INTELLIGENT MULTI-CAMERA VIDEO SURVEILLANCE

P. Hameete S. Leysen T. van der Laan I. Lefter L. Rothkrantz

Section Interactive Intelligence, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

Department Sewaco, The Netherlands Defence Academy, Nieuwe Diep 8, 1781 AC Den Helder, The Netherlands

L.J.M.Rothkrantz@tudelft.nl

Abstract: Video surveillance is found in different areas. Traditionally multiple security cameras are positioned throughout the area, linked to computer screens. Unfortunately humans are unable to monitor a huge number of screens and may lose their focus over time. In this paper we describe an automated intelligent multi-camera video surveillance system which is able to track people and detect suspicious situations. In this paper we will describe the implemented prototype and the results of experiments.

Key words: surveillance system, image processing, artificial intelligence

1. INTRODUCTION

Video surveillance is a type of surveillance that is found in different kinds of areas, such as public buildings, metro stations and military areas [3,4,5]. It has proven to be an effective tool to monitor large areas with limited resources. Because cameras and surveillance systems are continuously developed, these systems become increasingly cost efficient, allowing for larger systems as well. Traditionally, multiple static security cameras are positioned throughout the area, and attached to monitor screens. These screens are monitored by security personnel in order to detect suspicious behaviour. Examples of such situations are when a person enters a restricted area, when a person suddenly starts running or when a person has been following another person or a period of time. In case of a suspicious situation the security personnel is alarmed, and relevant information is displayed to allow the personnel to take action if required. Depending on the seriousness of the situation, further action may be taken in order to resolve it.
Fig. 1. Security employee inspecting video screens, Surveillance camera

Though computers may no be as good at computer vision and reasoning as human operators, they provide different advantages. For instance, computers are capable of working 24 hours a day, 7 hours a week. By using an automated intelligent multi-camera video surveillance system the security personnel could be supported in their work, allowing for less security flaws in monitoring the area.

The outline of the paper is as follows. In section 2 we give a problem definition and in section 3 we describe how we solved the problem. In section 4 we report the results of experiments end we end with a discussion and refereed papers.

2. PROBLEM DEFINITION

The main problem with the current camera surveillance systems lies within the nature of the tasks of the security personnel. They must passively watch multiple monitor screens simultaneously. Humans easily get tired and lose concentration, especially in situations when no situations of notice occur for a long time. At regular times the guard will inspect the area in person, leaving the cameras unattended and being exposed to potential attackers. Furthermore, humans are not capable of noticing every small detail, let alone of focusing on multiple things at the same time.

A second problem is the fact that camera footage of some areas is being recorded without supervision of a human operator, simply because it would be too expensive to constantly monitor that area. The data is stored as evidence in case some event occurs. This prevents incidents from being detected in a timely manner, preventing swift responses to potentially dangerous situations. Moreover, it is time-consuming to find the correct video images, especially when the event occurred many hours before it is detected, and the system consists of many different cameras.
By using computers to support the security personnel in their task to watch the monitor screens, some of these problems may be prevented. Computers, unlike humans, are capable of working continuously, without losing focus. Moreover, computers are well scalable to allow for monitoring a large number of security cameras simultaneously. Additionally, computers can communicate constantly, allowing each computer to have a complete and up-to-date view of the monitored area.

The goal of this project was to develop a system that is able to assist the security personnel in observing the NLDA area and alarming the control room in case of a suspicious situation. In order to achieve the project's goals, the prototype system utilizes the static security cameras located in the area. Each camera is linked to a client software application that is able to detect moving objects from the camera images. This client computes different features, such as the locations of the detected objects and whether they are human or not. Because suspicious behavior may occur over multiple cameras, these extracted features are then transmitted to a central server. The central server application gathers, combines and maintains the information from the different security cameras and then reasons about the situation. We summarize the research goal of the project as follows:

P1 The system must be capable of capturing data from multiple cameras, and then detect, identify and locate moving objects.

P2 The received data must be processed into different objects which may then be tracked by the system. The path history of GPS coordinates is then used to extract features for further reasoning.

P3 Given a certain context or region of interest in combination with extracted features the system must determine whether behavior is considered suspicious. Suspicious behavior must result in an alert.

P4 The server in the control room should visualize the gathered data and results of the reasoning process to the security personnel through a graphical user interface.

3 PROBLEM SOLUTION

The system consists of two applications: a Client written in C++ with [1,2] and a Server application written in Java. The Client application is attached to a security camera, and will then detect moving objects, classify them as either human or non-human, and determine their location relative to the camera. For each detected object it will then transmit this information to the Server application. This Server application gathers the information from the Clients and combines the information into actual objects in the monitored area. It then reasons about these
objects by using their history of GPS locations to detect object paths and speed and based on this information detects whether suspicious situations are occurring. The GUI Component showing a map of the area that is being monitored as well as the objects currently detected in the area and their path history. In graphs the speed of the objects has been plotted over time. Most importantly, the objects in the area will be displayed and potential alarms for these objects.

![Fig. 2. A global overview of the system](image.png)

### 3.1. Client image processing modules

The client makes use of the open source C++ computer vision library [1,2] for all basic computer vision operations. For the separation from foreground to background the client makes use of a background subtraction technique referred to as ‘Mixture of Gaussians (MoG)’, which is developed by Stauffer and Grimson [6]. This technique uses three dimensional Gaussians per pixel to model the image background. The three dimensions correspond with the red, green and blue (RGB) color values of each pixel. The main advantage of this background model is its capability to handle multi-modal backgrounds, such as waving trees and objects moving in and out the background scene. Whenever there is enough statistical evidence for a pixel to belong to the background it is classified as background. The client makes use of a slightly improved version of the original MoG algorithm, which was developed by Zivkovic and van der Heijden [7]. The input variables for the MoG algorithm were set according to what worked best in a series of experimental setups intended to simulate all possible situations. In the resulting MoG foreground ground mask detected objects are identified and labeled using the OpenCV [1,2]. This library contains functions for finding the different connected components in a binary mask and for thresholding these connected components. A minimum size of moving objects can be defined in order
to suppress noise. It is clear that filtering based on a correct size results in only the three moving objects being identified and labeled.

For human classification a Support Vector Machine (SVM) was used in conjunction with an algorithm called Histogram Oriented Gradients (HOG). This algorithm computes a feature vector of the given input image, which is then used by the SVM for classification. The Support Vector machine was trained on the MIT pedestrian dataset1. The HOG algorithm computes the oriented gradient of the input image and overlays the image with a grid of cells. Then, for each cell a histogram is created with bins based on the orientation, where each bin is weighted by the magnitude of the gradient. Next, overlapping blocks are created consisting of adjacent cells and the cells in each block are contrast normalized to get rid of change in light intensity over the image. The histograms are then put into a single vector, which is the feature vector used for classification. For a more comprehensive explanation of the histogram oriented gradient algorithm we refer to the original paper by Dalal and Triggs [8]. The parameters of the feature extraction algorithm HOG were set according to the original paper [8] as they have done research as to what works best in general.

3.2. Server design and implementation

The server design was challenging as its processes are mostly non-linear, compared to the Client which processes new camera frames linearly throughout the image processing pipeline. The server is responsible for storing, updating, displaying and reasoning about data. This has to happen dynamically and by asynchronous processes. The design decisions that were made to improve code quality and extendibility are discussed separately for each component. The essential server components are displayed in Fig. 3 and listed below:

- **Message Handler**: This component runs in a separate thread and creates connection handler objects that handle the connections from client to the server. Each newly created connection handler runs on its own dedicated thread to assure that the other components will not suffer from computational congestion.
- **Whiteboard**: This is the central component of the Server application. It is responsible for storing, maintaining and distributing data and embodies all the methods to do so. Information on the cameras, moving objects in the area, areas of interest and alerts are stored on the whiteboard.
- **Reasoner**: This component reasons about the objects and areas stored on the Whiteboard and adds generates to the whiteboard whenever it derives that a suspicious situation is occurring.
- **GUI**: This is a collection of components responsible for graphically displaying the data stored on the Whiteboard in a clear format. The design follows a clear separation between the view and control, according to the Model-View-Control (MVC) design pattern. A subcomponent called Map allows for plotting all the
latest object locations and location histories on a draggable and zoomable image of the area. The GUI is also responsible for displaying the speed of the objects over time and the alerts that have been sent by the reasoner. Lastly, the GUI has a Console component that displays informative messages on the system status.  

Util: This component consists of static utility methods that are used throughout the entire application, such as a method used for translating locations in a received camera frames to actual GPS locations.

Figure 3: a component diagram that describes the initial design of the server as well as interactions between components

The Reasoner puts the actual intelligence into the System. The Reasoner combines all the separate reasoning components, and ensures that they are executed in the correct order during the reasoning cycle. First, new camera Frames are processed by the Object Identifier. Then irrelevant objects are cleaned. Next the Areas of Interest are evaluated and finally the reasoning rules are applied. The Reasoner also allows for communication of the reasoning components with the Whiteboard and its stored objects. Moreover, it allows for addition of new rules. The responsibility of the Object Identifier is to match the MotionBoxes in the incoming frames from the Client applications to actual Objects of Interest that are followed by the system. A systematic figure that explains the process of the Object Identifier can be found in Figure 4.
4. EXPERIMENTAL RESULTS AND DISCUSSION

Testing Localization Accuracy The localization algorithm accuracy is not only dependent on the accuracy of the camera specific properties, it depends also on other camera aspects. There are many aspects that can create an error in the localization such as warping of the image due to lens inconsistencies. The sum of these errors (together with the measurement errors) have been measured to get an estimate of the total average error in a camera. Based on this empirical research, taking practical issues in mind, we conclude that the maximal deviation of the Euclidean distance tolerated from the real position is never more then half a meter.

Testing the Background Subtraction Background subtraction may be tested by comparing the results of the background subtraction to an image in which the ground truth foreground is selected. This is a time consuming process however. Instead we chose to manually assess the quality of the background subtraction, by simultaneously displaying the source image and the background subtraction result. By manually assessing the quality of the result we could fine-tune the parameters until the achieved result was clean enough for consequent image processing processes.
Scenario Based Testing and Debug Server application. In order to allow for testing of the Reasoner rules and areas we have developed an extension to the Server application, called the Debug Server. The Debug Server extends the GUI MapPanel component by adding a set of extra user interaction functionalities: drawing of areas, addition of test cameras and drawing of object paths to be linked to a selected test camera. Currently the Debug Server is an unofficial extension of the application and used for testing purposes only. In order to test the functioning of our different Areas Of Interest and reasoning rules we have developed many different scenario's using the Debug Server. These scenarios consisted of object tracks that should lead to alarms being fired. By playing back the scenario it can be confirmed if the Server application actually fires an alarm correctly. All currently implemented areas and rules have been scenario tested, and have been confirmed to work as intended. To give an example a simulation file was created consisting of four objects. One object was walking around, following a random path with curves. Another object followed a path that should be considered as following the first object. Two other objects were added that cross the paths of the first two objects to ensure that no false alerts are fired when more objects come into play. Many more simulations similar to this 'following' simulation were created using the Debug Server. The Debug Server allows for more scenarios to be very easily generated.

Performance Testing. In order to test the performance of the system we created a basic performance testing setup. Because it was impossible to perform this test at an actual scale by using a large number of cameras, we made use of simulation files generated by the Debug Server. These simulation files each represented a camera transmitting the location of a single object at a rate of two frames per second. This object is constantly zig-zagging in and out of a restricted area in order to simulate unrealistic heavy reasoning. Next, we added multiple of these simulated cameras to the Server application. Then, we calculated the amount of cycles that were performed while running the simulations, and divided this by the total time span of the simulations. We then calculated the average processing speed, measured in processing cycles per second, while the simulations were played. By incrementing the number of cameras that are simulated we increased the load on the Server application. This test was executed on a 3.1 GHz Intel Core i5-2400 quad-core CPU.

Table 5.2 shows the results of the performance test. Under low computational load the Server application scaled extremely well maintaining reasoning cycle speeds of over 300 cycles per second for 30 cameras and objects simultaneously. However, when a large amount of reasoning computations was performed on each of the objects (object in an area and following rule was enabled), the reasoning cycle speed drops to 3 per second. The Server application managed to process information of at most 20 cameras simultaneously under bad conditions, while
maintaining an average processing speed of 6 cycles per second. This ensures that quality requirement Q1 is met for monitoring security areas with up to 20 cameras when a single PC is used for the Server application.

Table 5.2: The results of a performance test for the server application

<table>
<thead>
<tr>
<th>Number of cameras/objects</th>
<th>Low reasoning load</th>
<th>Heavy reasoning load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000+</td>
<td>1000+</td>
</tr>
<tr>
<td>5</td>
<td>1000+</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>1000+</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>763</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>567</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>332</td>
<td>3</td>
</tr>
</tbody>
</table>

Scenario Based Testing We tested a couple of different scenarios (comprised in one test footage) listed below:

*Noise introduced by wind.* The tree in the footage has moving leaves which generates constant periodic motion. This motion should be filtered out over time by the Mixture of Gaussians model.

*Noise generated by camera movement.* The camera moves slightly generating large blocks of noise in the footage. The system should recover from this as fast as possible.

*Classification.* Classification was tested by introducing humans in the frame and cyclists. The system should not label the cyclists as humans. The footage includes a human moving over predefined locations whose real world location relative to the camera are known. In this way it could be checked if the localization algorithm works correctly.

The footage includes a human walking after which it runs which should trigger a human is running alert.

Each scenario described above passed the test. However, the system does have minor difficulties differentiating humans from cyclists. In practice this could lead to moderate amounts of false running alarms from clients observing areas containing a lot of cyclists.

5. CONCLUSION

For this Research Project an Intelligent Multi-camera Video Surveillance system has been developed, with a particular focus on the Netherlands Defence Academy area of the Koninklijk Instituut voor de Marine. This system is a proof of concept, capable of gathering information from multiple surveillance cameras, to then
reason about the situation. When suspicious behavior is detected, an alarm is displayed for the security personnel.

Now we discuss the results of research problems as defined in section 3 in more details:

**P1:** The system must be capable of capturing data from multiple cameras, and then detect, identify and locate moving objects.

For this sole purpose a separate Client application, written in C++ with the OpenCV library, has been developed. This application runs through an image processing pipeline. A camera is directly attached to the hardware that is running this application. Next, the moving objects are detected in real-time using a Mixture of Gaussians algorithm. By applying a Histogram of Oriented Gradients classifier, the moving objects are identified as either human or nonhuman.

Finally, by using a custom geometrical localization algorithm, the locations of the moving objects are determined. To allow the System to capture data from multiple cameras simultaneously a Client/Server model is introduced. When a Client application is booted up it identifies itself with the server to get a unique label. It is then allowed to transmit the computed features of the detected moving objects to a central Server application in realtime.

**P2:** The received data must be processed into different objects which may then be tracked by the system. The path history of GPS coordinates is then used to extract features for further reasoning.

The Server application cycles through the received information from each of its connected Client applications. For each of the detected objects it determines an accurate estimate of the GPS location. It then tries to match this detected object to an actual object of interest within the system, based on its GPS location. For each of the objects of interest it applies location matcher rules, to see if the new GPS location is a match to the existing object. Such a rule could for instance predict the next location of an object based on its average direction and speed. When it is statistically plausible that the new location is the new location of this object it is considered a match. When no match is found a new object of interest is created. Such matcher rules can easily be added to the Server by implementing an interface. Moreover, the Server ensures that objects that are considered irrelevant are removed from the Server application. A current rule for this discards objects that have not been seen for a time that is 1.5 times longer than the time it has been seen in total.

**P3:** Given a certain context or region of interest in combination with extracted features the system must determine whether behavior is considered suspicious. Suspicious behavior must result in an alert.

This goal has currently been achieved by implementing the system as a rule based Expert System. The Server application allows for definition of Areas of Interest. Such areas may be defined on the map of the area which is being monitored, and whenever an object is within the area a certain rule is applied. This rule may result
in an Alarm being fired. An example of such a rule is the rule for a SpeedLimit: whenever an object is within this area and its average speed over a certain timeframe exceeds the speed limit, it fires an alarm. New types of areas may easily be added to the system by extending the abstract AreaofInterest. New types of areas may easily be added to the system by extending the abstract AreaofInterest class.

Besides Areas of Interest the System allows for defining Rules and Comparative Rules. Such rules evaluate a single object, or compare each pair of objects respectively. The basic rule reasons about a single Object of Interest and its extracted features, an example of such a rule determines when an object starts running to then fire an alert. A comparative rule compares the computed features of two Objects of Interest. For example, this could be used to determine if one object is following another, based on its history of GPS locations. Like the object matchers, Rules and Comparative Rules may be easily added to the Server application by implementing an interface.

**P4:** The server in the control room should visualize the gathered data and results of the reasoning process to the security personnel through a graphical user interface.

Through a user centered design process a GUI has been developed that will visualize the gathered data and results of the reasoning process (alarms) to the security personnel. An important component of this GUI is the Map Panel, which shows the up-to-date locations of all objects in the area on a map. Each unique object is marked using a unique color throughout the GUI. The Map Panel allows zooming and dragging to survey the complete area. Besides the MapPanel the GUI contains a Console in which informative messages on the status of the connected Clients, and the Server system itself are displayed. Additionally, the GUI contains a plot in which the average speed of each of the objects is displayed over time. Again each object is labeled using its unique color. Finally, the GUI contains a table. This table is the crucial component of the GUI, as it displays the alarms that have been fired. Each alarm is colored according to its priority, and contains information on the time, type of event and source object of a suspicious event. Moreover, it displays the camera that last detected the suspicious event, such that the corresponding video may easily be retrieved.

### 7. Reference


**Information about the author(s):**

**P. Hameete S. Leysen T. van der Laan,** Delft University of Technology, MSc= students, Artificial Intelligence.

**I. Lefter,** Delft University of Technology, PhD student, Interactive intelligence

**L.J.M. Rothkrantz,** Delft University of Technology, The Netherlands Defence Academy, Professor Artificial Intelligence