applied sciences

Article

Active Safety System for Urban Environments with Detecting Harmful Pedestrian Movement Patterns Using Computational Intelligence [†]

Juan Chavat [1,*], Sergio Nesmachnow [1], Andrei Tchernykh [2,3,4] and Vladimir Shepelev [3]

1 Facultad de Ingeniería, Universidad de la República, 11200 Montevideo, Uruguay; sergion@fing.edu.uy
2 CICESE Research Center, 22860 Ensenada, BC, Mexico; chernykh@cicese.mx
3 Automobile Transport, South Ural State University, 454080 Chelyabinsk, Russia; shepelevvd@susu.ru
4 Ivannikov Institute for System Programming, 109240 Moscow, Russia
* Correspondence: juan.pablo.chavat@fing.edu.uy

† This is extend version of conference paper Computational Intelligence for Detecting Pedestrian Movement Patterns, in Ibero-American Congress of Smart Cities, Soria, Spain, 26–27 September 2018.

Received: 17 November; Accepted: 9 December 2020; Published: 17 December 2020

Abstract: This article presents a system for detecting pedestrian movement patterns in urban environments, by applying computational intelligence methods for image processing and pattern detection. The proposed system is capable of processing multiple images and video sources in real-time. Furthermore, it has a flexible design, as it is based on a pipes and filters architecture that makes it easy to evaluate different computational intelligence techniques to address the subproblems involved in each stage of the process. Two main stages are implemented in the proposed system: the first stage is in charge of extracting relevant features of the processed images, by applying image processing and object tracking, and the second stage is responsible for the patterns detection. The experimental analysis of the proposed system was performed over more than 1450 problem instances, using PETS09-S2L1 videos, and the results were compared with part of the Multiple Object Tracking Challenge benchmark results. Experiments covered the two main stages of the system. Results indicate that the proposed system is competitive yet simpler than other similar software methods. Overall, this article provides the theoretical frame and a proof of concept needed for the implementation of a real-time system that takes as input a group of image sequences, extracts relevant features, and detects a set of predefined patterns. The proposed implementation is a reliable proof of the viability of building pedestrian movement pattern detection systems.

Keywords: computational intelligence; image processing; pedestrian movement patterns; surveillance cameras

1. Introduction

Nowadays, there is a growing trend of installing security and surveillance cameras, with the main goal of increasing security in public spaces, streets, offices, and private facilities [1]. Standard systems that handle traditional security cameras do not use automatic methods for detecting incidences in real-time. These surveillance systems are mainly based on operational centers that are in charge of receiving, storing, and analyzing the images and take actions when certain events happen. Operational centers needed to process images from security cameras are operated by technical professionals that play the role of visualizing agents. The job of each visualizing agent consists in constantly observing an (often large) number of images from different sources (security cameras), detecting and generating alerts in case an event of interest is observed [2]. Because of the high personnel costs, most operational centers assign to each visualizing agent a significantly large number of image sources, which could exceed their capacity. Therefore, two phenomena occur, which reduce the surveillance capabilities of the operational center. On the one hand, a degradation of the global attention or the visualizing agents, which are forced to just pay attention to a reduced number of image sources at a time, ignoring events from the rest of the sources. On the other hand, the fact that visualizing agents are human can cause them to feel bored and/or fatigue due to the monotony of the task. It frequently causes poor results.

This article presents an approach applying computational intelligence to overcome the attentional problem of human visualizing agents that works in operational centers, making use of different techniques for image processing and pattern detection. A system capable of processing in real-time multiple image/video sources is proposed to help human visualizing agents in the process of identifying pedestrian movement patterns.

The system focuses on harmful patterns in pedestrian movements, considering common situations for pedestrian safety as defined by the World Health Organization [3], including patterns that can cause a risk of harm (robbery, agglomeration, prowling, etc.) or have negative impacts on pedestrian safety (running, sudden direction changes, walking in forbidden areas). Furthermore, the proposed system is flexible to include/detect different pedestrian movement patterns, which are relevant for each studied scenario.

The proposed system is based on a *filter and pipes* architecture that makes it easy to exchange and evaluate different computational intelligence techniques in each stage of the process. The system is comprised of two main stages. In the first stage, image processing and filtering techniques are applied to images generated from multiple sources, extracting relevant features of images and discarding the ones that are not of interest, according to pre-loaded rules. This stage allows visualizing agents to focus their attention efficiently, on important images. The second stage is responsible for the detection of patterns, taking into account typical situations arising in surveillance and security that are worth identifying (e.g., people running, agglomerations, prowling). The system architecture and design allow extending its capabilities without significant effort, as it is easily adaptable for detection of different types of events of interest and different computational intelligence methods can be incorporated easily in the flow of the system.

This article is an extended version of our conference article "Computational intelligence for detecting pedestrian movement patterns" [4] presented at Iberoamerican Congress on Smart Cities. The main contributions of the research reported in this article are multi-fold. First, we extend our understanding of what and why existing techniques and methods are applied in the areas of image processing and pattern detection. Secondly, we propose and study a system that conveniently combines the surveyed techniques and methods to be capable of processing multiple sources of images in real-time and detect potential events of interest. Specific computational intelligence methods (Hungarian algorithm, Kalman filters, classification) are applied to handle the particularities of pedestrian movements. Finally, to demonstrate the practical applicability of the proposed scheme and study its properties, we conduct experiments over several benchmark videos from PETS09-S2L1 [5] and the comparative analysis of results with state-of-the-art methods from the Multiple Object Tracking (MOT) Challenge competition. The insights from our study contribute to understanding the design methodology, the state-of-the-art of related works, architecture, and components of the proposed system. We proposed to develop a system based on simple image processing and computational intelligence techniques, to provide an efficient solution to be applied in real-time and realistic scenarios. Thus, we focused on the utilization of free software libraries and standard methods for pattern detection, which have been properly extended, configured, and integrated into the proposed system.

The rest of the article is structured as follows. Section 2 contains a brief theoretical introduction to image processing and pattern detection. A review of related work on recognition and pattern detection and tracking on surveillance systems is presented in Section 3. Section 4 presents the general architecture and design of the system. The main implementation details of the proposed system are described in Section 5. Sample results from the evaluation are presented in Section 6. Finally, Section 7 presents the conclusions and the main directions for future work.

2. Methodology: Image Processing and Pattern Detection

This section presents a brief introduction to image processing and pattern detection for surveillance systems.

2.1. Image Processing

Image processing is defined as the process of applying computational techniques in order to modify, improve, change the appearance, or obtain information from images [6]. A standard image processing flow includes five steps (capture, preprocessing, segmentation, feature extraction, and object identification), where the output of each phase is the input of the next one:

1. *Capture* consists in acquiring raw images from a source (e.g., surveillance cameras). In this step, noise and other types of degradation such as blurring, high contrast of the scene, and others, are always added to the image, depending on the specific device used [7].
2. *Preprocessing* applies techniques to remove or reduce the information in those images that are not of interest for solving the problem. The main goal of the preprocessing step is to improve those characteristics of the image that are important for solving the problem (e.g., contour and shine), by using mathematical tools.
3. *Segmentation* consists of splitting each image into regions that represent different objects or background, based on contour, connectivity, or in pixel based characteristics (e.g., shades of gray, textures, gradient magnitude). Some authors state that segmentation algorithms focus on two relevant properties of images: discontinuity and similarity [8], while others add a third property: connectivity [9]. The output of this step is a binary representation of the original image.
4. *Features extraction* consists in finding, selecting, and extracting relevant features of an image, which allow identifying objects of interest for the problem. The features identified in the image must satisfy three properties: robustness, discrimination, and invariance [6].
5. *Object identification* is responsible for categorizing the set of features extracted in the previous step. In this step, different decision models are applied, such as supervised classifiers.

2.2. Pattern Detection

Pattern detection is the study of how computer programs can observe a context, learn, and classify patterns of interest, allowing to make intelligent decisions [10]. A pattern detection system consists of procedures that partition the universe of classes in such a way as to be able to assign elements to classes depending on a set of characteristics of each studied element (the characteristics pattern). On the one hand, when patterns are unknown a priori, the process is called *pattern recognition*; on the other hand, when the patterns are known, the process is called *pattern matching*. The pattern detection process usually consists of three stages: segmentation, features selection, and extraction and classification. The main details of each stage are commented next:

1. *Segmentation* in pattern detection systems is similar to the segmentation applied in image processing systems. The goal of the segmentation stage is to simplify the input, resulting in a set of information that is easier to process.
2. *Feature extraction* takes as input the result of the segmentation stage. The input is processed to extract relevant information about specific objects, remove redundant/irrelevant information, to reduce the dimension of the problem. Quantitative (e.g., speed, distance) and qualitative (e.g., occupation, sex) features are extracted and used to build a vector of features. The goal is to select a subset of features (from the original set) in order to optimize a predefined target function. Feature selection can be done by applying statistical techniques. This procedure usually requires

a deep knowledge of the problem. Selection feature methods consist of three components: at least one evaluation criterion, a procedure or search algorithm and a stop criterion [11].

3. *Classification* processes features a vector to assign features to specific classes, according to predefined metrics. Classification methods (i.e., *classifiers*) depend on the nature of the problem and, in general, their performance also depends on the quality and number of extracted features. There are two main groups of classifiers: *supervised*, and *unsupervised* [12]. Supervised classifiers are based on a set of elements, known as *training data*, for which the class they belong is previously known by the classifier [13]. Some typical supervised methods are Bayesian, Support Vector Machine (SVM), *k*-nearest neighbors (*k*-NN) and neural networks, among others [14]. Unsupervised classifiers tries to discover the classes of a given problem from a set of elements of which the classes are unknown [13]. The number of classes to be discovered by an unsupervised classifier can be known in advance or not, depending exclusively on the datasets handled. Some typical unsupervised methods are Simple Link, ISODATA and *k*-means, among others [14].

2.3. Methodology Applied in the Proposed System

The proposed system consists of two stages for detecting pedestrian movement patterns. In the first stage, it receives images from different sources and applies image processing techniques to extract a set of features from the scene to detect objects of interest (e.g., pedestrians) and ignore the rest of the image. In a second stage, a specific module applies pattern analysis techniques to detect patterns fulfilled by objects of interest detected in the previous stage. The main details of the proposed system are presented in Section 4.

3. Related Works

Surveillance systems and pedestrian movement detection have been the subject of several articles in the related literature.

The survey by Valera and Velastin [15] highlighted the growing importance of intelligent surveillance systems with distributed architecture. Several important issues were identified, including: detection and recognition of moving objects; tracking and detection of anomalous movement patterns in video surveillance, behavior analysis; and recovering/storing images. Existent systems were classified in three categories, according to their *generation*: (i) a *first generation* of analog Closed Circuit TeleVision (CCTV) systems that performed fairly good but were not easy to distribute and store; (ii) a *second generation* of systems that combined computer vision technologies with CCTV for automation, which allowed increasing the surveillance efficiency by providing accurate event detection; and (iii) a *third generation* of automatic systems that cover large areas by a distributed deploy, combining sensors, robust tracking algorithms, and optimized algorithms for managing large volumes of information. Taking into account the classification by Valera and Velastin, the system proposed in our research is within the third generation, as complex pattern detection methods are included and parallel computing techniques are applied in order to deal with multiple distributed sources of data.

Piccardi [16] presented a review of background subtraction methods, describing the main features of seven algorithms and analyzing their performance (processing speed, memory utilization and accuracy). The studied methods included: Running Gaussian Average (RGA), Temporal median filter, Mixture of Gaussians (MOG), Kernel Density Estimation (KDE), Sequential Kernel Density approximation, and Concurrence of image variations. A performance analysis was reported, based on the complexity of applying the methods over each pixel (for the case of processing speed and memory utilization) and on categorizing into limited, intermediate or high (L, M, H) accuracy provided. Analysis showed that Running Gaussian Average obtained the best processing speed and the lower memory utilization, while Mixture of Gaussians and Kernel Density Estimation were the best methods regarding accuracy. Table 1 summarizes the main features (time and space complexity, and accuracy) of the methods studied by Piccardi.

Table 1. Performance of the background subtraction methods (adapted from the work of Piccardi [16]).

Method	Speed	Memory	Accuracy
Running Gaussian Average	O (1)	O (1)	L/M
Temporal Median Filter	O (n_s)	O (n_s)	L/M
Mixture of Gaussians	O (m)	O (m)	H
Kernel Density Estimation	O (n)	O (n)	H
Sequential Kernel Density Approximation	O ($m + 1$)	O (m)	M/H
Concurrence of Image Variations	O ($8n/N^2$)	O ($8nK/8N^2$)	M
Eigen-backgrounds	O (M)	O (n)	M

Lopez [17] presented a system capable of detecting apparent movement on images (caused by camera movements). Two types of methods were reviewed: those that detect movement from the difference of consecutive images and those that detect the movement from a single image (e.g., optical flow methods, occasionally used for background subtraction). The author concluded that methods for detecting from a single image have a higher computational cost and introduce noise to the process. Thus, global alignment methods using points with correspondence were used. The system proposed by Lopez obtains an aligned image without apparent movement and both original and aligned images are sent to a segmentation module composed of three layers that apply background subtraction, labeling, and process grouping. The output of the segmentation module results in a set of regions of interest (*blobs*). Blobs are sent to the tracking module that applies filters, including the algorithm by Stauffer and Grimson [18] and also Kalman filters [19] to decide if apparent movement is detected or not. Results close to 90% were achieved without using tracking and almost 100% using tracking.

Regarding pedestrian detection/tracking, Lefloch [20] presented a system for counting people based on detecting the moment in which a person crosses a previously established virtual line. The author proposed counting people on images captured with a camera placed on the roof, in order to eliminate occlusion problems. The proposed system first applies background subtraction to obtain a binary image, which is used to determine which pixels belong to the bottom and to the front of the area that has movement. After that, morphological operations (e.g., erosion, dilatation, opening, and closure) were applied to eliminate noise and also small, isolated areas that exhibit minimal movement. The resulting image is sent to a stage of detection and classification of blobs, which detects contiguous pixels and calculates its bounding box. Bounding boxes that do not meet certain criteria (e.g., wide-high ratio and area) are discarded and those that potentially contain people are identified.

Rodriguez et al. [21] addressed the problem of detecting and tracking people in very dense crowds. Specific problems arise in this scenario, such as occlusion and change of shape and location of people, which pose big challenges for detection. The proposed approach is based on detecting heads, in order to mitigate occlusion problems when detecting the whole body. Background subtraction and segmentation techniques are not useful in this type of scenario, because they cannot isolate people. Rodriguez et al. applied an object detector, trained to detect human heads, and density estimation algorithms, which provide information about the number of persons within a region (but not their locations). The detector was applied to all regions of the image, generating a map that contains scores that indicate the possible presence of people. The map of scores was combined with data obtained by the density estimation algorithms to obtain accurate detection results.

Dollar et al. [22] analyzed the best approaches for people detection. A set of evaluations were performed for varying scenarios and data sets, studying the performance and limitations of different approaches. Authors concluded that people detectors are far from perfection, even under the most favorable conditions. Significant performance degradation was detected when working with images of people whose area is below 30 pixels or when occlusion is greater than 35%. Several areas were identified to improve people detection, including dealing with images between 30 and 80 pixels, improving performance against occlusions, using movement patterns, and using temporary and context information together with monitoring information.

Leach et al. [23] studied the problem of detecting subtle behavior anomalies in surveillance videos. A decision-making process was presented, taking into account *social signals*, i.e., information about the scenario and social context. Social context is based on the premise that two individuals who share a high degree of information in their trajectories (direction, speed, proximity, and trajectory overlap) have similarities and a 'social dependency'. Four scenarios were considered: traffic, with a high number of trajectories; idle, with stationary trajectories; convergence and divergence, with separate or join trajectories; and general, not classifiable in the previous ones. Pedestrian detection [24] and a follow-up Tracking-Learning-Detection method [25] were applied. The process was focused on detecting human heads to reduce the problems generated by the occlusion. Experiments were performed on data from PETS 2007 and Oxford datasets. Effective results were reported, for example, true positive rate 0.78 and false positive rate 0.19, improving 0.13 in comparison with methods that do not take into account the social context. These results suggest that inferring social connections between people helps improve decision making in the detection process.

Cho and Kang [26] presented an abnormal behavior detection system based on hybrid agents for complex scenarios studying group interaction and individual behavior. In the proposed system, agents are categorized into static and dynamic. Static agents are located at fixed points, in order to compute temporal movement information (e.g., speed and direction) of the objects belonging to the background of the image, considering the optical flow variation. Dynamic agents are assigned to moving objects and they move according to the optical flow. Dynamic agents are responsible for calculating the information about social interaction between neighbors, using a Social interaction Force Magnitude (SFM) model and the potential energy of interaction. The proposed system divides each video into non-overlapping blocks and places a static agent and a dynamic agent in the center of each block of the initial frame. Then, the optical flow field between pairs of consecutive frames is computed, extracting information from the behavior of individuals and groups, using the static and dynamic agents. Feature descriptors are created from the extracted information of each block and then mapped to codewords by a clustering process. Codewords from all the blocks plus the bag-of-word method are used to create feature word vectors. The feature word vectors are then processed by a trained SVM classifier and determine the abnormality of the scene. The system was implemented in Visual C++, using OpenCV and LIBSVM libraries. The reported results allowed concluding that the proposed system outperformed the SFM method over PETS 2009 and UCSD datasets. In addition, the authors argued that static agents help to detect quick movements, movements in restricted areas and abnormal behaviors of individual subjects, while dynamic agents help to detect disordered movements (such as panic situations) and separation movements, which are poorly detected by simple agent systems.

Zhu et al. [27] presented a method for detecting abnormal events in crowded scenes based on modeling motion and context information. Low- or medium-level visual information from surrounding local regions was used as context information. The proposed method focuses on modeling the activity of a crowd using context and motion information, considering that all objects or regions with detected motion provide context information. The method extracts dense trajectories and applies noise filters to filter non-typical trajectories. After that, Multiscale Block–Local Binary Pattern coding on Three Orthogonal Plans (MB-LBP-TOP) was used to encode local context information, and the Multiscale Histogram of Frequency Coefficient (MHFC) feature descriptor was used to describe the motion information and extract dense trajectories. Authors claimed that extracting dense trajectories, preferably with noise screening, using MHFC descriptors has two benefits: (i) the method achieves independence of the processed data and, (ii) it is able to better capture the motion characteristics of trajectories. Based on context patterns and motion information, the model is trained using sparse coding and sparse reconstruction cost is applied to classify events into normal and abnormal. The system was implemented in MATLAB and the experimental analysis was performed over UCSD and Subway datasets, obtaining a processing time of 3.7 and 4.2 s/frame, respectively. The authors concluded that the proposed system computed better results than previously developed methods, arguing that improvements are due to the fact that the proposed model does not take into account only the movement of the crowd, but it also uses context information contributed by surrounding objects. False alarms were detected in case of great perspective distortion or when the size of the objects vary significantly. This issue could be solved using different scales of objects in the training phase.

The analysis of related works indicates that there is still room to contribute regarding efficient systems for detecting pedestrian movement patterns applying computational intelligence techniques.

4. The Proposed Detection System

This section describes the architecture and design features of the proposed system for detecting pedestrian movement patterns.

4.1. Architecture and Design

The proposed system was designed to be able to collect and process images generated from different data sources. In turn, a specific architecture was conceived to be flexible enough to allow replacing or adding new algorithms without significant effort. To assure efficiency, the concurrent processing of multiple data sources must be supported. The processes applied on the images are independent of each other and they adapt correctly to a chain pattern [15,20]. Furthermore, all the applied algorithms are able to take as input the output of their previous immediate(s). The aforementioned schema is modeled by the pipes and filters architecture [28].

Taking into account the previous comments and the review of related works, an architecture based on *pipes* and *filters* was proposed for the developed system. The system consists of two main modules. The *Recognition and Tracking* module is responsible for detecting and monitoring pedestrians (objects of interest); the *Pattern Detection* module analyzes the results of the first module to detect patterns based on (recent) historical information. Both modules support multiple concurrent executions using separate processes. The Pattern Detection module supports processing multiple unrelated data sources. The system also includes three auxiliary modules: *control panel*, *instance launcher* and *events generator*. The Control Panel module graphically shows the received events and sends command orders (which refers to requested actions) to the Instance Launcher. The Instance Launcher is responsible for the creation of Recognition and Tracking module instances, as separate processes. Finally, the Events Generator receives the information of detected patterns and injects them back to the Control Panel. The system modules and the exchanged information are described in Figure 1.

Advanced Message Queuing Protocol (AMQP) protocol is used for communications between modules. AMQP is an open and secure protocol that guarantees delivery on time (or the consequent expiration), uniqueness, and correct ordering of messages, and also data integrity. The following subsections describe each module of the system.

4.2. Recognition and Tracking Module

The Recognition and Tracking module consists of four stages, arranged in pipes and filters. The first filter receives raw images and applies background subtraction, resulting in a binary image. The second filter takes binary images, detects blobs (set of adjacent pixels that belongs to the front of the image) and transfers the set of blobs to the blobs filter, which discards those blobs that do not contain objects of interest and adds spatial information to those relevant blobs. The last stage takes the information of the objects of interest in the image space and associates each one of them to the position of previously detected objects; thus, calculating the movement of each object. The different stages that compose the tracking and recognizing module are presented in Figure 2.

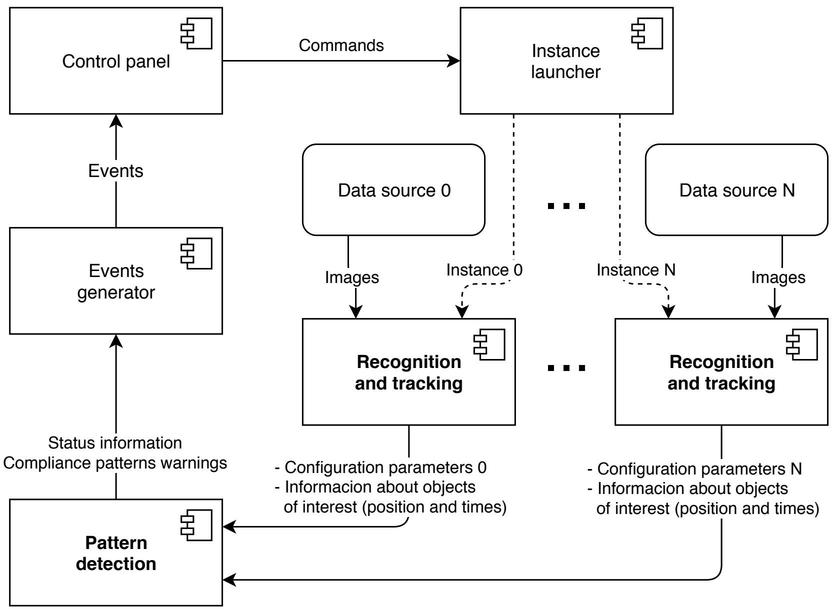

Control panel
Commands
Instance launcher
Events
Events generator
Data source 0
Data source N
Images
Instance 0
Instance N
Images
Recognition and tracking
Recognition and tracking
Status information
Compliance patterns warnings
- Configuration parameters 0
- Informacion about objects of interest (position and times)
- Configuration parameters N
- Informacion about objects of interest (position and times)
Pattern detection

Figure 1. Diagram of the architecture of the proposed system.

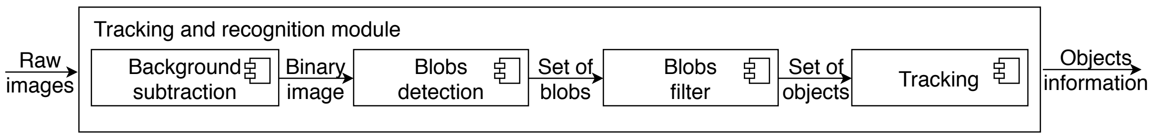


Figure 2. Diagram of components of the Recognition and Tracking module.

4.3. Pattern Detection Module

The Pattern Detection module receives information about objects of interest from multiple instances of the Recognition and Tracking module. The information is stored in a repository that contains the recent history for each object. Periodically, the module processes the last entries to identify a set of features, called *primitives*, which represent basic characteristics of the objects. Characteristics depend on the movement speed, direction, or another attribute of the object. A sequence of primitives plus a set of associated values of properties defines a *pattern*.

Single and multi-target primitives are used. Single-target primitives take into account only one object of interest, ignoring the rest of the objects in the scene, and multi-target primitives takes into account multiple objects in the scene. For example, single-target primitives can determine if a person is standing, walking, or running, depending only on the speed of movement of that person. On the other hand, a multi-target primitive can detect an agglomeration depending on the position of a group of persons for a period of time.

A specific method for pattern detection, based on previous work [29] is implemented. The proposed method for pattern detection takes into account the 'proximity' between an identified sequence of primitives and a set of previously established patterns. Proximity is evaluated using an error function that applies the concept of temporal distance, i.e., the total time of primitives within a sequence that are not included in the reference pattern.

Reference patterns are integrated into the system dynamically. For each primitive that integrates a pattern, a quantifier and a value are defined. For example, a primitive that takes into account the movement speed is fulfilled within a pattern if a pedestrian walks with a movement speed greater or equal to (*quantifier*) 5 km per hour (*value*).

The way patterns are defined and integrated into the system makes them generic, i.e., agnostic of specific information about the data sources and the scenes processed. For instance, an 'agglomeration' is defined by two relevant parameters: the number of people and the time that these people stay within a radius of less than a defined parameter distance. Parameters that define a given pattern keep unchanged along with the different data sources or the scenes, but can be modified according to specific needs.

4.4. Auxiliary Modules

Three auxiliary modules allow simplifying the operation of the main modules of the system and displaying results.

The Instance Launcher module starts instances of the pattern detection, control panel, and events generator modules. After that, it waits for the arrival of command orders. For instance, a command order can require to attend a new source of data, which causes a new instance of the Recognition and Tracking module to be launched. The Control Panel module consists of a web service and a web interface that allows final users to start new processing instances and visualize partial and final results. The Event Generator module stays idle while waiting for results generated by the patterns detector. Generated results, when available, are sent to the event generator. Based on the results received, the event generator generates web events that are sent to all web users using the control panel.

5. Implementation

This section describes the main decisions about technologies and algorithms taken during the implementation of the system.

5.1. Technology Selection

The search of technologies for implementing the system was based on a set of predefined conditions related to the main requirements of a pedestrian movement patterns detection system, including: (i) using a cross-platform programming language; (ii) develop over a programming language without technical complexities (not hard to type, has an automatic memory handler, etc.) and having a broad and active community; (iii) using free libraries and preferably open source; (iv) achieving good performance on all tasks covered by the system: image processing, pattern detection, message passing and management, etc.

After a literature and technology research [30,31], a group of configurations were selected for a deeper study: Matlab, OpenCV over C/C++, and OpenCV over Python. Both OpenCV and Python are free and open source. In addition, Python is a dynamically typed language and counts with an automatic memory manager. Python has a wide variety of free and open source scientific libraries and the community is broad and active. Regarding performance, Python is also an efficient option. The study allowed to conclude that the best choice for implementing the system was using OpenCV library (version 3.0.0 was selected) over Python (version 3.4.3).

5.2. Communication between Modules

The AMQP implementation from RabbitMQ [32] is used for the communication between modules. RabbitMQ was thought to support parallelism and be robust for managing messages. For the connection between Python and the RabbirMQ service, the pika library was used.

In AMQP, *exchange* elements provide the message delivery service, according to instructions about how and where to send them. There are four types of exchange elements: direct, topic, fanout, or header. All data in RabbitMQ are in JSON format, a standard, language independent, and simple format for data exchange.

An exchange of type 'direct' was defined between Recognition and Tracking and Pattern Detection modules. Each instance of the Recognition and Tracking module generates messages that are addressed to a unique queue attended by an instance of the Pattern Detection module. Messages exchanged between the two main modules are of two types: *configuration*, used to attend the instance of recognition and tracking that sends the message; and *data*, which contains precise information about objects of interest. The Pattern Detection module sends its results to an exchange of type topic. Each message contains a key that indicates the message type: commands, state information, and matched patterns warnings. The events generator module binds the exchange with a queue to receive the three types of messages, while the instance launcher module binds the exchange to receive just command messages.

5.3. Recognition and Tracking Module

The main implementation details of the four stages of the Recognition and Tracking module are described next.

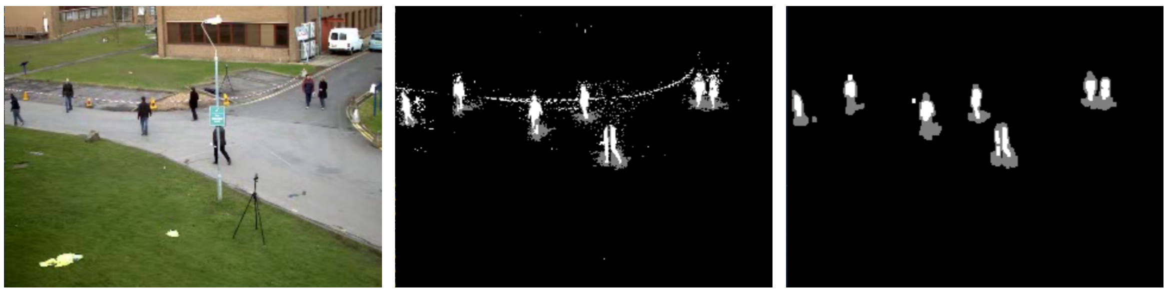
5.3.1. Background Subtraction

Algorithm 1 presents the steps followed by the component in charge of performing background subtraction. The background is subtracted directly from the raw image. First, background subtraction transforms the raw image to an image in grayscale (line 2 in Algorithm 1). The grayscale image allows processing less information and results in a lower processing time. After that, the grayscale image is blurred (line 3). Blurring is a technique for reducing the noise presented in the image [33]. Blur operations are made by the application of filters. In the proposed system, a Gaussian filter is applied. A Gaussian filter applies a convolution in each point of the image using a Gaussian kernel and then returns the summation as the final result. The implementation of the Gaussian filter used in the system is included in the OpenCV library. After blurring the image, the background is subtracted (line 4). Two different methods were integrated into the system for this purpose: Improved Mixture of Gaussians (MOG) [34] and *k*-NN [35]. MOG and *k*-NN are robust methods that provide support for handling dynamic backgrounds too. Both methods are included in OpenCV and a parameter in the configuration of each instance of the Recognition and Tracking module indicates which method is applied.

Algorithm 1 Background subtraction steps

```
1: frame ← raw image
2: grey image ← BGRToGrey(frame)
3: blurred image ← GaussianBlur(grey image)
4: binary image ← BackgroundSubtractor(blurred image)
5: binary image without noise ← MorphologicalOperations(binary image)
6: output ← binary image without noise
```

A binary image is obtained after background subtraction. In the binary image, some elements are detected incomplete or are too close to others, generating a not-desired union of blobs. Morphological operations are applied (line 5) to mitigate these problems. A variety of morphological operations are included in OpenCV: *erosion* allows separating elements that appear together by small contact areas; *dilatation* allows joining nearby elements by applying edge thickening; *opening* consists in applying first erosion and then dilatation; and *closing* is the result of applying first dilatation and then erosion. The result obtained after applying morphological operations is an image with less noise and better identified elements. In the proposed system, two operations are applied: erosion and dilatation. A sample of the processing is presented in Figure 3.



(**a**) (**b**) (**c**)

Figure 3. Background subtraction steps. (**a**) Raw image. (**b**) *k*-NN before morphological operations. (**c**) *k*-NN after morphological operations.

5.3.2. Blobs Detection

The proposed system includes two methods for processing the binary images for detecting blobs: *simple blob detector* (SBD) and *blob detection based on bounding boxes* (BBBD). SBD is a basic implementation of a blob extractor provided by OpenCV [36]. BBBD is a specific method implemented as part of the reported research. It operates in two phases: the first phase consists of detecting the contour of elements and the second phase performs a search of the minimum rectangles that contains the detected contours (the *bounding boxes*). Both methods return a set of rectangles that contains the blobs detected.

5.3.3. Blobs Classification

Blobs classification takes the set of rectangles as input and classifies them into *useful blobs*, i.e., those containing objects of interest, or as *not useful blobs* when not. Not useful blobs are discarded. This stage takes as input the set of blobs and not the entire image and avoids processing image areas without particular interest, therefore, resulting in a better performance. Three different techniques are implemented, which can be applied isolated or in combination with each other, to improve the results of the classification:

- *Aspect ratio* (AR) classifies blobs based on the relation (ratio) between their width and height. If the ratio is close to the average value of the objects of interest, AR indicates that the blob contains at least one object of interest. The major benefit of AR is its low computational cost. However, it tends to be inaccurate because the reference aspect ratio often varies significantly for different data sources. AR is not useful for discarding blobs (i.e., the fact that a blob fulfills the relation does not mean that it contains an object of interest) and wrongly discards blobs that do not comply with the established aspect ratio criterion due to the fact that they contain multiple objects of interest (e.g., objects close enough one of each other that conforms a unique blob).
- *Computational intelligence* uses the *default people detector*, a pre-trained learning algorithm included in OpenCV. Dalal and Triggs [37] demonstrated that a combination of Histograms of Oriented Gradients (HOG) for feature extraction, and Support Vector Machine (SVM) for the classification of the feature vectors, allows obtaining accurate detection results. The method is based on moving a gridded window all over the image, extracting the vectors of features (using HOG) and classifying them (using SVM) to decide if the image contains a person. Considering that the proposed system studies the movement of persons, the blob classification technique processes just those areas where movement was detected. Thus, the default people detector algorithm is applied just over each detected blob, reducing the computational cost of the processing. This method returns a set of rectangles that contains persons, some of them overlapped. To reduce and unify the number of rectangles, the *Non-Maximum Suppression* algorithm [38] is used.
- *Aspect ratio frequency* filters blobs depending on the frequency that similar blobs were filtered by computational intelligence algorithms. In this way it is possible to simulate a behavior close to the computational intelligence algorithms without having to execute them in each iteration. Aspect ratio frequency can be applied in two different ways: (i) directly filtering those blobs in which the frequency is greater than a threshold, or (ii) categorizing the blobs frequencies into three different levels and depending on that level, discarding the blob, requesting a computational intelligence method to process the blob, or keeping the blob.

5.3.4. Tracking

This stage determines the one-to-one correspondence between detected objects of interest in the current and previous frames. Specific techniques are applied to handle pedestrian movements, which have special features. Since pedestrians are autonomous and can make irregular movements, they differ from other pattern recognition systems, like the ones used to process traffic videos, where vehicles tend to move in more regular patterns and follow normalized rules.

A specific variant of the Hungarian algorithm [39] was developed for this purpose. The Hungarian algorithm receives as input a set of blobs, a set of objects of interest, and a cost function, and returns a correspondence between both input sets that optimizes the defined cost function. The original Hungarian algorithm only accepts inputs of the same size, thus the result is always surjective. A modified implementation was developed to allow the system to handle a different number of blobs than the number of objects of interest. This way, it is possible to process those cases where the number of blobs detected is lower than the objects of interest in the previous frame, or vice versa. In addition, the modified version declares invalid all correspondences in which the cost is greater than a certain threshold, assuming that the blobs do not correspond to the tracked objects.

The cost function used in the proposed system has three components, weighted according to specific parameters in the instance file configuration: (i) the distance between the position of an object in the previous frame and the current position of the blob; (ii) the distance between the predicted position of the object for the current frame and the current position of the blob; and (iii) the difference between the colors of the blob that contained the object in the previous frame and the color of the blob in the current frame.

The position of a blob is not always accurately adjusted to the shape of the objects. As a consequence, the raw trace of an object can suffer zig-zag movements, making it difficult to track the object and detect movement patterns. Kalman filters [19] are applied to avoid the zig-zag effect and to predict the next probable position of each object.

The Kalman filters method keeps the state of the objects, updating it in each frame based on a prediction and correction model (considering position, speed and acceleration for each person). The prediction uses a matrix, known as *transition matrix*, to predict the next state. The correction uses the information from the prediction plus a matrix, known as *noise matrix,* to calculate a correction of the predicted state. In addition, a *gain* parameter is used to smooth the trace, avoiding the zig-zag effect. Empirical results demonstrated that the smoothness achieved by a Kalman filters method is not enough to achieve an accurate tracking, as occlusions of objects are too frequent. In order to mitigate this problem, the system applies an extra smoothing process using Kalman filters smoothing in *fixed-lag smoother* mode. In a given time (state t), this method uses the data of the last $N-1$ Kalman filter iterations to improve state $t-N$. The cost of the extra processing method is a delay in the detection of patterns in the order of the time needed to process the last N iterations. The implementation for Kalman filter smoothing provided by the FilterPy library was used.

Two structures were implemented to store information of different objects and their tracking, and to resolve occlusions: *tracklets*, associated to a unique object, to store and update the relevant tracking information (position, color, frame when it appears, last frame when its object was not occluded, etc.) and *groups*, used to store tracklets and associate blobs to frames. Ideally, each group should have a single tracklet and a single blob associated. This is the case when the blob represents the object that is tracked by the tracklet. However, a single blob can be associated to a group with many tracklets. This is the case when a multiple occlusion occurs, i.e., multiple objects are close enough to form a single blob. In turn, when many blobs and many tracklets are associated to a single group, it is the case in which multiple objects where previously in occlusion and are splitting up at this moment. In this last case, the system divides the group in order to obtain a single blob per group.

Tracklets are updated or removed in each iteration of the tracking algorithm, depending on: (i) the groups they belong to, (ii) how long the tracklets have belonged to the group, or (iii) how long the *tracklet* has been in the system. On the other hand, a *tracklet* can be removed from the system due to three reasons: (i) the object tracked by the tracklet is the result of a ghost blob, i.e., a newly created blob resulting from noise in the cameras or in previous steps; (ii) low tracking confidence of the object, which happens when the tracklet has not been associated to a one-to-one group for a certain time; and (iii) because the object definitely disappears from the scene.

Three levels are considered for tracklet information updating: (i) *correction with maximum confidence*, when a tracklet is associated one-to-one to a group, the blob of the group represents the tracked object and the tracklet is updated with the information of position and appearance of the blob; (ii) *correction with minimum confidence*, when an object suffers multiple occlusion for a certain time, the tracklet is not associated one-to-one to a group, the predicted position is no longer trustworthy and the tracklet is updated with the position of the blob that represents the occlusion; and (iii) *prediction only*, when a tracklet was recently associated one-to-one to a group, it is assumed that the predicted position is reliable and no correction is made (e.g., when objects are occluded by a short time or two paths cross each other); this level makes it possible to keep tracking positions of the objects even when there is no blob assigned in the current iteration. The pseudocode in Algorithm 2 synthesizes how the system tracks the objects, frame by frame.

Algorithm 2 starts applying the Hungarian method to determine the correspondence between the blobs of the current frame and the objects detected in the previous frames (line 1). Then, it iterates over the blobs detected for the current frame (line 2) and, if the blob is in correspondence with a tracklet, this is added to the group of the tracklet (lines 3–4), otherwise a new tracklet is created, including the blob and a new group containing the tracklet (lines 6–7), because it means that the blob represents a new object in the system. The next step is iterating over the groups that have no blobs assigned (line 10). For each of these groups, the system looks for the closest blob that overlaps the tracklets of the group; if such a blob does not exist, the system looks for the closest blob. In the case that a blob was found, all the tracklets of the group are moved to the group that contains the blob in question (lines 11). Then, each blob is processed in accordance with the number of blobs assigned to it (lines 13–36). Three different situations are possible:

- The group has no assigned blobs. If the tracklet was recently updated with a blob position, then it will be updated in the current frame with the prediction values (line 17), otherwise, the tracklet is removed from the system, assuming that the objects that were tracked disappeared from the scene (line 19).
- The group has a single assigned blob. If the group has just one tracklet (the ideal case), the tracklet is updated with the blob data (line 24). Otherwise, if the group has more than one tracklet, it is the case where many objects are occluded in one blob. In this case, the system first deletes the tracklets with ghost blobs and then deletes all tracklets, except the most reliable, updating it with the blob data (line 26).
- The group has more than one assigned blob. This is the case where objects that were occluded are splitting up. To avoid any loss of information between iterations, each blob is assigned to a new group and the Hungarian algorithm is executed (up to two times, using different weights) between the tracklets of the current group and the blobs associated to it (line 30). After applying the Hungarian algorithm, if there are still many tracklets assigned to a single blob in a group, it is a situation where occlusions and/or ghost blobs happen. In this case, the method in line 26 is applied again.

Algorithm 2 Tracking steps

```
1:  calculate assignations between tracklets and blobs using Hungarian algorithm
2:  for each blob in the system do
3:    if assigned to a tracklet then
4:      add blob to the group of the tracklet
5:    else
6:      create tracklet containing blob
7:      create group containing the just created tracklet
8:    end if
9:  end for
10: for each group without assigned blobs do
11:   move tracklets to the group of the closest blob, or  the group of the blob that contains it
12: end for
```

Algorithm 2 *Cont.*

```
13: for each group in the system do
14:     if has no blobs assigned then
15:         for each tracklet in the group do
16:             if object disappears for a moment then
17:                 update position with prediction
18:             else
19:                 remove the tracklet
20:             end if
21:         end for
22:     else if has one blob then
23:         if has one tracklet then
24:             update it with the blob information
25:         else if has many tracklets then
26:             resolves with one-blob-to–many algorithm
27:         end if
28:     else
29:         if has more tracklets than blobs then
30:             resolves with many–to–many algorithm
31:         end if
32:     end if
33:     if the group has no tracklets then
34:         remove the group from the system
35:     end if
36: end for
```

5.4. Pattern Detection Module

The Pattern Detection module is capable of processing multiple data sources concurrently. The module consists of two stages:

1. The first stage receives messages from multiple instances of the Recognition and Tracking module and routes them depending on the source identifier. Two types of messages exist. The first type of message is the least common (but always necessary) and correspond to the ones used when processing requests by new instances of recognition and tracking. When a request arrives, the Pattern Detection module creates the structures to handle data from the new data source identified in the message, and configuration values in the message are applied to process data from the respective data source. The second type of message is used for communicating data of detected objects. When these data messages arrive, they are routed to the structures previously created to handle the data source.
2. The second stage receives data of the detected objects and has the patterns definition, the recent history of primitives fulfilled by each detected object, and all the logic needed to check patterns compliance. Patterns are defined as a sequence of primitives, defined by a 'primitive type', an 'event type' for each type of primitive, and a 'quantifier' that defines how the values of the met primitives are compared with that required by the pattern. For example, the primitives defined in the implemented system are `SPEED`, `DIRECTION`, and `AGGLOMERATION`. Each primitive type has events, for example, for the `SPEED` primitive, three events are possible: `WALKING`, `RUNNING` and `STOPPED`. The quantifiers defined in the proposed system are `LE`–lesser or equal, `GE`–greater or equal, `AX`–approximate, `EQ`–equal and `NM`–irrelevant value.

6. Validation and Results

This section presents validation results of the proposed system for detecting pedestrian movement patterns. Validation is performed for a case study in Montevideo, Uruguay, relying on security patterns, which are the most relevant for Ministerio del Interior, the ministry in charge of the public security.

6.1. Recognition and Tracking Module

The validation of the Recognition and Tracking module was performed using a video from scenario S2.L1 of the PETS09 dataset. PETS09 videos were recorded using a fixed camera installed at a higher height than the head of the people, at a rate of 7 FPS with a resolution of 768 $\times$ 576 pixels and natural light [5]. After 1:54 min of the video, 19 people get in and out of the scene and walk around, generating multiple occlusions among them and with objects of the scene.

A good performance of the module is characterized by an accurate tracking, where data are processed and sent to the Pattern Detection module in real-time (i.e., in less than a second). Thus, the metrics used in the experimental analysis focus on the final result of the module and not in partial filter results. In order to evaluate the processing efficiency, the time required to process each frame was collected and used to calculate the average and maximum processing time per frame. The MOTChallenge benchmark, a unified evaluation platform created by Leal-Taixé et al. [40], is used to evaluate the tracking accuracy. Experiments were executed on Xeon Gold 6138 processors with 128 GB of RAM, from the National Supercomputing Center (Cluster-UY), Uruguay [41].

The MOTChallenge benchmark consists of three components: (i) a public dataset including its own and well known videos (some of them including ground truth information, like PETS09-S2L1, which is used for the evaluation of the proposed system); (ii) a centralized evaluation method that allows the comparison of results obtained with different methods; and (iii) an infrastructure that makes the crowdsourcing of new data possible, new evaluation methods, and new notations (i.e., ground truth).

The evaluation method included in MOTChallenge provides several metrics. One of them, Multiple Object Tracking Accuracy (MOTA) was applied to evaluate the tracking accuracy of the proposed system. MOTA is a percentage that combines three indicators: false positives, false negatives, and identity changes of the tracked persons. The greater the MOTA value is, the more accurate is the tracking of the persons. Furthermore, the average and maximum difference between the number of persons in each frame (from the ground truth) and the detected tracklets and blobs in each frame are evaluated.

The system has a set of parameters that determine how accurate the module performs in a given scene. Thus, previously to the evaluation, a set of experiments was performed to find the best parameter configuration. Since the module has a pipes and filters architecture, the performance of each filter depends only on its configuration values, so finding the best configuration values for each filter results in the best configuration for the entire module. Taking into account these considerations, different values for 40 parameters were studied. A total number of 1458 experiments were performed. For each executed block, three configurations were selected to process the next block. To find the best performing configuration, the following three criteria were taken into account: (i) higher MOTA value (M-MOTA), (ii) lower average difference in the person counting (M-Count), and (iii) from the ones with higher MOTA value, the one with lower average processing time per frame (M-MOTA-TC). Full details of configuration experiments and best values are reported in the complete execution plan available in the project website fing.edu.uy/inco/grupos/cecal/hpc/APMP.

The baseline for the comparison was the manual configuration obtained by empirical tuning and used during the development of the system. The three configurations that achieved the best results in the configuration experiments used *k*-NN and were able to improve the MOTA value up to 14.8% and the person counting up to 34%. In addition, the highest MOTA value obtained by the system (52.7) was higher than the average MOTA value (36.6) obtained by algorithms in the 2D MOT 2015 benchmark [42]. However, it is worth noting that results for the proposed system correspond to an evaluation performed on a single scene, while results for the 2D MOT 2015 benchmark correspond to averages obtained from applying different algorithms to multiple scenes.

Regarding the other metrics, the obtained average processing time per frame was similar for all three configurations, between 0.024 and 0.046 s, being the blobs classification the filter that requires the most processing time. The maximum processing times per frame are in the range of 0.058 and 0.103 s. The complete average and maximum processing times for each stage are reported in Tables 2 and 3, respectively. An important result for the proposed system is that the average processing time is lower than 0.05 s for all cases. This value indicates that the proposed system allows processing in real-time a 20 FPS data source, which is twice the number of frames required for detecting pedestrian movements [43].

Table 2. Average execution time (in seconds) of the evaluation using the three best configurations.

Configuration	Background Subtraction	Blobs Detection	Blobs Classification	Tracking	Total
M-MOTA	0.00357	0.00051	0.03894	0.00294	0.04595
M-Count	0.00355	0.00053	0.03902	0.00296	0.04606
M-MOTA-TC	0.00356	0.00049	0.01712	0.00265	0.02381

Table 3. Maximum execution time (in seconds) of the evaluation using the three best configurations.

Configuration	Background Subtraction	Blobs Detection	Blobs Classification	Tracking	Total
M-MOTA	0.00552	0.00094	0.09065	0.00657	0.10369
M-Count	0.00559	0.00080	0.08779	0.00562	0.09980
M-MOTA-TC	0.00578	0.00091	0.04628	0.00556	0.05853

Execution time is not strongly linked to the resolution of the processed images. Image resolution only affects the first stage (background subtraction) and the second stage (transformation of the binary image to a set of blobs) of the Recognition and Tracking module. The third and subsequent stages use only the blobs and not the entire image. The two first stages have very low processing times compared with the following stages (one order the magnitude lower, as reported in Tables 2 and 3, so the impact of changing the resolution is not high).

Regarding the goal of person counting, the differences were always below four for two of the best configurations and below five for the other. From a total of 795 frames, no differences were registered in 533 frames for the best configuration. The system was robust, as the worst configuration also had no differences for 433 frames. The aforementioned values of the person counting metric are rather good, especially taking into account that multiple occlusions occur in the videos.

Results in Tables 4 and 5 highlight the accuracy of the proposed system. Table 4 reports the comparison of the number of blobs and number of trackings between the proposed system and ground truth (GT) in MOTChallenge. The obtained results suggest that the proposed system provides an accurate and robust method for recognition and tracking.

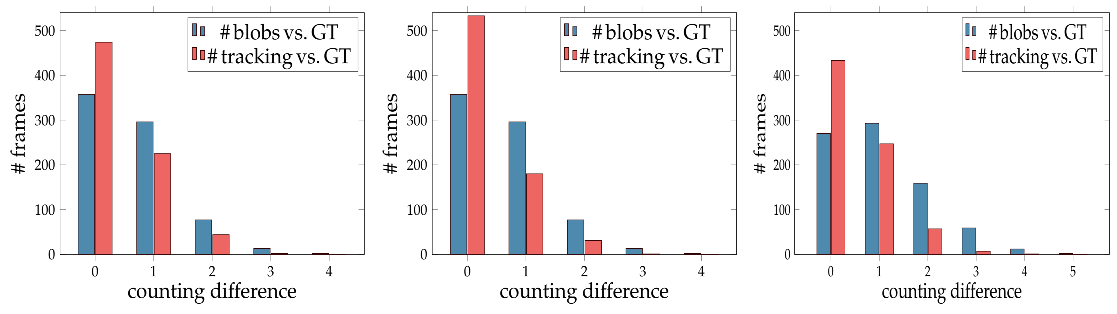
Table 4. Results of the differences in people counting for the experiments of the three best configurations.

Best Three Configurations	# Blobs vs. GT			# Trackings vs. GT		
	Mean	Min.	Max.	Mean	Min.	Max.
M-MOTA	0.67	0	4	0.43	0	3
M-Conteo	0.67	0	4	0.33	0	3
M-MOTA-TC	1.14	0	5	0.52	0	4

Table 5. Results of the Multiple Object Tracking (MOT) Challenge for the experiments of the three best configurations.

		Best Three Configurations		
		M-MOTA	**M-Count**	**M-MOTA-TC**
Metrics on MOT Challenge	Recall	78.6	78.9	75.3
	Precisión	76.3	74.5	76.5
	FAF	1.31	1.44	1.24
	MT	10	10	8
	PT	9	9	11
	ML	0	0	0
	FP	1040	1148	986
	FN	910	898	1053
	IDs	63	55	58
	FRA	253	235	234
	MOTA	52.7	50.7	50.8
	MOTP	64.2	64.1	64.3

Sample results for the counting difference for people for each best configuration are presented in the histograms in Figure 4, compared with the baseline GT. Values for counting using the number of blobs (# blobs vs. GT) and using the number of trackings (# tracking vs. GT) are reported. These results are representative of the behavior of the proposed system for other metrics and indicators.



(**a**) (**b**) (**c**)

Figure 4. Histogram of counting differences for people, for each best configuration. (**a**) Higher Multiple Object Tracking Accuracy (M-MOTA). (**b**) Lower average difference in the person counting (M-Count). (**c**) The one with lower average processing time per frame (M-MOTA-TC).

Histograms in Figure 4 indicate that the tree configurations have accurate values (differences near to zero). M-Count computed the best results (more than 500 frames with zero differences), M-MOTA is the second best and M-MOTA-TC computed the worst results. Values for counting using the number of blobs are accurate, but lower than using the number of tracking, for the three studied configurations.

Table 5 reports the results of the metrics included in MOTChallenge for the three best configurations. *Recall* corresponds to the percentage of persons detected (higher is better), *precision* is the percentage of persons correctly detected (higher is better), *FAF* measure the false alarms per frame (lower is better), *MT*, *PT* and *ML* are the number of ground truth trajectories that are tracked for at least 80 percent, between 20 and 80 percent, and less than 20 percent of its existence, respectively; *FP* and *FN* are false positives and negatives, respectively; *IDs* counts the changes of identity; *FRA* (for fragmentations) counts the interrupted or restarted trackings (lower is better); *MOTA* was previously introduced; and *MOTP* is the difference between the positions of the ground truth and correctly predicted positions (lower is better).

The scalability of the Recognition and Tracking module was validated regarding the number of sources (videos) that it can handle and process simultaneously. The proposed architecture (described in Figure 1) is robust and scalable, being able to work concurrently with many sources thanks to its parallel/distributed design that considers many instances of the Recognition and Tracking module.

Regarding the number of tracked objects (persons) in a single video, experiments were performed over sparse and medium-density scenes. However, the Recognition and Tracking module applies a parallel approach for blob classification, controlled by a specific parameter (PERSON_DETECTION_PARALLEL_MODE). Using this feature, the module is able to process many objects in the same scene and reducing the overall execution time in up to 20%.

6.2. Pattern Detection Module

The validation of the Pattern Detection module was performed over a recorded video that is representative of the current reality for recognition, tracking, and detection in nowadays surveillance systems in smart cities, such as Montevideo. The recorded video has a duration of 2:51 min, a resolution of 800 × 600 pixels and was recorded in natural light. In the video, nine people walk around and get in and out of the scene occasionally. Five events occur in the video, which can be detected by the four pre-loaded patterns in the proposed system: two agglomerations, two street robberies and one 90-degree turn. The Recognition and Tracking module was configured with the three best parameters values. No significant differences were found in the results between the executions that vary the configuration of the first module.

Experiments were oriented to evaluate the capability of the system to detect predefined patterns exactly, at the moment they occur. When the system detects an event that in fact occurred, a true positive is produced. In this case, the closer to the start of the event it is detected, the more accurate the system is. The evaluation also takes into account the number of false positives (i.e., when the system detects an event that did now occur in reality) and false negatives (i.e., events that occurred but were not detected) produced by the system during the processing. Thus, in addition to the timing accuracy of event detection, the number of false positives and false negatives during the processing was evaluated. The accuracy of the detection of true positive cases is evaluated considering the difference between the moment that an event is notified and the real starting time of the event. For false positives and false negatives, only the number of occurrences is taken into account.

Three of the five events were notified during the processing. All of them correspond to real events. Three true positives, two false negatives, and no false positive event notifications were recorded. True positive events were notified in a mean time of 8.3 s and a median of 4.0 s. This is an accurate time for event detection, which allows to take appropriate and timely decisions when a street robbery occurs.

As an example, the scene where the first street robbery occurs is shown in Figure 5. Blobs are represented by the violet rectangles and tracking is represented by the continuous lines. The robbery is detected in the leftmost blob: one person rapidly approaches another, steals the bag and runs fast, as represented by the violet line. This event was correctly detected by the proposed system.

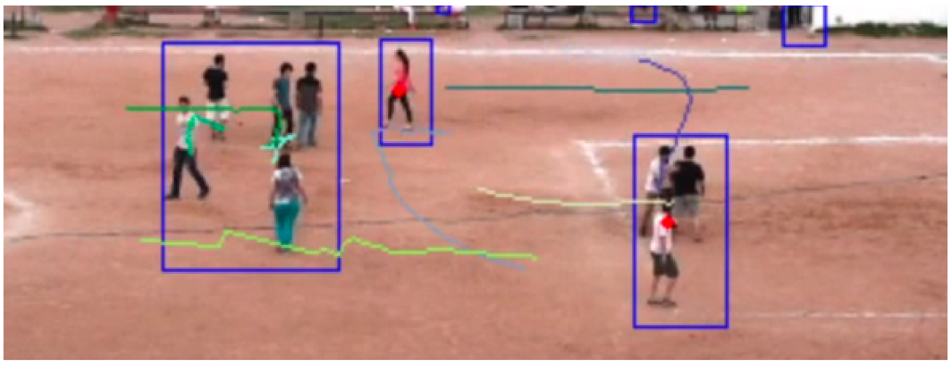


Figure 5. Scene where the first street robbery occurs.

The first not reported event was the second street robbery. This event was not detected due to an incorrect resolution of an occlusion. This case can be observed in Figure 6. The second not reported event corresponded to a 90-degree turn. From the empirical evaluation, it was observed that the detection patterns module has a high sensitivity to small variations of consecutive positions of objects. The fact that the system detects a sequence of small turns instead of a single turn suggests that, in order to detect the event correctly, it is necessary to use a longer history of the last positions of the person who turns.

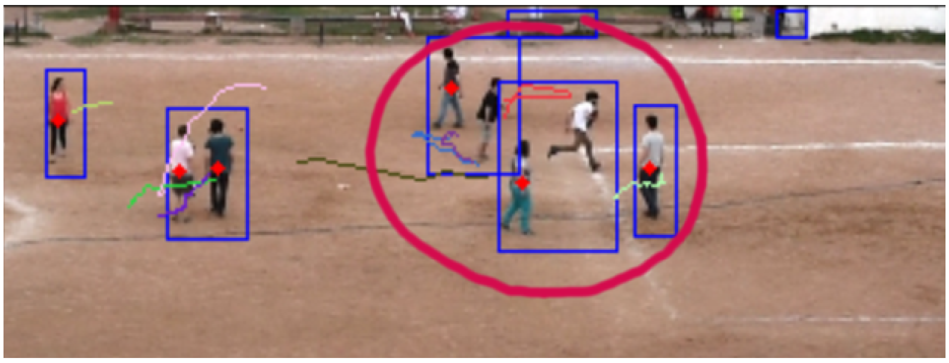


Figure 6. Scene where the second street robbery occurs (it was not detected by the system).

Several additional experiments were performed to evaluate the impact of relevant parameters of the captured videos: contrast and brightness. Real surveillance videos were modified varying the contrast and brightness of the captured scenes. Obtained results allowed concluding that contrast does not significantly impact the efficacy of pattern detection (variation on the prediction capabilities under 5% in the experiments performed). On the other hand, the study of the impact of brightness showed that it affects the successful prediction rate of the proposed system, especially in scenes/videos with poor light conditions (the successful prediction rate is below 50%). These results indicate that further enhancements are needed to properly work on hazy scenes (such as foggy/rainy days). Regarding approaches to overcome issues that emerge under poor illumination conditions, enhancements on two methods already included in the system can be developed: Kalman filters can be applied to predict values of pixels (or groups of pixels) to reduce the variation of illumination, and HOG can be applied to exploit information of the gradient of the images using local histograms that can be grouped in blocks to improve detection and tracking, by normalizing the final representation of each frame to make it less sensitive to illumination changes and distortion. Other methods, such as similarity techniques between frames, can also be applied to mitigate the impact of sudden illumination changes. These enhancements are proposed as one of the main lines for future work.

7. Conclusions and Future Work

This article presented a system for detecting pedestrian movement patterns, based on computational intelligence for image processing and pattern detection.

The proposed system is capable of processing in real-time multiple image/video sources. An architecture based on pipes and filters is used to allow an easy evaluation of different computational intelligence techniques in each stage of the processing. Two main stages are identified in the system, focusing on extracting relevant features of the processed images (implemented in the Recognition and Tracking module) and detecting movement patterns (implemented in the Pattern Detection module). Several techniques are applied for image processing and pattern detection. The proposed implementation fulfills important requirements for a pedestrian movement patterns detection system: it is cross-platform, open-source, and efficient.

The experimental analysis performed over more than 1450 problem instances covers the two main stages of the system. The system was evaluated using PETS09-S2L1 videos and the results were compared with part of the MOTChallenge benchmark results. Results suggest that the proposed system is competitive, yet simpler, than other similar software methods.

The main lines for future work include studying other algorithms for person detection that maintain the low processing cost and possibly improve the dynamical adaptation of blobs aspect ratios; and studying the auto-adaptation of parameters for different scenarios. In turn, to better handle scenes from foggy/rainy days, including support for scenes with different light conditions, and the integration of a specific module to automatically extract/learn the sequence of primitives from particular videos. Filters and estimators (e.g., velocity, depth) can be integrated to deal with the detection of people using moving cameras and other problems that emerge when background subtraction is not directly applicable. The integration of such modules is easily supported by the flexible pipes and filters architecture. Finally, the experimental analysis should be extended to account for the performance of the proposed system to deal with crowded scenarios. Further details about the proposed system are available on the project website fing.edu.uy/inco/grupos/cecal/hpc/APMP.

Author Contributions: J.C.: conceptualization, methodology, software, validation, research, writing; S.N.: conceptualization, methodology, software, validation, research, writing, supervision; A.T.: conceptualization, methodology, research, writing; V.S.: conceptualization, methodology, research, writing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: Part of the research was developed in project "Algoritmos de inteligencia computacional para detección de patrones de movimiento de personas" by J.C., J. Gómez and I. Silveira (advisor, S.N.).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. La Vigne, N.; Lowry, S.; Markman, J.; Dwyer, A. Evaluating the Use of Public Surveillance Cameras for Crime Control and Prevention. Technical Report. Urban Institute, Justice Policy Center, 2011. Available online: https://www.urban.org/research (accessed on 1 October 2018).
2. Kruegle, H. *CCTV Surveillance, Second Edition: Video Practices and Technology*; Butterworth-Heinemann: Newton, MA, USA, 2006.
3. World Health Organization. *Pedestrian Safety: A Road Safety Manual for Decision-Makers and Practitioners*; World Health Organization: Geneva, Switzerland; Foundation for the Automobile and Society: London, UK; Global Road Safety Partnership: Geneva, Switzerland; World Bank: Geneva, Switzerland, 2013.
4. Chavat, J.; Nesmachnow, S. Computational Intelligence for Detecting Pedestrian Movement Patterns. In *Smart Cities*; Springer International Publishing: Berlin/Heidelberg, Germany, 2019; pp. 148–163.
5. University of Reading. Performance Evaluation of Tracking and Surveillance. Available online: www.cvg.reading.ac.uk/PETS2009 (accessed on 26 October 2019).
6. Gonzalez, R.; Woods, R. *Digital Image Processing*; Pearson Education: London, UK, 2008.
7. Ramírez, B. *Procesamiento Digital de Imágenes: Fundamentos de la Imagen Digital*; Universidad Nacional Autónoma de México: Mexico City, Mexico, 2006.
8. Martín, M. *Técnicas Clásicas de Segmentación de Imagen*; Universidad de Valladolid: Valladolid, Spain, 2002.
9. Muñoz, J. Segmentación de Imágenes. 2009. Available online: www.lcc.uma.es/~munozp/documentos/procesamiento_de_imagenes/temas/pi_cap6.pdf (accessed on 26 October 2019).
10. Jain, A.; Duin, R.; Mao, J. Statistical pattern recognition: A review. *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 4–37. [CrossRef]
11. Ramírez, D. Desarrollo de un méTodo de Procesamiento de Imágenes para la Discriminación de Maleza en Cultivos de Maíz. Master's Thesis, Universidad Autónoma de Querétaro, Santiago de Querétaro, Mexico, 2012.
12. Webb, A. *Statistical Pattern Recognition*; John Wiley & Sons: Hoboken, NJ, USA, 2003.
13. Corso, C. *Aplicación de Algoritmos de Clasificación Supervisada usando Weka*; Facultad Regional Córdoba, Universidad Tecnológica Nacional: Buenos Aires, Argentina, 2009.
14. Carrasco, J.; Martínez, J. Reconocimiento de patrones. *Komput. Sapiens* **2011**, *2*, 5–9.
15. Valera, M.; Velastin, S. Intelligent distributed surveillance systems: A review. *IEE Proc. Image Signal Process.* **2005**, *152*, 192–204. [CrossRef]

16. Piccardi, M. Background subtraction techniques: A review. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, The Hague, The Netherlands, 10–13 October 2004; pp. 3099–3104.
17. López, H. Detección y Seguimiento de Objetos con Cámaras en Movimiento. Bachelor's Thesis, Universidad Autónoma de Madrid, Madrid, Spain, 2011.
18. Stauffer, C.; Grimson, E. Adaptive background mixture models for real-time tracking. In Proceedings of the 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Fort Collins, CO, USA, 23–25 June 1999; Volume 2.
19. Kalman, R. A new approach to linear filtering and prediction problems. *J. Basic Eng.* **1960**, *82*, 35–45. [CrossRef]
20. Lefloch, D. Real-Time People Counting System Using Video Camera. Master's Thesis, Université de Bourgogne, Dijon, France, 2007.
21. Rodriguez, M.; Laptev, I.; Sivic, J.; Audibert, J. Density-aware person detection and tracking in crowds. In Proceedings of the IEEE International Conference on Computer Vision, Barcelona, Spain, 7–13 November 2011; pp. 2423–2430.
22. Dollar, P.; Wojek, C.; Schiele, B.; Perona, P. Pedestrian Detection: An Evaluation of the State of the Art. *IEEE Trans. Pattern Anal. Mach. Intell.* **2012**, *34*, 743–761. [CrossRef] [PubMed]
23. Leach, M.; Sparks, E.; Robertson, N. Contextual anomaly detection in crowded surveillance scenes. *Pattern Recognit. Lett.* **2014**, *44*, 71–79. [CrossRef]
24. Felzenszwalb, P.; Girshick, R.; McAllester, D.; Ramanan, D. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *32*, 1627–1645. [CrossRef] [PubMed]
25. Kalal, Z.; Matas, J.; Mikolajczyk, K. Online learning of robust object detectors during unstable tracking. In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops, Kyoto, Japan, 27 September–4 October 2009; pp. 1417–1424.
26. Cho, S.; Kang, H. Abnormal behavior detection using hybrid agents in crowded scenes. *Pattern Recognit. Lett.* **2014**, *44*, 64–70. [CrossRef]
27. Zhu, X.; Jin, X.; Zhang, X.; Li, C.; He, F.; Wang, L. Context-aware local abnormality detection in crowded scene. *Sci. China Inf. Sci.* **2015**, *58*, 1–11. [CrossRef]
28. Garlan, D.; Shaw, M. An Introduction to Software Architecture. In *Advances in Software Engineering and Knowledge Engineering*; Ambriola, V., Tortora, G., Eds.; World Scientific Publishing: Singapore, 1993.
29. Van Huis, J.; Bouma, H.; Baan, J.; Burghouts, G.; Eendebak, P.; den Hollander, R.; Dijk, J.; van Rest, J. Track-based event recognition in a realistic crowded environment. In Proceedings of the SPIE–The International Society for Optical Engineering, Amsterdam, The Netherlands, 22–25 September 2014; Volume 9253, p. 92530E.
30. Mallick, S. Learn OpenCV (C++/Python). 2016. Available online: www.learnopencv.com/ (accessed on 26 October 2019).
31. Coelho, L. Why Python is Better than Matlab for Scientific Software. 2013. Available online: metarabbit.wordpress.com/2013/10/18/ (accessed on 26 October 2019).
32. Pivotal Software Inc. RabbitMQ–Messaging that just Works. Available online: www.rabbitmq.com/ (accessed on 26 October 2019).
33. Nixon, M.; Aguado, A. *Feature Extraction and Image Processing*; Academic Press: Amsterdam, The Netherlands, 2008.
34. Zivkovic, Z. Improved adaptive Gaussian mixture model for background subtraction. In Proceedings of the 17th International Conference on Pattern Recognition, 2004, ICPR 2004, Cambridge, UK, 23 August 2004; Volume 2, pp. 28–31.
35. Zivkovic, Z.; Van Der Heijden, F. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognit. Lett.* **2006**, *27*, 773–780. [CrossRef]
36. OpenCV Developers Team. OpenCV Simple Blob Detector. Available online: https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html (accessed on 26 October 2019).
37. Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 20–26 June 2005; pp. 886–893.
38. Rosebrock, A. Non-Maximum Suppression for Object Detection in Python. 2014. Available online: www.pyimagesearch.com/2014/11/17 (accessed on 26 October 2019).
39. Kuhn, H. The Hungarian method for the assignment problem. *Nav. Res. Logist. Q.* **1955**, *2*, 83–97. [CrossRef]

40. Leal-Taixé, L.; Milan, A.; Reid, I.; Roth, S.; Schindler, K. MOTChallenge: Multiple Object Tracking Benchmark. Available online: motchallenge.net/ (accessed on 26 October 2019).
41. Nesmachnow, S.; Iturriaga, S. Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay. In Proceedings of the 10th International Conference on Supercomputing in Mexico, Monterrey, Mexico, 25–29 March 2019; Volume 1151; pp. 188–202.
42. Leal-Taixé, L.; Milan, A.; Reid, I.; Roth, S.; Schindler, K. MOTChallenge Results 2D. Available online: motchallenge.net/results/2D_MOT_2015/ (accessed on 26 October 2019).
43. Honovich, J. Frame Rate Guide for Video Surveillance. 2014. Available online: ipvm.com/reports/frame-rate-surveillance-guide (accessed on 26 October 2019).