**AIM:**

The Python script performs H.261-like video encoding and decoding by analyzing a given video through motion estimation and motion compensation techniques. During encoding, the script estimates motion vectors between consecutive frames and generates two output videos: one displaying the motion vectors (motion estimation) and the other showing the compensated frames (motion compensation). In the decoding phase, the script reconstructs the original video from the motion estimation and compensation outputs and saves it back in the AVI format.

**ALGORITHM:**

**1. Video Encoding Process**

In the encoding process, we take an input video file (in AVI format) and perform two main tasks:

* Motion Estimation
* Motion Compensation

**STEP 1:** Open the input video: We start by opening the input AVI file, which

contains the original video.

**STEP 2:** Process each frame: We go through each frame in the video one by one.

* + - For the first frame, we simply save it as it is in the output files because it's an initial reference frame (this is called an Intra-frame).
    - For subsequent frames, we do the following:
      * Motion Estimation: We compare the current frame with the previous frame to find out how much the image has shifted or changed. This gives us what are called motion vectors, which are like arrows showing how the content in each block of the frame has moved.
      * Motion Compensation: Using the motion vectors, we can reconstruct the current frame by predicting how it should look based on the previous frame and its motion. This is called motion compensation.

**STEP 3:** Save the results:

* + - We save the motion estimation (the motion vectors) into one video file (MP4 format).
    - We save the motion compensation (the reconstructed frame) into another video file (MP4 format).

**2. Video Decoding Process**

**STEP 1:** Open the two MP4 files: We open the two files generated in the encoding process

one containing motion estimation and the other containing motion compensation.

**STEP 2:** Reconstruct each frame: We process each frame from the two MP4 files:

We read the frame from the motion compensation file, as this contains the actual

reconstructed video frame.

**STEP 3:** Save the reconstructed video: We write the reconstructed frame (from the motion

compensation video) into an AVI output file. This step reconstructs the original video by

combining the information from the two MP4 files.

**STEP 4:** Repeat for all frames: We repeat this for all the frames in the two MP4

files, eventually recreating the entire original video.

**STEP 5:** Open the MP4 file: We open the MP4 file containing either motion

estimation or motion compensation.

**STEP 6:** Read and write frames: We read each frame from the MP4 file and write

it directly into an AVI file.

**STEP 7:** Save the AVI file: After processing all the frames, we save the new AVI

file, which is now a copy of the original MP4 file but in AVI format.

**SOURCE CODE:**

import cv2

import numpy as np

import os

class H261VideoEncoder:

    def \_\_init\_\_(self, block\_size=16, search\_range=8):

        self.block\_size = block\_size

        self.search\_range = search\_range

def motion\_estimation(self, current\_frame, reference\_frame):

        height, width = current\_frame.shape

        motion\_vectors = np.zeros((height // self.block\_size, width // self.block\_size, 2), dtype=int)

        for y in range(0, height, self.block\_size):

            for x in range(0, width, self.block\_size):

                best\_match = (0, 0)

                min\_error = float('inf')

                current\_block = current\_frame[y:y + self.block\_size, x:x + self.block\_size]

                for dy in range(-self.search\_range, self.search\_range + 1):

                    for dx in range(-self.search\_range, self.search\_range + 1):

                        ref\_x = x + dx

                        ref\_y = y + dy

                        if 0 <= ref\_x < width - self.block\_size and 0 <= ref\_y < height - self.block\_size:

                            ref\_block = reference\_frame[ref\_y:ref\_y + self.block\_size, ref\_x:ref\_x + self.block\_size]

                            error = np.sum((current\_block - ref\_block) \*\* 2)

                            if error < min\_error:

                                min\_error = error

                                best\_match = (dy, dx)

                motion\_vectors[y // self.block\_size, x // self.block\_size] = best\_match

        return motion\_vectors

    def motion\_compensation(self, reference\_frame, motion\_vectors):

        height, width = reference\_frame.shape

        compensated\_frame = np.zeros\_like(reference\_frame)

        for y in range(0, height, self.block\_size):

            for x in range(0, width, self.block\_size):

                dy, dx = motion\_vectors[y // self.block\_size, x // self.block\_size]

                ref\_x = x + dx

                ref\_y = y + dy

                if 0 <= ref\_x < width - self.block\_size and 0 <= ref\_y < height - self.block\_size:

                    compensated\_frame[y:y + self.block\_size, x:x + self.block\_size] = \

                        reference\_frame[ref\_y:ref\_y + self.block\_size, ref\_x:ref\_x + self.block\_size]

        return compensated\_frame

    def draw\_motion\_vectors(self, frame, motion\_vectors):

        frame\_with\_vectors = cv2.cvtColor(frame, cv2.COLOR\_GRAY2BGR)

        for y in range(motion\_vectors.shape[0]):

            for x in range(motion\_vectors.shape[1]):

                dy, dx = motion\_vectors[y, x]

                start\_point = (x \* self.block\_size + self.block\_size // 2, y \* self.block\_size + self.block\_size // 2)

                end\_point = (start\_point[0] + dx, start\_point[1] + dy)

                cv2.arrowedLine(frame\_with\_vectors, start\_point, end\_point, (0, 0, 255), 1, tipLength=0.4)

        return frame\_with\_vectors

def process\_video(input\_path, output\_motion\_estimation, output\_motion\_compensation, block\_size=16, search\_range=8):

    encoder = H261VideoEncoder(block\_size=block\_size, search\_range=search\_range)

    cap = cv2.VideoCapture(input\_path)

    if not cap.isOpened():

        print("Error: Cannot open video file.")

        return

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

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

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

    frame\_count = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')  # Use XVID for broader compatibility

    out\_motion\_estimation = cv2.VideoWriter(output\_motion\_estimation, fourcc, fps, (frame\_width, frame\_height))

    out\_motion\_compensation = cv2.VideoWriter(output\_motion\_compensation, fourcc, fps, (frame\_width, frame\_height), isColor=False)

    prev\_frame = None

    print(f"Processing {frame\_count} frames...")

    for i in range(frame\_count):

        ret, frame = cap.read()

        if not ret:

            break

        gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

        if prev\_frame is None:

            # Intra-frame (first frame)

            print("Processing Intra-frame (first frame)...")

            out\_motion\_estimation.write(cv2.cvtColor(gray\_frame, cv2.COLOR\_GRAY2BGR))

            out\_motion\_compensation.write(gray\_frame)

        else:

            print(f"Processing P-frame ({i+1}/{frame\_count})...")

            motion\_vectors = encoder.motion\_estimation(gray\_frame, prev\_frame)

            motion\_estimation\_frame = encoder.draw\_motion\_vectors(prev\_frame, motion\_vectors)

            compensated\_frame = encoder.motion\_compensation(prev\_frame, motion\_vectors)

            out\_motion\_estimation.write(motion\_estimation\_frame)

            out\_motion\_compensation.write(compensated\_frame)

        prev\_frame = gray\_frame

        if i % 10 == 0:

            print(f"Processed {i}/{frame\_count} frames...")

    cap.release()

    out\_motion\_estimation.release()

    out\_motion\_compensation.release()

    print(f"Processing complete! Output files:")

    print(f" - Motion Estimation video: {output\_motion\_estimation}")

    print(f" - Motion Compensation video: {output\_motion\_compensation}")

def convert\_mp4\_to\_avi(input\_mp4, output\_avi):

    cap = cv2.VideoCapture(input\_mp4)

    if not cap.isOpened():

        print("Error: Cannot open MP4 video file.")

        return

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

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

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

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')  # Use XVID codec for AVI format

    out = cv2.VideoWriter(output\_avi, fourcc, fps, (frame\_width, frame\_height))

    print(f"Converting {input\_mp4} to {output\_avi}...")

    while True:

        ret, frame = cap.read()

        if not ret:

            break

        out.write(frame)  # Write the current frame to the AVI file

    cap.release()

    out.release()

    print(f"Conversion complete! Output file: {output\_avi}")

def process\_decoding(input\_motion\_estimation, input\_motion\_compensation, output\_avi):

    cap\_estimation = cv2.VideoCapture(input\_motion\_estimation)

    cap\_compensation = cv2.VideoCapture(input\_motion\_compensation)

    if not cap\_estimation.isOpened() or not cap\_compensation.isOpened():

        print("Error: Cannot open one or both MP4 video files.")

        return

    frame\_width = int(cap\_estimation.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

    frame\_height = int(cap\_estimation.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

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

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')  # Use XVID codec for AVI format

    out = cv2.VideoWriter(output\_avi, fourcc, fps, (frame\_width, frame\_height))

    print(f"Decoding {input\_motion\_estimation} and {input\_motion\_compensation} into {output\_avi}...")

    while True:

        ret\_estimation, frame\_estimation = cap\_estimation.read()

        ret\_compensation, frame\_compensation = cap\_compensation.read()

if not ret\_estimation or not ret\_compensation:

            break

        original\_frame = frame\_compensation  # or any other frame reconstruction logic

        out.write(original\_frame)  # Write the reconstructed frame to the AVI file

    cap\_estimation.release()

    cap\_compensation.release()

    out.release()

    print(f"Decoding complete! Output file: {output\_avi}")if \_\_name\_\_ == "\_\_main\_\_":

    input\_path = input("Enter the path to the input AVI video file for encoding: ").strip()

    output\_motion\_estimation = "motion\_estimation.mp4"

    output\_motion\_compensation = "motion\_compensation.mp4"

    if not os.path.exists(input\_path):

        print("Error: Input file does not exist.")

        exit(1)

    process\_video(input\_path, output\_motion\_estimation, output\_motion\_compensation, block\_size=16, search\_range=8)

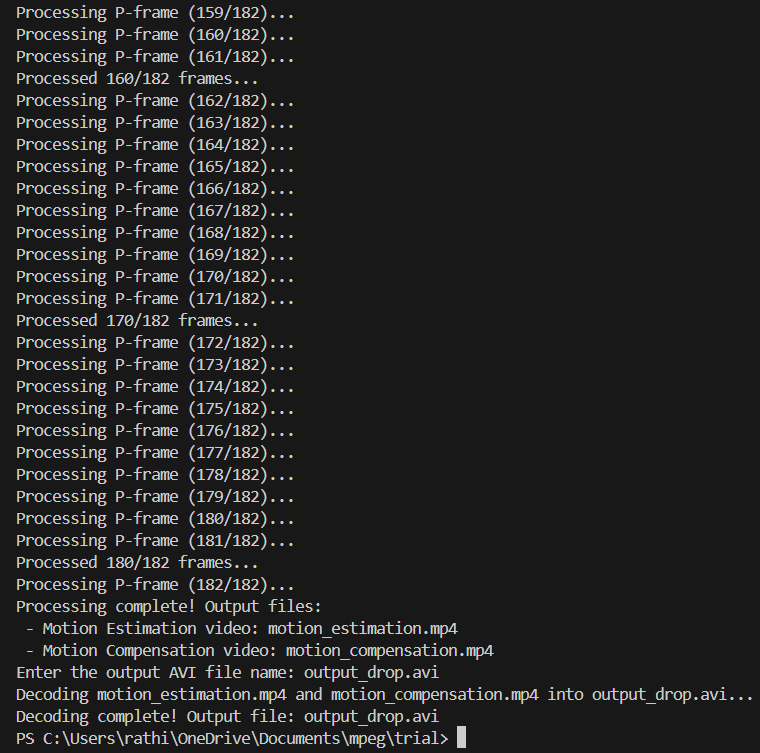
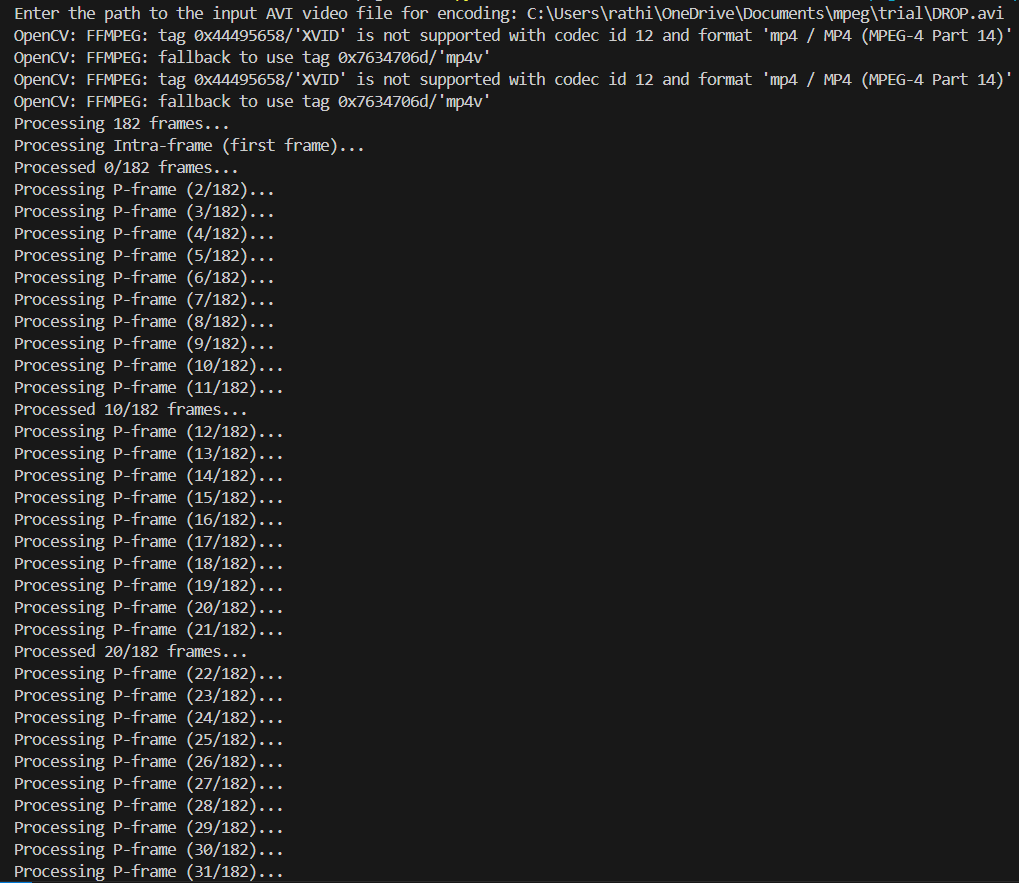
    output\_avi = input("Enter the output AVI file name: ").strip()

    process\_decoding(output\_motion\_estimation, output\_motion\_compensation, output\_avi)

**INPUT AND OUTPUT: INPUT FILE: INPUT.AVI**

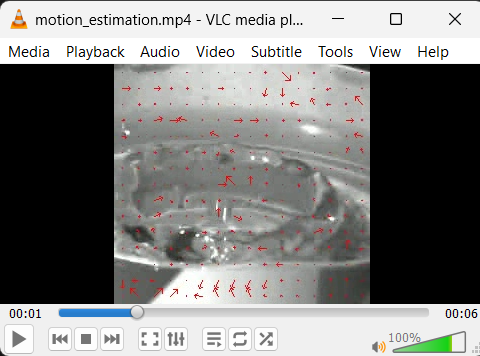
****

**OUTPUT:**

****

**ENCODING:**

**MOTION ESTIMATION:**

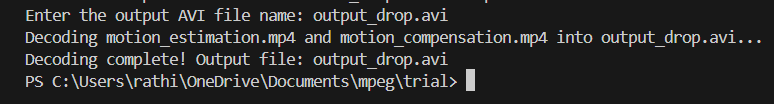
****

**MOTION COMPENSATION:**

****

**DECODING:**

****

****

**RESULT:**

The Python script performs H.261-like video encoding and decoding using motion estimation and motion compensation. Given an input AVI file, it encodes the video by extracting motion vectors and reconstructing frames, saving them into two MP4 files: one for motion estimation and the other for motion compensation. During decoding, the script takes these MP4 files, reconstructs the original video, and saves it as an AVI file. The process demonstrates motion-based video encoding and decoding.