**EX.NO:12 DATE: 03.04.2025**

**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.

**Algorithm:**

Step 1: Video Setup

1. Load the video using cv2.VideoCapture().
2. Read the first frame to prepare for manual object selection.

Step 2: Region Selection (ROI)

1. Manually select the bounding box by right-clicking to define corners.
2. Display the rectangle live on the frame.
3. Middle-click to reset coordinates.
4. After pressing ESC, crop the selected area.

Step 3: Feature Detection

1. Convert the cropped ROI to grayscale.
2. Use cv2.goodFeaturesToTrack() to detect strong corner features inside this ROI.
3. Filter the feature points to ensure they lie within the selected box.
4. Store the first valid point (for optical flow tracking).

Step 4: Optical Flow Initialization

1. Define Lucas-Kanade parameters, including window size, pyramid levels, and termination criteria.
2. Read the first frame again and convert to grayscale.
3. Initialize a mask to store drawing trails and begin frame counter and timer.

Step 5: Optical Flow Tracking Loop

1. Loop through all frames:
   * Convert each frame to grayscale.
   * If p0 (tracking point) exists:
     + Use cv2.calcOpticalFlowPyrLK() to calculate the new position of the tracked point.
     + If tracking was successful:
       - Extract good\_new and good\_old points.
       - Draw line and circle showing movement.
       - Update p0 and previous frame.
       - Display FPS on screen.
     + Else:
       - Set p0 = None (tracking lost).
2. Check if the tracked point goes out of bounds; if yes, reset p0.

Step 6: Feature Re-Detection (If Tracking Lost)

1. If p0 is None (no valid tracking point):

* Redetect new features in the current frame using cv2.goodFeaturesToTrack().
* Reset drawing mask to clean state.
* Continue tracking newly found point.

Step 7: Exit Condition

1. Press ESC to break the loop.
2. Release video capture and close all OpenCV windows.

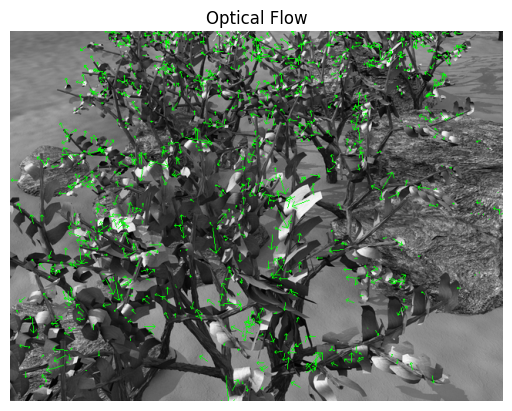
**Code:**

import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
  
def optical\_flow(old\_frame, new\_frame, window\_size=5, min\_quality=0.01):  
 max\_corners = 1000  
 min\_distance = 1  
 feature\_list = cv2.goodFeaturesToTrack(old\_frame, max\_corners, min\_quality, min\_distance)  
  
 w = window\_size // 2  
  
 old\_frame = old\_frame / 255.0  
 new\_frame = new\_frame / 255.0  
  
 kernel\_x = np.array([[-1, 1], [-1, 1]])  
 kernel\_y = np.array([[-1, -1], [1, 1]])  
 kernel\_t = np.array([[1, 1], [1, 1]])  
  
 fx = cv2.filter2D(old\_frame, -1, kernel\_x)  
 fy = cv2.filter2D(old\_frame, -1, kernel\_y)  
 ft = cv2.filter2D(new\_frame, -1, kernel\_t) - cv2.filter2D(old\_frame, -1, kernel\_t)  
  
 u = np.zeros(old\_frame.shape)  
 v = np.zeros(old\_frame.shape)  
  
 for feature in feature\_list:  
 j, i = feature.ravel()  
 i, j = int(i), int(j)  
  
 if i - w < 0 or i + w + 1 > old\_frame.shape[0] or j - w < 0 or j + w + 1 > old\_frame.shape[1]:  
 continue  
  
 I\_x = fx[i-w:i+w+1, j-w:j+w+1].flatten()  
 I\_y = fy[i-w:i+w+1, j-w:j+w+1].flatten()  
 I\_t = ft[i-w:i+w+1, j-w:j+w+1].flatten()  
  
 b = np.reshape(I\_t, (I\_t.shape[0], 1))  
 A = np.vstack((I\_x, I\_y)).T  
  
 if A.shape[0] > 1:  
 U = np.matmul(np.linalg.pinv(A), b)  
 u[i, j] = U[0][0]  
 v[i, j] = U[1][0]  
  
 return feature\_list, u, v  
  
  
def draw\_flow(img, features, u, v, scale=5):  
 img\_color = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR)  
 for pt in features:  
 x, y = pt.ravel()  
 x, y = int(x), int(y)  
 dx = int(u[y, x] \* scale)  
 dy = int(v[y, x] \* scale)  
 cv2.arrowedLine(img\_color, (x, y), (x+dx, y+dy), (0, 255, 0), 1, tipLength=0.3)  
 return img\_color  
  
  
def process\_images(image1\_path, image2\_path):  
 old\_frame = cv2.imread(image1\_path, cv2.IMREAD\_GRAYSCALE)  
 new\_frame = cv2.imread(image2\_path, cv2.IMREAD\_GRAYSCALE)  
  
 features, u, v = optical\_flow(old\_frame, new\_frame)  
 flow\_img = draw\_flow(new\_frame, features, u, v)  
  
 plt.imshow(cv2.cvtColor(flow\_img, cv2.COLOR\_BGR2RGB))  
 plt.title("Optical Flow")  
 plt.axis("off")  
 plt.show()  
  
  
def process\_video(video\_path):  
 cap = cv2.VideoCapture(video\_path)  
 ret, old\_frame = cap.read()  
  
 if not ret:  
 print("Cannot read video.")  
 return  
  
 old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY)  
  
 while True:  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 new\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 features, u, v = optical\_flow(old\_gray, new\_gray)  
 flow\_img = draw\_flow(new\_gray, features, u, v)  
  
 cv2.imshow("Optical Flow", flow\_img)  
 if cv2.waitKey(30) & 0xFF == 27:   
 break  
 old\_gray = new\_gray  
  
 cap.release()  
 cv2.destroyAllWindows()

process\_video("track\_Car.avi")

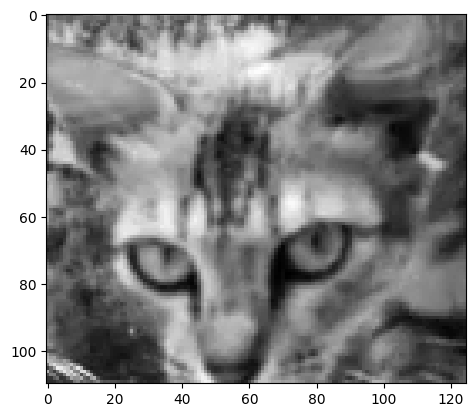


process\_images("grove1.png", "grove2.png")



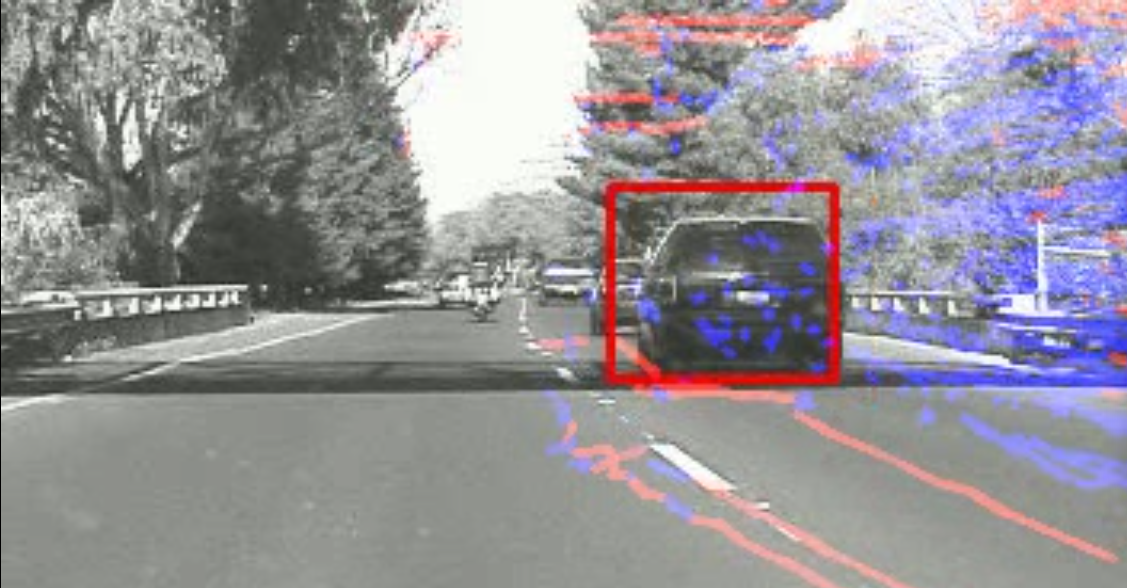
video\_path="Cat.mp4"   
video = cv2.VideoCapture(video\_path)  
ret,frame = video.read()  
x\_min,y\_min,x\_max,y\_max=36000,36000,0,0  
  
  
def coordinat\_chooser(event,x,y,flags,param):  
 global go , x\_min , y\_min, x\_max , y\_max  
 if event==cv2.EVENT\_RBUTTONDOWN:  
 x\_min=min(x,x\_min)   
 y\_min=min(y,y\_min)  
 x\_max=max(x,x\_max)  
 y\_max=max(y,y\_max)  
  
 cv2.rectangle(frame,(x\_min,y\_min),(x\_max,y\_max),(0,255,0),1)  
 if event==cv2.EVENT\_MBUTTONDOWN:  
 print("reset coordinate data")  
 x\_min,y\_min,x\_max,y\_max=36000,36000,0,0  
  
cv2.namedWindow('coordinate\_screen')  
cv2.setMouseCallback('coordinate\_screen',coordinat\_chooser)  
  
  
while True:  
 cv2.imshow("coordinate\_screen",frame)   
   
 k = cv2.waitKey(5) & 0xFF   
 if k == 27:  
 cv2.destroyAllWindows()  
 break  
  
  
cv2.destroyAllWindows()  
roi\_image=frame[y\_min:y\_max,x\_min:x\_max]  
  
roi\_gray=cv2.cvtColor(roi\_image,cv2.COLOR\_BGR2GRAY)   
  
feature\_params = dict(maxCorners=20,   
 qualityLevel=0.2,   
 minDistance=7,   
 blockSize=7)  
  
first\_gray = cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)  
  
points = cv2.goodFeaturesToTrack(first\_gray, mask=None, \*\*feature\_params)  
  
for point in points:  
 x, y = point.ravel()  
 if y\_min <= y <= y\_max and x\_min <= x <= x\_max:  
 selected\_point = point  
 break  
  
if selected\_point is not None:  
 p0 = np.array([selected\_point], dtype=np.float32)  
  
plt.imshow(roi\_gray,cmap="gray")

<matplotlib.image.AxesImage at 0x1f73521bc50>



import time  
  
lk\_params = dict(winSize=(7, 7),   
 maxLevel=2,   
 criteria=(cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, 10, 0.03))  
  
cap = cv2.VideoCapture(video\_path)  
  
ret, old\_frame = cap.read()  
  
width = old\_frame.shape[1]  
height = old\_frame.shape[0]  
  
mask = np.zeros\_like(old\_frame)  
  
frame\_count = 0  
start\_time = time.time()  
  
old\_gray = first\_gray  
  
while True:  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 if p0 is not None:  
 p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, frame\_gray, p0, None, \*\*lk\_params)   
 good\_new = p1[st == 1]   
 good\_old = p0[st == 1]  
  
  
 if len(good\_new) > 0:  
 a, b = good\_new[0].ravel()  
 c, d = good\_old[0].ravel()  
   
 mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)  
 frame = cv2.circle(frame, (int(a), int(b)), 5, (0, 255, 0), -1)  
  
 img = cv2.add(frame, mask)  
  
 elapsed\_time = time.time() - start\_time  
 fps = frame\_count / elapsed\_time if elapsed\_time > 0 else 0  
 cv2.putText(img, f"FPS: {fps:.2f}", (width - 200, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE\_AA)  
  
 cv2.imshow('frame', img)  
  
 old\_gray = frame\_gray.copy()  
 p0 = good\_new.reshape(-1, 1, 2)  
  
 else:  
 p0 = None  
  
 if not (25 <= a < width):  
 p0 = None

selected\_point\_distance = 0  
  
 if p0 is None:  
 p0 = cv2.goodFeaturesToTrack(frame\_gray, mask=None, \*\*feature\_params)  
 mask = np.zeros\_like(frame)  
 selected\_point\_distance=0  
   
 frame\_count += 1  
  
 k = cv2.waitKey(25)  
 if k == 27:  
 break  
  
cv2.destroyAllWindows()  
cap.release()



**Inference**

The implemented system successfully tracks a manually selected object, such as a ball, across video frames using the Lucas-Kanade Optical Flow method. By detecting and following strong feature points within a user-defined region of interest (ROI), the algorithm computes the object’s motion and visually represents its trajectory in real time. It also incorporates a recovery mechanism to re-detect features if tracking fails, making it robust for short occlusions or rapid movements. Overall, the approach demonstrates an effective method for motion tracking in videos using classical computer vision techniques.

**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.