ÉCOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE ET D'ANALYSE DES SYSTÈMES - RABAT

TRAITEMENT DE DONNÉES MULTIMÉDIA
RAPPORT : PROJET

Key Points Detection and Tracking for Event Detection in Crowded Environments

Élèves :
Mohamed STIFI
Hafssa BOUNIA
Ayoub EL ASSIOUI

Enseignant :
Pr. Sanaa FAKIHI

6 janvier 2025

Acknowledgments

First and foremost, we express our heartfelt gratitude to The Almighty Allah for granting us the strength and patience needed to successfully complete this work.

We would also like to extend our sincere thanks to our supervisor, Pr. Sanaa El Fakihi, for her invaluable guidance, trust, and generosity. Her unwavering support and the resources he provided were essential to the successful completion of this project. Throughout the duration of our work, he offered insightful guidance and demonstrated remarkable patience, helping us navigate the challenges we encountered.

Finally, we would like to thank all those who have contributed, both directly and indirectly, to the completion of this project. Your support has been invaluable, and your contributions have played a crucial role in helping us reach this gratifying outcome.

Abstract

This report presents a detailed overview of a key points detection and tracking system developed for the analysis of dynamic scenes, particularly focused on event detection in crowd environments. The system integrates various methodologies such as Harris Corner Detection, optical flow tracking, and models for both magnitude and direction, to capture the movement and interactions of key points within a video. Through a structured approach that involves detection, tracking, and grouping, the system is designed to handle the complexity of real-world scenarios, where the goal is to identify and analyze significant events in crowded scenes. This report explores the theoretical foundations of the key methods used, the architecture of the implementation, and the results obtained through various stages of the process.

Résumé

Ce rapport présente un aperçu détaillé d'un système de détection et de suivi de points clés développé pour l'analyse de scènes dynamiques, en particulier axé sur la détection d'événements dans des environnements bondés. Le système intègre diverses méthodologies telles que la détection de coins de Harris, le suivi par flux optique, ainsi que des modèles pour la magnitude et la direction, afin de capturer les mouvements et les interactions des points clés au sein d'une vidéo. À travers une approche structurée impliquant la détection, le suivi et le regroupement, le système est conçu pour gérer la complexité des scénarios réels, où l'objectif est d'identifier et d'analyser des événements significatifs dans des scènes de foule. Ce rapport explore les bases théoriques des principales méthodes utilisées, l'architecture de l'implémentation et les résultats obtenus à travers les différentes étapes du processus.

Introduction

The analysis of dynamic scenes in video data is a key component in various fields such as computer vision, surveillance, and event detection. In particular, the ability to detect and track key points across frames provides valuable insights into the movement and interactions within a scene[1]. This is especially important in crowded environments, where the identification of significant events, such as unusual group behaviors or sudden movements, is crucial for applications in security, crowd management, and public safety.

This report discusses the development of a system that employs various computer vision techniques for detecting and tracking key points in video sequences. The system integrates methods such as Harris Corner Detection for identifying stable key points, optical flow tracking for capturing movement, and statistical models to evaluate the magnitude and direction of motion. Additionally, the system utilizes strategies like block grouping and group tracking to enhance its capability in complex, crowded scenarios. Through the application of these methods, the system aims to accurately detect and analyze events in real-time video data.

By combining these techniques, the system provides a robust framework for monitoring and understanding dynamic scenes, offering significant potential for use in event detection and related applications.

Chapitre 1

Generalities

In this chapter, we explore fundamental techniques and models essential for analyzing and detecting events in dynamic environments, particularly in crowd scenes. We begin by discussing optical flow, a powerful method for tracking the apparent motion of objects between consecutive frames, which plays a crucial role in understanding movement patterns in videos. We then dive into the Lucas-Kanade method, a widely used approach for estimating optical flow in a local neighborhood, making it particularly effective for detecting small motions and tracking individual objects. Finally, we introduce the Von Mises mixture and the unidimensional Gaussian mixture models, which are used for probabilistic modeling of directional data and general distributions, respectively. These models are important for analyzing the complex movement patterns in crowd scenes, where understanding the orientation and grouping of individuals is crucial.

1.1 Optical Flow

Optical flow refers to the apparent motion of objects between two consecutive frames in a video, which results from the movement of either the object or the camera. It is represented as a 2D vector field, where each vector indicates the displacement of a point from the first frame to the second. The image below illustrates this concept

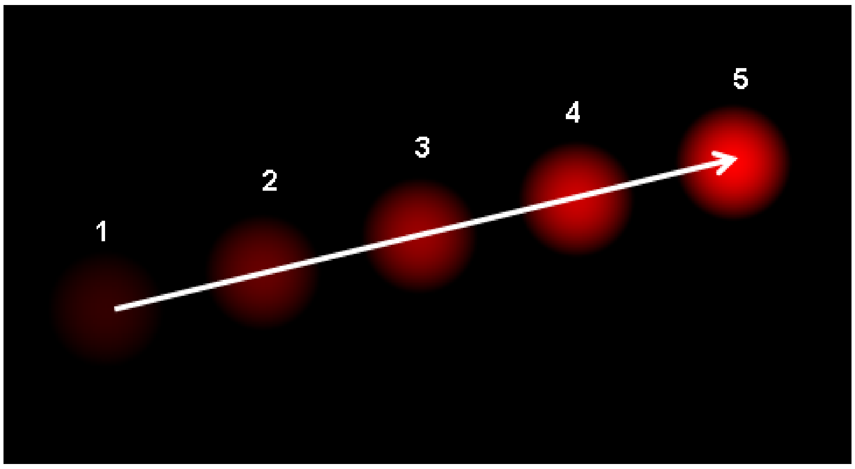


Figure 1.1 – Illustration of optical flow showing the displacement of a ball in 5 consecutive frames. The arrow represents the displacement vector.[2]

In this example, the ball moves across five consecutive frames, and the displacement of each point is captured by the vectors shown. Optical flow has numerous applications in fields such as :

— Structure from Motion (SfM)
— Video Compression
— Video Stabilization

The optical flow method operates under several assumptions :

1. The pixel intensities of an object remain constant between consecutive frames.
2. Neighboring pixels move in a similar manner.

Let's consider a pixel $I(x, y, t)$ in the first frame. Here, an additional dimension—time—has been introduced. This pixel moves by a distance (dx, dy) in the next frame taken after a time interval dt. Since the pixel remains the same and its intensity does not change, we can express this as :

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Next, by applying a Taylor series approximation to the right-hand side, canceling out common terms, and dividing by dt, we obtain the following equation :

$$f_x u + f_y v + f_t = 0$$

Where :

— $f_x = \frac{\partial f}{\partial x}$
— $f_y = \frac{\partial f}{\partial y}$
— $u = \frac{dx}{dt}$
— $v = \frac{dy}{dt}$

This equation is known as the **Optical Flow Equation**. In it, f_x and f_y represent the image gradients, while f_t denotes the gradient along time. However, (u, v)—the optical flow vectors—remain unknown. Since we have one equation with two unknowns, additional constraints are required to solve the equation. One popular method for solving this is the **Lucas-Kanade method**.[3]

1.2 Lucas-Kanade Method

As previously mentioned, one of the assumptions made in optical flow is that neighboring pixels exhibit similar motion. The **Lucas-Kanade method** leverages this assumption by considering a 3x3 patch around a point, meaning that all 9 points within this patch are assumed to have the same motion. For each of these 9 points, the gradients f_x, f_y, and f_t are computed. This transforms the problem into solving 9 equations with two unknown variables, which is over-determined. The optimal solution is then obtained using the **least squares method**.

The final solution to this system of equations is given by :

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_x^2 & \sum_i f_x f_y \\ \sum_i f_x f_y & \sum_i f_y^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_x f_t \\ -\sum_i f_y f_t \end{bmatrix}$$

Where the terms f_x, f_y, and f_t are the image gradients in the x, y directions and along time, respectively. This equation represents a two-variable, two-equation problem that can be solved to obtain the optical flow vectors u and v.

From the user's perspective, the method is relatively simple : we provide a set of points to track, and the algorithm returns the optical flow vectors for those points. However, there are challenges when dealing with large motions. The algorithm works well for small motions, but its performance degrades when the motion between consecutive frames is large. To overcome this, **pyramidal optical flow** is used. By applying image pyramids, the scale of motion is adjusted : when we move to higher levels in the pyramid, small motions are smoothed out, and large motions appear as smaller ones. This allows the Lucas-Kanade method to work effectively even for large motions.[3]

Pseudocode for the Lucas-Kanade Method

```
Input:
    - Image sequence (I1, I2)
    - Points to track (x, y)

Output:
    - Optical flow vectors (u, v)

1. Compute the gradients:
    - fx = I/x  (gradient in x direction)
    - fy = I/y  (gradient in y direction)
    - ft = I/t  (gradient along time)

2. For each point (x, y) in the image:
    - Take a 3x3 patch around the point (x, y)
    - Compute the sums:
        -  fx^2,  fx * fy,  fy^2
        -  fx * ft,  fy * ft

3. Solve the system of equations:
    - [u, v] = [ fx^2,  fx * fy ;  fx * fy,  fy^2]^-1 * [- fx * ft; - fy * ft]

4. Apply pyramidal approach if large motion is detected:
    - Build an image pyramid for each frame
    - Apply Lucas-Kanade on each level of the pyramid, starting from the coarsest leve
    - Refine the flow vectors at each level

5. Return the optical flow vectors (u, v) for each point.
```

1.3 Von Mises Mixture and Unidimensional Gaussian Mixture

In statistical modeling, mixture models are used to represent data that come from multiple distributions. These models are particularly useful when the data is assumed to come from a combination of several underlying distributions, each contributing to the overall behavior of the dataset. Below, we explain two types of mixture models : the **Von Mises mixture** and the **Unidimensional Gaussian mixture**.

Von Mises Mixture

The **Von Mises distribution** is a continuous probability distribution on the unit circle, used primarily for modeling angular data or directional data. It is often referred to as the "circular normal distribution" because it behaves similarly to the normal distribution but is specifically designed to handle data that is periodic in nature, such as angles or orientations.

A **Von Mises mixture** is a combination of multiple Von Mises distributions, each with its own mean and concentration parameter. The probability density function (PDF) of a Von Mises mixture is given by[4] :

$$f(\theta) = \sum_{k=1}^{K} \pi_k \frac{1}{2\pi I_0(\kappa_k)} e^{\kappa_k \cos(\theta - \mu_k)}$$

Where :
- θ is the angular variable (the direction or orientation),
- μ_k is the mean direction (location of the peak),
- κ_k is the concentration parameter (similar to the inverse variance in the normal distribution),
- $I_0(\kappa_k)$ is the modified Bessel function of the first kind of order zero,
- π_k is the mixing coefficient for the k-th component (weights of the mixture components).

The Von Mises mixture model is used in applications such as modeling the distribution of orientations in motion detection, robotic navigation, and various biological phenomena where data is circular in nature.

Unidimensional Gaussian Mixture

A **Gaussian mixture** is a probabilistic model that represents the presence of subpopulations within an overall population, where each subpopulation is modeled by a Gaussian (normal) distribution. In the case of a **unidimensional Gaussian mixture**, the data is assumed to be generated from a mixture of several Gaussian distributions, each with its own mean, variance, and mixing coefficient.

The probability density function (PDF) of a unidimensional Gaussian mixture is expressed as :

$$f(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \sigma_k^2)$$

Where :

- x is the data point,
- μ_k is the mean of the k-th Gaussian component,
- σ_k^2 is the variance of the k-th Gaussian component,
- $\mathcal{N}(x|\mu_k, \sigma_k^2)$ is the normal distribution with mean μ_k and variance σ_k^2,
- π_k is the mixing coefficient (the weight of the k-th Gaussian component), which satisfies $\sum_{k=1}^{K} \pi_k = 1$.

[5]

A unidimensional Gaussian mixture is widely used in various applications, such as clustering, density estimation, and anomaly detection. It can model data that comes from multiple underlying Gaussian distributions, making it flexible and useful in many real-world scenarios where data is not normally distributed but instead represents a combination of several different Gaussian distributions.

1.4 Conclusion

The concepts discussed in this chapter lay the groundwork for the practical implementation of event detection in crowd scenes, which is the focus of the next chapter. The combination of optical flow techniques and probabilistic models, such as the Lucas-Kanade method and mixture models, provides a robust framework for detecting and analyzing movement patterns in dynamic and crowded environments. By leveraging these methods, we can effectively detect events and extract meaningful information from video data.

Chapitre 2

Implementation

This chapter outlines the detailed processes and methodologies employed in the development of the key points detection and tracking system for the PETS09-S2L1 challenge. This section provides an in-depth look into the architecture of the project, including the series of steps followed to ensure a robust solution. The chapter covers crucial components such as Harris Corner Detection for key point identification, optical flow for tracking, and the integration of models for direction and magnitude computation. Additionally, the grouping of image blocks and group tracking techniques are explored, leading to the final output of the system.

2.1 Project Architecture

```
project/

 detection/
    point_detection.py          # Extraction and tracking of points of interest
    magnitude_direction_model.py          # Magnitude & direction model
 grouping/
    block_grouping.py         # Block clustering
    group_tracking.py           # Group tracking

 main.py          # Entry point of the project; coordinates the modules
                    # for the overall pipeline execution
 test.py          #  Script for testing the functionality of different modules
                    # to ensure the code works as expected
```

2.2 Project steps

The proposed approach consists of several steps (Figure 2.1). The first step involves extracting a set of key points from the current image. These points are then tracked in the subsequent image using optical flow computation techniques. Static points are removed to focus on moving subjects. The scene is divided into blocks, with each motion vector assigned to its corresponding block.

The block classification step groups neighboring blocks that share similar orientation and velocity. Finally, the groups are tracked, and both the magnitude model and the directional model are trained.

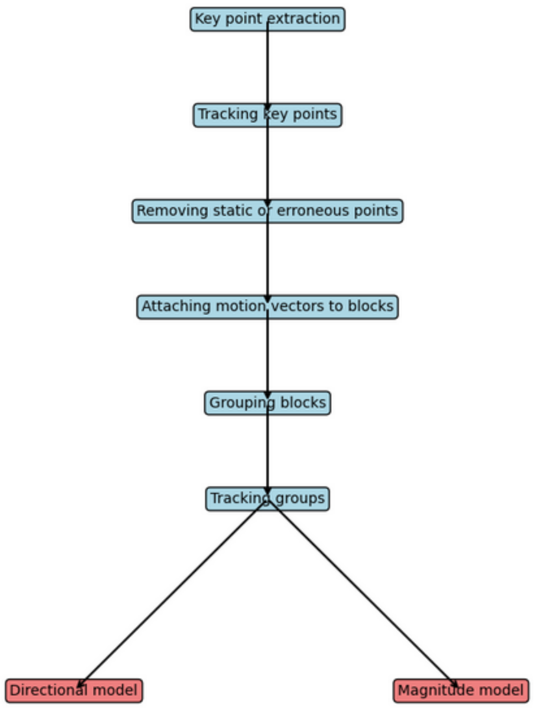


FIGURE 2.1

2.3 Detection and tracking of key points

The code uses two primary algorithms :

— **Harris Corner Detection** for detecting interest points in the image.

— **Lucas-Kanade Optical Flow** for tracking these points between consecutive frames.

The functions in the code are described below.

2.3.1 Harris Corner Detection

The function `detect_harris_points` detects interest points using the Harris corner detection algorithm. It works as follows :

1. Convert the input frame from color to grayscale using `cv2.cvtColor`.
2. Apply the Harris corner detection using `cv2.cornerHarris` with configurable parameters for the block size (`block_size`), kernel size (`ksize`), and sensitivity factor (`k`).
3. Dilate the response using `cv2.dilate` to highlight the corners more clearly.
4. Define a threshold as a fraction (0.01) of the maximum Harris response value, and find the points where the Harris response exceeds this threshold.
5. Convert these points into a list of `cv2.KeyPoint` objects to be used by other OpenCV functions.

6. Return the key points as a numpy array.

The function returns the coordinates of the key points detected in the image.

2.3.2 Tracking Key Points with Optical Flow

The function `track_points` tracks the key points between two consecutive frames using the Lucas-Kanade optical flow method. The steps are as follows :

1. Convert both the previous and current frames into grayscale images.
2. Use `cv2.calcOpticalFlowPyrLK` to compute the optical flow, which estimates the new positions of the key points in the current frame.
3. Iterate over the points and calculate the movement vectors by subtracting the previous position from the new position.
4. Filter out points with small movement (below a threshold defined by `min_movement`) and compute the magnitude and angle of the movement vector.
5. Return the valid points with significant movement, their new positions, and the corresponding movement vectors.

This function returns the updated key points and their corresponding movement vectors.

2.3.3 Dividing Image into Blocks

The function `divide_into_blocks` divides the image into blocks of a specified size and associates the movement vectors of the tracked points with their respective blocks. The process is as follows :

1. Determine the height and width of the frame.
2. For each tracked point, calculate the block in which it belongs by dividing the coordinates by the block size.
3. Group the points into blocks using a dictionary, where the keys are the block coordinates and the values are lists of points within the block.
4. Return the dictionary containing the blocks and their associated points.

This function is useful for analyzing the movement of points in localized regions of the image.

2.3.4 Results

The video used to execute the code is from the PETS09-S2L1 challenge, which is part of the PETS (Performance Evaluation of Tracking and Surveillance) series. This specific video is a part of the second session (S2) of the 2009 PETS challenge, where the focus is on surveillance and tracking of objects in crowded and dynamic environments. The challenge involves detecting and tracking the movement of objects, such as pedestrians, in video sequences captured by surveillance cameras.

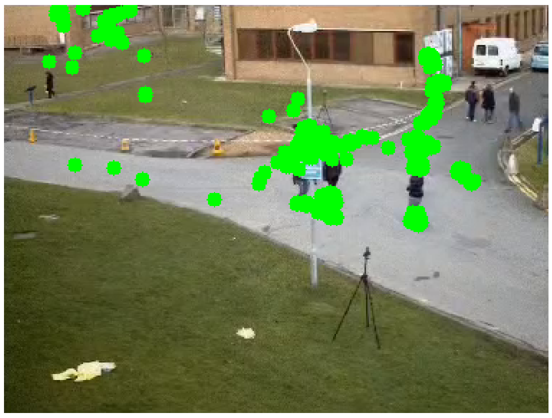


FIGURE 2.2 – First frame showing the objects

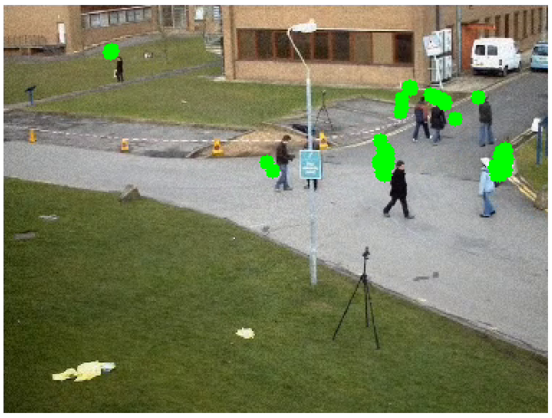


FIGURE 2.3 – Second frame after tracking the movement of the object.

2.4 Magnitude and the direction models

This section presents models for the magnitude and direction of movement vectors. Specifically, it discusses the use of von Mises mixture models to describe the directional probabilities of the movement vectors and Gaussian Mixture Models (GMM) to model the magnitudes of the vectors. These models are applied at the block level to analyze the motion patterns in different regions of the image.

The implementation consists of two primary models :

— **Von Mises Mixture Model** for modeling the direction of movement vectors.

— **Gaussian Mixture Model (GMM)** for modeling the magnitude of the movement vectors.

The functions in the code are explained below.

2.4.1 Von Mises Mixture Model

The class `VonMisesMixture` implements a mixture of von Mises distributions, which are commonly used to model circular data, such as angles or directions. The key steps are :

1. **Initialization :** The model is initialized with a default number of components (`n_components`). The weights are set equally for all components, and the means are evenly spaced over the interval $[0, 2\pi)$.
2. **Density Calculation :** The von Mises density function is calculated using the formula described in the previous chapter
3. **Fitting the Model :** The fitting process involves the Expectation-Maximization (E-M) algorithm tailored for circular data. In the E-step, the responsibilities of each component are computed based on the current parameters. In the M-step, the parameters (weights, means, and concentrations) are updated to maximize the likelihood.
4. **Prediction :** The model calculates the probability density for a given angle by summing the contributions from all components.

The mixture model allows for the representation of complex directional distributions, which are useful for analyzing movement patterns in video or image sequences.

2.4.2 Computing Directional Probabilities

The function `compute_directional_probabilities` computes the directional probability distribution for each block of movement vectors using the von Mises mixture model. For each block, the angles of the movement vectors are extracted, and the von Mises mixture model is fitted to these angles. The resulting model for each block is stored in the `block_probabilities` dictionary.

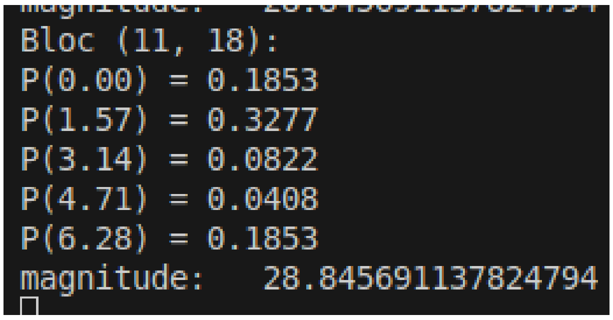
2.4.3 Gaussian Mixture Model for Magnitudes

The function `compute_magnitude_model` models the magnitude of movement vectors using a Gaussian Mixture Model (GMM). The steps are as follows :

1. **Collecting Magnitudes :** The magnitudes of the movement vectors from all blocks are collected into a single array.
2. **Training the GMM :** A GMM is trained on the collected magnitudes. The number of components in the GMM is determined by the number of samples available. If there are enough samples, the model uses a default number of components (`default_components`). If there are too few samples, the number of components is adjusted accordingly.
3. **Fitting the GMM :** The GMM is fit to the magnitude data, and the means of the Gaussian components are extracted.
4. **Computing Block Magnitudes :** For each block, the average magnitude is computed by taking the mean of the means of the GMM components. If there are no vectors in a block, the block magnitude is set to `None`.

The resulting model provides an estimate of the average magnitude of movement vectors for each block.

2.4.4 Results



```
Bloc (11, 18):
P(0.00) = 0.1853
P(1.57) = 0.3277
P(3.14) = 0.0822
P(4.71) = 0.0408
P(6.28) = 0.1853
magnitude:   28.845691137824794
```

FIGURE 2.4 – The directional probabilities and magnitude for the bloc (11,18)

2.5 Block grouping

This section describes the process of grouping blocks in an image or video based on their directional and magnitude similarities. The blocks are classified into groups where each group consists of blocks that share similar movement directions and magnitudes. The grouping is based on two key criteria :

- The angular difference between the directions of the blocks (modeled using the von Mises distribution).
- The difference in the average magnitude of the movement vectors in the blocks.

The goal of this process is to cluster the blocks into coherent regions that exhibit similar motion patterns.

The code involves grouping blocks using a threshold-based approach. The key steps are outlined below.

2.5.1 Block Similarity Evaluation

The function `group_blocks` groups blocks that have similar directions and magnitudes. The function works as follows :

1. **Similarity Check :** The function `are_blocks_similar` compares two blocks to check if they are similar in terms of their direction and magnitude.
   - **Direction Similarity :** The difference between the main directions of the two blocks is calculated. The direction is represented by the mean of the von Mises distribution for each block. The angular difference is calculated using :

     $$\text{angle\_diff} = \min\left(|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\right)$$

     where θ_1 and θ_2 are the directions of the two blocks. The blocks are considered similar if the angular difference is below a specified threshold (`threshold_angle`).
   - **Magnitude Similarity :** The magnitude of each block is retrieved from the `block_magnitudes` dictionary. The blocks are considered similar if the absolute difference in magnitude is below a specified threshold (`threshold_magnitude`).
2. **Block Grouping :** The blocks are grouped using a depth-first search (DFS) approach. Starting from an unvisited block, the algorithm iteratively checks all neighboring blocks (in a 4-connected neighborhood) to see if they are similar to the current block. If they are similar, the neighboring block is added to the current group. This process continues until no more similar blocks can be added to the group.
3. **Group Formation :** The groups are stored in the `groups` list, where each group is a set of block coordinates. The process ensures that all blocks are visited and grouped accordingly.

2.5.2 Neighboring Blocks and Group Expansion

The algorithm uses a stack to manage the expansion of groups. For each block, the algorithm checks its immediate neighbors (up, down, left, right) to see if they meet the similarity criteria. If a neighbor is similar, it is added to the group and further explored.

2.5.3 Final Grouping Output

The function `group_blocks` returns a list of groups, where each group is a set containing the coordinates of the blocks that belong to that group. The grouping process ensures that the blocks within each group have similar motion characteristics.

2.5.4 Results

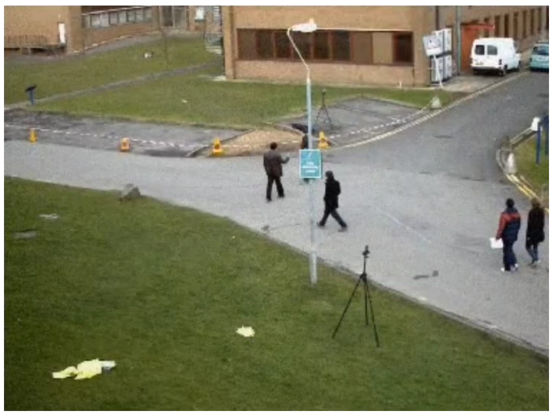


FIGURE 2.5 – Original frame

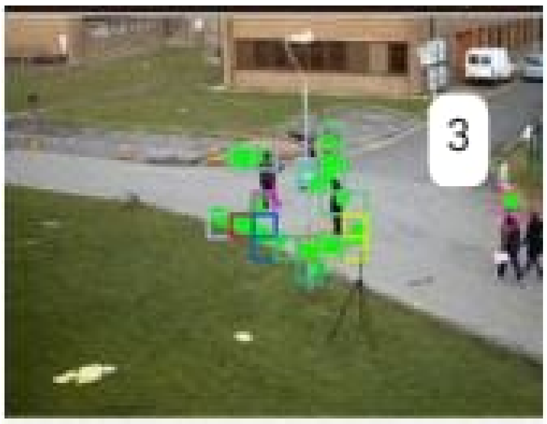


FIGURE 2.6 – Similar blocks visualized on the frame

As it can be seen the two people who are heading to the road three are considered similar (the pink squares).

2.6 Group tracking

This section describes the process of tracking groups of blocks across two consecutive frames. The groups are tracked by calculating the centroids of the blocks in each group and matching groups from the previous frame to those in the current frame based on the distance between their centroids. The tracker aims to maintain the identity of groups across frames, even when new groups appear or old groups disappear.
The code involves the creation of a `Tracker` class that is responsible for tracking the groups across frames. The key steps are outlined below.

2.6.1 Tracker Initialization

The `Tracker` class is initialized with a minimum distance (`min_distance`) that determines how close two centroids must be to be considered a match. The class maintains two lists :

— `previous_groups` : A list of groups from the previous frame.
— `current_groups` : A list of groups from the current frame.

2.6.2 Centroid Calculation

The function `compute_centroid` computes the centroid (the geometric center) of a given group of blocks. The centroid is calculated by averaging the coordinates of all blocks in the group. The formula for computing the centroid of a group is :

$$\text{centroid}_x = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \text{centroid}_y = \frac{1}{n}\sum_{i=1}^{n} y_i$$

where (x_i, y_i) are the coordinates of the blocks in the group, and n is the number of blocks in the group.

2.6.3 Group Matching and Update

The function `update_groups` matches the groups from the previous frame to the groups in the current frame. The process works as follows :

1. The centroids of the groups from both frames are computed.
2. The Euclidean distance between each pair of centroids from the previous and current frames is calculated :
$$\text{distance} = \sqrt{(x_{\text{prev}} - x_{\text{curr}})^2 + (y_{\text{prev}} - y_{\text{curr}})^2}$$
where $(x_{\text{prev}}, y_{\text{prev}})$ and $(x_{\text{curr}}, y_{\text{curr}})$ are the centroids of the previous and current frames, respectively.
3. If the distance between a pair of centroids is less than the specified minimum distance (`min_distance`), the groups are considered a match.
4. If no match is found for a group in the previous frame, it is considered to have disappeared.
5. If a group in the current frame does not have a match in the previous frame, it is considered a new group.

The function returns a list of matches, where each match is represented as a tuple (i, j), indicating that group i from the previous frame corresponds to group j from the current frame. If there is no match, j is set to `None`.

2.6.4 Results

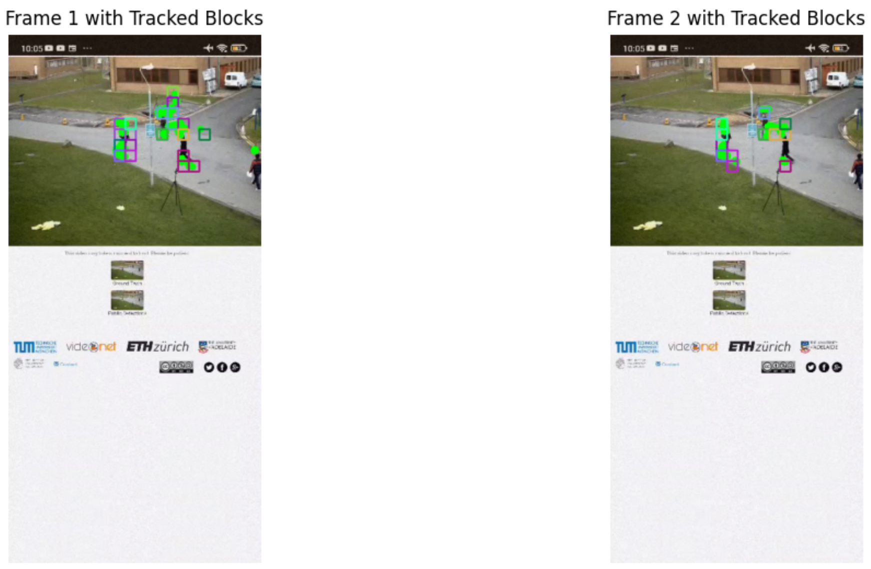


FIGURE 2.7 – Visualization of the Tracked blocks

2.7 Conclusion

In conclusion, the implementation of the key points detection and tracking system is a comprehensive process involving several interconnected components. The use of Harris Corner Detection, optical flow tracking, and directional and magnitude models ensures accurate and efficient performance. Furthermore, the block grouping and group tracking strategies enhance the system's capability to handle complex scenarios. Through careful design and testing, the system is able to successfully detect, track, and group key points, offering a solid foundation for applications in video analysis and related fields, especially in event detection in crowd scenes

Conclusion

In conclusion, the key points detection and tracking system developed in this project represents a comprehensive and versatile approach to analyzing dynamic scenes, particularly in crowded environments. The system effectively integrates multiple techniques, including Harris Corner Detection, optical flow tracking, and advanced models for magnitude and direction, enabling precise detection and tracking of key points across frames. By leveraging block grouping and group tracking methods, the system enhances its ability to handle complex scenarios, ensuring robust performance even in crowded and chaotic settings.

The success of this system lies in its ability to accurately detect, track, and group key points, providing a solid foundation for identifying significant events within crowd scenes. Moving forward, the next phase involves defining specific rules and criteria that characterize an event within a crowd. With these event definitions in place, the system can be further refined to detect and analyze targeted events in real time, offering valuable insights for applications in surveillance, crowd management, and event monitoring. Overall, this project lays the groundwork for scalable, efficient video analysis systems that can be tailored to various real-world use cases, contributing to enhanced safety, security, and understanding of crowd behavior.

Bibliographie

[1] A Combined Corner and Edge Detector. Opencv : Optical flow, 1988.

[2] Wikipedia. Optical flow.

[3] OpenCV. Opencv : Optical flow, 2024.

[4] John Bentley. Modelling circular data using a mixture of von mises and uniform distributions. page 4, 2006.

[5] DataMListic. Gaussian mixture models (gmm) explained, 2023.