21MIA1025 - Abhineswari M - CSE4076 - Lab Assignment 02

1. Setup and basic extraction

Objective: Install the necessary tools and libraries, and extract frame information from a video . Steps:

1. Install ffmpeg and ffmpeg-python:

o Install the ffmpeg tool and the ffmpeg-python libr avideo.

In [1]:
```
!pip install ffmpeg-python
```

```
Requirement already satisfied: ffmpeg-python in c:\users\abhineswari madan\anaconda\envs\tensorflowgpu\lib\site-packages (0.2.0)
Requirement already satisfied: future in c:\users\abhineswari madan\anaconda\envs\tensorflowgpu\lib\site-packages (from ffmpeg-python) (1.0.0)
```

In [2]:
```
import ffmpeg

# Path to the input video file
input_video = 'football.mp4'
print(input_video)
```

```
football.mp4
```

2. Extract Frame Information:

o Extract frame information from a sample video.

In [3]:
```
import subprocess
import json

ffprobe_executable = r'C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffprobe.exe'

# Run ffprobe to get video information
result = subprocess.run([ffprobe_executable, '-v', 'error', '-show_entries', 'stream=codec_name,width,height,r_frame_rate,nb_frames,duration', '-of', 'json', input_video],
                        stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

# Parse the JSON output
probe = json.loads(result.stdout)

# Print the entire JSON output to inspect its structure
print(json.dumps(probe, indent=4))

# Extract and print information about the video streams
for stream in probe.get('streams', []):
    # Determine if the stream is video by checking for video-specific keys
    if 'width' in stream and 'height' in stream:
        print(f"Codec: {stream.get('codec_name', 'N/A')}")
        print(f"Width: {stream.get('width', 'N/A')}")
        print(f"Height: {stream.get('height', 'N/A')}")
        print(f"Frame rate: {stream.get('r_frame_rate', 'N/A')}")
        print(f"Number of frames: {stream.get('nb_frames', 'N/A')}")
        print(f"Duration (seconds): {stream.get('duration', 'N/A')}")
        print("-------")
```


```
{
    "programs": [],
    "stream_groups": [],
    "streams": [
        {
            "codec_name": "h264",
            "width": 736,
            "height": 480,
            "r_frame_rate": "60000/1001",
            "duration": "14.601667",
            "nb_frames": "591"
        },
        {
            "codec_name": "aac",
            "r_frame_rate": "0/0",
            "duration": "14.651791",
            "nb_frames": "631"
        }
    ]
}
```


```
Codec: h264
Width: 736
Height: 480
Frame rate: 60000/1001
Number of frames: 591
Duration (seconds): 14.601667
-------
```

In [4]:
```
# Output pattern for frame images
output_pattern = 'frame_%04d.png'

# Path to the ffmpeg executable
ffmpeg_executable = r'C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffmpeg.exe'

# Run ffmpeg command to extract frames
subprocess.run([
    ffmpeg_executable,  # Use the full path to ffmpeg
    '-i', input_video,
    '-vf', 'fps=1',  # Adjust frame rate as needed
    output_pattern
], check=True)

print(f"Frames extracted and saved as {output_pattern}")
```

```
Frames extracted and saved as frame_%04d.png
```

In [5]:
```
output_pattern = r'C:\Users\Abhineswari Madan\Videos\frames\frame_%04d.png'
print(output_pattern)
```

```
C:\Users\Abhineswari Madan\Videos\frames\frame_%04d.png
```

2. Frame Type Analysis

Analyze the extracted frame information to understand the distribution of I,P and B frames in a video.

Steps:

1. Modify the Script:

o Count the number of I, P, and B frames. o Calculate the percentage of each frame type in the video.

In [6]:
```
import subprocess
import json
import matplotlib.pyplot as plt

# Paths to executables
ffprobe_executable = r'C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffprobe.exe'
ffmpeg_executable = r'C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffmpeg.exe'

# Path to the input video file
input_video = 'football.mp4'

# Run ffprobe to get frame type information
result = subprocess.run([ffprobe_executable, '-v', 'error', '-select_streams', 'v:0', '-show_entries', 'frame=pict_type', '-of', 'json', input_video],
                        stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

# Parse the JSON output
frames_info = json.loads(result.stdout)
frame_types = [frame['pict_type'] for frame in frames_info.get('frames', [])]

# Count occurrences of each frame type
frame_count = {'I': 0, 'P': 0, 'B': 0}
for frame_type in frame_types:
    if frame_type in frame_count:
        frame_count[frame_type] += 1

# Calculate total frames and percentages
total_frames = sum(frame_count.values())
frame_percentages = {ft: (count / total_frames) * 100 for ft, count in frame_count.items()}

# Print frame counts and percentages
print("Frame Counts:")
for frame_type, count in frame_count.items():
    print(f"{frame_type} frames: {count} ({frame_percentages[frame_type]:.2f}%)")
```

```
Frame Counts:
I frames: 15 (2.54%)
P frames: 576 (97.46%)
B frames: 0 (0.00%)
```

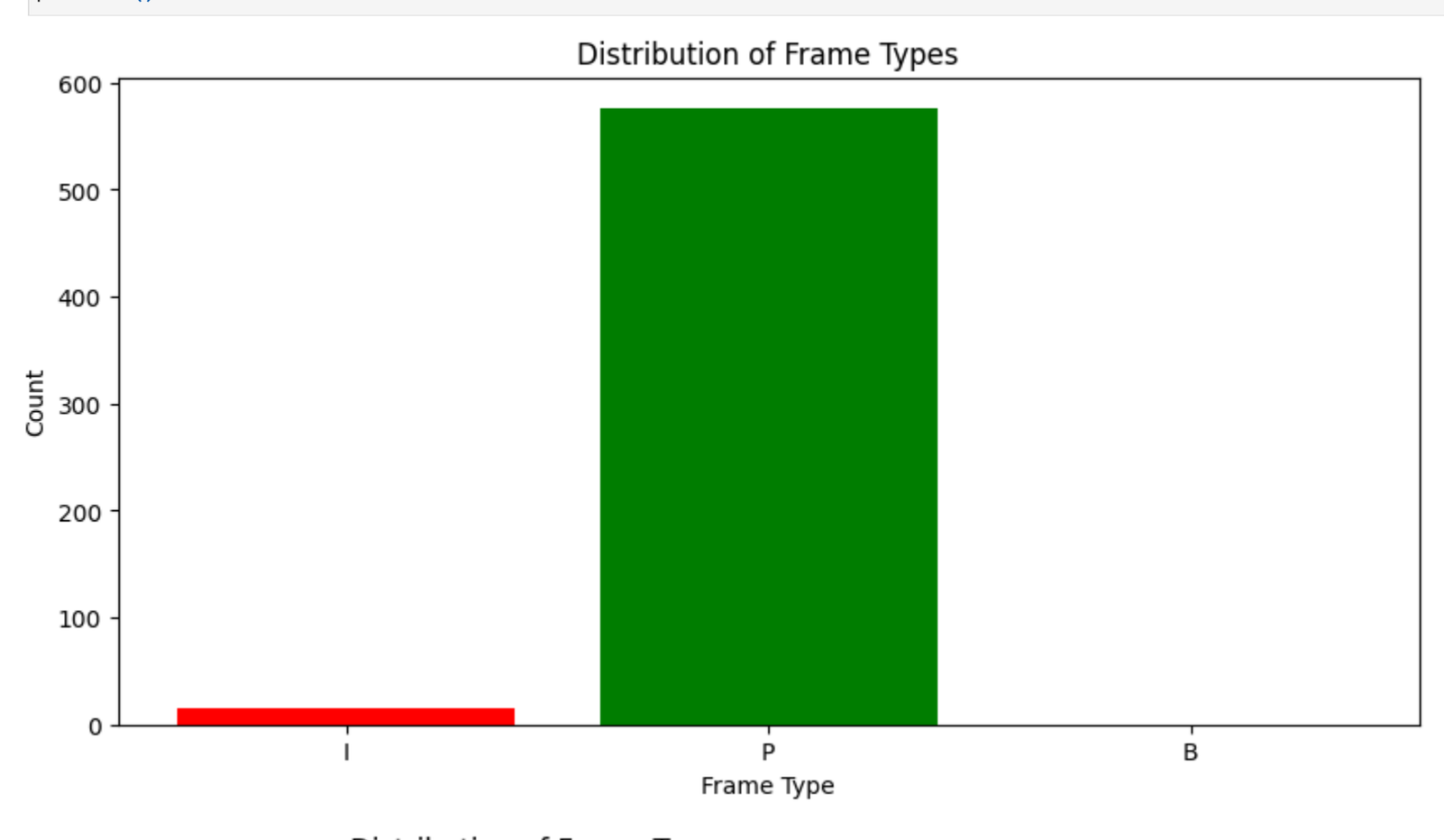
2. Analyze Frame Distribution:

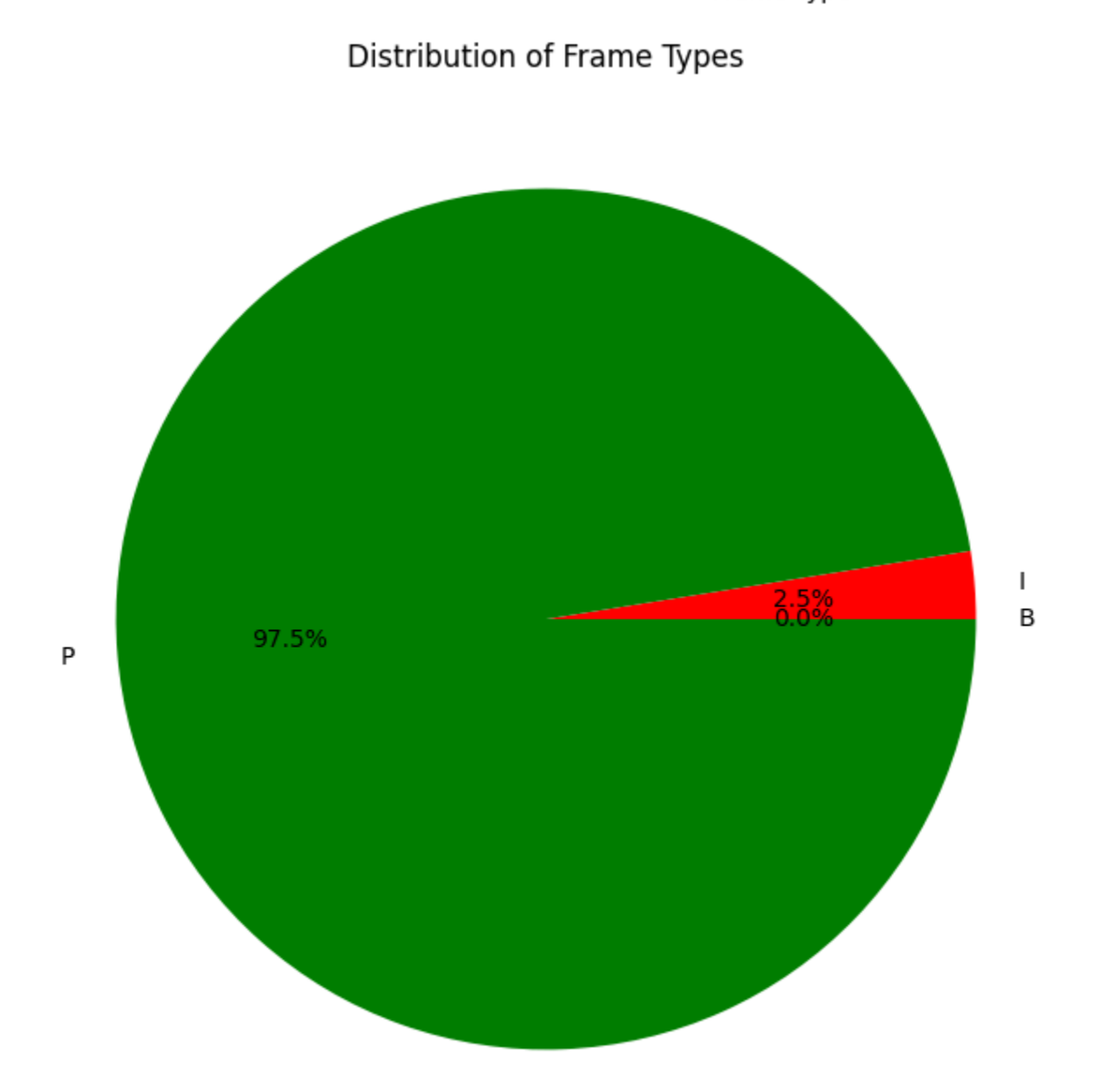
o Plot the distribution of frame types using a library like matplotlib. o Plot a pie chart or bar graph showing the distribution of frame types using matplotlib.

In [7]:
```
# Plot distribution of frame types
frame_labels = list(frame_count.keys())
frame_values = list(frame_count.values())

# Bar chart
plt.figure(figsize=(10, 5))
plt.bar(frame_labels, frame_values, color=['red', 'green', 'blue'])
plt.xlabel('Frame Type')
plt.ylabel('Count')
plt.title('Distribution of Frame Types')
plt.show()

# Pie chart
plt.figure(figsize=(8, 8))
plt.pie(frame_values, labels=frame_labels, autopct='%1.1f%%', colors=['red', 'green', 'blue'])
plt.title('Distribution of Frame Types')
plt.show()
```





3. Visualizing Frames

Objective: Extract actual frames from the video and display them using Python.

Steps:

1. Extract Frames:

o Use ffmpeg to extract individual I, P, and B frames from the video. o Save these frames as image files.

2. Display Frames:

o Use a library like PIL (Pillow) or opencv-python to display the extracted frames.

Tasks:

1. Save I, P, and B frames as separate image files using ffmpeg.
2. Use PIL or opencv-python to load and display these frames in a Python script.
3. Compare the visual quality of I, P, and B frames.

In [8]:
```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\abhineswari madan\anaconda\envs\tensorflowgpu\lib\site-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in c:\users\abhineswari madan\anaconda\envs\tensorflowgpu\lib\site-packages (from opencv-python) (1.26.4)
Note: you may need to restart the kernel to use updated packages.
```

In [16]:
```
import subprocess
import cv2
from PIL import Image

# Paths to executables
ffmpeg_executable = r'C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffmpeg.exe'

# Path to the input video file
input_video = 'football.mp4'

# Output patterns for frame images
output_patterns = {
    'I': 'I_frame_%04d.png',
    'P': 'P_frame_%04d.png',
    'B': 'B_frame_%04d.png'
}

# Command templates for extracting frames
commands = {
    'I': [ffmpeg_executable, '-i', input_video, '-vf', 'select=eq(pict_type\\,I)', '-vsync', 'vfr', output_patterns['I']],
    'P': [ffmpeg_executable, '-i', input_video, '-vf', 'select=eq(pict_type\\,P)', '-vsync', 'vfr', output_patterns['P']],
    'B': [ffmpeg_executable, '-i', input_video, '-vf', 'select=eq(pict_type\\,B)', '-vsync', 'vfr', output_patterns['B']]
}

# Extract frames
for frame_type, command in commands.items():
    print(f"Extracting {frame_type} frames with command: {' '.join(command)}")
    try:
        result = subprocess.run(command, check=True, stderr=subprocess.PIPE, text=True)
        print(f"{frame_type} frames extracted and saved.")
    except subprocess.CalledProcessError as e:
        print(f"Error extracting {frame_type} frames:")
        print(e.stderr)

# Function to display images using OpenCV
def display_image_opencv(image_path):
    img = cv2.imread(image_path)
    if img is not None:
        cv2.imshow(f'{image_path} - Frame', img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print(f"Error: Unable to read image {image_path}")

# Function to print image details
def print_image_details(image_path):
    img = cv2.imread(image_path)
    if img is not None:
        height, width, _ = img.shape
        print(f"{image_path} - Dimensions: {width}x{height}")
    else:
        print(f"Error: Unable to read image {image_path}")

# Example usage: Display the first I-frame, P-frame, and B-frame
example_frame_paths = {
    'I': 'I_frame_0001.png',
    'P': 'P_frame_0001.png',
    'B': 'B_frame_0001.png'
}

for frame_type, path in example_frame_paths.items():
    print(f"Displaying {frame_type} frame...")
    display_image_opencv(path)
    print_image_details(path)
```

```
Extracting I frames with command: C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffmpeg.exe -i football.mp4 -vf select=eq(pict_type\,I) -vsync vfr I_frame_%04d.png
I frames extracted and saved.
Extracting P frames with command: C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffmpeg.exe -i football.mp4 -vf select=eq(pict_type\,P) -vsync vfr P_frame_%04d.png
P frames extracted and saved.
Extracting B frames with command: C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffmpeg.exe -i football.mp4 -vf select=eq(pict_type\,B) -vsync vfr B_frame_%04d.png
B frames extracted and saved.
Displaying I frame...
I_frame_0001.png - Dimensions: 736x480
Displaying P frame...
P_frame_0001.png - Dimensions: 736x480
Displaying B frame...
Error: Unable to read image B_frame_0001.png
Error: Unable to read image B_frame_0001.png
```

Must note that not all videos contain a significant number of B-frames. B-frames are sometimes not present in all the videos, especially if the codec settings or the video encoding profile doesnt use them extensively.

4. Frame Compression Analysis

Objective: Analyze the compression efficiency of I, P, and B frames.

Steps:

1. Calculate Frame Sizes:

o Calculate the file sizes of extracted I, P, and B frames. o Compare the average file sizes of each frame type.

In [10]:
```
import os
import glob

# Define the patterns for the frames
patterns = {
    'I': 'I_frame_*.png',
    'P': 'P_frame_*.png',
    'B': 'B_frame_*.png'
}

# Function to calculate average file size for a given pattern
def calculate_average_size(pattern):
    files = glob.glob(pattern)
    sizes = [os.path.getsize(file) for file in files]
    if sizes:
        return sum(sizes) / len(sizes)
    return 0

# Calculate average sizes for I, P, and B frames
average_sizes = {frame_type: calculate_average_size(pattern) for frame_type, pattern in patterns.items()}

# Print the results
print("Average Frame Sizes (in bytes):")
for frame_type, size in average_sizes.items():
    print(f"{frame_type} Frames: {size:.2f} bytes")
```

```
Average Frame Sizes (in bytes):
I Frames: 452012.07 bytes
P Frames: 507658.36 bytes
B Frames: 0.00 bytes
```

2. Compression Efficiency:

o Discuss the role of each frame type in video compression. o Analyze why P and B frames are generally smaller than I frames.

Types of Frames:

I Frames (Intra-coded frames): These frames are essentially full images and are used as reference frames for P and B frames. They are larger in size because they contain more data, representing complete picture information.

P Frames (Predictive-coded frames): These frames store only the differences from the previous frame (I or P). They are smaller than I frames because they encode only changes, not the full image.

B Frames (Bi-directional predictive-coded frames): These frames use both previous and future frames for prediction, allowing even more efficient compression. They are usually smaller than both I and P frames.

Why I Frames Are Larger?

Full Image Data: I frames are keyframes that contain complete image data without reference to other frames, making them inherently larger.

Data Redundancy: P and B frames reduce redundancy by encoding differences or using data from surrounding frames, which reduces their size.

Why P and B Frames Are Smaller?

Prediction-Based Compression: P frames use past frames to predict data, while B frames use both past and future frames, leading to reduced data size due to the high level of data prediction and compression.

Temporal Redundancy: The temporal redundancy between consecutive frames is exploited in P and B frames, which compresses the data more efficiently compared to I frames.

5: Advanced Frame Extraction

Objective: Extract frames from a video and reconstruct a part of the video using only I frames.

Steps:

1. Extract and Save I Frames:

o Extract I frames from the video and save them as separate image files.

In [11]:
```
import subprocess
import glob
import os

# Paths and settings
input_video = 'football.mp4'
ffmpeg_executable = r'C:\Users\Abhineswari Madan\Downloads\ffmpeg-build\ffmpeg-full_build\bin\ffmpeg.exe'
output_frames_pattern = 'I_frame_%04d.png'
reconstructed_video = 'I_frames_reconstructed.mp4'
frame_rate = 5  # Reduced frame rate for the new video

# Step 1: Extract and Save I Frames
def extract_I_frames():
    command = [
        ffmpeg_executable,
        '-i', input_video,
        '-vf', 'select=eq(pict_type\\,I)',
        '-vsync', 'vfr',
        output_frames_pattern
    ]
    subprocess.run(command, check=True)
    print("I frames extracted and saved.")

# Step 2: Reconstruct Video from I Frames
def reconstruct_video_from_frames():
    # Create a list of the extracted I frames
    frames = sorted(glob.glob(output_frames_pattern))

    if not frames:
        print("No I frames found for reconstruction.")
        return

    # Build the FFmpeg command to create a new video from the frames
    command = [
        ffmpeg_executable,
        '-framerate', str(frame_rate),
        '-i', output_frames_pattern,
        '-c:v', 'libx264',
        '-r', str(frame_rate),
        '-pix_fmt', 'yuv420p',
        reconstructed_video
    ]

    subprocess.run(command, check=True)
    print(f"Reconstructed video saved as {reconstructed_video}.")
```

In [12]:
```
# Execute the extraction and reconstruction
extract_I_frames()
reconstruct_video_from_frames()
```

```
I frames extracted and saved.
No I frames found for reconstruction.
```

In [13]:
```
import os

# Directory where I frames are saved
frames_directory = 'C:/Users/Abhineswari Madan'  # Update this path

# Check if directory exists
if not os.path.exists(frames_directory):
    print(f"Directory does not exist: {frames_directory}")
else:
    # List files in the directory
    frames_files = [f for f in os.listdir(frames_directory) if f.startswith('I_frame_') and f.endswith('.png')]

    # Print out found files
    print("I frames found:")
    for frame_file in frames_files:
        print(frame_file)

    # Check if any files were found
    if not frames_files:
        print("No I frames found for reconstruction.")
    else:
        # Proceed with reconstruction
        pass
```

```
I frames found:
I_frame_0001.png
I_frame_0002.png
I_frame_0003.png
I_frame_0004.png
I_frame_0005.png
I_frame_0006.png
I_frame_0007.png
I_frame_0008.png
I_frame_0009.png
I_frame_0010.png
I_frame_0011.png
I_frame_0012.png
I_frame_0013.png
I_frame_0014.png
I_frame_0015.png
```

2. Reconstruct Video:

o Use the extracted I frames to reconstruct a portion of the video. o Create a new video using these I frames with a reduced frame rate.

In [14]:
```
import cv2
import os

# Directory where I frames are saved
frames_directory = 'C:/Users/Abhineswari Madan'
output_video_path = 'C:/Users/Abhineswari Madan/reconstructed_video.mp4'  # Update this path

# Desired frame rate for the new video
frame_rate = 10  # Adjust as needed

# Get list of I frames sorted by file name
frames_files = sorted([f for f in os.listdir(frames_directory) if f.startswith('I_frame_') and f.endswith('.png')])

# Check if there are I frames
if not frames_files:
    print("No I frames found for reconstruction.")
else:
    # Load the first frame to get the width and height
    first_frame_path = os.path.join(frames_directory, frames_files[0])
    first_frame = cv2.imread(first_frame_path)

    if first_frame is None:
        print(f"Error loading the first frame: {first_frame_path}")
    else:
        height, width, _ = first_frame.shape

        # Create a video writer object
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec for mp4 files
        video_writer = cv2.VideoWriter(output_video_path, fourcc, frame_rate, (width, height))

        # Write frames to the video file
        for frame_file in frames_files:
            frame_path = os.path.join(frames_directory, frame_file)
            frame = cv2.imread(frame_path)

            if frame is None:
                print(f"Error loading frame: {frame_path}")
            else:
                video_writer.write(frame)

        # Release the video writer object
        video_writer.release()
        print(f"Reconstructed video saved to: {output_video_path}")
```

```
Reconstructed video saved to: C:/Users/Abhineswari Madan/reconstructed_video.mp4
```

Frame rate information comparison

In [15]:
```
import cv2
import os

# Paths
input_video_path = 'football.mp4'  # Path to the original video
frames_directory = 'C:/Users/Abhineswari Madan'  # Path where I frames are saved
output_video_path = 'C:/Users/Abhineswari Madan/reconstructed_video.mp4'  # Path for the new video

# Desired frame rate for the new video
reduced_frame_rate = 10  # Adjust as needed

# Get the original frame rate
def get_original_frame_rate(video_path):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        raise IOError(f"Error opening video file: {video_path}")
    fps = cap.get(cv2.CAP_PROP_FPS)
    cap.release()
    return fps

# Get original frame rate
try:
    original_frame_rate = get_original_frame_rate(input_video_path)
    print(f"Original frame rate: {original_frame_rate} fps")
except Exception as e:
    print(f"Error: {e}")

# Get list of I frames sorted by file name
frames_files = sorted([f for f in os.listdir(frames_directory) if f.startswith('I_frame_') and f.endswith('.png')])

# Check if there are I frames
if not frames_files:
    print("No I frames found for reconstruction.")
else:
    # Load the first frame to get the width and height
    first_frame_path = os.path.join(frames_directory, frames_files[0])
    first_frame = cv2.imread(first_frame_path)

    if first_frame is None:
        print(f"Error loading the first frame: {first_frame_path}")
    else:
        height, width, _ = first_frame.shape

        # Create a video writer object
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec for mp4 files
        video_writer = cv2.VideoWriter(output_video_path, fourcc, reduced_frame_rate, (width, height))

        # Write frames to the video file
        for frame_file in frames_files:
            frame_path = os.path.join(frames_directory, frame_file)
            frame = cv2.imread(frame_path)

            if frame is None:
                print(f"Error loading frame: {frame_path}")
            else:
                video_writer.write(frame)

        # Release the video writer object
        video_writer.release()
        print(f"Reconstructed video saved to: {output_video_path}")
        print(f"Reduced frame rate: {reduced_frame_rate} fps")
```

```
Original frame rate: 40.47483164022372 fps
Reconstructed video saved to: C:/Users/Abhineswari Madan/reconstructed_video.mp4
Reduced frame rate: 10 fps
```

In [ ]:

In [ ]: