School: .... Campus: ....

Academic Year: .... Subject Name: .... Subject Code: ....

Semester: .... Program: .... Branch: .... Specialization: ....

Date: ....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement :

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

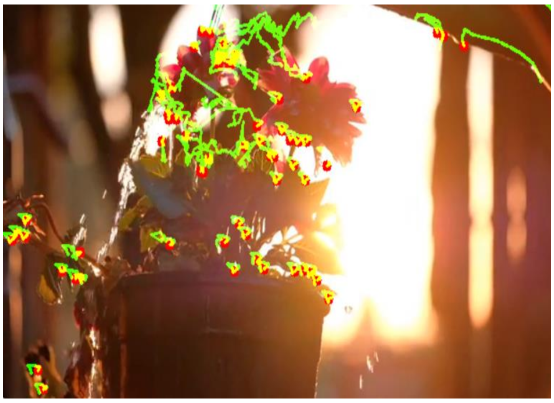
1. **Lucas-Kanade optical flow**
2. **HornSchunck**
3. **Block matching ebma Thresholding**
4. **Block matching dbma Thresholding**

**As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.

* Testing Phase: Compilation of Code (error detection)

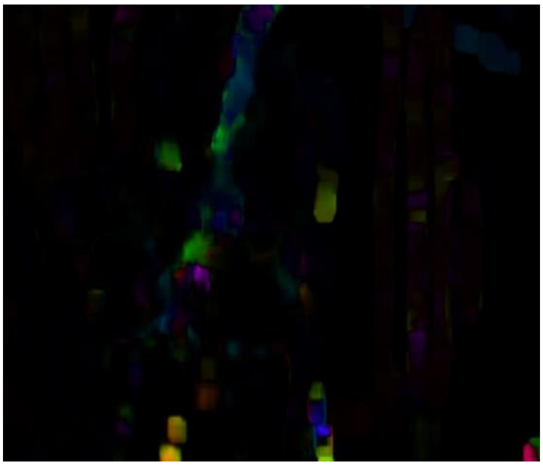
1) *Lucas-Kanade optical flow*

```
import cv2
import numpy as np
cap = cv2.VideoCapture('sample-5.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))
ret, old_frame = cap.read()
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_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0,
None, **lk_params)
    good_new = p1[st == 1]
    good_old = p0[st == 1]
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        a, b = new.ravel().astype(int)
        c, d = old.ravel().astype(int)
        mask = cv2.line(mask, (a, b), (c, d), (0, 255, 0), 2)
        frame = cv2.circle(frame, (a, b), 5, (0, 0, 255), -1)
    img = cv2.add(frame, mask)
    cv2.imshow('frame', img)
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break
    old_gray = frame_gray.copy()


    p0 = good_new.reshape(-1, 1, 2)


cap.release()


cv2.destroyAllWindows()
```



2) *Horn Schunck optical flow*

```
import cv2
import numpy as np
cap = cv2.VideoCapture('sample-5.mp4')
hs_params = dict(alpha=0.001, # smoothness parameter   iterations=100)
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
flow = None
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    if not ret:
        break
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    if flow is None:

        flow = cv2.calcOpticalFlowFarneback(old_gray, frame_gray, None,
 0.5, 3, 15, 3, 5, 1.2, 0)
    else:
        flow = cv2.calcOpticalFlowFarneback(old_gray, frame_gray, flow,
 0.5, 3, 15, 3, 5, 1.2, 0)
    mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
    mask = np.zeros_like(frame)
    mask[..., 1] = 255
    mask[..., 0] = ang * 180 / np.pi / 2
    mask[..., 2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)
    img = cv2.cvtColor(mask, cv2.COLOR_HSV2BGR)
    cv2.imshow('frame', img)
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break
    old_gray = frame_gray.copy()
cap.release()
cv2.destroyAllWindows()cap.release()
cv2.destroyAllWindows()
```



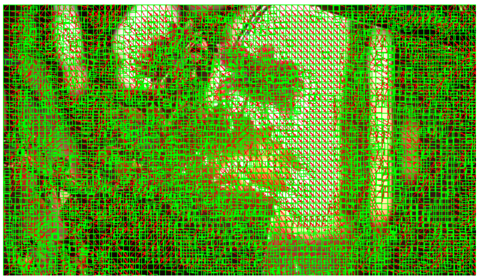
*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

Testing Phase: Compilation of Code (error detection

4) ***EBMA matching***

```
import cv2
import numpy as np
video = cv2.VideoCapture('sample-5.mp4')
block_size = 16
search_range = 16
while video.isOpened():
    ret, frame = video.read()
    if not ret:
        break
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    for y in range(0, gray_frame.shape[0] - block_size, block_size):
        for x in range(0, gray_frame.shape[1] - block_size, block_size):
            current_block = gray_frame[y:y+block_size, x:x+block_size]
            min_sad = float('inf')
            best_match = (0, 0)
            for dy in range(-search_range, search_range+1):
                for dx in range(-search_range, search_range+1):
                    x_search = x + dx
                    y_search = y + dy
                    if x_search < 0 or x_search + block_size >= gray_frame.shape[1] or y_search < 0 or 
y_search + block_size >= gray_frame.shape[0]:
                        continue
                    search_block = gray_frame[y_search:y_search+block_size, x_search:x_search+block_size]
                    sad = np.sum(np.abs(np.subtract(current_block, search_block)))
                    if sad < min_sad:
                        min_sad = sad
                        best_match = (dx, dy)
            cv2.arrowedLine(frame, (x+block_size//2, y+block_size//2), (x+block_size//2+best_match[0], 
y+block_size//2+best_match[1]), (0, 0, 255), 1)
    cv2.imshow('Motion Vectors', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video.release()
cv2.destroyAllWindows
```

4) ***DBMA matching***



```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
def dbma_block_matching(video_path, block_size=8, search_range=16):
cap = cv2.VideoCapture(video_path)
prev_frame = None
frame_count = 0
while True:
ret, frame = cap.read()
if not ret:
break
if frame_count > 0: # Start matching from the second frame
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray_prev_frame = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
output_frame = frame.copy()
for i in range(0, frame.shape[0], block_size):
for j in range(0, frame.shape[1], block_size):block = gray_frame[i:i+block_size, 
j:j+block_size]best_match_pos = dbma_search(block, gray_prev_frame, block_size, 
search_range)cv2.arrowedLine(output_frame, (j, i), (j + best_match_pos[1], i + best_match_pos[0]), (0, 0, 
255), 1)cv2.rectangle(output_frame, (j, i), (j + block_size, i + block_size), (0, 255, 0), 1)match_x = j + 
best_match_pos[1]
match_y = i + best_match_pos[0]
cv2.rectangle(output_frame, (match_x, match_y), (match_x + block_size, match_y + block_size), (0, 255, 0), 
1)
cv2_imshow(output_frame)
prev_frame = frame # Store for the next iteration
frame_count += 1
if cv2.waitKey(1) == ord("q"):
break cap.release() cv2.destroyAllWindows()def dbma_search(block, search_area, block_size, 
search_range):best_match_pos = (0, 0)
min_sad = float('inf')
video_path = '/content/sample-5.mp4'
dbma_block_matching(video_path)
```

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

In our experiment, we compared Lucas-Kanade and Horn-Schunck optical flow algorithms, alongside block matching with both exhaustive and dynamic programming-based motion estimation (EBMA and DBMA) with thresholding. We learned

Lucas-Kanade optical flow: A differential method commonly used for estimating the motion of objects in image sequences by solving an equation for each pixel.

Horn-Schunck: An optical flow algorithm based on the minimization of an energy functional, providing a smooth and dense flow field estimation.

Block matching EBMA (Exhaustive Block Matching Algorithm) Thresholding: A motion estimation technique that divides the image into blocks and exhaustively searches for the best matching block in the subsequent frame, followed by thresholding for motion detection.

Block matching DBMA (Dynamic Block Matching Algorithm) Thresholding: Similar to EBMA, DBMA optimizes block matching using dynamic programming, enhancing motion estimation accuracy with thresholding for effective motion detection.

ASSESSMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Signature of the Faculty:

Regn. No. :

✱As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.