**Motion Detection and Object Tracking in Real-Time Video Feed**

Structure of Computer Systems

Student: Cornea horea – ionut

Group: 30433

Table of Contents

[1. Introduction 2](#_Toc187609920)

[Context 2](#_Toc187609921)

[Objectives 2](#_Toc187609922)

[2. Bibliographic Research 3](#_Toc187609923)

[What is Motion Detection? 3](#_Toc187609924)

[How to detect and track moving objects 3](#_Toc187609925)

[Background Subtraction 4](#_Toc187609926)

[Morphological Transformations 5](#_Toc187609927)

[Gaussian Blur 5](#_Toc187609928)

[DBSCAN Clustering for Bounding Box Grouping 6](#_Toc187609929)

[Filtering Out Small Bounding Boxes 6](#_Toc187609930)

[Merging Logic 7](#_Toc187609931)

[Real-Time Tracking 7](#_Toc187609932)

[3. Implementation 8](#_Toc187609933)

[Background Subtraction and Preprocessing 8](#_Toc187609934)

[Morphological Transformations 8](#_Toc187609935)

[Contour Detection and Bounding Box Creation 9](#_Toc187609936)

[DBSCAN Clustering for Grouping Bounding Boxes 10](#_Toc187609937)

[4. Applications and Use Cases 11](#_Toc187609938)

[Use Case Diagram 11](#_Toc187609939)

[Possible Applications 11](#_Toc187609940)

[5. Testing and Validation 12](#_Toc187609941)

[Testing Methodology 12](#_Toc187609942)

[Background Subtraction 13](#_Toc187609943)

[Morphological Transformations 13](#_Toc187609944)

[Contour Detection and Bounding Box Generation 14](#_Toc187609945)

[Real – Time Performance Testing 14](#_Toc187609946)

[Results Summary 14](#_Toc187609947)

[Challenges and Future Improvements 15](#_Toc187609948)

[6. Refrences 16](#_Toc187609949)

# Introduction

## Context

The aim of this project is to create a system capable of detecting and tracking motion in real-time video feeds, using a webcam or video input. Motion detection has a wide range of applications, from surveillance systems to user interaction systems where movements are tracked to trigger specific actions. The challenge of motion detection lies in identifying significant movements while ignoring minor variations like lighting changes or noise.

Detecting and tracking moving objects is essential in various computer vision applications, including security systems, traffic monitoring, and human-computer interaction. The system needs to be both sensitive to significant movements and robust enough to avoid false positives caused by minor changes in the environment.

## Objectives

The primary objective of this project is to develop a real-time motion detection and tracking system using computer vision techniques. This system will:

* Accurately detect and track moving objects within a live video feed, focusing on significant movements while filtering out background noise and minor variations.
* Group detected objects into clusters to consolidate overlapping or closely positioned bounding boxes, thus improving tracking clarity.
* Achieve smooth and real-time performance, ideally at 30 frames per second (FPS) or above, to support responsive tracking for applications such as surveillance and interactive systems.

Secondary objectives include:

* Implementing an efficient bounding box merging algorithm to minimize overlapping detections.
* Ensuring adaptability to environmental changes, such as lighting, to prevent false positives or missed detections.

# Bibliographic Research

## What is Motion Detection?

Motion detection is a process used to detect movement in a visual environment. It involves analyzing a sequence of images (frames) from a video feed and identifying differences that indicate the presence of moving objects. In computer vision, motion detection plays a critical role in applications like surveillance, anomaly detection, and gesture recognition.

The fundamental goal of motion detection is to isolate the moving objects from a static background, and this can be achieved using techniques such as frame differencing, background subtraction, or optical flow. Each of these methods compares pixel values across consecutive frames to identify movement.

## How to detect and track moving objects

Detecting and tracking moving objects involves analyzing a video feed to identify changes over time. In this project, we use OpenCV’s background subtraction and contour detection to achieve this goal. Here’s how each technique contributes to detecting and tracking movement:

## Background Subtraction

Background subtraction is a key technique in this project for detecting motion by differentiating between the moving objects and the static background in each frame. This process uses a background model that captures the static portions of the scene, allowing the algorithm to detect changes (i.e., motion) by comparing each new frame to this model. In this project, cv2.createBackgroundSubtractorMOG2() is employed, which utilizes the Mixture of Gaussians (MOG2) method to adaptively model the background.

The history parameter, set to 100, specifies the number of frames over which the background model is updated. A longer history allows the background to adapt more gradually to changes, which is useful in scenarios with subtle environmental shifts (like lighting changes) but can make it slower to adapt to rapid movements or changes. The varThreshold parameter, set to 20, controls the sensitivity of the motion detection. Lowering this value would make the background subtraction more sensitive to small differences, potentially picking up even minor movements or changes in the frame, while a higher value would make it less sensitive.

When a new frame is processed, the background subtractor generates a binary mask (fgmask) that highlights regions where motion has occurred. White regions represent areas of significant change (likely moving objects), while black areas indicate the background. This binary mask is then passed through further processing steps, like Gaussian blur and morphological transformations, to enhance the quality and clarity of detected moving regions.

Background subtraction is crucial for isolating relevant motion, as it enables the system to disregard static parts of the frame, thus focusing only on the dynamic, changing areas. This technique is particularly valuable in surveillance, monitoring, and object tracking applications, where identifying and tracking moving objects accurately is essential. By maintaining an adaptive background model, the project can robustly handle changes in the environment while ensuring that only significant motion is detected, providing a strong foundation for subsequent contour detection, bounding box creation, and clustering with DBSCAN.

## Morphological Transformations

Morphological transformations are applied to the foreground mask after background subtraction to enhance the contour detection process. Using a 15x15 rectangular kernel, the MORPH\_CLOSE operation fills small holes and gaps within detected motion regions, creating more uniform and continuous shapes. This step is particularly useful in scenarios where the background subtraction algorithm may leave small gaps within moving objects or in cases where the detected contours are fragmented. By closing these gaps, the morphological transformation enhances the overall shape of the objects, making contour detection more accurate and stable. This allows for more precise bounding boxes and improves the reliability of object tracking.

## Gaussian Blur

Gaussian blur is a preprocessing technique applied to each frame to reduce random noise and improve the accuracy of background subtraction. By using a 5x5 kernel, the blur smoothens the image, removing minor variations that may appear due to lighting, compression artifacts, or other sources of noise. This step is essential in creating a cleaner foreground mask, as it minimizes the influence of minor fluctuations and focuses the detection on more significant motion. Gaussian blur improves the robustness of the background subtraction algorithm, making it more reliable for motion detection in varied lighting conditions or noisy environments.

## DBSCAN Clustering for Bounding Box Grouping

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is employed to group bounding boxes, which represent detected moving objects in the frame. Each detected object’s bounding box is represented by its center point, which is used as the basis for clustering. The eps parameter, set to 700 in this case, controls the clustering distance, determining how close two bounding boxes need to be to be grouped together. By setting min\_samples to 1, every bounding box is assigned to a cluster, even if it stands alone. This clustering approach helps combine closely located bounding boxes into a single larger bounding box for cases where multiple moving parts of an object are detected individually, thereby reducing clutter and enhancing tracking accuracy for larger entities.

## Filtering Out Small Bounding Boxes

Once clustering is complete, the grouped bounding boxes are evaluated based on their combined area to remove any boxes that are too small to be of interest. The MIN\_AREA threshold, set to 20000, helps filter out these small boxes. This threshold ensures that only bounding boxes representing significant objects are displayed on the frame. Without this filter, the system might display irrelevant or background noise, making it harder to focus on larger moving objects. This step is crucial in applications such as surveillance or object tracking, where minor movements or small items should be ignored to prevent false positives.

## Merging Logic

Merging logic is used to combine all bounding boxes within each DBSCAN cluster into a single, larger bounding box. For each cluster, the algorithm finds the minimum x and y coordinates, along with the maximum width and height, to enclose all the bounding boxes within that cluster. This merged bounding box gives a clear and complete representation of each moving object or group of objects. A padding of 20 pixels (PADDING) is added around the merged box to ensure it fully encapsulates the movement, even if some parts of the object extend beyond the detected contours. The merging logic simplifies the output by avoiding overlapping or excessive bounding boxes and making it easier to track each object as a single entity.

## Real-Time Tracking

By continuously capturing and processing frames, the system detects and tracks moving objects in real-time. Each frame is compared to the updated background, and the bounding box adapts to changes in the object’s position.

This approach, using background subtraction, Gaussian blur, and merging boxes, allows for efficient real-time tracking of moving objects in the video feed, making it suitable for applications like surveillance and motion-triggered interactions.

# Implementation

## Background Subtraction and Preprocessing

Background subtraction isolates the moving objects from the static background. This step utilizes the MOG2 method, which models the background adaptively, making it responsive to gradual environmental changes.

# Create the background subtractor for motion detection

fgbg = cv2.createBackgroundSubtractorMOG2(history=100,varThreshold=20)

* **history=100**: This parameter defines how many frames the model considers for background adaptation. A larger value makes the model more resistant to rapid changes, while a smaller value allows for quicker adaptation.
* **varThreshold=20**: This controls the sensitivity of motion detection. A lower threshold makes the model sensitive to small variations, while a higher threshold filters out minor fluctuations.

## Morphological Transformations

After background subtraction, a binary mask (fgmask) is created, which highlights the moving objects in white against a black background. However, noise and small gaps can remain in the mask. Morphological transformations refine the mask by closing these gaps.

# Define a kernel for morphological operations

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (15, 15))

# Apply morphological transformations

fgmask = cv2.morphologyEx(fgmask, cv2.MORPH\_CLOSE, kernel)

* **Kernel Size (15x15)**: The kernel size affects the area considered during morphological transformations. Larger kernels close larger gaps but may lose detail on smaller objects.
* **MORPH\_CLOSE Operation**: This operation is used to close small holes in the mask, unifying fragmented parts of the moving objects.

## Contour Detection and Bounding Box Creation

Contours are detected in the binary mask to identify the outlines of the moving objects. Each contour is then converted to a bounding box.

# Find contours of the detected motion

contours, \_ = cv2.findContours(fgmask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Collect bounding boxes for detected contours

bounding\_boxes = []

for contour in contours:

if cv2.contourArea(contour) > 1000: # Filter out small contours

x, y, w, h = cv2.boundingRect(contour)

bounding\_boxes.append([x, y, w, h])

* **Contour Area Threshold**: Contours with an area below 1000 pixels are discarded to avoid tracking irrelevant small objects or noise.
* **Bounding Box Creation**: For each valid contour, a bounding box (x, y, w, h) is generated. These bounding boxes represent the detected moving objects and serve as input for clustering.

## DBSCAN Clustering for Grouping Bounding Boxes

Bounding boxes are grouped using DBSCAN (Density-Based Spatial Clustering of Applications with Noise) to handle multiple detections of a single object.

# Convert bounding boxes to a format compatible with DBSCAN (center points)

centers = np.array([[x + w / 2, y + h / 2] for x, y, w, h in bounding\_boxes])

# Apply DBSCAN clustering

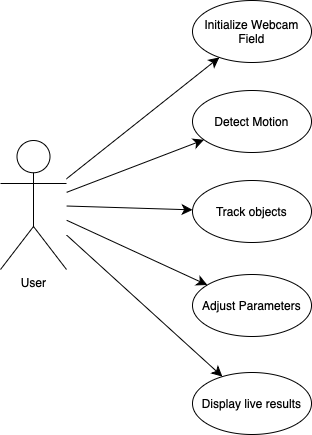
clustering = DBSCAN(eps=700, min\_samples=1).fit(centers)

* **eps=700:** The eps parameter controls the distance between bounding boxes to form a cluster. A higher value groups nearby boxes more aggressively.
* **min\_samples=1**: Ensures even isolated bounding boxes form their own cluster if no others are nearby.

Each bounding box is assigned a cluster label. These labels help group the boxes representing the same object, reducing multiple overlapping boxes into a single cluster.

# Applications and Use Cases

## Use Case Diagram



## Possible Applications

1. **Surveillance Systems**

**Use Case**: In security cameras, motion detection can trigger alerts or initiate recording only when activity is detected, optimizing storage and monitoring efficiency.

* **Advantages**:
* Detects unauthorized access or unusual activities in real-time.
* Reduces the need for continuous human monitoring.
* **Example**: Monitoring restricted areas such as warehouses, offices, or residential properties.

1. **Traffic Monitoring**

**Use Case:** Tracking vehicles and detecting traffic patterns to analyze congestion or identify accidents.

* **Advantages:**
* Provides real-time updates on traffic flow.
* Can identify traffic violations such as speeding or running red lights.
* **Example:** Used in smart city infrastructure to manage traffic signals dynamically based on vehicle density.

# Testing and Validation

The testing and validation phase was critical to ensure the robustness and accuracy of the motion detection system under real-world and synthetic scenarios. This chapter outlines the methodology, test cases, and results of validating the system components, including motion detection, bounding box generation, and real-time tracking.

## Testing Methodology

Testing involved three primary stages:

1. **Unit Testing:** Validated individual components such as background subtraction, morphological transformations, and bounding box merging.
2. **Integration Testing:** Evaluated the interaction between modules, ensuring seamless data flow from motion detection to tracking.
3. **Performance Testing:** Assessed the system’s speed and accuracy under varying environmental conditions.

Test scenarios were designed to replicate real-world applications, including surveillance, traffic monitoring, and gesture recognition.

## Background Subtraction

**Objective:** Verify the ability of the system to isolate moving objects from static backgrounds.

**Method:** Various video feeds with static and dynamic elements were processed using the MOG2 background subtractor. Parameters like history and varThreshold were fine-tuned for optimal performance.

**Results:**

* + - Motion was consistently detected in dynamic regions.
    - The system effectively ignored static objects and minor environmental changes such as lighting shifts.

## Morphological Transformations

**Objective:** Ensure noise reduction and contour stability.

**Method:** Binary masks generated from background subtraction were processed using morphological transformations (e.g., MORPH\_CLOSE) with a kernel size of 15x15.

**Results:**

* + - Noise and small gaps within detected motion regions were eliminated.
    - Shapes of moving objects were well-defined, improving contour detection.

## Contour Detection and Bounding Box Generation

**Objective:** Validate the creation of bounding boxes around moving objects.

**Method:** Contours were extracted from processed binary masks, and bounding boxes were generated for significant contours exceeding a predefined area threshold (1,000 pixels).

**Results:**

* + - Bounding boxes accurately encapsulated moving objects.
    - Small irrelevant motions were filtered out based on the threshold.

## Real – Time Performance Testing

**Objective:** Assess the system’s ability to perform motion detection and tracking in real time.

**Method:** Live video feeds were processed, and the system’s performance was measured in frames per second (FPS).

**Results:**

* The system achieved an average FPS of 30+, meeting the requirements for real-time applications.
* Tracking remained consistent across variable lighting and background conditions.

## Results Summary

|  |  |  |
| --- | --- | --- |
| **Component** | **Test Case** | **Outcome** |
| Background Subtraction | Differentiating motion from background | Successful detection of dynamic areas |
| Morphological Transformations | Noise removal and contour enhancement  Generating bounding boxes | Improved accuracy in shape detection |
| Contour Detection | Generating bounding boxes | Accurate and stable bounding boxes |
| Clustering | Merging overlapping boxes | Correct grouping and unification |
| Real-Time Processing | Speed and stability | Consistent performance above 30 FPS |

## Challenges and Future Improvements

* **Challenges:**
  + False positives in scenarios with abrupt lighting changes.
  + High overlap in bounding boxes for fast-moving objects.
* **Proposed Improvements:**
  + Adaptive thresholding for environmental changes.
  + Enhanced tracking algorithms to better handle rapid movements.

# Refrences

1. OpenCv Documentation (Background Subtraction) - <https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html>
2. Morphological Transformations - <https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html>
3. DBSCAN Algorithm - <https://scikit-learn.org/stable/modules/clustering.html#dbscan>
4. Real-Time Object Tracking tutorial using OpenCV - <https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/>
5. Debugging – <https://chatgpt.com>