CourseName: Computer Vision Lab **Course Code:** CSP-422

Experiment: 3.1

Aim: Write a program to determine the effectiveness of incorporating optical flow analysis into object tracking algorithms.

Software Required: Any Python IDE (e.g.: PyCharm, Jupyter Notebook, GoogleCollab)

Technique used:

Optical flow analysis and object tracking are important techniques in computer vision that deal with the motion of objects in images or video sequences.

Optical flow analysis is often integrated into object tracking algorithms to enhance the tracking process. Optical flow provides information about the motion of pixels between consecutive frames in a video sequence. By incorporating optical flow, object tracking algorithms can better understand and predict the movement of objects in the scene.

This integration is valuable for addressing challenges such as occlusion, scale variation, and rotation in object tracking. It helps to maintain the continuity of tracking even when objects undergo complex motion patterns. The combination of optical flow analysis and object tracking contributes to more robust and accurate tracking systems, especially in dynamic and challenging environments.

Here are key points related to optical flow analysis and object tracking algorithms:

- Motion Estimation
- Initial Object Localization
- Feature Extraction
- Object Representation
- Tracking Initialization
- Tracking Update
- Handling Occlusion
- Adaptive Adjustments
- Improved Robustness

Steps:

1. Import the required libraries, including datetime, numpy, cv2, sys, and os. Import cv2_imshow from google.colab.patches for displaying images in Colab.
2. Define the path to the video file.
3. Open Video Capture and read the input video or capture frames from a camera feed.
4. Generate initial corners of detected object. Set limit, minimum distance in pixels, and quality of object corner to be tracked.
5. Get the first video frame.
6. Create a directory to save frames
7. Object Tracking Loop, Iterate through each frame in the video sequence:
    a. Read each frame
    b. Convert the frame to grayscale
    c. Apply Lucas-Kanade optical flow for object tracking
    d. Draw lines and circles for object tracking
    e. Combine the original frame with the drawn lines
    f. Display the frame with the tracked objects and result.
    g. Save each frame as an image
    h. Update variables for the next iteration
8. Release Video Capture and Convert frames to a video
9. Create a video file and write frames to it and evaluate the performance.

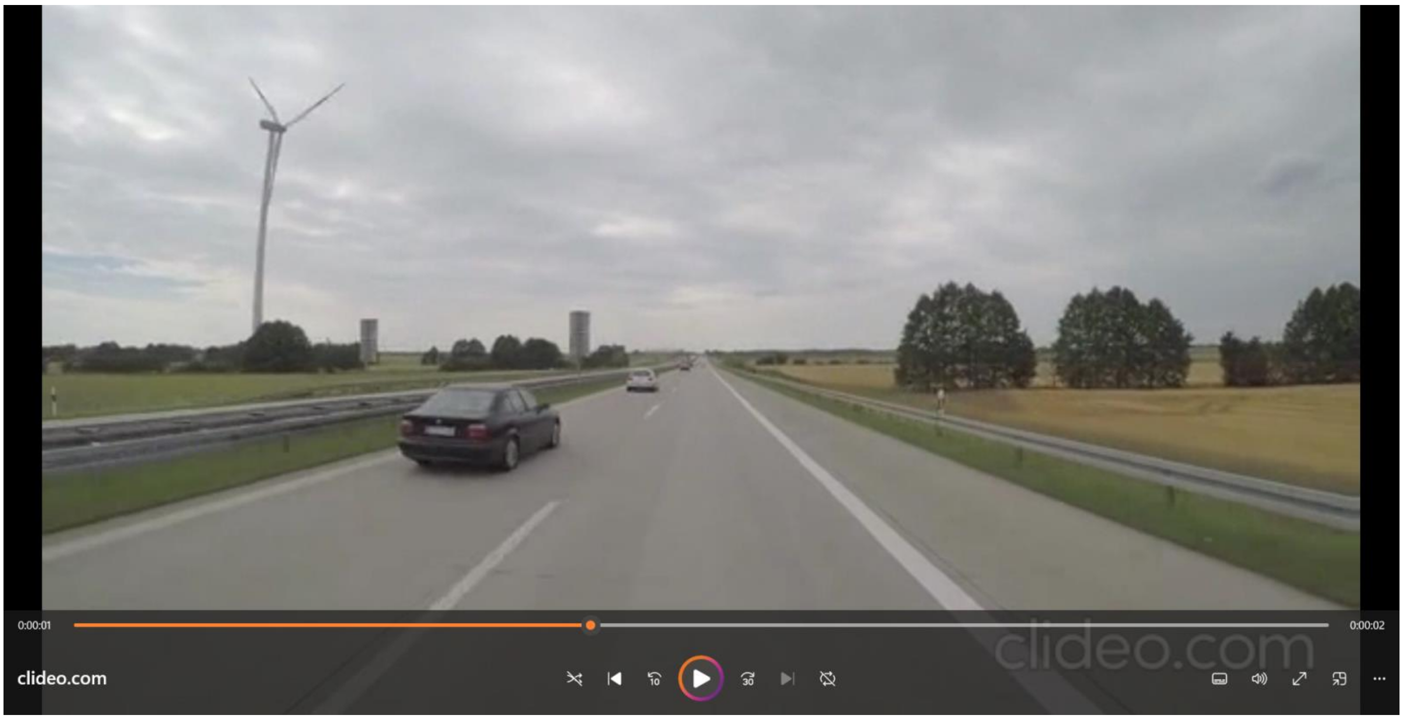
Implementation:

```
# Import Libraries
import datetime
import numpy as np
import cv2
import sys
import os
from google.colab.patches import cv2_imshow
```

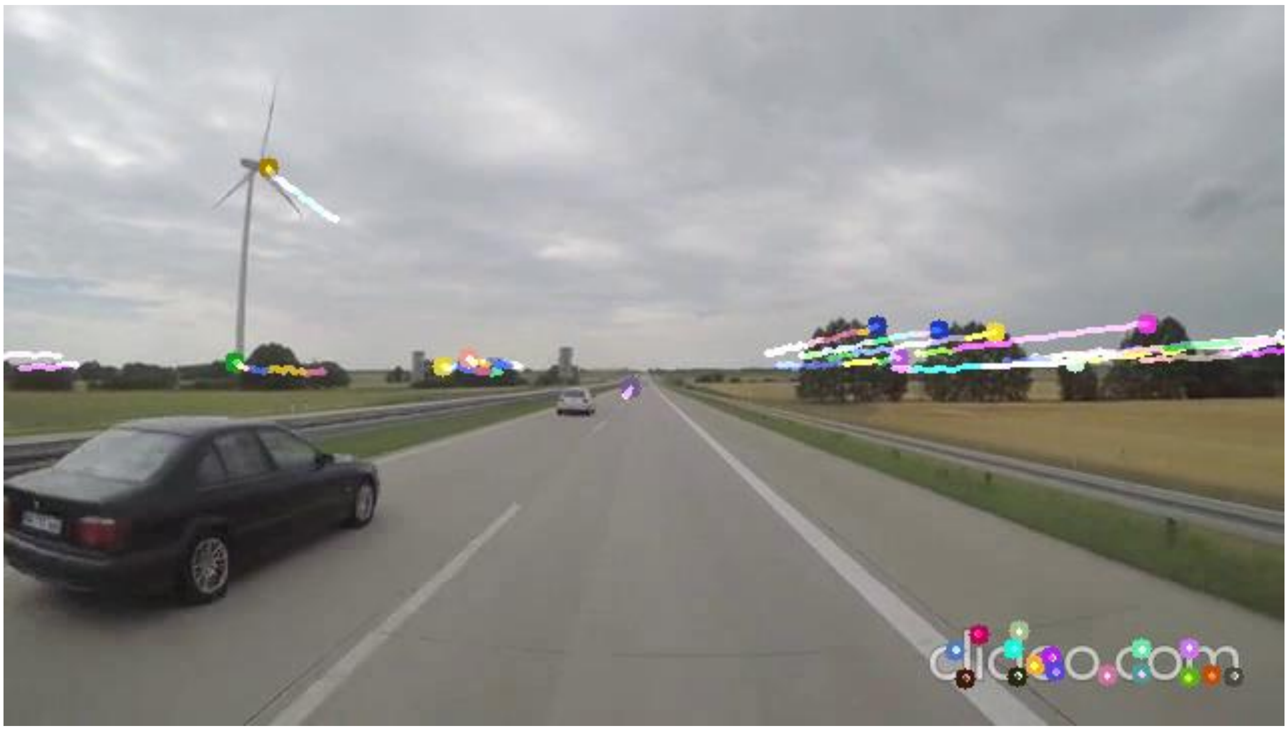
```
# Define the path to the video file
video_path = './cars.mp4'  # Adjust the path to your uploaded video

# Open Video Capture
cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print("[ERROR] cannot open video file")
    sys.exit()

# Generate initial corners of detected object
# Set limit, minimum distance in pixels, and quality of object corner to be tracked
parameters_shi_tomasi = dict(maxCorners=100, qualityLevel=0.3, minDistance=7)
parameter_lucas_kanade = dict(winSize=(15, 15), maxLevel=2,
criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
colours = np.random.randint(0, 255, (100, 3))

# Get the first video frame
ok, frame = cap.read()
if not ok:
    print("[ERROR] cannot get frame from video")
    sys.exit()
frame_gray_init = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

edges = cv2.goodFeaturesToTrack(frame_gray_init, mask=None,
**parameters_shi_tomasi)
canvas = np.zeros_like(frame)

# Create a directory to save frames
frames_directory = './'
os.makedirs(frames_directory, exist_ok=True)

frame_count = 0  # Count frames for saving

# Object Tracking Loop
while True:
    # Read each frame
    ok, frame = cap.read()
    if not ok:
        print("[INFO] end of file reached")
        break
    # Convert the frame to grayscale
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply Lucas-Kanade optical flow for object tracking
    update_edges, status, _ = cv2.calcOpticalFlowPyrLK(frame_gray_init, frame_gray,
edges, None, **parameter_lucas_kanade)
    new_edges = update_edges[status == 1]
    old_edges = edges[status == 1]

    # Draw lines and circles for object tracking
    for i, (new, old) in enumerate(zip(new_edges, old_edges)):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv2.line(canvas, (int(a), int(b)), (int(c), int(d)),
colours[i].tolist(), 2)
        frame = cv2.circle(frame, (int(a), int(b)), 5, colours[i].tolist(), -1)

    # Combine the original frame with the drawn lines
    result = cv2.add(frame, mask)

    # Display the tracking result
    cv2_imshow(result)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    # Save each frame as an image
    cv2.imwrite(frames_directory + f"frame_{frame_count:04d}.jpg", frame)
    frame_count += 1

    # Update variables for the next iteration
    frame_gray_init = frame_gray.copy()
    edges = new_edges.reshape(-1, 1, 2)

# Release Video Capture
cv2.destroyAllWindows()
cap.release()

# Convert frames to a video
files = os.listdir(frames_directory)
```

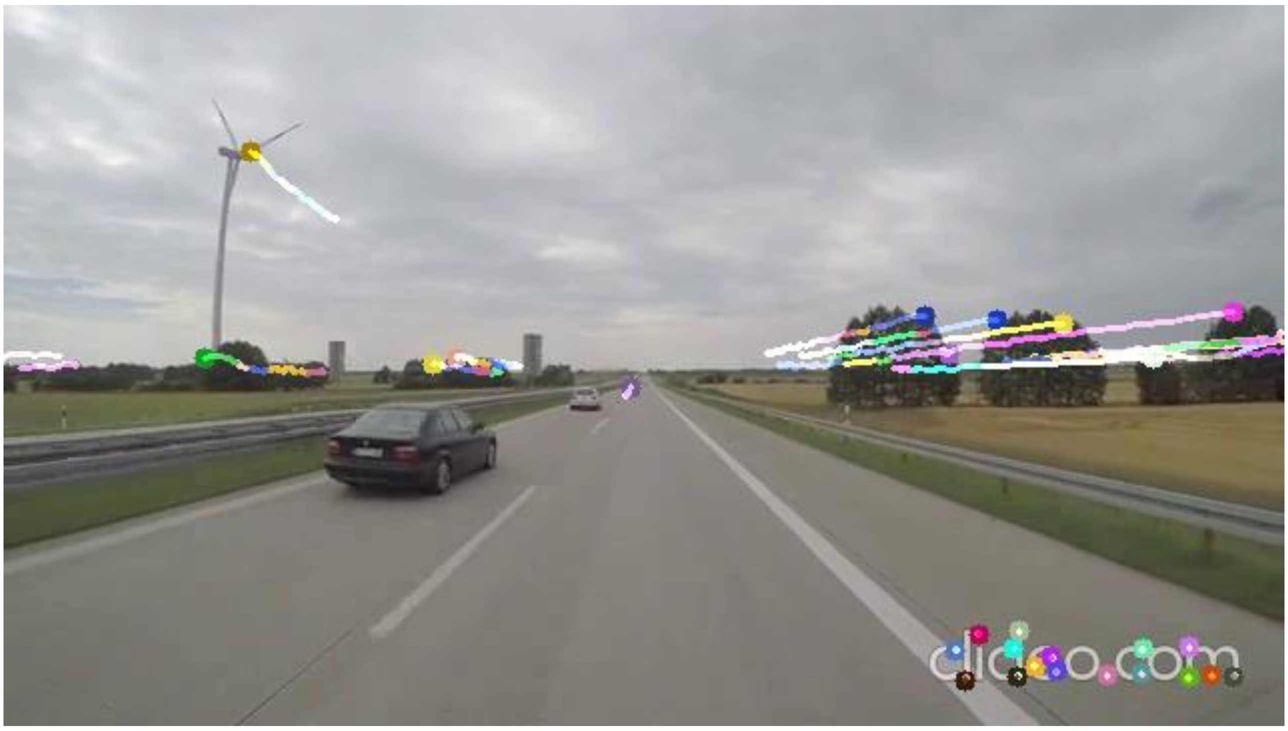
```
files.sort()

if len(files) > 0:
    img = cv2.imread(frames_directory + files[0])
    height, width, layers = img.shape

    # Create a video file and write frames to it
    fourcc = cv2.VideoWriter_fourcc(*'MP4V')
    video = cv2.VideoWriter('/content/created_video.mp4', fourcc, 30, (width,
height))

    for file in files:
        img = cv2.imread(frames_directory + file)
        video.write(img)

    # Release Video Writer
    cv2.destroyAllWindows()
    video.release()
```

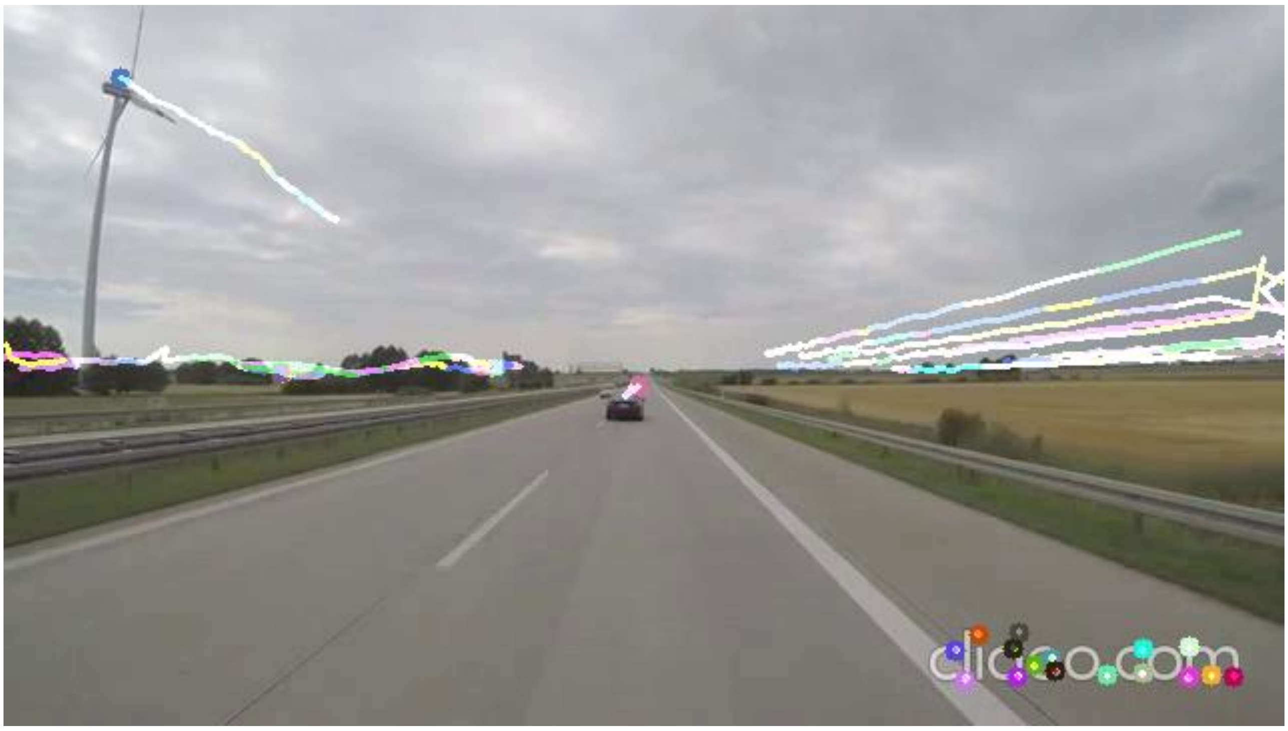
Normal Video:



Output Video






clideo.com