Computer Vision and Image Processing
Alma Mater Studiorum — a.a. 2024/2025

MANTIS
Intrusion Detection Project

Bocconcelli Francesca, Fantini Luca

"Development of a computer vision system aimed at intrusion detection"

1 Summary

We developed a software system that uses automatic video analysis to detect objects (intruders) that don't belong to a static reference scene (background). The system identifies moving objects by comparing each video frame to the background and then classifies which of these objects are people. By using known techniques, like background subtraction, the system can effectively detect human presence, making it useful for security and surveillance applications. This approach helps automate monitoring and improve safety by alerting users to any unusual activity.

2 Introduction

2.1 Problem of Motion Detection

Motion detection is the process of identifying changes in the position of an object relative to its background. It plays a crucial role in a variety of fields, such as intelligent video surveillance, traffic monitoring, event detection, and people tracking. While there are multiple modalities for detecting motion, including infrared sensors, radar, and sound, the camera-based approach is widely used due to its effectiveness in computer vision applications. Video-based motion detection relies on capturing frames from a video feed and classifying the pixels of each frame into two sets: one representing the background and the other representing the foreground, where the object of interest resides. [1]

Several traditional methods are employed for motion detection, each with its own advantages and limitations. These methods can be used individually or in combination to improve detection accuracy.
The most widely used approach is **background subtraction**, which calculates the difference between a reference background frame and the current frame to identify moving objects. Although simple, it can be enhanced by techniques such as edge background, Gaussian mixtures, kernel density estimation (KDE), and temporal median filtering.
Another approach, **frame differencing**, calculates the difference between consecutive frames. It is computationally simpler compared to other methods but may fail to capture the full contours of the detected objects, leading to inaccurate results. This can be addressed by applying morphological operations for further refinement.
Similarly, the **temporal difference** method calculates pixel-wise differences between successive frames but struggles in scenarios where objects are moving too slowly or too quickly, as the small frame-to-frame differences may cause the object to be missed.
Lastly, the **optical flow** method, while computationally expensive, provides a more detailed approach by estimating the flow of objects between frames and clustering them based on optical flow distribution. This method offers reliable motion detection with more comprehensive information but is sensitive to noise, making it particularly useful for real-time applications.

2.2 Our Task and Challenges

Our task was to develop a software system that, based on automatic video analysis, can detect objects (intruders) that do not belong to a static reference scene (background) and establish which of such objects are persons.
In the analyzed video, a **person moves continuously** within an indoor environment at varying speeds. For a significant portion of the sequence, the subject is shown in close-up view, occupying a large part of the video frame. Due to this constant movement, there is no time interval in which

the background is visible without the subject, making it challenging to apply a standard background subtraction algorithm, as there is no static reference frame for comparison. Instead, the background must be estimated dynamically, even in the presence of a continuously moving object.
The first part of the video shows the subject moving within the environment while the lighting gradually changes (darkening). The main challenge in this scenario is adapting the background model to handle these **continuous variations in illumination**. A different type of illumination change, which is not present in this video sample and is not specifically addressed in our approach, is a sudden shift in lighting. Since this would cause a diffuse difference across the entire frame, a standard background model might temporarily misclassify a large number of pixels. However, our method is designed to adapt over time, allowing it to recover as the algorithm stabilizes.
In the second part, the subject interacts with objects in the scene, first removing one and then introducing another that was not originally present. This presents challenges for basic background subtraction methods, which typically incorporate "quickly" added or removed **stationary objects** into the background model. However, we aim to keep this distinction visible over time. In principle, it is possible to use a *selective update* for the background models in order to not absorb detected foreground objects; in Section 4 we will discuss our approach and considerations.
Moreover, objects were not only added but also removed, making it necessary to develop a mechanism that allows the algorithm to distinguish between an introduced object (true object) and a removed one (false object).
Furthermore, the video is grayscale so it can happen more frequently a known problem known as **"camouflage"**. The intensity of the foreground is very similar to the background so the frame difference will be near zero and thus the foreground or part of it is wrongly classified. This is a real challenge and we tried to achieve slightly better result with using the postprocessing phase. An approach that could effectively address this problem is the use of additional features, such as disparity or depth, which would help to more clearly distinguish the objects.
The presence of **noise** in the video was another challenge, because the irregularities caused by the noise made it difficult for the algorithm to accurately identify true motion, leading to inaccuracies and reducing the overall performance of the detection process.
Given these characteristics, the solution must adapt to these requirements accordingly.

3 Algorithms Comparison

In the scientific literature on this topic, numerous methods are presented, each addressing a range of problems based on assumptions and hypotheses about the scenario, the camera, the specific challenges and the spatio-temporal complexity of the algorithm.
The first decision we faced was selecting the most suitable motion detection technique. As mentioned in the Section 2.1, there are several possible approaches: background subtraction, optical flow, frame differencing, and temporal differencing. We chose background subtraction due to its strong performance, low memory requirements, and ability to deliver reliable results. Now, let's take a closer look at the main methods of background subtraction.

Background model	References
Basic	Running average [14, 84, 196, 70] Temporal median [110] Motion history image [16, 118]
Parametric	Single Gaussian [182] Gaussian Mixture Model (GMM) [156, 72, 74, 202, 48, 46, 146, 58] Background clustering [26, 126, 82] Generalized Gaussian Model [4, 80] Bayesian [138, 94] Chebyshev inequality [118]
None-Parametric	Kernel Density Estimation (KDE) [42, 122, 190, 116]
Data-driven	Cyclostationary [140] Stochastic K-nearest neighbors (KNN) [6, 64]. Deterministic KNN [202] Hidden Markov Model (HMM) [158]
Matrix Decomposition	Principal Component Analysis (PCA) [124, 188, 96, 36, 150] Sparsity and dictionary learning [136, 194]
Prediction Model	Kalman filter [78, 198] Weiner filter [168]
Motion Segmentation	Optical flow segmentation [180, 114, 102] GMM and Optical flow segmentation [126, 200]
Machine Learning	1-Class SVM [32] SVM [100, 62, 60] Neural networks [104, 106, 152]

Figure 1: Overview of 8 families of motion detection methods. [2]

As shown in the table in Figure 1, there are various approaches to create and maintain a background model. Despite the many variants of motion detection algorithms, no single method seems to effectively address all the challenges encountered in real-world scenarios. The research in this field is vast; for instance, the paper "Analysis of Computer Vision-Based Techniques for Motion Detection" [2] presents a classification of motion detection methods into eight different families. An even wider collection of algorithms and techniques is summarized in "Traditional and Recent Approaches in Background Modeling for Foreground Detection: An Overview" [3]. Moreover, another increasingly popular approach is the use of deep learning for motion detection tasks. Naturally, there is a wide range of methods available, from highly complex to more straightforward ones, as real-world scenarios vary significantly. Depending on the complexity and variability of the situation, one might require more advanced algorithms or simpler solutions.
For our task, which involved a relatively simple and static indoor environment, we chose to focus on exploring traditional approaches to motion detection. In addition to the requirements expressed in Section 2.2, our goal was to identify methods that strike a balance between simplicity, efficiency, low resource consumption and accurate results.
To guide this analysis, we referred to the paper "Background Subtraction Techniques: A Review" [4], from which we extracted the table in Figure 2. This table provides a comprehensive comparison of several traditional methods, considering factors such as speed, memory usage and qualitative accuracy. The time complexity $O(1)$ has been defined as the complexity required by the running Gaussian average, in which, for each pixel, the classification is just a thresholded difference and the background model update adapts just one or two parameters.
Given:

- n: the total number of samples in the sample set.
- m: the number of Gaussian distributions used in the Mixture of Gaussians method.
- n_s: the sub-sampled number of samples taken from the full sample set in the Temporal Median Filter.

Method	Spd	Mem	Acc
Running Gaussian average [1,2]	*1*	*1*	*L/M*
Temporal median filter [3,4]	n_s	n_s	*L/M*
Mixture of Gaussians [5]	m	m	*H*
Kernel density estimation (KDE) [7]	n	n	*H*
Sequential KD approximation [11]	$m + 1$	m	*M/H*
Cooccurence of image variations [12]	$8n/N^2$	nK/N^2	*M*
Eigenbackgrounds [13]	*M*	n	*M*

Figure 2: Background subtraction method and performance analysis [4]

We observed that lighter methods, such as the Mixture of Gaussians (especially the OpenCV MOG2 implementation, which uses only 3 to 5 Gaussian distributions, leading to low memory consumption) and the Temporal Median Filter (which uses a small number of frames for background computation), showed promising results.

Next, we evaluated the performance of OpenCV's background subtraction implementations.
As shown in Figure 3, there are numerous ready-to-use algorithms, each with its own set of parameters to configure their behavior. Many of these algorithms overlap with those discussed in our previous analysis. We tested the methods without parameter tuning and just running them on the task. This approach allowed us to observe how each method performs in its default configuration, providing an initial comparison before any optimization or customization of the parameters.
We also reviewed the relevant papers, if available, to understand the assumptions and scenarios under which the algorithms were developed. For our particular scenario, among the tested methods, MOG2 (Mixture of Gaussians) and CNT performed particularly well.

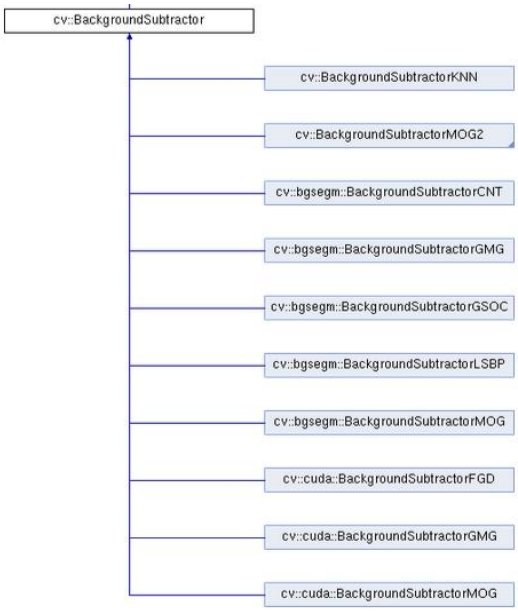

Figure 3: Background Subtraction algorithms in OpenCV.

Following this initial evaluation, we chose to focus on testing three key algorithms: **MOG2** (Mixture of Gaussians), the **Temporal Median Filter**, and **CNT**. The next subsections will provide a detailed analysis of each method and, after that, we will describe how we incorporated them into our solution in the Section 4. Moreover, it is possible to observe the results of those tests from this Jupyter Notebook file: `experiments/IntrusionDetectionComparison.ipynb`

3.1 Temporal Median Filter

The Temporal Median Filter is a simple yet effective method for estimating the Background Reference Frame. It works by computing the pixel-wise median across the last N frames, which helps create a robust reference for background subtraction. Outliers or moving objects are unlikely to occupy the median position in the sorted frame buffer, effectively excluding them. This approach has shown good performance even when frames are subsampled over time, i.e., by introducing a counting variable to skip some frames and store others in the frame buffer.
A key advantage of the Temporal Median Filter is its simplicity and efficiency. However, one limitation is that it requires a buffer to store recent pixel values, resulting in slightly higher memory consumption compared to methods that do not rely on frame history.
Despite this, the results remain quite good, especially for our specific video challenges. To handle gradual illumination changes, it was crucial to maintain an up-to-date background frame throughout the video. If the background was not kept "fresh", these subtle lighting variations would have been wrongly classified as foreground. Thus, the new frames must be appened in the frame buffer quickly enough to capture the changes in the median value in a timely manner.
One notable drawback of the median filter is that it does not offer a rigorous statistical description of the background and does not provide a measure of deviation to adapt the subtraction threshold. The temporal filtering methods are sensitive to compression artifacts, global illumination changes and incapable to detect moving objects once they become stationary. Despite these limitations, the Temporal Median Filter was chosen for our motion detection pipeline due to its simplicity, ease of implementation, and satisfactory performance in handling "long" horizon differences.

In the next section, we will delve into the role of the Temporal Median Filter within our algorithm and its integration into the overall motion detection system.

3.2 MOG2

Let's refer again to Figure 2. From this analysis, the **Running Gaussian Average** appears to be the least complex method, with a time complexity of $O(1)$. This approach is a simplified version of the Mixture of Gaussians (MoG) model, where each pixel is represented by a single Gaussian distribution. Specifically, for each pixel, only the mean μ and variance σ^2 of the Gaussian function are stored.

Although the Running Gaussian Average has a lower computational complexity than MoG, relying on a single Gaussian makes it less effective because pixel values often exhibit complex distributions, so this method could have difficulties in handling dynamic environments, illumination changes, and intermittent object occlusions. To address these limitations, we opted for the **Mixture of Gaussians**, which models each pixel with multiple Gaussian components. In this approach, instead of a single Gaussian, each pixel is represented by N Gaussian distributions (typically 3 to 5), where each component describes a different mode of the background model.
Specifically, we used the OpenCV implementation, which typically employs $m \approx 3..5$ Gaussian distributions, resulting in a complexity of $O(m)$. Despite the increased computational cost, this method achieves higher accuracy (H) and better adaptability to real-world scenarios.

The fundamental assumption of MOG is that images of a scene without intrusive objects exhibit regular behavior that can be well described by a probabilistic model. If a statistical model of the scene is available, an intrusive object can be detected by identifying parts of the image that do not fit the model.
The Mixture of Gaussians (MoG) model is an extension of pixel-wise background subtraction, where the scene is represented by a **probability density function** for each pixel. In this approach, the probability density of a pixel is modeled as a weighted sum of several Gaussian components, where each component is defined by its mean $\hat{\mu}_m$ and variance $\hat{\sigma}_m^2$, with weights $\hat{\pi}_m$ indicating their contribution. This allows for more flexibility in representing complex pixel intensity distributions. A pixel in a new image is considered part of the background if its value is well described by the mixture of Gaussian components. This method improves upon simpler models by accommodating dynamic environments and variations in illumination.
The GMM parameters are updated recursively based on a new data sample. If no component is close enough to the new sample, a new component is created with initial values, and if the maximum number of components is reached, the least significant component is discarded. The closeness is determined by calculating the Mahalanobis distance between the sample and the component's mean, which measures how far the sample is from the component's distribution, considering its variance. In the OpenCV implementation, this threshold for determining closeness is controlled by the parameter **VarThreshold**, which sets the maximum allowable distance for a sample to be considered close to a component. In the Figure 4 we can see the comparison between various VarThreshold values.

Figure 4: MOG2 foreground masks, with same parameters but VarThreshold $= 3, 4, 5$ respectively

In MOG2, the number of components for each pixel are selected dynamically. This allows for the suppression and removal of components that are not sufficiently supported by the data, discarding components when their weight becomes negative. In the OpenCV implementation, it is possible to set the maximum number of guassian functions thanks to the parameter **NMixture**.
At time t we have a training dataset $\mathcal{X}_T = \{x^{(t)}, ..., x^{(t-T)}\}$ in which T is a reasonable time adaptation period. Intrusive foreground objects are typically represented by additional components in the model with small weights ($\hat{\pi}_m$), which contribute little to the overall model and can be ignored when approximating the background. The background, on the other hand, is captured by the first B components with the largest weights, which represent the more stable and persistent aspects of the scene:

$$\hat{p}(\vec{x}|\mathcal{X}_T; \text{BG} + \text{FG}) = \sum_{m=1}^{M} \hat{\pi}_m \mathcal{N}(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m^2 I) \quad \rightarrow \quad \hat{p}(\vec{x}|\mathcal{X}_T; \text{BG}) \sim \sum_{m=1}^{B} \hat{\pi}_m \mathcal{N}(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m^2 I)$$

As we can see in the second formula, $\hat{p}$ is the conditional probability of having the "color" $\vec{x}$ given the current dataset $\mathcal{X}_T$ and being a BG (background) pixel, in fact it's using only the first B components. The number of background components is determined by a threshold c_f, which

defines the maximum proportion of data that can belong to foreground objects without affecting the background model:

$$B = \arg\min_{b} \left\{ \sum_{m=1}^{b} \hat{\pi}_m > (1 - c_f) \right\}$$

If a new object remains static for a period, its weight (initially assigned to a non-background component) gradually increases. Once the weight exceeds the threshold c_f, the object is considered part of the background. This behavior is quantified in the OpenCV implementation[1] by the **Background Ratio**, defined as $fTB = 1 - c_f$, which indicates the proportion of the background in the scene relative to the total image or frame.
The time required for an object to become part of the background is linked to the background ratio and can be approximately expressed as $\frac{\log(1-c_f)}{\log(1-\alpha)}$, where α is the learning rate, usually kept as $\alpha = 1/T$. For example, with $c_f = 0.3$ (i.e., background ratio = 0.7), it takes approx. 21.2 frames (1.77 seconds at 12 FPS) for an object to be considered part of the background. [5] In the following Table we can see some examples values derived from the previous formula.

Frames of stability comparison					
c_f	Background Ratio	History	Learning Rate	Number of Frames	Seconds (FPS=12)
0	1	60	0.01 (= 1/60)	0	0
0.3	0.7	60	0.01	21.2	1.76
0.3	0.7	600	0.001 (= 1/600)	213.8	17.81
0.5	0.5	60	0.01	41.24	3.43
0.7	0.3	60	0.01	71.6	5.96
0.7	0.3	600	0.001	721.7	60.14
1	0	60	0.01	$\to \infty$	$\to \infty$

In the MOG2 algorithm, the model adapts by selecting the minimum number of Gaussians needed to represent each pixel, ensuring that the sum of their weights accounts for $1 - c_f$ percent of the observed data. If a new object doesn't fit into any existing component, a new Gaussian is created. The more frequently an object is seen, the higher its weight becomes, eventually contributing to the background once it surpasses the threshold. Objects that are rarely seen retain small weights and have little impact on the background model, allowing the model to dynamically adapt based on object stability and frequency in the scene.
Another important parameter for MOG2 is **History**, which defines the number of previous frames that are considered when updating the background model. This parameter influences how much the model is affected by older frames: a larger history value means that the model considers more frames, resulting in slower adaptation, while a smaller history allows the model to respond more quickly to recent changes in the scene.

3.3 CNT

The **BackgroundSubtractorCNT** algorithm is designed with an emphasis on efficiency, speed, and simplicity. Unlike more complex methods, such as MOG2, which model pixel color distributions over time, CNT adopts a frame count approach. The core idea behind CNT is that if a pixel remains unchanged for a specific number of consecutive frames, within a tolerance for lighting variations, it is classified as background. This frame-counting mechanism makes CNT computationally lighter and faster, which is particularly beneficial for low-end hardware platforms, such as the Raspberry Pi, where CNT has been shown to run more than twice as fast as MOG2.
Despite its performance advantages, CNT comes with certain trade-offs. One of the main drawbacks of the CNT algorithm is the presence of noise, which can result in background artifacts lingering across several frames. These artifacts negatively affect the clarity of the segmentation, as some areas in the frame may appear incomplete or noisy. For instance, parts of the foreground, such as clothing items like shirts, may remain well-defined, but other areas, such as pants, may appear poorly segmented or entirely missing. Additionally, ghosting effects are observed, indicating difficulty in distinguishing moving objects from the background. This ghosting is especially noticeable when the scene changes rapidly, leading to inconsistencies in the segmentation and a loss of foreground stability.
In comparison to more robust methods like MOG2, CNT's segmentation results tend to be less

[1] https://github.com/opencv/opencv/blob/3e43d0cfca9753bcc4983f610b75d70c3f25f0cd/modules/video/src/bgfg_gaussmix2.cpp#L359

reliable. While MOG2 adapts well to complex, dynamic backgrounds and provides a higher degree of consistency in foreground extraction, CNT prioritizes performance and real-time processing capabilities. This makes CNT an excellent choice for applications where speed is crucial, but the trade-off is reduced reliability in handling more intricate dynamic environments. Furthermore, the algorithm benefits from parallelization, which allows it to scale effectively with processing power when available.
Overall, CNT's lightweight design and C++ optimizations make it an excellent choice for real-time applications that require high speed, such as object detection on resource-constrained devices. However, to ensure the best performance, careful tuning of its parameters is required to balance responsiveness with background stability, as rapid changes in the scene can introduce unwanted ghosting effects.

4 Algorithms Structure

We structured our motion detection approach as a **multi-time-horizon approach** algorithm, distinguishing between long-term and short-term elements. This means dividing the detection phase into two tasks:

- *Short-term elements*: objects that move quickly across the scene (fast-moving objects), such as a person, which rarely remains in the same position between consecutive frames.
- *Long-term elements*: objects that move slowly or have been recently displaced, such as books or furniture that have been moved or removed.

By structuring the algorithm this way, we were able to keep long-term elements in the foreground instead of let them be absorbed into the background. We chose this approach because we believed that any element that had been moved at least once should remain in the foreground, preserving a trace of its movement. For example, in a security camera system, it is good to have an indication of a removed object, making it easier to quickly identify which objects have been stolen.

From the previous section, we have identified three main methods for motion detection, namely Temporal Median Filter, MOG2 and CNT. We observed that algorithms work well but are suited for different aspects of motion detection.
The **Temporal Median Filter** is particularly effective at detecting moved object in the scene with more precision than MOG2. For example, in some images, the Temporal Median retained objects such as books, much longer than MOG2. We used it to detect the long-term elements.
On the other hand, MOG2 and CNT performed well in detecting the shape of fast-moving objects, such as a person in the scene. Since the CNT algorithm performed slightly worse with respect to MOG2, because of ghosting and noise, we decided to exclude it from our final approach. So, **MOG2** was employed to detect short-term elements and to identify regions of interest, which, when subtracted from the Temporal Median foreground, helped to identify long-term elements.

Let's briefly describe the main steps of the implemented algorithm, represented also in the Figure 5.

`Step [1] and [2]: Temporal Median Filter and MOG2`
As a way to subsample over time, we introduced a counting variable, which helped as understand when to skip some frames. After capturing the current frame, if the time subsampling say so, we appended the frame to the frame buffer and recomputed the Temporal Median Filter Background. Right after that, we applied the MOG2 foreground check and updated its background model.
After every foreground mask computation, we have added a Post Processing step, that, more often that not, helps to obtain better segmentation. We divided it into three main steps, *denoising*, *pixel addition* and *contour filling*.
Please refer to the Jupyter Notebook `experiments/PostProcessing.ipynb` for further details.

`Step [3] and [4]: RoI extraction and fgmasks combination`
First of all, we performed the background subtraction between the estimated reference frame and the updated temporal median background, obtaining a foreground mask representing various objects.
At this point we have two different foreground masks obtained from the two different algorithms. To improve the global foreground mask we combine them only in the Region of Interest (RoI) in which the person is detected. This RoI is dictated by the foreground estimated from the MOG2

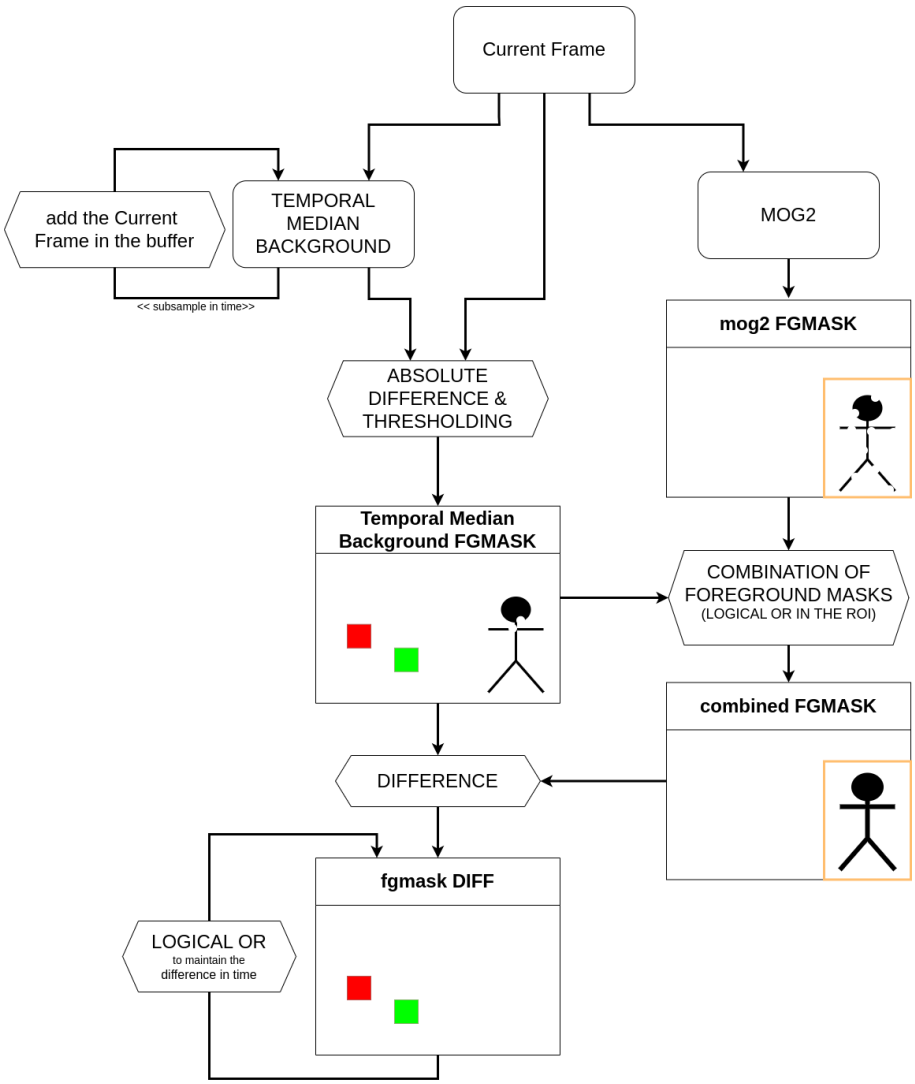

Figure 5: MANTIS algorithm structure

and then used in logical OR with the one estimated from the Temporal Median Filter.

`Step [5]: Static Object Detection`
To detect the long-term moved elements, we performed a bitwise AND between the foreground mask obtained from the Temporal Median Filter and the inverted (logical NOT) of the previously described combined mask. This operation allowed us to use the difference between the temporal median filter and the combined mask to identify objects that are now present in the foreground mask of the temporal median but were not detected in the MOG2's foreground. By doing so, we highlighted the difference between the two time horizons.

`Step [6]: Preserve Seen Differences`
Even if we have used different time scales, at some point also the Temporal Median Filter will absorb the books in its background model as soon as their pixel's values reach the median of the buffer. As previously said, we wanted to keep highlighted all the seen differences, to do so, we perform a logical OR with the previously detected frame differences in order to maintain them in the foreground mask even if the model is no more getting them.

`Step [7]: Connected Component Analysis`
At each frame, we have done a Connected Component Analysis in order to detect blobs and classify them as person or objects. In particular, we discriminated blobs relying on their size.
Our algorithm is not an object tracking algorithm but rather a motion detection one, so it doesn't really track in time the same blob. We could have accomplish that with the motion of the centroids of the detected blobs but it would have introduced a lot of tuning. To execute this task, an extra

layer of algorithms is essential that utilizes the prior segmentation to track the objects.
In this step we also analyzed if the detected objects have been removed from the scene, based on the Canny Edge Differencing. We defined a specific method that, estimating some region of interest (RoI) related to the objects, defines if an object is a "ghost", computing, inside the RoIs, the canny difference between the actual frame and the background.
In particular we can notice that, subtracting one image from the other, if in the first one there are no edges for the RoI defined, the result is near zero. Using this observation, we defined this assumption: if the object was removed from the scene, the difference between the frame and the background is near zero, instead the difference between the background and the frame is higher than zero.
However, the objects will be absorbed in the background after few frames, so the difference between the frame and the background will become always zero. For that reason, we inserted a second condition to identify "phantom" objects: if the object removed some frames ago from the scene, the difference between the background and the long-term mask is near zero, instead the difference between the long-term and the background is higher than zero.

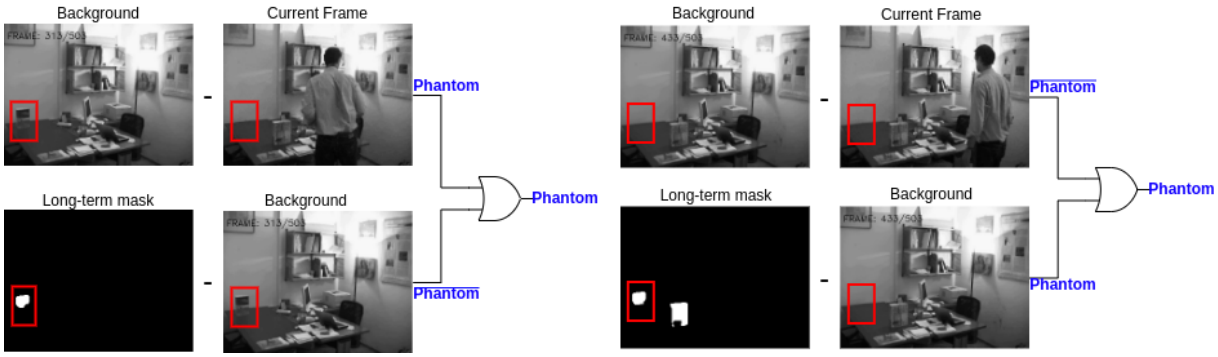

Figure 6: Phantom detection algorithm. On the left, we observe short-term recognition, while on the right, we see long-term recognition, where the background has "absorbed" the changes, but the mask of the seen differences has not.

4.1 Masked Array

In the very few seconds of the video in which the person is moving right across the same spot over and over, both Temporal Median and MOG2 are "absorbing" him into the background model.
To improve this part we decided to try implementing a **selective update** for the temporal median using the `numpy.MaskedArray` module in order to insert in the buffer a `masked frame` i.e. a frame and its associated boolean validity mask. The results obtained by this approach, on the TemporalMedian foreground mask was qualitative better. But the main problem were:

- higher computational time
- due to combination with MOG2, the problem was only half solved.

In our approach the RoI in which the combination of the foreground masks happens, is dictated by the short-time horizon algorithm, i.e. MOG2. A quasi-static object wouldn't result in the combined foreground mask, even if the TemporalMedian is correctly detecting it. For these reasons, we decided against incorporating this part. For further details please cfr. the Jupyter Notebook: `experiments/MaskedArray.ipynb`.

References

[1] Sumati Manchanda and Shanu Sharma. "Analysis of computer vision based techniques for motion detection". In: 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence). 2016, pp. 445–450. DOI: 10.1109/CONFLUENCE.2016.7508161.

[2] Pierre-Marc Jodoin, Yi Wang, and Marc Droogenbroeck. "Overview and Benchmarking of Motion Detection Methods". In: CRC Press, June 2014. Chap. 24. ISBN: 978-1-4822-0537-4. DOI: 10.1201/b17223-30.

[3] Thierry Bouwmans. "Traditional and recent approaches in background modeling for foreground detection: An overview". In: Computer Science Review 11-12 (2014), pp. 31–66. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2014.04.001. URL: https://www.sciencedirect.com/science/article/pii/S1574013714000033.

[4] M. Piccardi. "Background subtraction techniques: a review". In: 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583). Vol. 4. 2004, 3099–3104 vol.4. DOI: 10.1109/ICSMC.2004.1400815.

[5] Zoran Zivkovic and Ferdinand van der Heijden. "Efficient adaptive density estimation per image pixel for the task of background subtraction". In: Pattern Recognition Letters 27.7 (2006), pp. 773–780. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.11.005. URL: https://www.sciencedirect.com/science/article/pii/S0167865505003521.