x)
   * Computes the square root.
2. numpy.ravel()
   * Flattens a multi-dimensional array.
3. numpy.ones\_like(array)
   * Creates an array of ones with the same shape.
4. numpy.zeros\_like(array)
   * Creates an array of zeros with the same shape.
5. numpy.random.randint(low, high, shape)
   * Generates random integers in the specified range.

**ALGORITHM :**

**OPTICAL FLOW - LUCAS KANADE (Two-frame & Full Video Version)**

1. Load Video and Initial Frames

* Read first two frames using cv2.VideoCapture().
* Convert to grayscale using cv2.cvtColor().

2. Feature Detection

* Use cv2.goodFeaturesToTrack() to identify corners (features) in the first frame.

3. Optical Flow Calculation

* Use cv2.calcOpticalFlowPyrLK() to track features from frame 1 to frame 2 (or between video frames).
* Calculate the displacement between old and new positions.

4. Filtering by Movement

* For each feature point, compute the Euclidean distance between old and new points.
* If below a threshold (e.g., 2 pixels), discard the motion (insignificant movement).

5. Visualize Motion

* Draw lines and circles to show the movement of features.
* Use color coding (e.g., red for large movement, green/blue for smaller).

6. Video Writing (Full Version)

* Create VideoWriter object.
* Write each processed frame to a video file (optical\_flow\_output.mp4).

7. Display Results

* Display the processed frames using cv2.imshow() until the end or ESC is pressed.

**KLT FEATURE TRACKING**

This is an extended form of Lucas-Kanade where tracked features are updated frame-to-frame over time.

1. Initialization

* Read the first frame, convert to grayscale.
* Use cv2.goodFeaturesToTrack() to extract initial tracking points.

2. Feature Tracking in Loop

* For each new frame:
  + Convert to grayscale.
  + Calculate optical flow from previous to current frame.
  + Update only the successfully tracked points.
  + Visualize the motion with unique colors per feature.

3. Handle Feature Loss

* If no points are tracked, re-detect features in the current frame.

4. Visualize

* Draw updated paths using cv2.line() and cv2.circle().

**CODE :**

**OPTICAL FLOW - LUCAS KANADE (Two-frame Version)**

import cv2

import numpy as np

cap = cv2.VideoCapture("op.mp4")

ret,frame1 = cap.read()

if not ret:

print("Error: Could not read frame")

cap.release()

exit()

ret,frame2 = cap.read()

if not ret:

print("Error:Could not read frame")

cap.release()

exit()

gray1 = cv2.cvtColor(frame1,cv2.COLOR\_BGR2GRAY)

gray2 = cv2.cvtColor(frame2,cv2.COLOR\_BGR2GRAY)

feature\_params = dict(maxCorners=400,qualityLevel=0.1,minDistance=7,blockSize=7)

p0 = cv2.goodFeaturesToTrack(gray1,mask=None,\*\*feature\_params)

lk = dict(winSize=(21,21),maxLevel=5,criteria=(cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT,10,0.03))

p1,st,err = cv2.calcOpticalFlowPyrLK(gray1,gray2,p0,None,\*\*lk)

good\_new = p1[st==1]

good\_old = p0[st==1]

movement\_threshold = 2

for i,(new,old) in enumerate(zip(good\_new,good\_old)):

a,b = new.ravel()

c,d = old.ravel()

movement\_distance = np.sqrt((a - c) \*\* 2 + (b - d) \*\* 2)

if movement\_distance < movement\_threshold:

continue

color = (0, 255, 0)

if movement\_distance > 5:

color = (0, 0, 255)

frame2 = cv2.line(frame2,(int(c), int(d)), (int(a), int(b)), color, 2)

frame2 = cv2.circle(frame2,(int(a), int(b)),5, color, -1)

cv2.imshow("Optical Flow (Two frames)",frame2)

cv2.waitKey(0)

cv2.destroyAllWindows()

cap.release()



**OPTICAL FLOW** - **LUCAS KANADE (Full Video Version)**

import cv2

import numpy as np

cap = cv2.VideoCapture("op.mp4")

ret, old\_frame = cap.read()

if not ret:

print("Error: Could not read video.")

cap.release()

exit()

old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY)

feature\_params = dict(maxCorners=400, qualityLevel=0.1, minDistance=7, blockSize=7)

feature\_mask = np.ones\_like(old\_gray, dtype=np.uint8) \* 255

feature\_mask[:, :150] = 0

p0 = cv2.goodFeaturesToTrack(old\_gray, mask=feature\_mask, \*\*feature\_params)

lk\_params = dict(winSize=(21, 21), maxLevel=5,

criteria=(cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, 10, 0.03))

mask = np.zeros\_like(old\_frame)

movement\_threshold = 2

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

fps = cap.get(cv2.CAP\_PROP\_FPS)

width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

out = cv2.VideoWriter("optical\_flow\_output.mp4", fourcc, fps, (width, height))

while True:

ret, frame = cap.read()

if not ret:

break

frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, frame\_gray, p0, None, \*\*lk\_params)

if p1 is None:

break

good\_new = p1[st == 1]

good\_old = p0[st == 1]

for new, old in zip(good\_new, good\_old):

a, b = new.ravel()

c, d = old.ravel()

movement\_distance = np.sqrt((a - c) \*\* 2 + (b - d) \*\* 2)

if movement\_distance < movement\_threshold:

continue

color = (255, 0, 0)

if movement\_distance > 5:

color = (0, 0, 255)

mask = cv2.line(mask, (int(c), int(d)), (int(a), int(b)), color, 2)

frame = cv2.circle(frame, (int(a), int(b)), 5, color, -1)

img = cv2.add(frame, mask)

cv2.imshow("Optical Flow - Full Video", img)

out.write(img)

if cv2.waitKey(30) & 0xFF == 27:

break

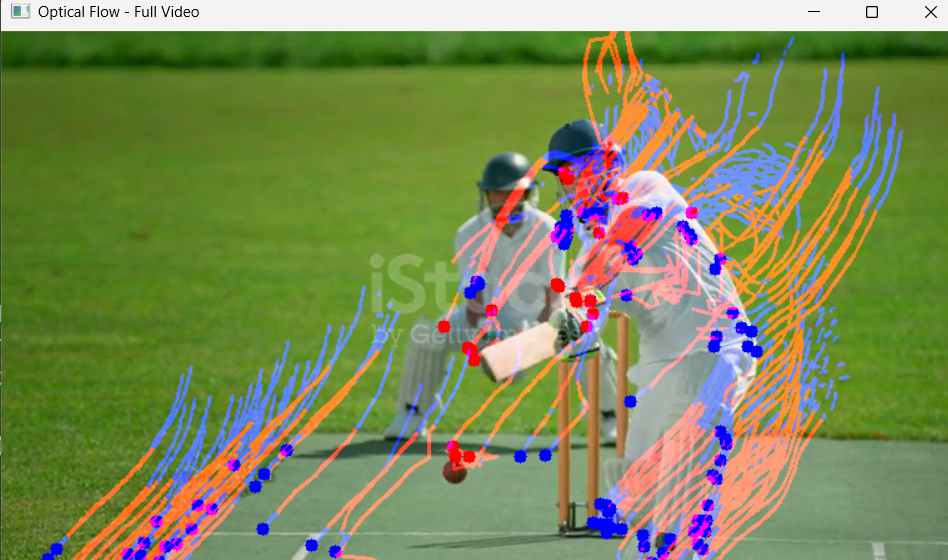
old\_gray = frame\_gray.copy()

p0 = good\_new.reshape(-1, 1, 2)

cap.release()

out.release()

cv2.destroyAllWindows()



**KLT FEATURE TRACKING**

import cv2

import numpy as np

cap = cv2.VideoCapture('mi.mp4')

feature\_params = dict(maxCorners=100,

qualityLevel=0.3,

minDistance=7,

blockSize=7)

lk\_params = dict(winSize=(15, 15),

maxLevel=2,

criteria=(cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, 10, 0.03))

color = np.random.randint(0, 255, (100, 3))

ret, old\_frame = cap.read()

old\_frame = cv2.resize(old\_frame, (512, 512))

old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY)

p0 = cv2.goodFeaturesToTrack(old\_gray, mask=None, \*\*feature\_params)

mask = np.zeros\_like(old\_frame)

while True:

ret, frame = cap.read()

if not ret:

break

frame = cv2.resize(frame, (512, 512))

frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, frame\_gray, p0, None, \*\*lk\_params)

if p1 is not None:

good\_new = p1[st == 1]

good\_old = p0[st == 1]

new\_mask = np.zeros\_like(mask)

for i, (new, old) in enumerate(zip(good\_new, good\_old)):

a, b = new.ravel()

c, d = old.ravel()

new\_mask = cv2.line(new\_mask, (int(a), int(b)), (int(c), int(d)), color[i % 100].tolist(), 2)

frame = cv2.circle(frame, (int(a), int(b)), 5, color[i % 100].tolist(), -1)

mask = new\_mask

img = cv2.add(frame, mask)

old\_gray = frame\_gray.copy()

p0 = good\_new.reshape(-1, 1, 2)

else:

p0 = cv2.goodFeaturesToTrack(frame\_gray, mask=None, \*\*feature\_params)

mask = np.zeros\_like(frame)

cv2.imshow('KLT Feature Tracking', img if 'img' in locals() else frame)

if cv2.waitKey(30) & 0xFF == 27:

break

cap.release()

cv2.destroyAllWindows()



**RESULT :**

The estimation of Optical Flow of first two Frames of the video and the full video and Feature Tracking using Kanade Lucas Tomasi Algorithm are implemented successfully.