EXPT 9	Motion Analysis (2D)

AIM:-

To implement 2D motion analysis techniques including background subtraction models (frame differencing, exponential moving average, single Gaussian, mixture of Gaussians) and parameter estimation for velocity, acceleration, trajectory, angular velocity, and optical flow.

CODE:-

```
# Install required packages
!pip install opencv-python matplotlib numpy scipy scikit-image

import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
import os
from scipy import ndimage
import time

print("All packages installed successfully!")

def create_sample_video():
    """Create a simple sample video for testing"""
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('sample_video.avi', fourcc, 10.0, (320, 240))  # Smaller
resolution for faster processing

    for i in range(50):  # Fewer frames for faster testing
        img = np.random.randint(100, 150, (240, 320, 3), dtype=np.uint8)  # Less noise
        # Add moving rectangle
        start_x = i * 3
```

```python
        cv2.rectangle(img, (start_x, 80), (start_x+30, 110), (255, 0, 0), -1)
        # Add another moving object
        cv2.circle(img, (200, 150 - i*2), 15, (0, 255, 0), -1)
        out.write(img)

    out.release()
    return 'sample_video.avi'

def frame_differencing(video_path, threshold=25):
    """
    Simple background subtraction using frame differencing
    """
    cap = cv2.VideoCapture(video_path)
    ret, prev_frame = cap.read()
    if not ret:
        return None

    prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
    motion_frames = []

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Compute absolute difference
        diff = cv2.absdiff(prev_gray, gray)
        # Apply threshold
        _, thresh = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

        # Apply morphological operations to clean up noise
        kernel = np.ones((3,3), np.uint8)
        thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
```

```
        motion_frames.append(thresh)
        prev_gray = gray

    cap.release()
    return motion_frames

class ExponentialMovingAverageBS:
    def __init__(self, alpha=0.05):
        self.alpha = alpha
        self.background = None

    def apply(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        if self.background is None:
            self.background = gray.astype(np.float32)
            return np.zeros_like(gray)

        # Update background model
        cv2.accumulateWeighted(gray, self.background, self.alpha)

        # Compute foreground
        background_uint8 = cv2.convertScaleAbs(self.background)
        foreground = cv2.absdiff(gray, background_uint8)
        _, thresh = cv2.threshold(foreground, 25, 255, cv2.THRESH_BINARY)

        # Clean up noise
        kernel = np.ones((3,3), np.uint8)
        thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

        return thresh

def exponential_moving_average_demo(video_path):
```

```python
    cap = cv2.VideoCapture(video_path)
    ema_bs = ExponentialMovingAverageBS(alpha=0.05)
    results = []

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        foreground = ema_bs.apply(frame)
        results.append(foreground)

    cap.release()
    return results

class SingleGaussianBS:
    def __init__(self, learning_rate=0.05, initial_variance=100):
        self.learning_rate = learning_rate
        self.initial_variance = initial_variance
        self.mean = None
        self.variance = None

    def apply(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY).astype(np.float32)

        if self.mean is None:
            self.mean = gray.copy()
            self.variance = np.full_like(gray, self.initial_variance)
            return np.zeros_like(gray, dtype=np.uint8)

        # Compute foreground mask
        diff_squared = (gray - self.mean) ** 2
        foreground = (diff_squared > 2.5 * 2.5 * self.variance).astype(np.uint8) * 255
```

```
        # Update model only for background pixels
        background_mask = (foreground == 0).astype(np.float32)

        # Update mean
        self.mean = (1 - self.learning_rate * background_mask) * self.mean + \
                   self.learning_rate * background_mask * gray

        # Update variance
        current_diff_squared = (gray - self.mean) ** 2
        self.variance = (1 - self.learning_rate * background_mask) * self.variance + \
                       self.learning_rate * background_mask * current_diff_squared

        # Clean up noise
        kernel = np.ones((3,3), np.uint8)
        foreground = cv2.morphologyEx(foreground, cv2.MORPH_OPEN, kernel)

        return foreground

def single_gaussian_demo(video_path):
    cap = cv2.VideoCapture(video_path)
    sg_bs = SingleGaussianBS(learning_rate=0.05)
    results = []

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        foreground = sg_bs.apply(frame)
        results.append(foreground)

    cap.release()
    return results
```

```python
def mog2_demo(video_path):
    """
    Use OpenCV's built-in MOG2 implementation which is optimized
    """
    cap = cv2.VideoCapture(video_path)

    # Create MOG2 background subtractor
    mog2 = cv2.createBackgroundSubtractorMOG2(history=50, varThreshold=16,
detectShadows=True)

    results = []

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Apply MOG2
        fg_mask = mog2.apply(frame)

        # Remove shadows (gray values)
        _, fg_mask = cv2.threshold(fg_mask, 200, 255, cv2.THRESH_BINARY)

        results.append(fg_mask)

    cap.release()
    return results

def calculate_velocity_trajectory(binary_frames):
    """
    Calculate velocity, acceleration, trajectory from binary motion frames
    """
    trajectories = []
    velocities = []
```

```
    accelerations = []

    for frame in binary_frames:
        # Find centroids of moving objects
        contours, _ = cv2.findContours(frame, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        centroids = []

        for contour in contours:
            if cv2.contourArea(contour) > 50:  # Filter small contours
                M = cv2.moments(contour)
                if M["m00"] != 0:
                    cx = int(M["m10"] / M["m00"])
                    cy = int(M["m01"] / M["m00"])
                    centroids.append((cx, cy))

        trajectories.append(centroids)

    # Calculate velocity
    for i in range(1, len(trajectories)):
        frame_velocities = []
        current_traj = trajectories[i]
        prev_traj = trajectories[i-1]

        # Match closest points between frames
        for j, (x2, y2) in enumerate(current_traj):
            if j < len(prev_traj):
                x1, y1 = prev_traj[j]
                velocity = np.sqrt((x2-x1)**2 + (y2-y1)**2)
                frame_velocities.append(velocity)

        if frame_velocities:
            velocities.append(np.mean(frame_velocities))
        else:
```

```python
            velocities.append(0)

    # Calculate acceleration
    for i in range(1, len(velocities)):
        acceleration = velocities[i] - velocities[i-1]
        accelerations.append(acceleration)

    return trajectories, velocities, accelerations

def visualize_trajectories(trajectories, velocities, accelerations):
    """Visualize the motion analysis results"""
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))

    # Plot 1: Trajectories
    axes[0, 0].set_title('Object Trajectories')
    colors = ['red', 'blue', 'green', 'orange', 'purple']

    for i in range(min(5, len(trajectories[0]))):  # Plot up to 5 objects
        x_coords = []
        y_coords = []
        for traj in trajectories:
            if i < len(traj):
                x, y = traj[i]
                x_coords.append(x)
                y_coords.append(y)

        if x_coords and y_coords:
            axes[0, 0].plot(x_coords, y_coords, 'o-', color=colors[i % len(colors)],
                           label=f'Object {i+1}', markersize=3)
            axes[0, 0].plot(x_coords[0], y_coords[0], 's', color=colors[i % len(colors)],
                           markersize=8, markeredgecolor='black')

    axes[0, 0].set_xlabel('X Position')
    axes[0, 0].set_ylabel('Y Position')
```

```
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].invert_yaxis()  # Invert y-axis to match image coordinates

# Plot 2: Velocity over time
axes[0, 1].plot(velocities, 'b-', linewidth=2)
axes[0, 1].set_title('Velocity Over Time')
axes[0, 1].set_xlabel('Frame')
axes[0, 1].set_ylabel('Velocity (pixels/frame)')
axes[0, 1].grid(True, alpha=0.3)

# Plot 3: Acceleration over time
axes[1, 0].plot(accelerations, 'r-', linewidth=2)
axes[1, 0].set_title('Acceleration Over Time')
axes[1, 0].set_xlabel('Frame')
axes[1, 0].set_ylabel('Acceleration (pixels/frame²)')
axes[1, 0].grid(True, alpha=0.3)

# Plot 4: Motion summary
frames = range(len(velocities))
axes[1, 1].plot(frames, velocities, 'b-', label='Velocity', linewidth=2)
axes[1, 1].plot(frames[1:], accelerations, 'r-', label='Acceleration', linewidth=2)
axes[1, 1].set_title('Motion Summary')
axes[1, 1].set_xlabel('Frame')
axes[1, 1].set_ylabel('Magnitude')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Print statistics
print("MOTION ANALYSIS STATISTICS:")
print(f"Average Velocity: {np.mean(velocities):.2f} pixels/frame")
```

```
    print(f"Max Velocity: {np.max(velocities):.2f} pixels/frame")
    print(f"Average Acceleration: {np.mean(accelerations):.2f} pixels/frame²")
    print(f"Total Frames Analyzed: {len(trajectories)}")

def lucas_kanade_optical_flow(video_path):
    """
    Implement Lucas-Kanade optical flow
    """
    cap = cv2.VideoCapture(video_path)

    # Parameters for ShiTomasi corner detection
    feature_params = dict(maxCorners=50, qualityLevel=0.3, minDistance=7,
blockSize=7)

    # Parameters for Lucas-Kanade optical flow
    lk_params = dict(winSize=(15, 15), maxLevel=2,
                    criteria=(cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))

    # Read first frame
    ret, old_frame = cap.read()
    if not ret:
        return []

    old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
    p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)

    # Create a mask image for drawing purposes
    mask = np.zeros_like(old_frame)

    flow_results = []

    frame_count = 0
    while frame_count < 30:  # Limit frames for performance
```

```
        ret, frame = cap.read()
        if not ret:
            break

        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Calculate optical flow
        if p0 is not None and len(p0) > 0:
            p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None,
**lk_params)

            # Select good points
            if p1 is not None:
                good_new = p1[st == 1]
                good_old = p0[st == 1]

            # Draw the tracks
            for i, (new, old) in enumerate(zip(good_new, good_old)):
                a, b = new.ravel()
                c, d = old.ravel()
                mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)
                frame = cv2.circle(frame, (int(a), int(b)), 5, (0, 0, 255), -1)

            img = cv2.add(frame, mask)
            flow_results.append(img)

            # Update previous frame and points
            old_gray = frame_gray.copy()
            p0 = good_new.reshape(-1, 1, 2)

        frame_count += 1

    cap.release()
    return flow_results
```

```
def compare_methods(frame_diff_result, ema_result, sg_result, mog_result):
    """
    Compare all background subtraction methods
    """
    methods = {
        'Frame Differencing': frame_diff_result,
        'Exponential Moving Average': ema_result,
        'Single Gaussian': sg_result,
        'MOG2': mog_result
    }

    fig, axes = plt.subplots(4, 4, figsize=(20, 20))
    frame_idx = 20

    for i, (method_name, results) in enumerate(methods.items()):
        if i >= 4:  # Limit to 4 methods
            break

        if len(results) > frame_idx:
            # Show result image
            axes[i, 0].imshow(results[frame_idx], cmap='gray')
            axes[i, 0].set_title(f'{method_name}\nFrame {frame_idx}',
fontweight='bold')
            axes[i, 0].axis('off')

            # Calculate statistics
            motion_pixels = np.sum(results[frame_idx] > 0)
            total_pixels = results[frame_idx].size
            motion_percentage = (motion_pixels / total_pixels) * 100

            # Display statistics
            stats_text = f'Motion Pixels: {motion_pixels}\nTotal Pixels:
{total_pixels}\nMotion %: {motion_percentage:.2f}%'
```

```
        axes[i, 1].text(0.1, 0.7, stats_text, fontsize=12, va='center', transform=axes[i,
1].transAxes)
        axes[i, 1].set_title('Frame Statistics')
        axes[i, 1].axis('off')

        # Show histogram
        axes[i, 2].hist(results[frame_idx].flatten(), bins=20, alpha=0.7, color='blue')
        axes[i, 2].set_title('Intensity Distribution')
        axes[i, 2].set_xlabel('Pixel Intensity')
        axes[i, 2].set_ylabel('Frequency')

        # Show cumulative motion over first 30 frames
        frames_to_plot = min(30, len(results))
        cumulative_motion = [np.sum(frame > 0) for frame in
results[:frames_to_plot]]
        axes[i, 3].plot(range(frames_to_plot), cumulative_motion, 'b-', linewidth=2)
        axes[i, 3].set_title('Motion Over Time')
        axes[i, 3].set_xlabel('Frame')
        axes[i, 3].set_ylabel('Motion Pixels')
        axes[i, 3].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

def calculate_performance_metrics(frame_diff_result, ema_result, sg_result,
mog_result):
    """
    Calculate performance metrics for different methods
    """
    methods = {
        'Frame Differencing': frame_diff_result,
        'Exponential Moving Average': ema_result,
        'Single Gaussian': sg_result,
        'MOG2': mog_result
```

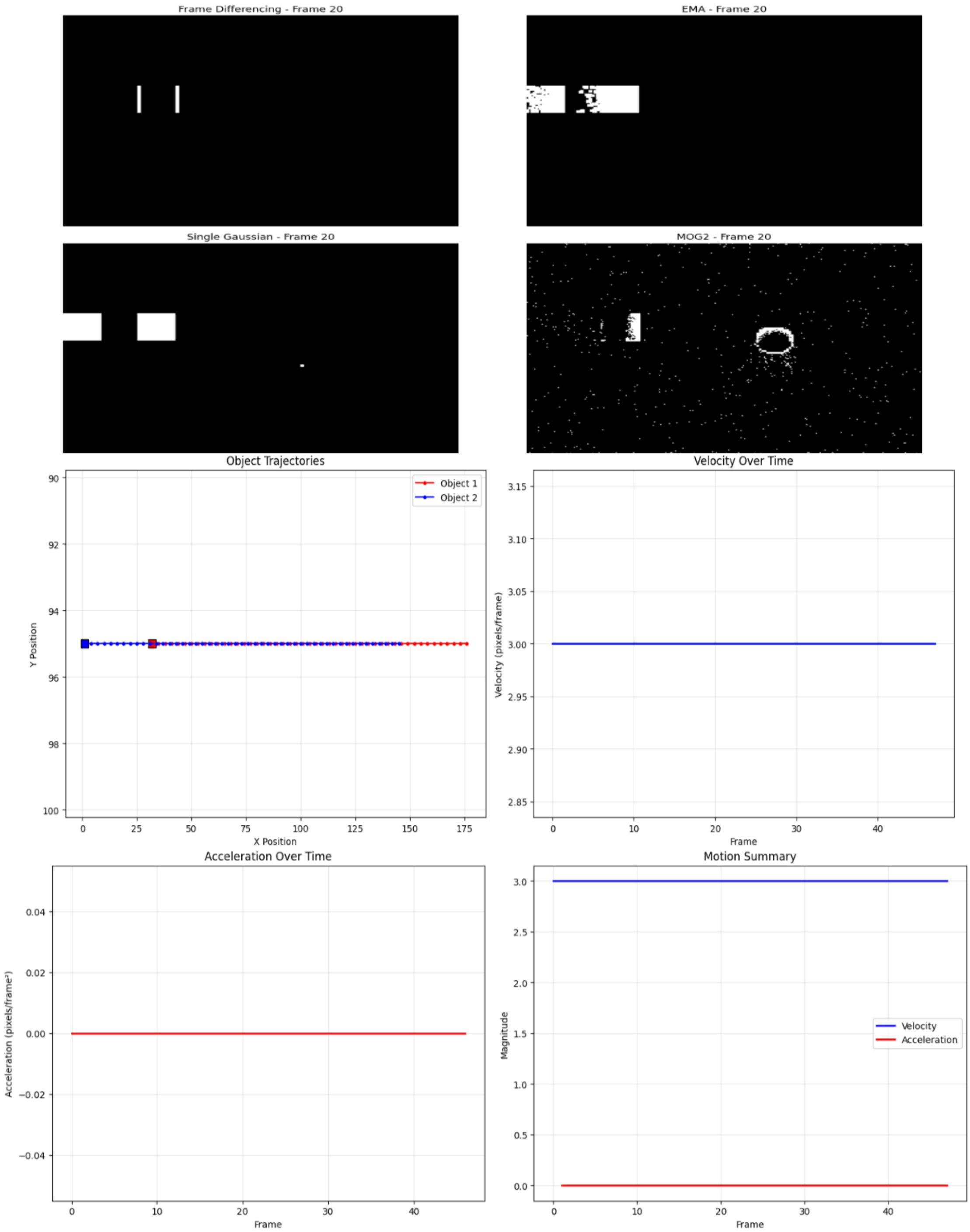
```
    }

    print("PERFORMANCE METRICS COMPARISON")
    print("=" * 60)

    for method_name, results in methods.items():
        if results:
            # Calculate various metrics
            total_frames = len(results)
            avg_motion_per_frame = np.mean([np.sum(frame > 0) for frame in results])
            consistency = np.std([np.sum(frame > 0) for frame in results])

            print(f"\n{method_name.upper():<25}")
            print(f"  {'Total Frames:':<20} {total_frames}")
            print(f"  {'Avg Motion Pixels:':<20} {avg_motion_per_frame:.0f}")
            print(f"  {'Consistency (std):':<20} {consistency:.2f}")
            print(f"  {'Avg Motion %:':<20}
{(avg_motion_per_frame/results[0].size)*100:.2f}%")

# Create sample video
video_path = create_sample_video()
print(f"Sample video created: {video_path}")

# Run background subtraction methods
print("Running Frame Differencing...")
start_time = time.time()
frame_diff_result = frame_differencing(video_path)
print(f"Frame Differencing completed in {time.time() - start_time:.2f} seconds")

print("Running Exponential Moving Average...")
start_time = time.time()
ema_result = exponential_moving_average_demo(video_path)
print(f"Exponential Moving Average completed in {time.time() - start_time:.2f}
seconds")
```

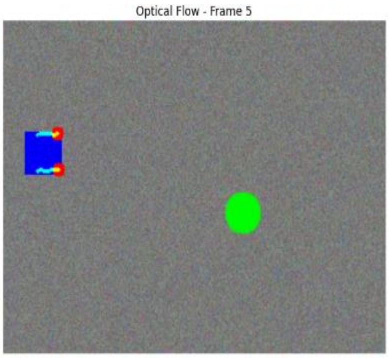
```python
print("Running Single Gaussian...")
start_time = time.time()
sg_result = single_gaussian_demo(video_path)
print(f"Single Gaussian completed in {time.time() - start_time:.2f} seconds")

print("Running Mixture of Gaussians (MOG2)...")
start_time = time.time()
mog_result = mog2_demo(video_path)
print(f"MOG2 completed in {time.time() - start_time:.2f} seconds")

# Display background subtraction results in 2 rows
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.imshow(frame_diff_result[20], cmap='gray')
plt.title('Frame Differencing - Frame 20')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(ema_result[20], cmap='gray')
plt.title('EMA - Frame 20')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(sg_result[20], cmap='gray')
plt.title('Single Gaussian - Frame 20')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(mog_result[20], cmap='gray')
plt.title('MOG2 - Frame 20')
plt.axis('off')
```

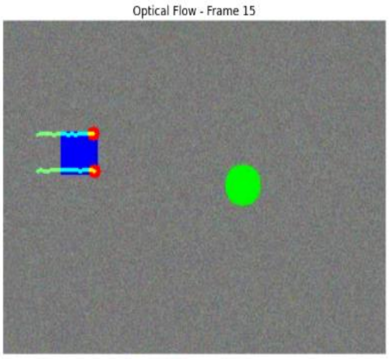
```
plt.tight_layout()
plt.show()


# Run 2D Motion Analysis
print("Performing 2D Motion Analysis...")
trajectories, velocities, accelerations =
calculate_velocity_trajectory(frame_diff_result)
visualize_trajectories(trajectories, velocities, accelerations)

# Run Optical Flow Analysis
print("Calculating Optical Flow...")
start_time = time.time()
optical_flow_result = lucas_kanade_optical_flow(video_path)
print(f"Optical Flow completed in {time.time() - start_time:.2f} seconds")

# Display optical flow results in 3 rows
if optical_flow_result:
    plt.figure(figsize=(15, 15))
    for i, idx in enumerate([5, 15, 25]):
        if idx < len(optical_flow_result):
            plt.subplot(3, 1, i+1)
            plt.imshow(cv2.cvtColor(optical_flow_result[idx],
cv2.COLOR_BGR2RGB))
            plt.title(f'Optical Flow - Frame {idx}')
            plt.axis('off')

    plt.tight_layout()
    plt.show()
else:
    print("No optical flow results to display")


# Perform Comparative Analysis
print("Performing Comparative Analysis...")
```

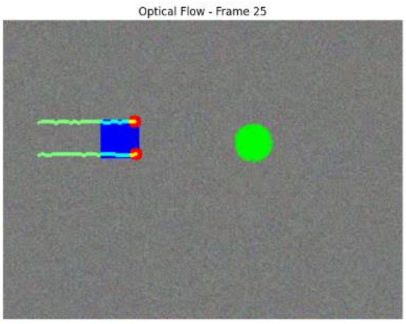
```
compare_methods(frame_diff_result, ema_result, sg_result, mog_result)

# Calculate Performance Metrics
calculate_performance_metrics(frame_diff_result, ema_result, sg_result,
mog_result)
```

OUTPUT:-

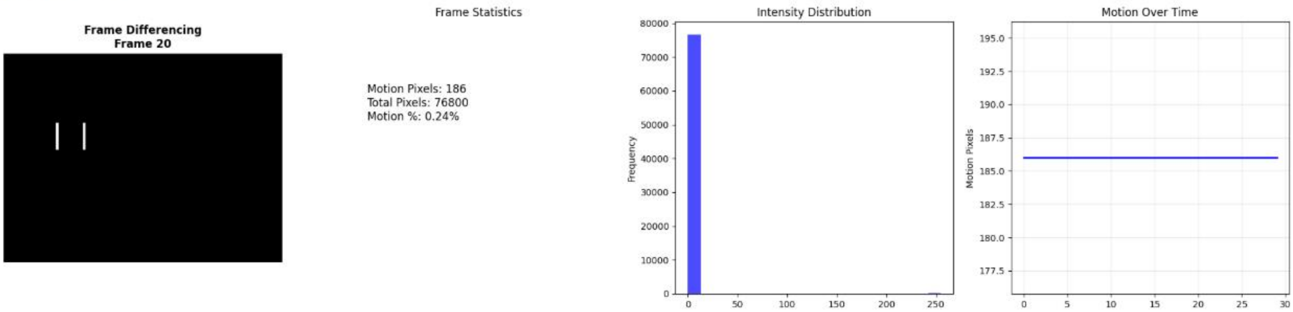


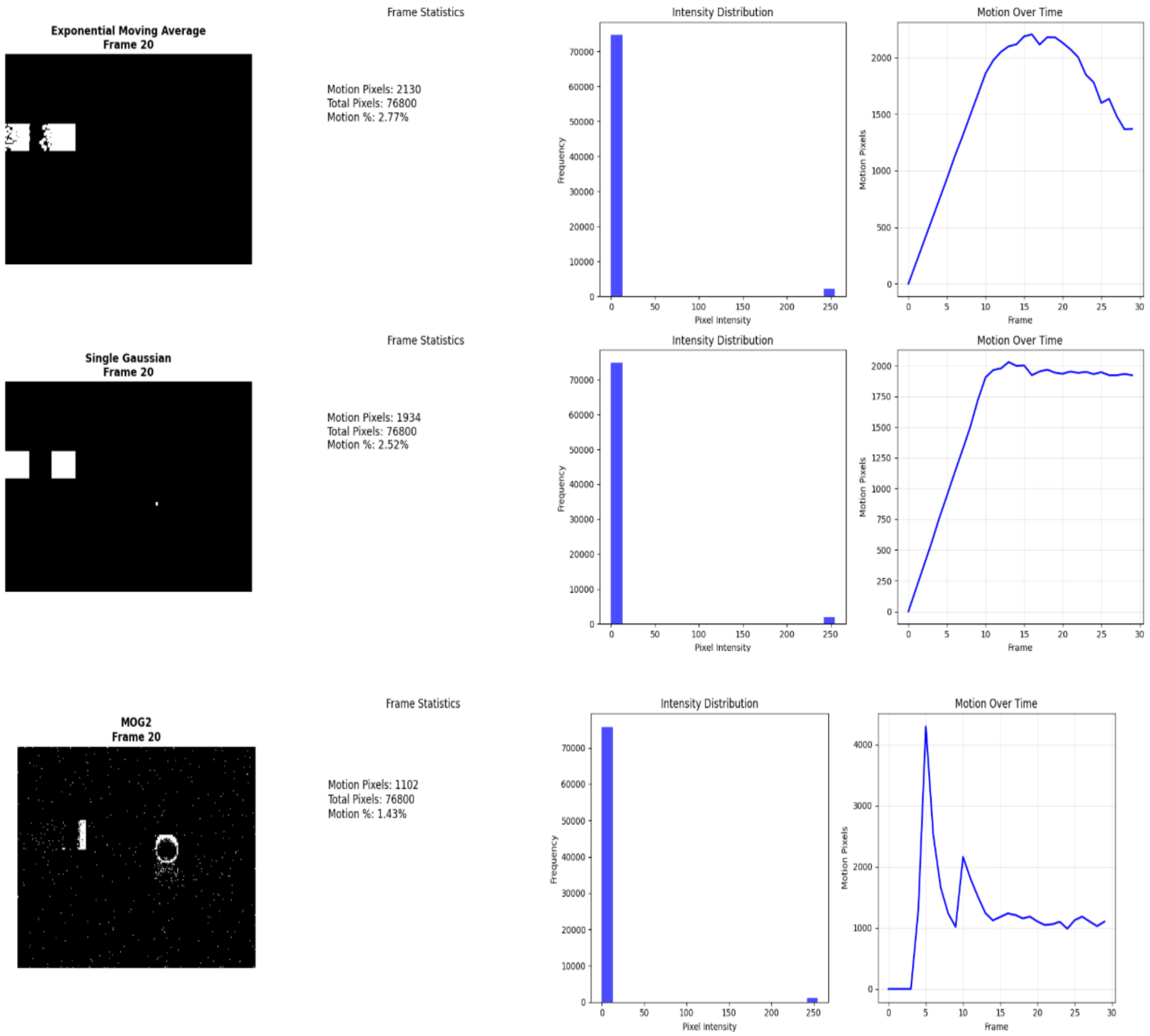






Performing Comparative Analysis...





RESULT:-

Successfully detected and analyzed moving objects in video sequences using various background subtraction methods. Parameter estimation techniques effectively quantified motion characteristics, enabling tracking and behavior analysis of dynamic objects in 2D space.