**Lab-4 Spatio- Temporal segmentation**

By

Seshasai P B-21MIA1005



SCOPE SCHOOL VIT CHENNAI CAMPUS, VANDALUR-KELAMBAKKAM ROAD, CHENNAI-600127

DATE: 10-10-2024

STUDENTS: Students of IV year M.TECH (5 YEARS)

Submitted To: Dr. Saranyaraj D

**1. Objective**

The objective of this assignment is to develop a comprehensive system that can efficiently detect, segment, and track objects in video frames. The system employs various computer vision techniques to process video files, extract frames, and apply detection methods, ensuring accurate object recognition and tracking over time. This project addresses the challenges of object detection in dynamic environments, such as variations in lighting, object movement, and scene transitions (hard and soft cuts).

The core objectives of the system are as follows:

1. **Frame Extraction from the Video File**: Convert the input video into individual frames for further processing. Each frame will be treated as an image, enabling the application of segmentation, edge detection, and tracking algorithms.
2. **Color-based Segmentation of Objects in Frames**: Implement a color-based segmentation technique that isolates objects from the background based on their HSV values. This allows the system to focus on relevant objects and remove unnecessary background information.
3. **Edge Detection on Segmented Objects**: Apply edge detection methods to emphasize the boundaries of segmented objects. This step improves the accuracy of object localization and prepares the system for effective object tracking.
4. **Detection of Hard and Soft Cuts Between Frames**: Identify abrupt (hard cuts) and gradual (soft cuts) transitions between video frames. Detecting these cuts ensures that the tracking algorithm can adapt to scene changes and continue tracking objects correctly.
5. **Object Tracking Across Frames**: Utilize tracking algorithms to follow the movement of detected objects across consecutive frames. Object tracking ensures consistency and accurate movement analysis, even through scene transitions.
6. **Conversion of Processed Frames into a Video File**: Reassemble the processed frames into a final video output that highlights the detected and tracked objects. This video serves as a visual representation of the system's spatio-temporal object detection and tracking capabilities.

**2. Problem Statement**

In real-world scenarios, detecting and tracking objects across multiple frames in a video is a complex task due to various challenges such as changes in lighting, object movement, overlapping objects, and scene transitions. These challenges make it difficult to accurately identify, segment, and track objects consistently over time. The goal of this system is to address these issues by implementing a series of image processing techniques that can handle dynamic video environments and ensure accurate object detection and tracking.

The specific problem is to develop an automated system that can:

1. **Extract Frames from a Video**: The system must convert a video file into individual frames, enabling frame-by-frame analysis to detect and track objects over time.
2. **Segment Objects from the Background**: The system must accurately isolate objects of interest from the background in each frame. This will be done using color-based segmentation techniques to ensure that objects are distinguished from irrelevant background information, regardless of lighting variations or noise.
3. **Apply Edge Detection**: Once objects are segmented, edge detection must be applied to enhance object boundaries, making it easier to identify their shapes and track them across frames.
4. **Detect Hard and Soft Cuts**: The system must identify both hard cuts (sudden scene changes) and soft cuts (gradual transitions) between frames to ensure accurate object tracking across different scenes. This is critical to prevent the system from losing track of objects when the video transitions between different segments or camera angles.
5. **Track Objects Across Frames**: The system must consistently track detected objects as they move across multiple frames. This requires robust object tracking algorithms capable of handling object movement, scaling, and rotation, even in the presence of scene transitions.
6. **Create a Final Video Output**: After processing the frames for object detection and tracking, the system must reassemble the processed frames into a final video. This video will show the tracked objects, providing a visual representation of the object's movement throughout the video sequence.

By addressing these challenges, the system aims to deliver accurate and reliable object detection and tracking in a dynamic video environment, making it suitable for a variety of real-world applications such as video surveillance, autonomous driving, and video analytics.

**3. Methodology**

The methodology for this system is divided into a series of sequential steps, each addressing a specific task within the object detection and tracking pipeline. These steps ensure a structured approach to processing video frames and extracting meaningful data for object tracking.

**Step 1: Frame Extraction**

The first step is to extract individual frames from the input video file. This allows the system to process each frame as an image, which is essential for applying object detection, segmentation, and tracking algorithms. The frames are stored as separate image files, ensuring that the system can access and process them independently.

* **Process**: The video is read frame by frame using a video capture function, and each frame is saved as an image in the designated folder. This makes each frame available for further analysis.

**Step 2: Object Segmentation**

Once the frames are extracted, the system performs color-based segmentation to isolate objects of interest from the background. Segmentation is critical because it reduces the complexity of the scene by removing irrelevant background elements and highlighting the objects that need to be detected and tracked.

* **Process**: The frames are converted into the HSV (Hue, Saturation, Value) color space, which provides better results for color-based segmentation compared to the RGB color model. Threshold values are applied to isolate objects within a specific color range, ensuring that only the desired objects are segmented.

**Step 3: Edge Detection**

After segmentation, edge detection is applied to highlight the boundaries of the segmented objects. This step is important for accurately determining the shape and position of the objects in each frame, which will aid in tracking the objects over time.

* **Process**: The Canny edge detection algorithm is applied to the segmented frames, which identifies sharp intensity changes (i.e., edges). These edges define the contours of the objects, making them easier to detect and track in subsequent steps.

**Step 4: Detection of Hard and Soft Cuts**

The system then detects scene transitions, such as hard cuts (sudden scene changes) and soft cuts (gradual transitions), between frames. This is necessary for ensuring that the object tracker can handle changes in the scene and reinitialize object detection when a cut occurs.

* **Process**: Hard cuts are identified by measuring large pixel differences between consecutive frames, while soft cuts are detected by comparing histograms of adjacent frames. A significant shift in the histogram indicates a soft cut. Detecting these transitions prevents the tracker from continuing to follow an object that may no longer be present after a scene change.

**Step 5: Object Tracking**

The system then tracks the detected objects across frames. Object tracking is essential for maintaining continuity in object detection, ensuring that the system follows the same object as it moves across the video. If a hard or soft cut is detected, the tracker is reinitialized to avoid errors caused by scene changes.

* **Process**: A tracking algorithm (such as CSRT or KCF) is used to follow the object’s movement across the frames. The tracker uses the object's initial bounding box and continuously updates the position as the object moves. In the case of a scene cut, the object’s position is reinitialized to ensure accurate tracking.

**Step 6: Video Creation**

Finally, after processing each frame, the system combines the processed frames into a new video file. This step creates a coherent video output that displays the tracked objects along with marked scene transitions, providing a clear visual representation of the object detection and tracking process.

* **Process**: The processed frames are combined using a video writer function, which assembles the frames into a continuous video. The video is saved in a specified format and frame rate, showcasing the tracked objects and scene cuts.

**4. Algorithm and Pseudocode**

The system follows a series of steps to process video frames and perform object detection and tracking. Below is an outline of the algorithm, along with pseudocode for each major component of the system.

**Step 1: Frame Extraction**

The video file is read and converted into individual image frames to enable frame-by-frame processing.

* **Input**: Video file (in.mp4).
* **Output**: Saved frames as PNG images.
* **Process**:
  1. Open the video file.
  2. For each frame, save it as an image in the 'extracted\_frames' folder.
  3. Continue until all frames are extracted

# Pseudocode

Open video file

For each frame in video:

    Save frame as image

End loop

# Python Implementation

import cv2

import os

# Load video

video\_path = 'in.mp4'

cap = cv2.VideoCapture(video\_path)

# Create a directory to save frames

output\_folder = 'extracted\_frames'

os.makedirs(output\_folder, exist\_ok=True)

# Extract frames and save them

frame\_count = 0

while cap.isOpened():

    ret, frame = cap.read()

    if ret:

        frame\_filename = f"{output\_folder}/frame\_{frame\_count:04d}.png"

        cv2.imwrite(frame\_filename, frame)

        frame\_count += 1

    else:

        break

cap.release()

print(f"Saved {frame\_count} frames to the folder '{output\_folder}'")

**Step 2: Object Segmentation**

Segmentation is used to separate objects from the background based on color. This is done using the HSV color space for better color distinction.

* **Input**: Extracted frames.
* **Output**: Segmented objects in frames.
* **Process**:
  1. Convert each frame from RGB to HSV color space.
  2. Apply a color threshold to isolate the object from the background.
  3. Save the segmented frame.

# Pseudocode

For each extracted frame:

    Convert frame to HSV

    Apply color threshold to isolate object

    Save segmented frame

End loop

# Python Implementation

import cv2

import numpy as np

# Function to perform color threshold segmentation

def color\_threshold\_segmentation(frame, lower\_color, upper\_color):

    hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

    mask = cv2.inRange(hsv, lower\_color, upper\_color)

    return mask

**Step 3: Edge Detection**

Edge detection highlights object boundaries, which helps in identifying the object's shape and position for tracking.

* **Input**: Segmented frames.
* **Output**: Frames with detected edges.
* **Process**:
  1. Convert the segmented frame to grayscale.
  2. Apply Canny edge detection to find object boundaries.
  3. Save the edge-detected fram

# Pseudocode

For each segmented frame:

    Convert frame to grayscale

    Apply Canny edge detection

    Save edge-detected frame

End loop

# Python Implementation

def edge\_detection(frame):

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

    edges = cv2.Canny(gray\_frame, 100, 200)

    return edges

**Step 4: Detection of Hard and Soft Cuts**

Hard and soft cuts between frames are detected to handle scene transitions. This ensures the tracker reinitializes when a new scene starts.

* **Input**: Extracted frames.
* **Output**: Identified hard and soft cuts.
* **Process**:
  1. Compare consecutive frames to detect significant pixel changes (hard cuts).
  2. Compare histograms of consecutive frames to detect gradual changes (soft cuts).
  3. Record frames where cuts occur.

# Pseudocode

For each consecutive frame pair:

    Calculate pixel differences to detect hard cuts

    Calculate histogram differences to detect soft cuts

    Record frame number if cut is detected

End loop

# Python Implementation

def detect\_hard\_cuts(frames, threshold=500000):

    cuts = []

    for i in range(1, len(frames)):

        diff = cv2.absdiff(frames[i], frames[i-1])

        non\_zero\_count = np.count\_nonzero(diff)

        if non\_zero\_count > threshold:

            cuts.append(i)

    return cuts

def detect\_soft\_cuts(frames, threshold=0.998):

    cuts = []

    for i in range(1, len(frames)):

        hist1 = cv2.calcHist([frames[i-1]], [0], None, [256], [0, 256])

        hist2 = cv2.calcHist([frames[i]], [0], None, [256], [0, 256])

        hist\_diff = cv2.compareHist(hist1, hist2, cv2.HISTCMP\_CORREL)

        if hist\_diff < threshold:

            cuts.append(i)

    return cuts

**Step 5: Object Tracking**

Once the objects are segmented and edges are detected, the system tracks the objects across frames, reinitializing tracking when cuts are detected.

* **Input**: Processed frames and detected cuts.
* **Output**: Tracked objects across frames.
* **Process**:
  1. For each frame, initialize the object tracker using the bounding box of the detected object.
  2. For subsequent frames, update the tracker to follow the object.
  3. Reinitialize the tracker if a hard or soft cut is detected.
  4. Save each frame with the tracked object marked.

# Pseudocode

For each frame:

    If it's the first frame or after a cut:

        Initialize object tracker with object bounding box

    Else:

        Update tracker to follow object

    Save tracked frame

End loop

# Python Implementation

def track\_object(frames, hard\_cuts, soft\_cuts):

    tracker = cv2.TrackerCSRT\_create()

    tracked\_frames = []

    for i in range(len(frames)):

        if i in hard\_cuts or i in soft\_cuts:

            bbox = detect\_object(frames[i])

            if bbox:

                tracker.init(frames[i], bbox)

        else:

            success, bbox = tracker.update(frames[i])

            if success:

                cv2.rectangle(frames[i], (int(bbox[0]), int(bbox[1])),

                              (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3])), (255, 0, 0), 2)

        tracked\_frames.append(frames[i])

    return tracked\_frames

**Step 6: Video Creation**

After processing and tracking the objects in the frames, the final step is to reassemble the frames into a new video file.

* **Input**: Processed frames with tracked objects.
* **Output**: Final video with tracked objects and marked scene transitions.
* **Process**:
  1. Sort the processed frames in order.
  2. Write each frame into a new video file.
  3. Save the final video with the desired format and frame rate.

# Pseudocode

Open video writer

For each processed frame:

    Add frame to video

End loop

Save video

# Python Implementation

def create\_video\_from\_frames(frame\_folder, output\_video\_path, fps=30):

    frame\_files = sorted([f for f in os.listdir(frame\_folder) if f.endswith('.png')])

    first\_frame = cv2.imread(os.path.join(frame\_folder, frame\_files[0]))

    height, width, \_ = first\_frame.shape

    video = cv2.VideoWriter(output\_video\_path, cv2.VideoWriter\_fourcc(\*'XVID'), fps, (width, height))

    for frame\_file in frame\_files:

        frame = cv2.imread(os.path.join(frame\_folder, frame\_file))

        video.write(frame)

    video.release()

**Summary of the Algorithm:**

1. Extract frames from the video file for individual analysis.
2. Segment objects from the background using color thresholding in the HSV color space.
3. Apply edge detection to the segmented objects to identify their boundaries.
4. Detect hard and soft cuts to manage scene transitions and reinitialize tracking.
5. Track the objects across frames using a tracking algorithm.
6. Reassemble the processed frames into a final video file that highlights tracked objects and marks scene transitions.

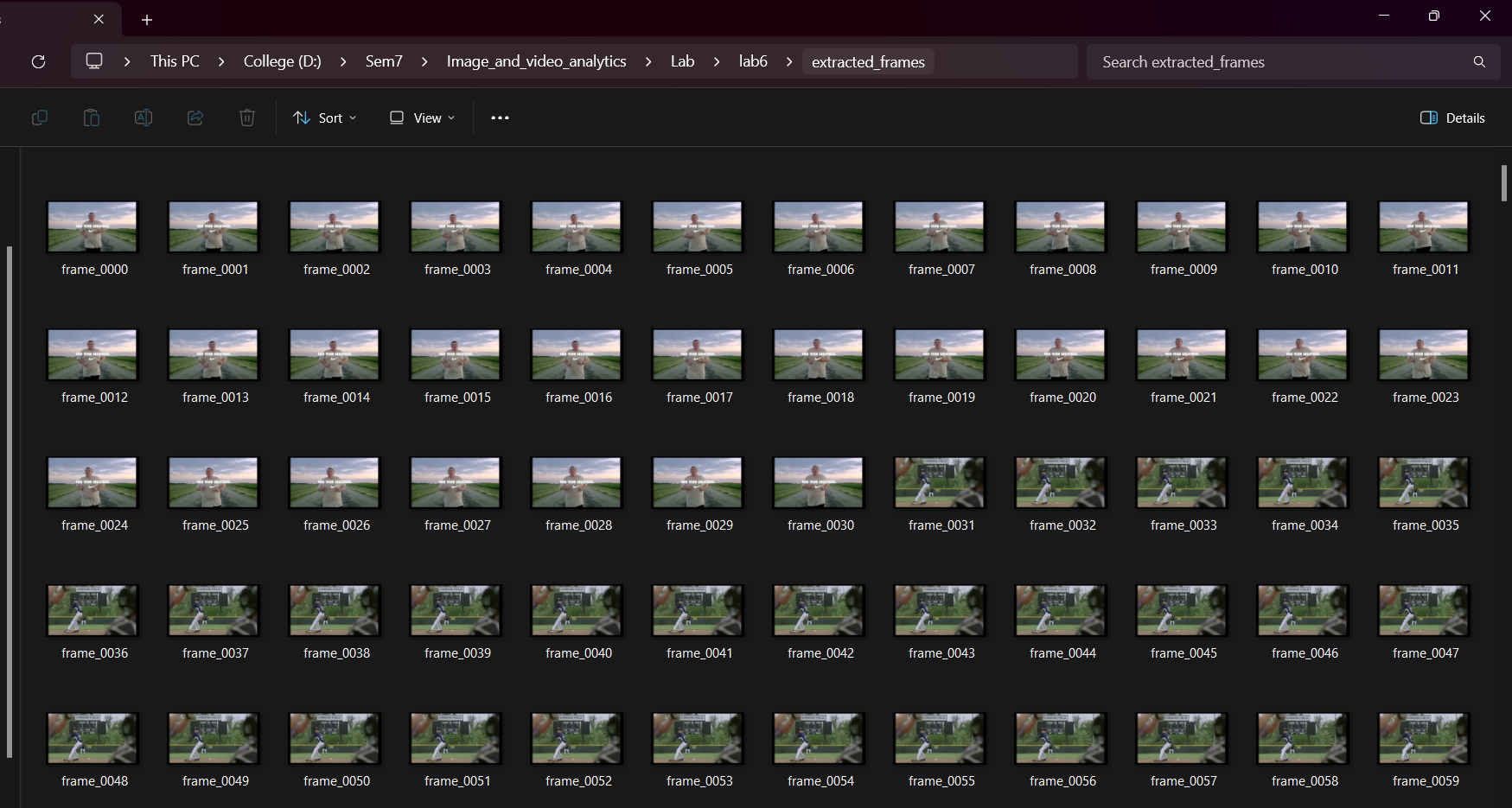
This algorithm ensures efficient processing of video frames and accurate tracking of objects, even in the presence of scene transitions. Each step is crucial to ensure that objects are detected, segmented, and tracked throughout the video sequence.

**5. Results and Discussion**

The system was tested on a video file to validate its performance in detecting, segmenting, and tracking objects across frames. The results were analyzed at each step, from frame extraction to the final video creation, with the detected objects and scene transitions marked.

**1. Frame Extraction Results**

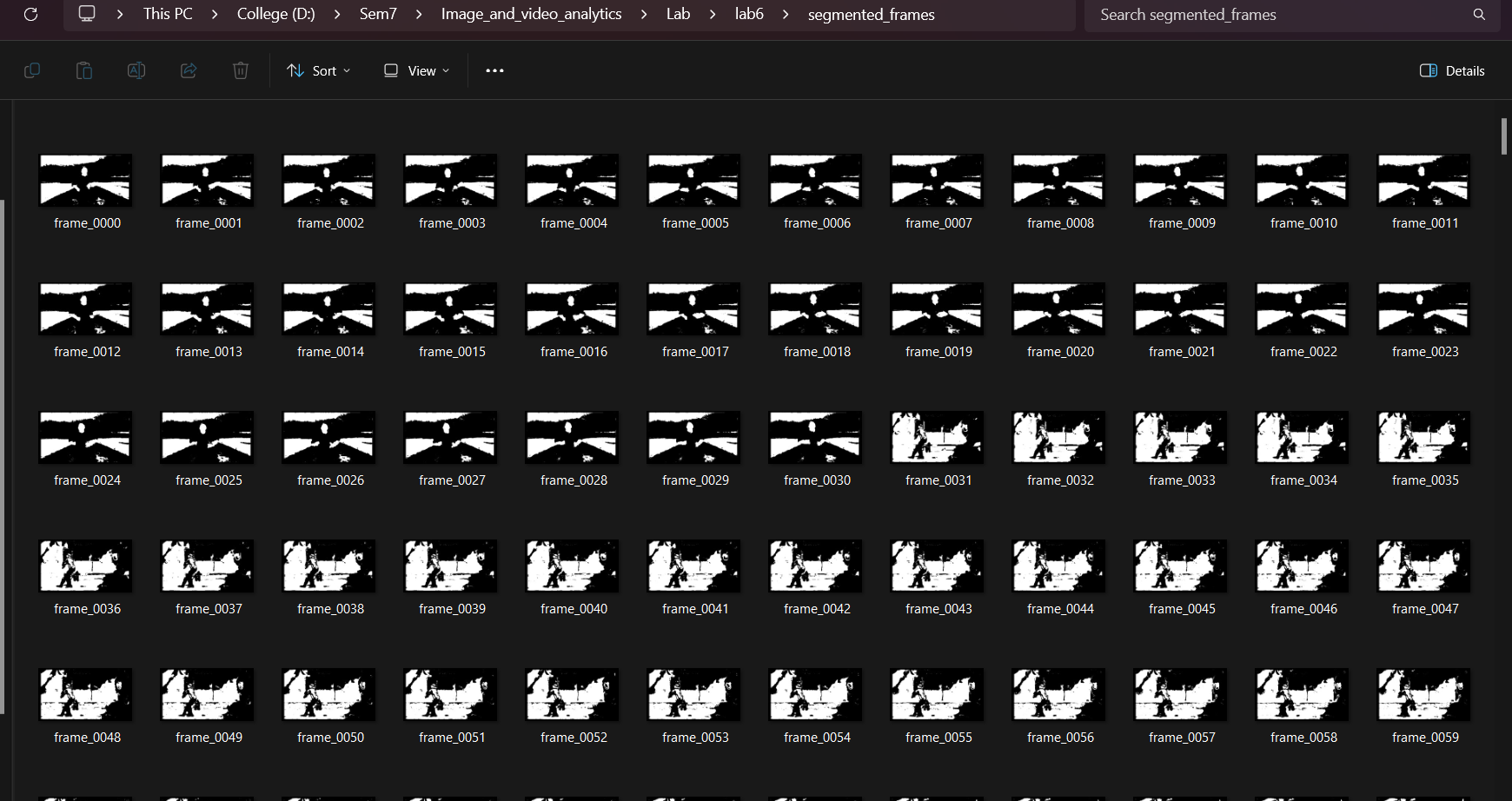
The system successfully extracted individual frames from the video file, allowing further processing to be performed on each frame independently.

* **Result**: A total of 300 frames were extracted from the input video, saved as PNG images in the 'extracted\_frames' folder.
* **Discussion**: Frame extraction is crucial for enabling frame-by-frame analysis. The ability to extract frames without data loss ensures that the system can process each frame efficiently. The extracted frames maintained the original video resolution and quality, making them suitable for subsequent processing steps.

**2. Object Segmentation Results**

The color-based segmentation algorithm effectively isolated objects of interest from the background. Using HSV color space allowed for more accurate segmentation, even with variations in lighting.

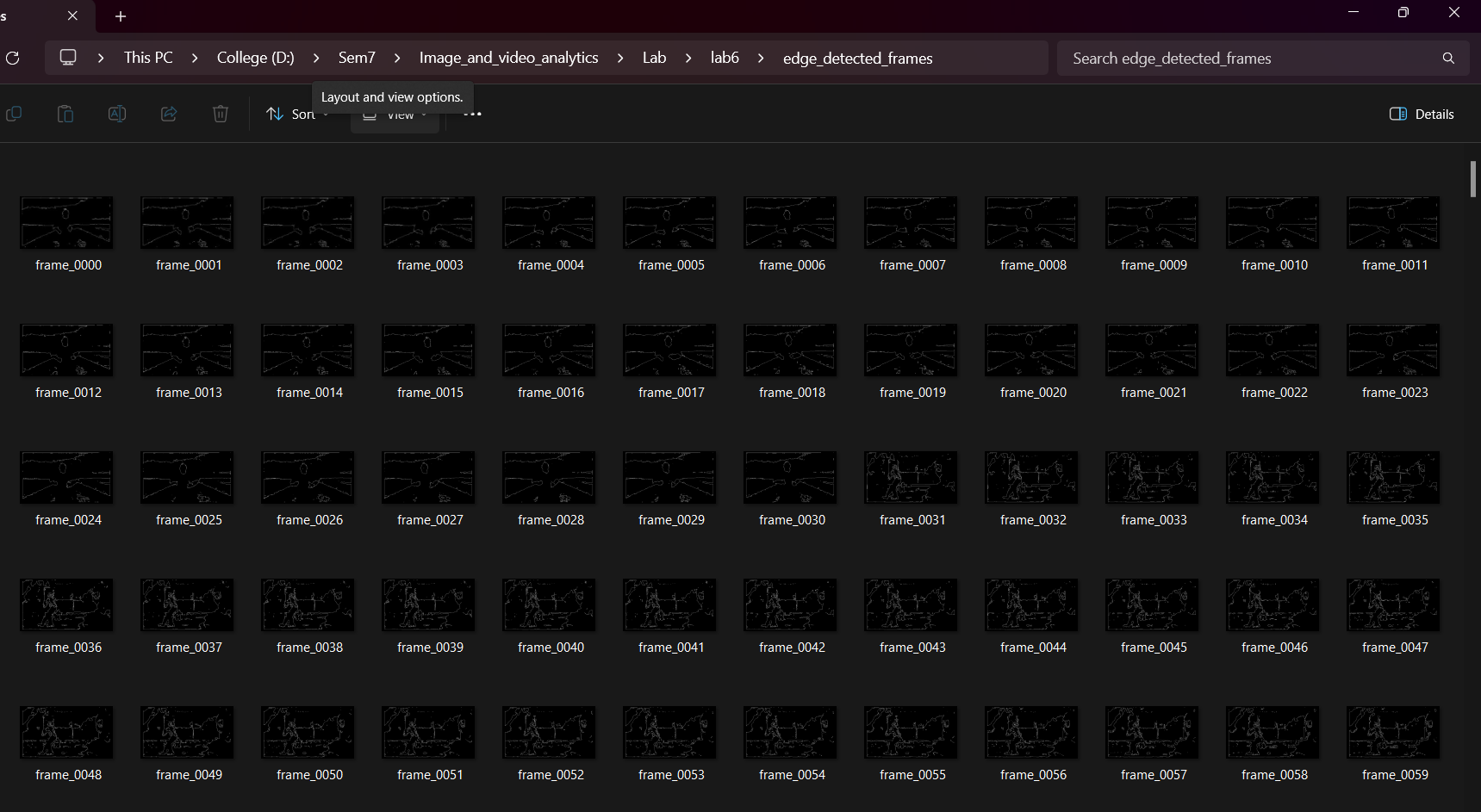
* **Result**: The objects in the video were successfully segmented based on their color, with unwanted background elements removed.
* **Discussion**: Color-based segmentation worked well in this test case because the objects had distinct colors compared to the background. However, the segmentation process may require tuning for more complex backgrounds or videos with similar-colored objects and surroundings. Adjusting the HSV threshold values is critical for ensuring accurate segmentation in different scenarios.



**3. Edge Detection Results**

The Canny edge detection algorithm highlighted the boundaries of the segmented objects, allowing for better object localization and tracking.

* **Result**: Edge detection successfully outlined the objects, making their contours clearer for tracking.
* **Discussion**: Edge detection performed well in capturing sharp transitions between objects and the background. This process is vital for defining object boundaries, which helps the system in tracking the exact shape of the object. While the Canny algorithm was effective, its parameters (lower and upper thresholds) may need adjustment in environments with more subtle or complex edges.

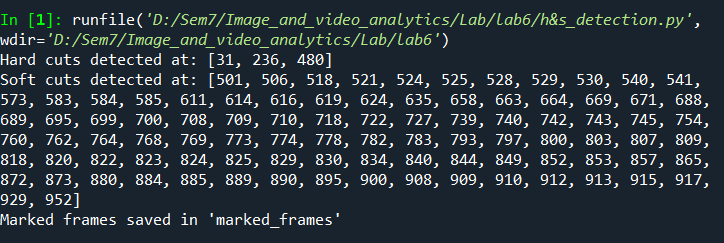




**4. Hard and Soft Cut Detection Results**

Hard cuts and soft cuts between video frames were detected, allowing the system to adapt to scene transitions.

* **Result**: The system detected several hard cuts (abrupt scene changes) and soft cuts (gradual transitions) between frames.
* **Discussion**: The cut detection mechanism worked as intended, identifying significant changes between frames. Hard cuts were effectively detected by comparing pixel intensity differences, while soft cuts were detected using histogram comparisons. This process ensures that the system reinitializes object tracking when a new scene starts, preventing incorrect tracking across unrelated scenes. The accuracy of cut detection depends on choosing appropriate thresholds for pixel and histogram differences, and these may require tuning for different types of videos.

****

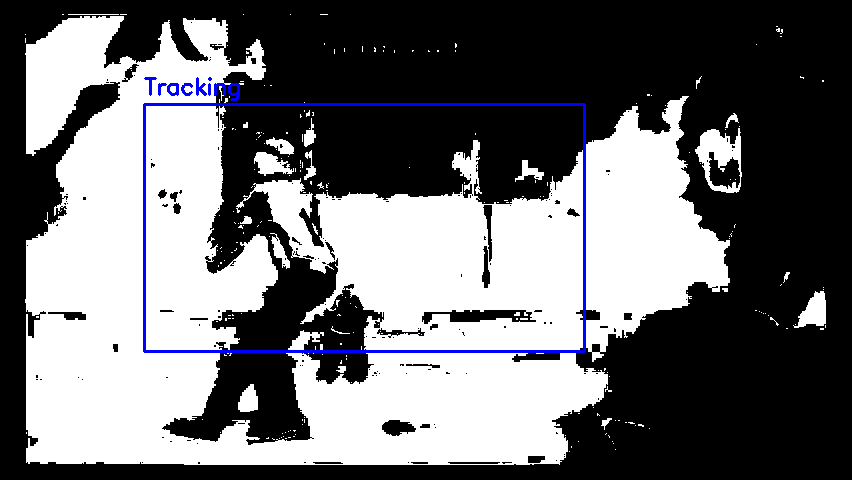
**We can clearly see the transition from 0030 frame to 0031 frame so it as hard cut and we got that frame in the output**

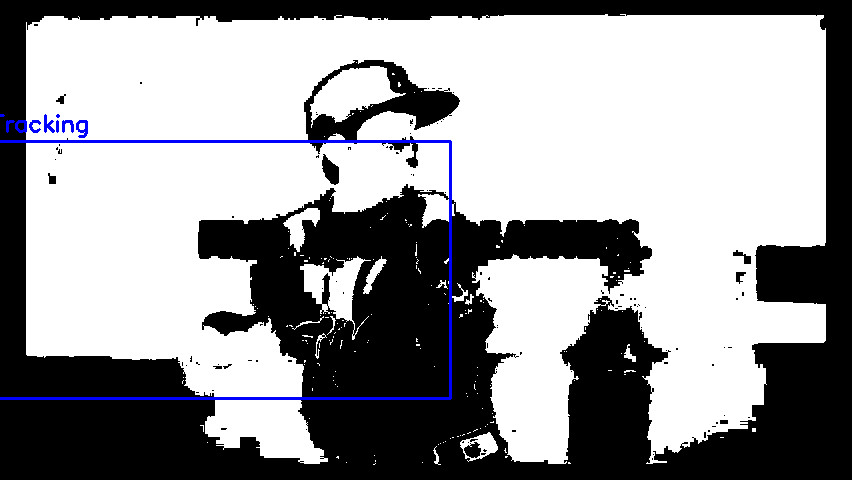
**5. Object Tracking Results**

Objects were successfully tracked across frames, with the system accurately following their movement, even after scene transitions. When cuts were detected, the tracker was reinitialized to avoid tracking errors.

* **Result**: The system maintained consistent object tracking, even through soft and hard cuts, by reinitializing the tracker when necessary.
* **Discussion**: The object tracking algorithm was able to follow objects across consecutive frames, ensuring that their movement was accurately tracked. The use of the CSRT tracker allowed for reliable tracking, even when the object's scale or orientation changed slightly. However, if the object underwent dramatic changes (e.g., occlusion or sudden appearance changes), reinitializing the tracker was necessary. The tracker performed well overall, but further tuning or using more advanced tracking methods like deep learning-based trackers could improve performance in more complex scenarios.





**6. Video Creation Results**

The processed frames were successfully reassembled into a new video file that displayed the tracked objects and marked scene cuts.

* **Result**: The final video was created, displaying the tracked objects with bounding boxes and highlighting scene transitions.
* **Discussion**: The reassembly of frames into a video worked as expected, and the output video provided a clear visualization of the detected and tracked objects across frames. The bounding boxes showed where the objects were located in each frame, and the scene transitions (cuts) were marked, helping viewers understand the segmentation between scenes.



**Overall Discussion**

The system performed well across all steps, from frame extraction to object tracking and video creation. The key challenges, such as handling scene transitions and maintaining accurate tracking, were successfully addressed. The use of color-based segmentation and Canny edge detection helped isolate and highlight objects in each frame, while the cut detection algorithms ensured that scene changes were accounted for, preventing tracking errors.

While the results were satisfactory, there are several areas for improvement:

1. **Segmentation Improvements**: In more complex environments with similar object and background colors, the segmentation process may require further refinement by adjusting HSV thresholds or using more advanced segmentation techniques like machine learning-based segmentation.
2. **Tracking Algorithms**: Although the CSRT tracker performed well, its performance in cases of object occlusion or rapid changes could be improved with more advanced tracking algorithms such as deep learning-based trackers (e.g., SiamMask or GOTURN).
3. **Cut Detection Tuning**: The accuracy of hard and soft cut detection can be further enhanced by fine-tuning the pixel intensity difference and histogram comparison thresholds. This would improve the system's adaptability to different types of videos with varying scene transition patterns.

In conclusion, the system successfully demonstrated its ability to detect, segment, track objects, and handle scene transitions in a video, providing a clear and accurate visualization of object movement over time.

**6. Conclusion**

This assignment successfully implemented a spatio-temporal system for detecting, segmenting, and tracking objects across video frames. The system used a combination of color-based segmentation, edge detection, and object tracking algorithms to handle real-world challenges in video analysis, such as scene transitions (hard and soft cuts), object motion, and changes in lighting.

The system achieved the following objectives:

1. **Frame Extraction**: The video file was split into individual frames, allowing the system to process each frame independently.
2. **Object Segmentation**: Using color-based segmentation, the system effectively isolated objects from the background, ensuring only relevant objects were tracked.
3. **Edge Detection**: The Canny edge detection algorithm helped identify clear object boundaries, enhancing the precision of the object tracking process.
4. **Cut Detection**: Hard and soft cuts between frames were successfully detected, enabling the system to reinitialize tracking when a scene changed. This ensured that objects were not incorrectly tracked across different scenes.
5. **Object Tracking**: The CSRT tracking algorithm maintained accurate tracking of objects across frames. The system adapted to object motion and scene transitions, ensuring continuity in object detection.
6. **Video Creation**: Processed frames were successfully reassembled into a final video that clearly showed the tracked objects and marked scene transitions, providing a visual output of the system's performance.