ABSTRACT

Title of Dissertation: Petri Net Models for Event Recognition in Surveillance Videos

Nagia M. Ghanem, Doctor of Philosophy, 2007

Dissertation directed by: Professor Larry Davis
Department of Computer Science

Video surveillance is the process of monitoring the behavior of people and objects within public places, e.g. airports and traffic intersections, by means of visual aids (cameras) usually for safety and security purposes. As the amount of video data gathered daily by surveillance cameras increases, the need for automatic systems to detect and recognize suspicious activities performed by people and objects is also increasing.

The first part of the thesis describes a framework for modeling and recognition of events from surveillance video. Our framework is based on deterministic inference using Petri nets. Events can be composed by combining primitive events and previously defined events by spatial, temporal and logical relations. We provide a graphical user interface (GUI) to formulate such event models. Our approach automatically maps each of these models into a set of Petri net filters that represent the components of the event.

Lower-level video processing modules, e.g. background subtraction, tracking and classification, are used to detect the occurrence of primitive events. These primitive events are then filtered by Petri nets filters to recognize composite events of interest. Our framework is general enough and we have applied it to many surveillance domains.

In the second part of the thesis, we address the problem of detecting carried objects. Detecting carried objects is the main step to solve the problem of left object detection. We present two approaches to the left object detection problem. Both approaches poses the problem as a classification problem. For both approaches, we trained SVM classifiers [19] on a laboratory database that contains examples of people seen with and without two common objects, namely backpacks and suitcases. We used a boosting technique, AdaBoost [20], to select the most discriminative features used by the SVMs and to enhance the performance of the classifiers. We give recognition results for each approach and then compare both approaches and describe the advantages of each one. We also compare the performance of both approaches on real world videos captured at the Munich airport.

Petri Net Models for Event Recognition in Surveillance Videos

by

Nagia M. Ghanem

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

DEDICATION

To my parents, my husband, Farah and Sarah

ACKNOWLEDGEMENTS

First of all, I thank Allah (swt) for helping me complete this work and I ask Him to accept it as a good deed and help me use this knowledge in a way that pleases Him.

I would like to thank my advisor, Dr. Larry Davis, for allowing me the opportunity to pursue my research within one of the leading computer vision labs. I also thank him for all the support and guidance he provided to me during the past years.

I would also like to thank both Dr. David Doermann and Dr. Daniel De-Menthon. Their discussions and comments are always helpful.

I am very thankful to my parents who have always given me the love and support I need throughout my entire life. Everyday, I feel blessed by their prayers. May Allah (swt) help me give it back to them.

I cannot express my gratitude to my husband, Moustafa. Without his help, support and encouragement, I would not be able to complete this work. No wards can express my appreciation to him as he stands beside me during the most difficult times.

Last but definitely not least, I would like to thank my two lovely daughters; Farah and Sarah for their unconditional love and patience during completing this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Video surveillance is the process of monitoring the behavior of people and objects within public places, e.g. airports, metro stations and traffic intersections, by means of visual aids (cameras) usually for safety and security purposes. As the amount of video data gathered daily by surveillance cameras increases, the need for automatic systems to detect and recognize suspicious activities performed by people and objects is also increasing. Manual detection and recognition of these activities would require system operators to monitor a large number of cameras simultaneously to detect any suspicious activity and report it in a timely manner. Even in the case of searching video archives for previous events, as in criminal cases when we are trying, for example, to track a suspicious person back in time to determine where he came from and with whom he has interacted, a significant amount of human instrument is required and the process is subject to human errors and fatigue.

Thus, automating the process of event detection and recognition is one important task of computer vision research. The goal is to interpret uncertain data computed by lower level vision modules into high level semantics representing humans activity. There are three main challenges here. First, event modeling and representation should be general enough to handle variabilities in event durations and in the different ways the same event might be performed by different actors or in different places. It is also important to be able to compose events by combining simpler ones using temporal and logical relations. Second, event recognition should be done efficiently so that the large number of irrelevant observations from low level vision does not affect the performance of the recognition process. Finally, the recognition process should also be able to handle uncertainties and failures in low-level vision modules.

The problem of event detection and recognition is usually posed as an inference problem, where some inference mechanism is applied to available knowledge (output of lower level vision modules) to infer the occurrence of these events in the video data. Both stochastic inference [1, 2, 3, 4, 5, 6] and deterministic inference [7, 8, 9, 10] have been proposed for the problem of event detection and recognition. Methods using stochastic inference assume that activity structures are known in advance or can be easily learned from training data. Then, some stochastic inference method is used to infer the occurrence of events in video. On the other hand, methods using deterministic inference usually assume that events can be decomposed into subevents, some of which can be directly detected by perceptual methods, accounting for a variety of temporal constraints. Then constraint propagation algorithms can be used to infer the event occurrences. This can be useful in cases where event structures are not known in advance and when training data is not available. This is usually true, as what is more important in surveillance is the detection of rare events (that raise safety and security concerns) for which training data is natural conditions is exceptionally difficult to acquire.

The first part of the thesis describes a framework for modeling and recognition of events from surveillance video. Our framework is based on deterministic inference using Petri nets. Events can be composed by combining primitive events and previously defined events by spatial, temporal and logical relations. We provide a graphical user interface (GUI) to formulate such event models. Our approach automatically maps each of these models into a set of Petri net filters that represent the components of the event. Lower-level video processing modules (background subtraction, tracking, etc.) are used to detect the occurrence of primitive events. These primitive events are then filtered by Petri nets filters to recognize composite events of interest. This approach is general enough to be applied to any surveillance domain (car parks, airports, indoor scenes, etc.). Inference about temporal, spatial and logical relations between events is performed by the engine independently of the characteristics of the primitive events.

A Petri net is an abstract model of the flow of information in a system [11]. Using Petri nets as a representation and as a filtering mechanism has the following advantages:

- Petri nets can be used for both deterministic and stochastic inference of event occurrences.
- Petri nets have a nice graphical representation that uses just a few types of elements. This representation has a well-defined semantics so that it is easy to understand the model and to learn the language.
- Petri nets have a precise mathematical model that can be used for analysis. For example, there are well-defined algorithms for detecting deadlock and inconsistency in the data.
- Petri nets can be used to represent sequentiality, concurrency and synchronization of events.

- Petri nets can be used to represent events in a top-down fashion at various levels of abstraction, i.e. they can be used to model a composite event hierarchically from simpler event models.

- Compared to classical rule-based expert systems, in terms of efficiency, Petri nets are known to be as efficient as expert systems. The RETE algorithm, used in most expert systems implementations to improve speed [12], is applicable to Petri nets [13]. The main idea is to exploit temporal data redundancies (coming from the markings that are not changed during transition firing).

- At any time during the interpretation process, the positions of tokens in the Petri net summarize what happened in the past (keep history) and predict what will happen in the future. In this way, composite events are recognized incrementally and there is no need to reevaluate past events.

In the second part of the thesis, we address the problem of carried object detection. One important problem in understanding human activities is to detect whether a person is carrying an object or not at different times. For example, if a person is carrying an object at time t1 and not carrying it at time t2, we can infer that the person has dropped the object or give it to another person between times t1 and t2. Another problem is detecting left packages in public places. Detection of left packages is among the goals of many visual surveillance systems of these places for security and safety concerns. In some cases, the left package can be detected by the background modeling component of the surveillance system [14, 15, 16, 17, 18]. On the other hand, if the package is left in an unseen place (e.g. behind a pillar or in a trash bin), then these methods will fail to detect it. In this case, we can infer that a package is being left if we detect that a person is carrying a package at one time and not carrying it at a later time.

Examples where the second approach can be useful include the following. A person enters a room carrying an object, deposits the object and then after a short time he exits wearing the same clothes but without the object. Another example is a person in a public place is observed by one or more cameras, then he disappears for a short period of time (e.g., behind a pillar) where he drops or picks up an object and he reappears. Even in cases where this person is continuously observed, he may drop or pickup objects in places not easily observed by surveillance cameras (e.g., in a trash bin). In these examples, direct detection of the left object itself is not possible, but could be inferred by deciding whether the owner is carrying an object at a given time but not carrying it at a later time.

In this part of the thesis, we present two approaches to the left package detection problem. We assume we have different instances of the same person within different cameras and at different times, and that the time separation between different instances of the same person is small, so that he does not change clothes between these instances. Both approaches poses the problem as a classification problem. The first approach, direct classification of silhouettes, classifies the subject's silhouettes for each instance directly to determine whether he is carrying an object or not. The second approach, appearance change detection, determines whether there is a significant change in human appearance between two different instances or not that might be due to an object being carried at one time but not the other. If there is a significant change in the human appearance, additional analysis is conducted to decide whether the person has dropped an object or acquired one from the scene.

For both approaches, we trained SVM classifiers [19] on a laboratory database that contains examples of people seen with and without two common objects, namely backpacks and suitcases. We used a boosting technique, AdaBoost [20], to select the most discriminative features used by the SVMs and to enhance the performance of the classifiers. We give recognition results for each approach and then compare both approaches and describe the advantages of each one. We have also tested both approaches on real world data captured at an airport.

1.2 Contributions

The contributions of this thesis are:

- We provide a framework for modeling and recognition of events from surveillance video.
  - We define an ontology for event modeling. Ontology entities include: objects, states, events and relations. Events are either primitive events or composite events that can be built hierarchically from simpler events joined by temporal and logical relations.
  - We define a mapping from each ontology entity into a set of Petri nets models.
  - We develop a GUI, through which users formulate ad-hoc queries about events.
  - We provide a generic mapping from users' queries into a set of Petri nets models that are used for detecting and recognizing event queries.
  - We support the ability to define negative events easily in our framework. A negative event is detected when an important observation is missing or not detected. Negative events are of special importance in surveillance application, e.g. security guard does not return with 15 minutes.

    - We tested the system extensively on real world examples in many domains. Events modeled and detected by the system range from primitive events with one actor to composite events with many actors and temporal and logical relations.

- We developed two machine learning approaches to detect carried objects.
    - We apply it to the problem of left package detection in the framework of our event modeling and recognition system.
    - We generate a large pool of features capturing the shape and color models of different instances of a person.
    - We use a boosting technique, AdaBoost, to select the most discriminative features and provide them to a set of Support Vectors Machine (SVM) classifiers.
    - We train the classifiers on a large database recorded in our laboratory and we test the method extensively on data recorded in the laboratory and on real world data captured at an airport. High recognition rates were obtained.

1.3 Organization

The rest of the thesis is organized as follows. In Chapter 2, we describe the Petri nets event modeling and recognition system in detail. Chapter 3 provides experimental results for the event modeling and recognition system. In Chapter 4, two approaches to solve the problem of carried object detection are described along with the classifier design and training. Chapter 5 provides experimental results for the carried object detection methods based on data captured at an airport. We conclude and provide directions for future research in Chapter 6.

1.4 Related Work

In this section, we review previous work related to topics discussed in this thesis and compare our work with other work. First, we discuss work related to event recognition in surveillance video. Second, the use of Petri nets as an inference mechanism in rule-based expert systems is discussed. Finally, we discuss work related to the problems of human appearance change detection and left package detection.

1.4.1 Event Recognition

Recognition of events from video data is usually posed as an inference problem, where some inference mechanism is applied to available knowledge to infer the occurrence of these events in the video data. Both deterministic and stochastic inferences have been applied to recognize events from video data. First, we survey methods using stochastic inference. Then we discuss methods using deterministic inference and compare our Petri nets-based approach with these methods.

Stochastic Inference

Stochastic inference methods have been applied to event recognition from video data. Examples include Hidden Markov models, stochastic grammars and Bayesian networks.

Hidden Markov models (HMMs) were chosen to recognize American sign language [1]. HMMs are suitable for recognizing sequential events with different temporal durations but not for activities involving more than one actor. Coupled HMMs (CHMMs) were introduced to alleviate this problem by coupling the states of two HMMs to model interaction between persons[2]. For activities involving more than two persons, the model is complex and the number of parameters is large and difficult to learn from training data.

Stochastic context free grammars (SCFG) are used in [3] to recognize high-level activities. The input for this grammar is assumed to be primitive events recognized at a lower level by HMMs. The limitations of this approach is that representing temporal and spatial relations between events is difficult. Also, inferring the grammar rules and their probabilities for each new domain is difficult.

Bayesian networks have also been used by many researchers. Buxton et al. used Bayesian Belief Networks (BBN) for video interpretation in a traffic surveillance application [21]. For simple tasks, like monitoring overtaking and giveaway behavior involving just two vehicles, this approach works well. But, if the task involves complex multiple object interpretation or plan-like behaviors, the approach may not scale well.

The system described in [4] supplies textual descriptions for dynamic activities occurring in a dynamic scene that include vehicles and pedestrians. There are two levels of description. In the first level, the object level, each tracked object is assigned a behavior agent, which uses a Bayesian network to infer fundamental features of the objects' trajectories. In the second level, the inter-object interaction level, a situation agent is created dynamically when two objects are in close proximity. But this system does not provide ways to handle situations involving more than two objects. In [22] Bayesian networks are used to recognize several activities in a football match. Dynamic Belief Networks (DBN) are used in [23] in a hierarchical fashion to interpret video taken from a moving airplane, where humans make up a few pixels in the video. The highest-level scenario recognition DBNs are built from smaller DBN, which can be used in more than one higher-level network. The structure of the DBN is given in advance and the statistical parameters are learned from data. Smaller networks can be built and experimented with separately.

In [5], three levels of events are described. Bayesian networks are used to infer the likelihood of simple events from the mobile objects' properties. At the second level, complex single thread events correspond to a linearly ordered time sequence of simple events (or other complex events). A Probabilistic finite state automaton is used to represent and recognize these events. At the third level, multiple thread events correspond to two or more single-thread events with logical and temporal relationships between them. Many actors may participate in the same event. Allen's interval-to-interval relations are used to describe temporal relations between subevents. The recognition is done by propagating temporal constraints and the likelihood degrees of subevents along the event graph. The advantage of this approach is that it can verify and propagate temporal constraints when events are uncertain, while other techniques for constraint satisfaction and propagation techniques usually assume that events and their durations are deterministic.

A particular form of dynamic Bayesian networks, Recurrent Bayesian Networks (RBNs), have been used for the recognition of human behaviours through the temporal evolution of their visual features in [6]. Although RBNs have the advantage of independence from the time scale of events, the learning problem is tedious and how to represent different temporal and spatial relations is not clear.

Deterministic Inference

There have been many methods that apply deterministic inference to detect events in video data. Most of these methods assume that events can be decomposed into subevents, some of whose occurrences can be directly detected by perceptual methods, and between which there exist a variety of temporal constraints. Then, constraint propagation algorithms can be used.

In PNF-networks [7], Allen's temporal relationships [24] are used to express parallelism and mutual exclusion between different subevents. Then, Allen's interval algebra network is mapped into a simpler 3-valued domain (past- now- future) network, a PNF-network, to allow fast detection of actions and subactions. The arc consistency algorithm AC-2 is used to propagate temporal constraints. This algorithm is linear in the number of constraints. But the computation of PNF restriction is NP-hard.

Declarative models described in [8] are used to describe activities at many levels (states of the scene, events and scenarios). The activities are described by the conditions between the objects of the scene. Then a classical constraint satisfaction algorithm, AC-4 (Arc Consistency-4), is used to reduce the processing time for the process of recognizing activities in video sequences.

To increase the efficiency of processing temporal constraints, Vu et al. [9] suggest that in a preprocessing step, scenario models are decomposed into simpler scenario models containing at most two sub-scenarios. Then, the recognition of these simpler scenarios just tries to link two scenario instances instead of trying to link together a whole set of combinations of scenario models. However, this method cannot be applied to partially ordered events, where there is single order of events.

Petri nets have been suggested in [10] as an inference mechanism to represent the dynamic evolution of a car parking scene with humans and vehicles. A symbolic language is defined to capture the logical and algebraic conditions that are handled in a set of prototypes. An Activity prototype is a set of logical and algebraic relations holding on a finite set of objects and scene elements. A Plan prototype is a set of relations between some activity prototypes and some state conditions. The plan prototype is interpreted as a Petri net. Places are associated with activities prototypes and state conditions. Transitions are associated with logical conditions and constraints.

Our approach extends this work in the following ways:

- Precise use of the state-of-the-art ontology in video surveillance. We define automatic mappings of ontology entities into Petri nets. Generic queries can be modeled by this automatic mapping instead of manual creation of nets as in [10].

- Support of temporal logic is provided by our approach, not by the approach in [10].

- We represent each event instance by a token that encapsulates information about participants in this instance. This makes the total number of nets is the same as the number of event models. For each new event instance, [10] creates a new Petri net. So, the total number of existing nets the sum of the number of instances of all events. In our approach, all instances of the same event are represented by one Petri net and event instances are represented by tokens in the corresponding Petri net. So, at any time, the total number of existing nets is fixed and small compared to the number of events.

1.4.2 Petri Nets as an Inference Mechanism

Petri nets have been used as an inference mechanism for rule-based expert systems. The rest of this section will survey the use of Petri nets in rule-based expert systems.

In 1987, Sahaoui et al. showed the similarities between a rule-based expert system and a Petri net: transitions can represent rules, markings can represent facts and the token player can represent the inference engine. They also showed that using the Petri net representation increases the efficiency of rule-based expert systems by providing parallelism and pipelining. Since then, many expert systems were developed that use Petri nets as a knowledge representation that guides the inference process. In 1988, Murata and Zhang [25] used a predicate/transition net model for a subset of Horn clause logic programs. In 1993, Hura [26] provided a framework for automating the construction and maintenance of rule-based expert systems using Petri nets as a representation for the knowledge base. In [27], Petri nets were used to implement logic programs with negation. In [28], Petri nets were used for reasoning in propositional logic in real-time, a Petri net is constructed for the given proposition logic rule-based expert system; then, using Petri net analysis techniques, all logically implied propositions are deduced in real-time.

Many issues need to be addressed to make Petri nets suitable for application as an inference mechanism in a vision system where data is usually uncertain and incomplete, and where real-time response time is desired. These issues include dealing with uncertainty and efficient implementations.

Researchers have dealt with uncertainty in Petri nets for different purposes. Stochastic Petri nets [29], [30] are a class of Petri nets in which the firing times are considered random variables, and a probability distribution over all transition firing times is formed. Looney was the first to apply Petri nets to fuzzy rule-based reasoning using propositional logic [31], where transitions serve as rules, places serve as propositions, and markings are assigned fuzzy values between 0 and 1. Following Looney, many researchers developed algorithms for reasoning using fuzzy Petri nets. Examples include work done by Chen et al. [32], by Konar et al. [33] and by Scarpelli et al. [34]. Cardoso et al. [35] proposed a possibilistic Petri net model that combined possibility theory and Petri nets to lead to a tool for qualitative representation of uncertain knowledge about a system's state.

The issue of efficient implementation of the Petri nets has also been addressed by many researchers. The RETE algorithm has been applied to reduce the complexity of Petri nets and to achieve linear performance in the number of knowledge base rules [12]. It has been also shown that Petri nets can improve the use of working memory by splitting it into partitions corresponding to places. Petri nets also reduce the tree sizes used in testing [36].

1.4.3 Carried Object Detection

One important problem in understanding human activities is to detect whether a person is carrying an object or not at different times. For example, if a person is carrying an object at time t1 and not carrying it at time t2, we can infer that the person has dropped the object or give it to another person between times t1 and t2. Another problem is detecting left packages in public places.The problem of left package detection has attracted many researchers in the last few years due to increasing concerns about safety in public places, like airports and train stations. In [37], a system is presented that is able to detect if a person carries an object. Spengler and Schiele propose an approach [17] for detecting abandoned objects and tracking people using the CONDENSATION algorithm in monocular sequences. A distributed surveillance system for the detection of abandoned objects in public environments is presented in [15] and [38]. In [18], a multicamera surveillance and tracking system for monitoring airport activities is discussed. In [14], abandoned objects are detects in real-world conditions utilizing logic to differentiate between abandoned objects and stationary people. In [39] abandoned objects are detected using a double background subtraction method. In [40] objects are tracked using a trans-dimensional Markov Chain Monte Carlo tracking model, then the problem of determining if a luggage item is left unattended is solved by analyzing the output of the tracking system in a detection process. In [16], left luggage detection is considered in an event recognition framework where events are represented as hypotheses and recognized in a Bayesian inference framework.

Most of these approaches assumes the left package can be detected by the background modeling component of the surveillance system. If the package is left in an unseen place (e.g. behind a pillar or in a trash pin), then these methods will fail to detect it. The two approaches presented in this thesis differ from these methods in that they don't depend on detecting the left package itself instead they try to infer if a package is left by measuring the differences between different instances of the person. We assume we have different instances of the same person within different cameras and at different times, and that the time separation between different instances of the same person is small, so that he does not change clothes between these instances. Both approaches poses the problem as a classification problem. The first approach, direct classification of silhouettes, classifies the subject's silhouettes for each instance directly to determine whether he is carrying an object or not. The second approach, appearance change detection, determines whether there is a significant change in human appearance between two different instances or not that might be due to an object being carried at one time but not the other. If there is a significant change in the human appearance, additional analysis is conducted to decide whether the person has dropped an object or acquired one from the scene.

For both approaches, we trained SVM classifiers [19] on a laboratory database that contains examples of people seen with and without two common objects, namely backpacks and suitcases. We used a boosting technique, AdaBoost [20], to select the most discriminative features used by the SVMs and to enhance the performance of the classifiers. We give recognition results for each approach and then compare both approaches and describe the advantages of each one.

Chapter 2

Petri Net Models for Event Recognition

In this chapter, we describe our system and its components in details. The main objective of the system is to detect events in surveillance videos based on event models provided by the user. We assume that events can be composed by combining primitive events and previously defined events by spatial, temporal and logical relations. We provide a graphical user interface (GUI) to formulate such event models. Our approach automatically maps each of these models into a set of Petri net filters that represent the components of the event. Lower-level video processing modules (background subtraction, tracking, etc.) are used to detect the occurrence of primitive events. These primitive events are then filtered by Petri nets filters to recognize composite events of interest. This approach is general enough to be applied to any surveillance domain (car parks, airports, indoor scenes, etc.). Inference about temporal, spatial and logical relations between events is performed by the engine independently of the characteristics of the primitive events.

In this chapter, we give details about our system. First, in section 2.1, we provide some background information about Petri nets, their structure and dynamics. In section 2.2, we define an ontology for events. This ontology describes the main concepts in a surveillance domain, like objects, states, events and relations. In section 2.3, an overview of the system and its basic modules is given. Modeling events as Petri nets constructs is described in section 2.4. Petri net models derived from user-defined events are used to detect and recognize events as explained in section 2.5 along with some examples.

2.1 Background

In this section, we describe the basic concepts of Petri nets, their structure and their dynamics. A Petri net is an abstract model of the flow of information in a system [11]. A marked Petri net is a quintuple $(P; T; I; O; M)$, where:

- P = $\{p_1; p_2; ...\}$ is the set of n_p places (drawn as circles in the graphical representation);
- T = $\{t_1; t_2; ....\}$ is the set of n_t transitions (drawn as bars);
- I is the transition input relation and is represented by means of arcs directed from places to transitions;
- O is the transition output relation and is represented by means of arcs directed from transitions to places;
- M = $\{m_1; m_2; .....\}$ is the marking. The generic entry m_i is the number of tokens (drawn as black dots) in place p_i in marking M.

The graphical structure of a Petri net is a bipartite directed graph: the nodes belong to two different classes (places and transitions) and the edges (arcs) are allowed to connect only nodes of different classes.

The dynamics of a Petri net is obtained by moving the tokens in the places by means of the following execution rules:

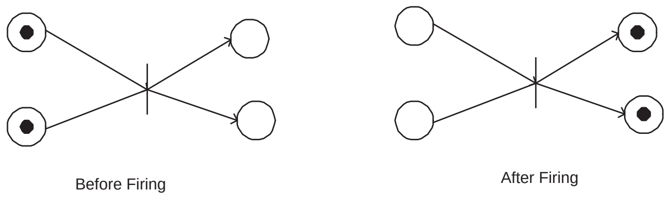


Figure 2.1: Simple Petri Net Before and After Firing

- A transition is enabled in a marking M if all its input places carry at least one token;
- an enabled transition fires by removing one token [1] per arc from each input place and adding one token per arc to each output place.

Figure 2.1 shows a Petri net with one transition. The transition has two input places and two output places. It is shown before and after the firing. Firing the transition removes one token from every input place and inserts a token in every output place.

For more information about Petri nets basics, readers can refer to [11]. One of the main disadvantages of ordinary Petri nets is that for large complex systems the sizes of the nets are unmanageable. High Level Petri Nets(HLPN) are Petri nets whose tokens carry information represented by data structures. HLPN also provides hierarchical structures where compact and manageable descriptions can be obtained while preserving many properties when nets are composed [41].

2.2 Event Ontology

An ontology is a data model that represents a set of concepts within a domain and the relationships between those concepts. It is used to reason about the objects within that

[1]**MOS** Note that more than one token can be removed, if desired, as explained later in this chapter.

domain. In this section, an ontology for event recognition is described. The ontology is described in terms of a Geometric Scene Description (GSD). A GSD is a quantitative object-level scene interpretation in terms of recognized objects and their (possibly varying) locations in the scene. It is assumed that the intermediate vision layer provides this GSD.

2.2.1 Objects

Tracked objects are assumed to be provided by the intermediate vision layer. The following properties are examples of what a GSD can describe.

- Class: Mobile/Contextual.
- Attributes: Color, Position,Orientation.
- Type: Person/Car/Door/Region-of-interest.
- Identifier.

2.2.2 States

A state is defined as a conceptual entity with one or more object for which a qualitative predicate is true over a time interval. Examples are:

- One-object states: Moving/Still.
- Two-objects states: Two mobile objects: Far from/Near.

  One mobile object and one contextual object: Inside/Outside.

2.2.3 Event

We define an event as a significant occurrence that happens at a given place and time. One or more objects may be involved in an event. An event may be primitive or composite.

Primitive Events

A primitive event is the simplest type of events inferred directly from the observables in the video data (e.g. position, trajectory, speed, etc.). Examples are:

- One-object events: Move/Stop, Accelerate/Deaccelerate.
- Two-objects events: Two mobile objects: Approach/Leave, Pickup/Putdown. One mobile object and one contextual object: Enter Area/Exit Area, Open/Close.

Composite Events

A composite event, or a scenario, is composed of states and simpler events connected by spatial, temporal or logical relations. Examples of composite events are:

- Sequences: A sequence is a succession of two or more events.
- Repetitions: Detecting more than one occurrence of the same event may have a special meaning in its context. For example, the different occurrences of the event may be performed by different mobile objects with respect to the same contextual object.
- Negative Events: A negative event is triggered by the absence of some critical observation (e.g a security guard has left his duty position and does not return within 15 minutes).

2.2.4 Relations

Logical Relations

Logical Relations (e.g. AND, OR, NOT) are used in their usual meaning to express different compositions of events.

Temporal Relations

A binary temporal relation is a relation between two events. As an event is represented by an interval or by one point in time. Point-interval temporal logic [42], which is an extension to Allen's interval logic [24], is used to handle different possibilities, which are

- both events are intervals
- both events are points
- one event is an interval and the other is a point.

It is suitable also for representing incomplete information. For example, if there are two events represented by intervals X,Y and it is required to detect instances of X and Y where X's startpoint happens during interval Y. In this case, the relation between the endpoint of X and interval Y is not known (or not significant).

Spatial Relations

A binary spatial relation is a relation between two spatial entities. These entities may be points, lines or regions. A spatial relation can be topological, directional or distance relation [43]. Topological and directional relations are qualitative relations while a distance relation is a quantitative measure of the distance between two objects. A primitive spatial relation is a combination of a topology and a direction.

2.3 System Overview

Figure 2.2 shows the system overview. With the goal of designing a general system that can be configured in different settings, we provide the user a graphical user interface (GUI) that can be used to provide contextual information about the scene by drawing polygons around regions of interest and naming them. Through the GUI, the user can also specify events to be modeled and recognized by building event templates hierarchically from primitive events and previously defined events and joining them by spatial, temporal and logical relations. The Petri net for the event query is inferred from the Petri nets of its components.

The input video is preprocessed by low level vision modules that detect objects by background subtraction. The detected objects are classified and tracked across frames to provide object trajectories. Object trajectories are analyzed to detect primitive events that are parts of the final event query. The detected primitive events represent inputs to Petri net-based recognition modules.

Once an event is recognized by the system, it is reported to the user through a panel so appropriate actions can be taken. The panel displays a keyframe for the event and other information including event time with the ability to rerun video streams where the event takes place.

We will discuss the GUI, the object detection and tracking and the primitive event detection modules in the following subsections. Section 2.4 discusses the Petri net-based event modeling and section 2.5 discusses composite event recognition based on Petri net models.

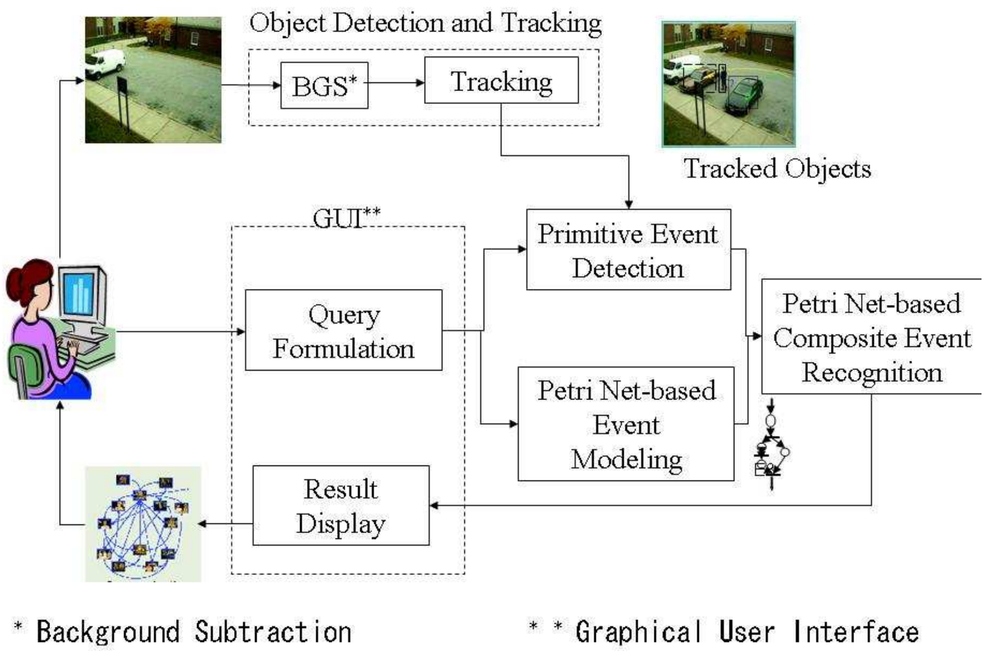

Object Detection and Tracking
BGS*
Tracking
Tracked Objects
GUI**
Primitive Event Detection
Query Formulation
Petri Net-based Composite Event Recognition
Petri Net-based Event Modeling
Result Display
* Background Subtraction
* * Graphical User Interface

Figure 2.2: System Overview

2.3.1 Graphical User Interface (GUI)

With the goal of designing a general system that can be configured in different settings, we are providing the user a graphical user interface (GUI) that can be used to build event models hierarchically and provide contextual information about the scene. Figure 2.3.a is a snapshot of the query design interface.

For each input video source, the view captured by the camera is displayed so that ROIs can be marked on it. There are five lists where information about event models can be edited (added, deleted or modified). These lists are:

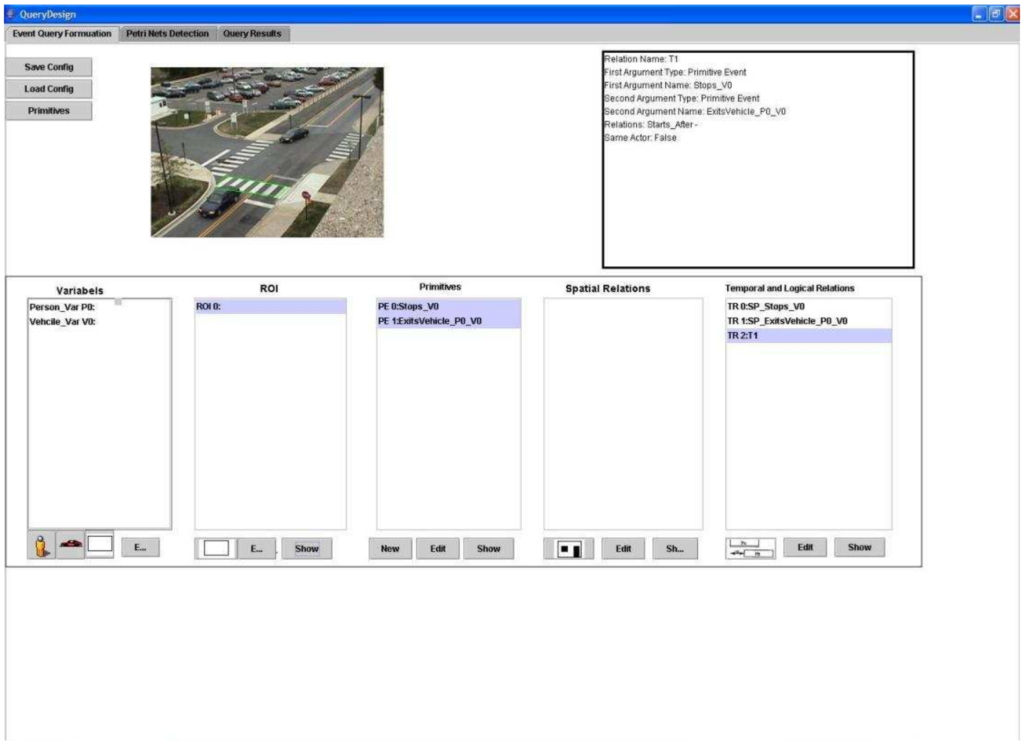
- **Variables**: Variables of the event model should be defined before building the event model. A variable is defined by its name and type. The name is a string unique for this variable. The type is one of the following: Person, Vehicle, ROI or Other.

- **ROIs**: Regions of interest can be marked by drawing polygons around them. Each region should also have a unique name.

- **Primitives**: Primitive events that are part of the event model can be selected from a library of predefined primitive events and then assigned variables. The same primitive event can be used more than once in building the same event model but each time with different variables. For example, if the event model has two vehicle variables: $V1$ and $V2$, the same primitive event *Stops* can be used twice, each time with different variable so we will have *Stops(V1)* and *Stops(V2)* as parts of the event model.

- **Spatial Relations**:

- **Temporal and Logical Relations**: Event models are built incrementally by defining new relations on existing primitive events and existing event models. Defining a new temporal or logical relation requires selecting two operands. These two operands are either primitive events from the **Primitives** list, two previously defined relations from **Temporal and Logical Relations** list or one from each list. A separate window is used to allow the user to enter a name for the relation and select the relationship between the starting and ending points of the two operands. A snapshot of this window is shown in Figure 2.3.b.



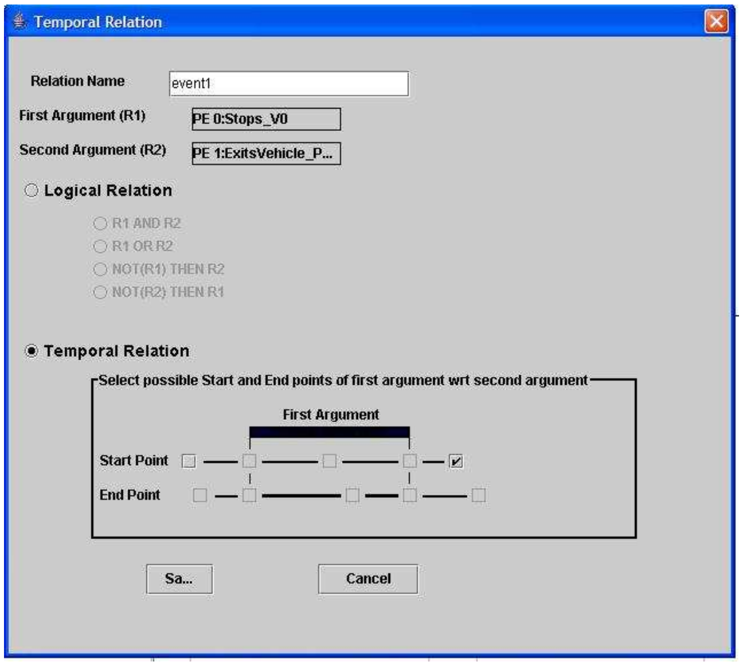
Relation Name: T1
First Argument Type: Primitive Event
First Argument Name: Stops_V0
Second Argument Type: Primitive Event
Second Argument Name: ExitsVehicle_P0_V0
Relations: Starts_After -
Same Actor: False

Person_Var P0:
Vehcile_Var V0:

ROI 0:

PE 0:Stops_V0
PE 1:ExitsVehicle_P0_V0

TR 0:SP_Stops_V0
TR 1:SP_ExitsVehicle_P0_V0
TR 2:T1


Temporal Relation
Select possible Start and End points of first argument wrt second argument
First Argument
Start Point
End Point
Sa...
Cancel

Relation Name: event1
First Argument (R1): PE 0:Stops_V0
Second Argument (R2): PE 1:ExitsVehicle_P...

Logical Relation
- R1 AND R2
- R1 OR R2
- NOT(R1) THEN R2
- NOT(R2) THEN R1

[✓ Temporal Relation]

(b) Logical and Temporal Relation Design Interface

Figure 2.3: Snapshots from Graphical User Interface

2.3.2 Object Detection and Tracking

Detecting and tracking moving objects are widely used as low-level tasks of computer vision applications, such as video surveillance and robotics. Software development of low-level tasks is especially important because it influences the performance of all higher levels of various applications.

Many surveillance systems use background modeling to detect moving objects. One of the problems with most algorithms is the need of an empty scene for initialization. Often this is hard to obtain, and each time something changes in the scene, the initialization needs to be redone. Other problems include changing illumination, waving trees, water, scene changes and shadows. On the other hand, multiple object tracking has been also a challenging research topic in computer vision. It has to deal with the difficulties existing in single object tracking, such as changing appearances, non-rigid motion, dynamic illumination and occlusion, as well as the problems related to multiple object tracking including inter-object occlusion, multi-object confusion. Good surveys about object detection and tracking algorithms can be found in [44, 45].

With the goal of developing a high level event modeling and recognition module independent of these lower level vision modules, we have designed our system so that the detection and tracking results are either pre-computed or performed online by the system. Experiments in Chapter 3 shows examples of both cases.

2.3.3 Primitive Event Detection

As mentioned in Section 2.2, a primitive event is the simplest type of events inferred directly from the observables in the video data (e.g. position, trajectory, speed, etc.).

Primitive events are detected by a separate module whose function is to interpret the data provided by the object detection and tracking module and keep information about objects state in the scene, their position, speed and acceleration. Any change in these features signals the detection of an event.

In the following, we will give some examples of the detection and recognition of six commonly used primitive events, which are: *“Appears, Disappears, Moves, Stops, Enters_ROI, Exits_ROI”*.

An instance of the primitive event *Appears* is said to be detected of an object appears for the first time and remains as a foreground object for at least k frames. On the other hand, an existing object that cannot be tracked for at least k frames is said to disappear, an instance of the primitive event *Disappears* is said to be detected.

Based on the trajectories computed by the object detection and tracking module, the motion of a tracked object is described in terms of its position, speed and acceleration. When the object is moving, it starts decelerating and when the change in object position during a specified number of frames is under a given threshold, an instance of the primitive event *Stops* is detected. On the other hand, if the object is not moving and then it starts accelerating, an instance of the primitive event *Moves* is detected.

For each ROI, there is a boolean that indicates whether or not the object is inside the ROI. A change in the value of that boolean indicates that either *Enters_ROI* or *Leaves_ROI* is detected.

To Test if an object is inside or outside a given ROI, we test whether there is overlap between the bounding box around the object and the ROI polygon. But this can result in a large number of false positives as while the bounding box of an object may overlap the ROI polygon, the object itself may be completely outside the ROI. Figure 2.4a shows a pedestrian whose bounding box intersects a crosswalk ROI, which causes the primitive event *Enters_ROI* to be detected, while the pedestrian is completely outside the crosswalk.

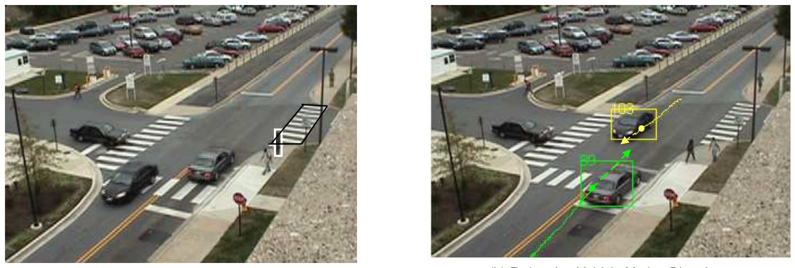

(a) False Positive Example (b) Estimating Vehicle Motion Direction

Figure 2.4: Primitive Event Detection

To solve this problem, we use the bounding box information only. For example, for a pedestrian to be inside a ROI polygon, the bottom line of its bounding box should be inside the ROI polygon. In other words, the pedestrian's feet are inside the ROI polygon, since the bottom line of the bounding box usually touches the pedestrian's feet. For a vehicle to be inside a ROI polygon, we need to ensure that its wheels are inside the polygon. Since we only have the 2D information about the objects, it is not practical to find the wheels. Instead, we estimate the bounding box side that touches the front side of the vehicle. To do this, we first measure the direction in which the vehicle is moving and then identify the bounding box side in this direction. Assuming that this side represents the front of the vehicle, to detect a vehicle *Enters_ROI*, we test whether this line lies inside or at least intersects the ROI polygon. Figure 2.4b demonstrates this approach. We also wait until the overlap area is above a predefined percentage of the ROI area. This approach reduces the false positives rate.

2.4 Petri Net-Based Event Modeling

2.4.1 Notations

In this section, we will describe some notations about Petri net elements that we are going to use in our framework. These notations include examples of ways into which HLPNs extend the ordinary Petri nets [13, 41].

Transitions

- **Immediate Transitions:** The same as transitions in ordinary Petri nets. This means that the transition fires immediately when every input place has the required tokens for firing.
- **Conditional Transitions**: A conditional transition has additional firing conditions that should be satisfied for the transition to fire. In other words, the transition fires when every input place has the required tokens and the associated conditions are satisfied. A conditional transition is represented by a thin bar.
- **Composite Transitions**: In order to simplify the structure of large nets, composite transitions can be used as normal transitions in a Petri net but represent subnets themselves. A composite transition is connected to the outside net by a set of incoming and outgoing edges and places. A composite transition is represented by an unfilled rectangle.

Note that for all these types of transitions, the conditions can be set based on the *number* of tokens that satisfy a certain condition in the input places. In this case, when the transition fires, more than one token can be removed from each of the input places. This can be useful in a number of applications as explained in the next section.

Tokens

- **Regular Tokens**: Regular tokens are used for marking only and don't hold any specific information. A regular token is drawn as a black filled circle.

- **Colored Tokens**: Colored tokens hold information represented by data structures suitable for the application.

2.4.2 Event Modeling

In our framework, an event in modeled as a Petri net whose structure is derived from the event structure. Each token is represented by an array, where each object variable has a position in the array. Different instances of the same event are represented by tokens, one for each instance. Each token of them will have the same structure, i.e. array, but, maybe, with different values for the variables.

The simplest case is for a primitive event. Figures 2.5.a and 2.5.b show examples of the Petri net models for the primitive events *E1: Stops(V)* and *E2: Exits_Vehicle(P,V)*. From the figures we can see that, for a primitive event, the Petri net model consists of a source place, a conditional transition, and a sink place:

- **Source Place**: A dummy token representing the primitive event is initially placed in the source place. The associated variables of the primitive event are unassigned in the array representing the token. These variables are set when the conditional transition fires.

- **Conditional Transition**: The firing condition for this transition represents the occurrence of the primitive event, as detected by the lower vision modules. At that time, the variables of the dummy token are instantiated by the values obtained from the lower vision modules and the token is moved from the source place to the sink place. A new dummy token, with unassigned variables, is placed in the source place. Note that this represents a self loop in the Petri net. We do not show these self loops in the models for sake of clarity of the figures.

- **Sink Place**: Tokens reaching this place represent detected instances of the event model.

As mentioned above, a token in this model is an array of length equals the number of variables in the primitive event. Tokens in the first example, in Figure 2.5.a, are arrays of one entry, for the variable *V*, whereas for the second example, in Figure 2.5.b, tokens are arrays of two entries, for the variables *P* and *V*.

As explained before, composite events are built incrementally by joining simpler events, two at a time, by temporal and logical relations. In the same way, models for these composite events are constructed from models of its subevents joined by appropriate connections, i.e. transitions and places, to reflect these temporal and logical relations. For example, Figure 2.5.c shows the Petri net model for the composite event *E3: E1(V)* ***Before*** *E2(P,V)*, whereas Figure 2.5.d shows the Petri net model for the composite event *E4: E1(V1)* ***And*** *E2(P,V2)*. From the figures we can see that, for a composite event, the Petri net model consists of a source place, one or more composite transitions, and a sink place:

- **Source Place**: Similar to primitive events, a dummy token representing the union of all different variables of its subevents is initially placed in the source place. The associated variables of the primitive event are unassigned in the array representing the token.

- **Composite Transition**: Each composite transition represents a subevent.

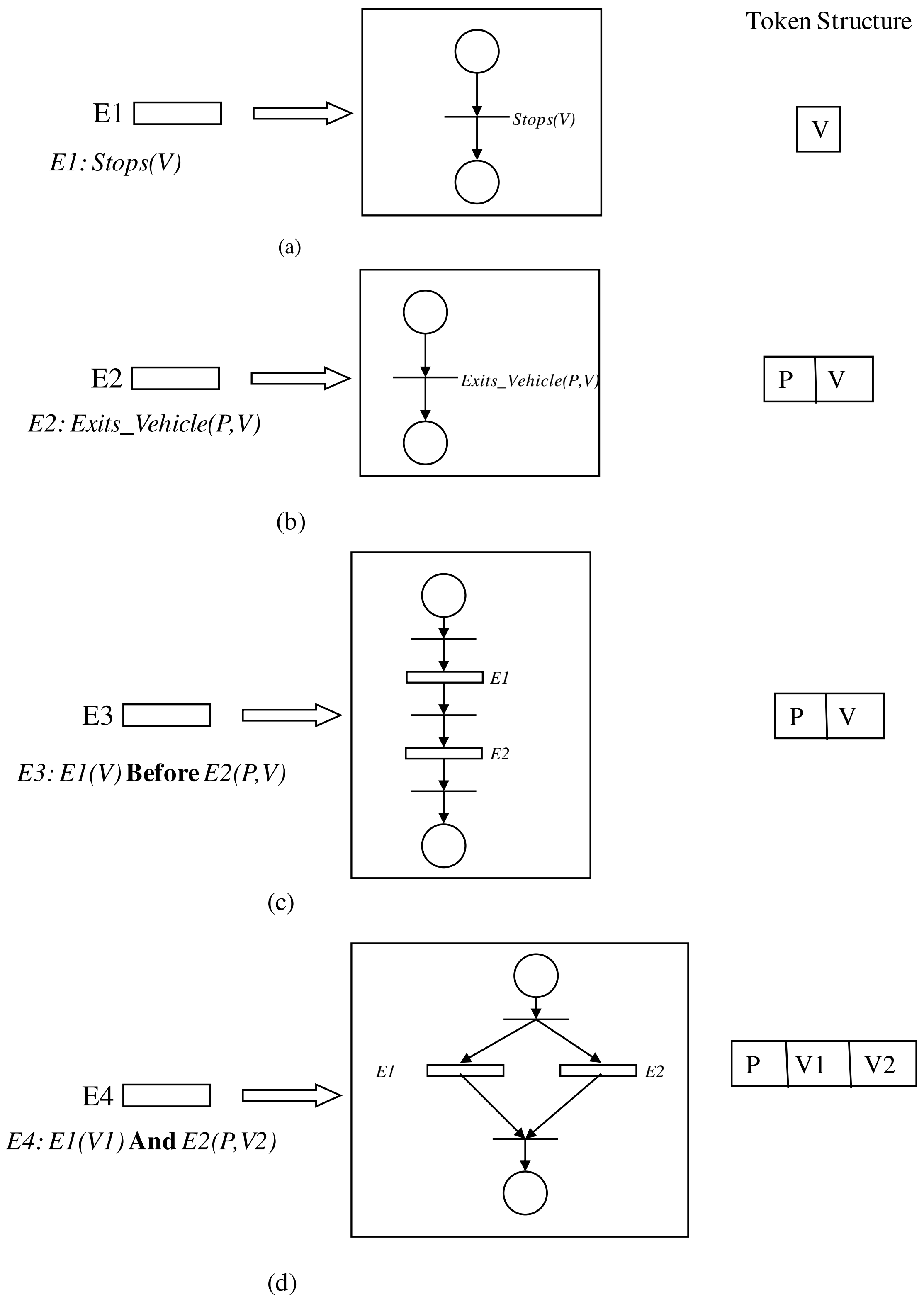


Figure 2.5: Petri Net Models for Example Events

- **Sink Place**: Similar to primitive events, tokens reaching this place represent detected instances of the composite event.

Tokens in a composite event is the union of the tokens of the subevents such that every variable has only one entry. For example, tokens in this model are arrays of two entries, for the variables *P* and *V*. Tokens in this model are arrays of three entries, for the variables *R*, *V1* and *V2*.

2.5 Petri Net-Based Composite Event Recognition

The detected primitive events are the input for the Petri net recognition module whose function is to recognize composite events. The use of Petri nets for event recognition has two important advantages:

- Petri nets reduce the number of checked events whenever a primitive event is detected.
- Petri nets facilitate the process of binding labels (generated by the tracking module) to token variables.

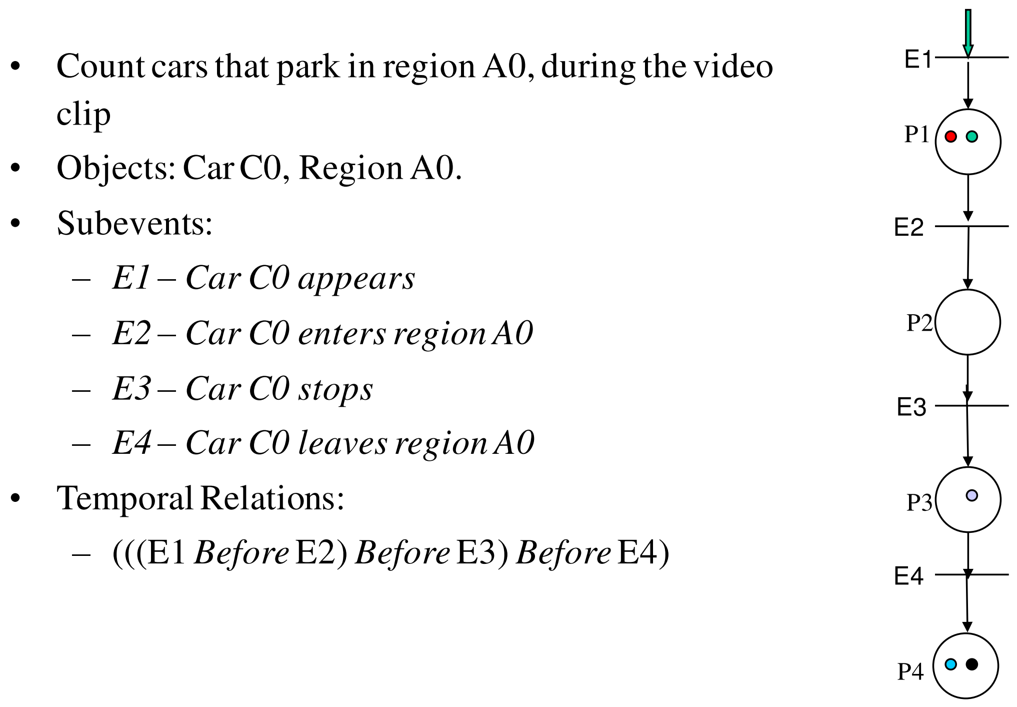
For each composite event to be recognized, we maintain a list of enabled transitions. An enabled transition is a transition where all its input places have tokens but the associated event has not occurred yet. The Petri nets are reevaluated only when a primitive event is detected. When this occurs, we check only the list of enabled transitions to test if any of them is waiting for this primitive to fire. So, we need not check all transitions in the net. When a transition is enabled and the associated primitive is detected, the transition fires. Firing a transition removes tokens from input places, inserts tokens in output places and updates the list of enabled transitions. The fact that we check only the list of enabled transitions provides an efficient implementation, since usually the number of enabled transitions is small.

When a primitive event is detected, its objects have to be matched with tokens from input places. If there are more than one input place to the same transition, then tokens from these places have also to be matched to see whether there is a combination of actors that satisfy the event so far. For a given transition to fire, every possible combination of tokens is tested and a new token is placed in the output place only if a match occurs. The fact that only a small number of these combinations will match reduces the expected number of times this matching process is required. In this way, the Petri net transitions act as filters to filter the large amount of detected primitive events and only keeps information about the relevant ones.

In the following, we give examples to illustrate basic ideas described in this section and the previous sections.

Example 1

Assume we have a parking area and we want to count the number of vehicles that used this area during a given period of time. Here, we have two variables, a variable representing the vehicle, $V0$, and a variable representing the parking area region, $R0$. The Petri net model in this case is a sequence of the following primitives: "*Appears(V0), Enters_ROI(R0, V0), Stops(V0)* and *Leaves_ROI(R0,V0)*". In Figure 2.6, the Petri net corresponding to this sequential order is shown. Variable $V0$ is assigned many labels during the recognition process. Whenever a car appears, a new token is inserted in the first place, P1. Whenever a car enters the parking area, its token is moved from P1 to P2, and so on. At the end of the detection, the number of tokens in the place P4 is the number of cars that stopped in the parking area and then left, and the number of tokens



- Count cars that park in region A0, during the video clip
- Objects: Car C0, Region A0.
- Subevents:
  - *E1 – Car C0 appears*
  - *E2 – Car C0 enters region A0*
  - *E3 – Car C0 stops*
  - *E4 – Car C0 leaves region A0*
- Temporal Relations:
  - (((E1 *Before* E2) *Before* E3) *Before* E4)

Figure 2.6: Petri Net Representation for Counting Cars

in P3 are the number of cars that stopped in the parking area and have not left yet.

Example 2

Another event, is *Vehicle_Exchange* event. In this event, two vehicles enter the parking area and park. Then a person leaves one vehicle and enters the second vehicle. After that, the second vehicle should leave. In this example, there are three variables, $V0$ and $V1$ representing the vehicles and variable $P0$ representing the person. The Petri net for this event is shown in Figure 2.7. In this event, we are not interested which vehicle arrives first, so there is no relation between E1 and E2. Whenever a vehicle arrives in the parking area and parks, a token is placed in both places P1 and P2. A token in P1 will not be moved to P3 until a person exits the vehicle represented by this token. Now, a token in P3 represents the combination of this person and this vehicle, and hence

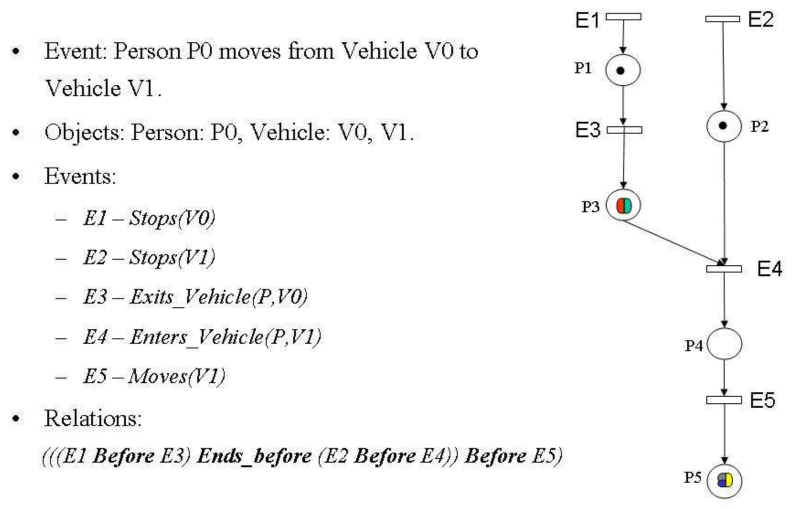

- Event: Person P0 moves from Vehicle V0 to Vehicle V1.
- Objects: Person: P0, Vehicle: V0, V1.
- Events:
  - *E1 – Stops(V0)*
  - *E2 – Stops(V1)*
  - *E3 – Exits_Vehicle(P,V0)*
  - *E4 – Enters_Vehicle(P,V1)*
  - *E5 – Moves(V1)*
- Relations:
  (((E1 ***Before*** *E3)* ***Ends_before*** *(E2* ***Before*** *E4))* ***Before*** *E5)*

Figure 2.7: Petri Net Representation for Car Exchange Event

contains two colors. In the same way, tokens from P2 and P3 are not matched until the primitive event *Enters_Vehicle* is detected with person matching the person in P3's token and a car matching the car in P2's token. A new token is created (now representing the two vehicles and the person) and inserted in P4.

Example 3 - Negative Events

As mentioned in Section 2.2.3, a negative event is triggered by the absence of some critical observation, e.g. a security guard has left his duty position and does not return within 15 minutes. To detect the absence of event X, we need to set a limiting event Y, so that if Y is detected we can safely infer that X has not occurred. For the given example:

- Event(X) – Security guard appears in his duty position.

- Event(Y) – 15 minutes has passed since last guard appearance.

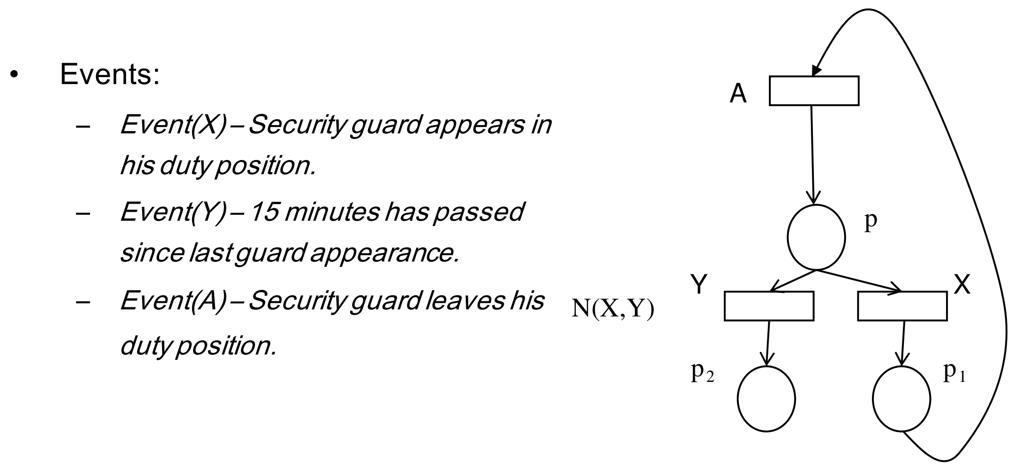
In this case, events X and Y are represented by transitions that share the same input place p, as shown in Figure 2.8. If a token t in place p performs event X before Y, token t moves to place p1 and is ignored. If t does not perform event X until event Y is detected, token t moves to place p2 and participates as an output of negative event N(X,Y).

Note also that this example shows a non-trivial loop, where the security guard periodically returns to the duty position. If event X is detected, a token is placed in place p1, indicating that the guard is in his duty position. When the guards leaves the position, represented by the firing of transition A, a new token is placed in the common place p and the timer for event Y is reset, starting a new loop. Figure 2.9 shows a generalization of the negative event example, where the guard has to visit a number of positions in sequence periodically within a certain amount of time. The figure also show a longer loop.

Example 4 - Counting Events

By a counting event we mean an event that requires the Petri net to count the number of certain events. Examples include counting the number of people or cars that enter a certain area of interest (as in Example 1), detecting events that involve more than one entity, etc.

Counting events can be detected by counting the tokens in places that satisfy certain conditions. In additions, the conditions of the transitions can be set such that the transition do not fire unless a number of tokens are available.



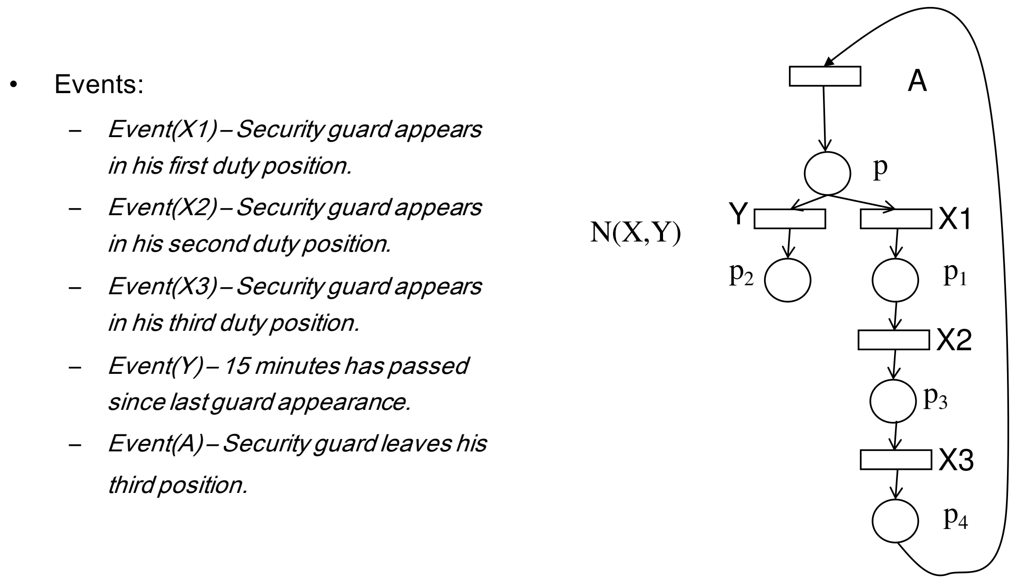
- Events:
  - *Event(X) – Security guard appears in his duty position.*
  - *Event(Y) – 15 minutes has passed since last guard appearance.*
  - *Event(A) – Security guard leaves his duty position.*

Figure 2.8: Petri Net Representation for Negative Event N(X,Y)

Figure 2.10 show the car exchange example, where the user is interested of exchanges that involves more than two persons from one car to the other. Note that the Petri net model for this case is exactly the same as the one in Figure 2.7, with the exception of the condition on the composite transition E5. The transition E5 will be enabled if the vehicle moves ***and*** the number of tokens that have the structure $(V1, Pi)$, for any i. is more than two.

2.6 Conclusion

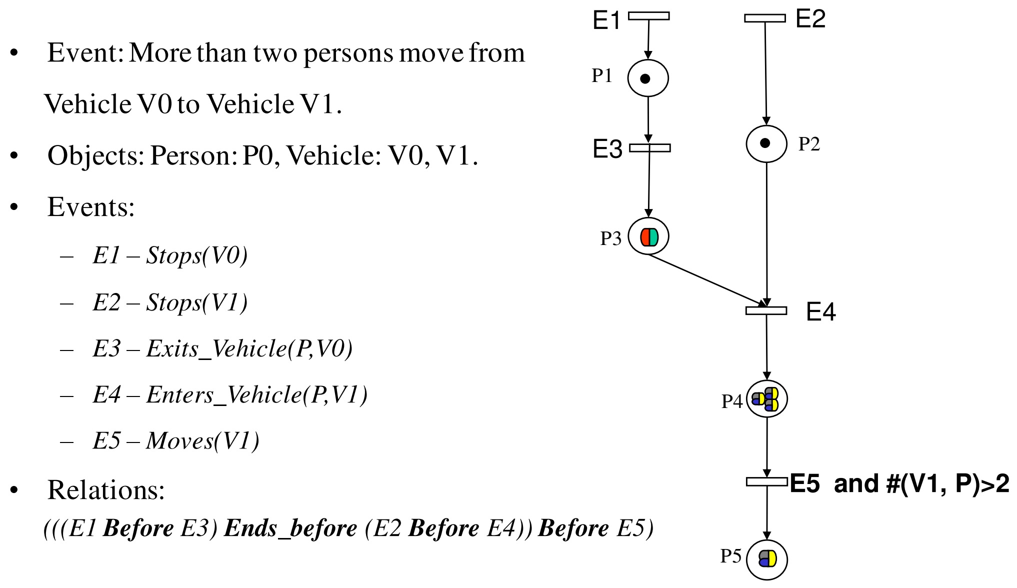
We have described our event modeling and recognition system in details. Using our event modeling approach based on Petri nets, we have shown that models for new events can be built easily by combing simpler event models by temporal, logical and spatial relations using our GUI. The mapping into Petri net models are performed automatically. Petri nets also provide a formal and natural method that is easy to understand. One of the main advantages of event recognition based on Petri net models is that the large number



- Events:
  - *Event(X1) – Security guard appears in his first duty position.*
  - *Event(X2) – Security guard appears in his second duty position.*
  - *Event(X3) – Security guard appears in his third duty position.*
  - *Event(Y) – 15 minutes has passed since last guard appearance.*
  - *Event(A) – Security guard leaves his third position.*

Figure 2.9: Petri Net Representation for the generalization of example 3

of irrelevant observations does not affect the performance of the recognition process.



- Event: More than two persons move from Vehicle V0 to Vehicle V1.
- Objects: Person: P0, Vehicle: V0, V1.
- Events:
  - *E1 – Stops(V0)*
  - *E2 – Stops(V1)*
  - *E3 – Exits_Vehicle(P,V0)*
  - *E4 – Enters_Vehicle(P,V1)*
  - *E5 – Moves(V1)*
- Relations:
  (((E1 ***Before*** *E3)* ***Ends_before*** *(E2* ***Before*** *E4))* ***Before*** *E5)*

Figure 2.10: Petri Net Representation for Car Exchange Event involving multiple persons

Chapter 3

Experiments I

Our goal is to evaluate the performance of the system across different natural scenes and in detecting events with increasing level of complexity ranging from primitive events involving one object to more complex events involving multiple objects and multiple logical and temporal relations. In the first set of experiments, we applied our system on a dataset of video sequences provided as a part of ETISEO project [46]. The length of these sequences range from 800 frames to 3000 frames. In these sequences, required events to be detected range from single primitives to simple sequences of two or three primitives. In the second set of experiments, we applied our system on a longer video sequence (14 minutes$= 14 \times 60 \times 30 = 25200 frames$) to detect and recognize events and violations performed by pedestrians and vehicles in a traffic intersection. Events in this case are more complex and include multiple actors and multiple logical and temporal relations.

3.1 ETISEO Dataset

We applied our system on a dataset of video sequences. Those videos are provided as a part of ETISEO project [46], a research project sponsored by the French government, whose aim is to evaluate vision techniques for video surveillance applications. It focuses on the treatment and interpretation of videos involving pedestrians and (or)vehicles, indoors or outdoors, obtained from fixed cameras. Information about these video sequences is provided in Table 3.1. The first sequence (ETI-VS2-AP-11) was taken at an apron scene, from two camera views. The second sequence (ETI-VS2-RD-6) and the fifth sequences (ETI-VS2-RD-10) were taken at different road scenes, from only one camera. The third sequence (ETI-VS2-BE-19) was taken at a building entrance, from a camera monitoring the outdoor scene and another camera monitoring the indoor scene of the entrance. The fourth sequence (ETI-VS2-MO-1) was taken at a metro station, from one camera view. Information about events to be detected for each sequence is provided along with other contextual information about the scenes. Table 3.1 shows the set of events modeled and detected by our system. More information can be obtained from [46]. Figure 3.1 shows frames representing each of these sequences, with regions of interest marked up.

3.1.1 Event Definitions

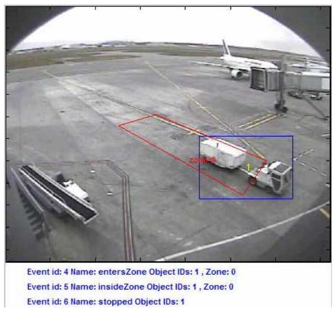
In this section, we will discuss the representation of the events to be detected in our Petri nets framework. The primitive library used has nine primitive events. In the following, we list these primitives along with the time span associated with each primitive, based on ETISEO definitions and other assumptions we made.

- **Appears**: the object appears and remains as a foreground object for at least 10 frames +/- 5.

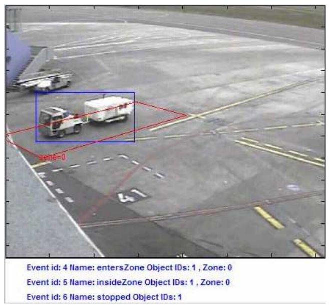
- **Moves**: the object starts moving and the change in object position for the last k frames is above a given threshold +/- 5.

Table 3.1: ETISEO Video Sequences Summary

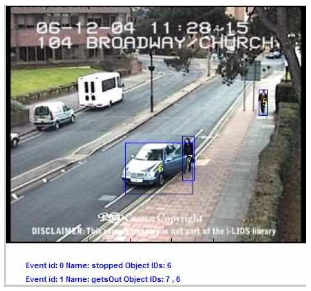
Sequence Name	ETI-VS2-AP-11	ETI-VS2-RD-6	ETI-VS2-BE-19	ETI-VS2-MO-1	ETI-VS2-RD-10
Number of available views	2	1	2	1	1
Scene information	Outdoor Apron	Outdoor Road	Indoor-Outdoor Building Entrance	Indoor Metro	Outdoor Road
Number of frames	804	1200	1025	1255	2936
Objects types	vehicle	person vehicle	person vehicle	person bag	person vehicle
Events	stopped; inside_zone; enters_zone; empty_area	gets_in; gets_out; stopped	stopped; gets_out; inside_zone; enters_zone; exits_zone; changes_zone; door_control	waiting; picks_up; puts_down; exchange_object	gets_in; gets_out; stopped



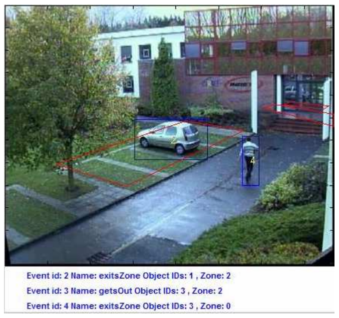
```
Event id: 4 Name: entersZone Object IDs: 1 , Zone: 0
Event id: 5 Name: insideZone Object IDs: 1 , Zone: 0
Event id: 6 Name: stopped Object IDs: 1
```



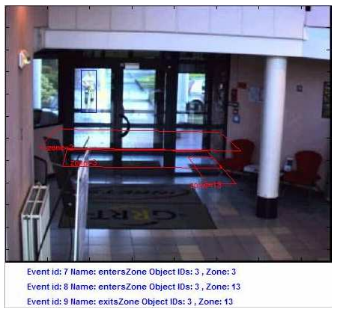
```
Event id: 4 Name: entersZone Object IDs: 1 , Zone: 0
Event id: 5 Name: insideZone Object IDs: 1 , Zone: 0
Event id: 6 Name: stopped Object IDs: 1
```



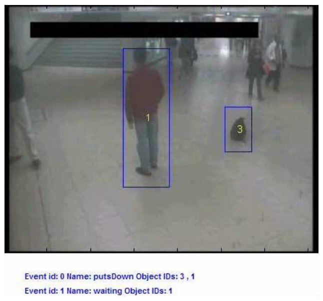
```
Event id: 0 Name: stopped Object IDs: 6
Event id: 1 Name: getsOut Object IDs: 7 , 6
```



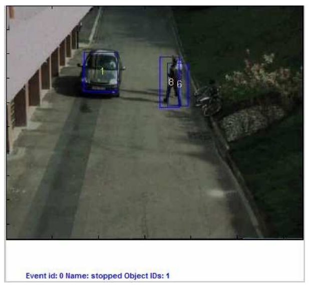
```
Event id: 2 Name: exitsZone Object IDs: 1 , Zone: 2
Event id: 3 Name: getsOut Object IDs: 3 , Zone: 2
Event id: 4 Name: exitsZone Object IDs: 3 , Zone: 0
```



```
Event id: 7 Name: entersZone Object IDs: 3 , Zone: 3
Event id: 8 Name: entersZone Object IDs: 3 , Zone: 13
Event id: 9 Name: exitsZone Object IDs: 3 , Zone: 13
```



```
Event id: 0 Name: putsDown Object IDs: 3 , 1
Event id: 1 Name: waiting Object IDs: 1
```



```
Event id: 0 Name: stopped Object IDs: 1
```

Figure 3.1: Annotated Frames from ETISEO Sequences

- **Stops**: the change in object position for the last k frames is below a given threshold +/- 5.
- **Enters_ROI**: first part of the vehicle in the zone until the whole vehicle is inside zone OR first foot inside zone for a person +/- 5 frames.
- **Exits_ROI**: first part of the vehicle outside zone until the whole vehicle is outside zone OR last foot inside zone for a person +/- 5 frames.
- **Enters_vehicle**: the person disappears in the vicinity of the vehicle +/- 5.
- **Exits_vehicle**: the person appears in the vicinity of the vehicle +/- 5.
- **puts_down**: last frame carried object is connected_with the person +/- 10 frames.
- **picks_up**: 1st frame the carried object is connected_with the holder +/- 10 frames.

In our system, we model other events as simple scenarios (a sequence of two or three primitives). In the following, we list these scenarios along with its components based on ETISEO definitions and other assumptions we made.

- **inside_zone**:
  - **Variables** Vehicle: V, ROI: R
  - **Primitive Events** P1: *Enters_ROI*(V,R), P2: *Exits_ROI*(V,R)
  - **Scenario inside_zone** :P1 *Before* P2
- **empty_area**:
  - **Variables** Vehicle: V, ROI: R
  - **Primitive Events** P1: *Enters_ROI*(V,R), P2: *Exits_ROI*(V,R)

  - **Scenario empty_area** :P2 *Before* P1

- **gets_in**:
  - **Variables** Vehicle: V, Person: P
  - **Primitive Events** P1: *moves_towards*(V,P), P2: *Enters_vehicle*(V,P)
  - **Scenario gets_in** :P1 *Before* P2

- **gets_out**:
  - **Variables** Vehicle: V, Person: P
  - **Primitive Events** P1: *Exits_vehicle*(V,P), P2: *moves_away_from*(V,P)
  - **Scenario gets_out** :P1 *Before* P2

- **changes_zone**:
  - **Variables** Object: O, ROI: R1, ROI: R2
  - **Primitive Events** P1: *Exits_ROI*(O,R1), P2: *Enters_ROI*(O,R2)
  - **Scenario changes_zone** :P1 *Before* P2

  Object O may be a person or vehicle.

- **exchange_object**: the second person holds the object + 30 frames
  - **Variables** Object: O, Person: P1, Person P2: R2
  - **Primitive Events** P1: *puts_down*(P1, O), P2: *picks_up*(P2, O)
  - **Scenario changes_zone** :(P1 *Before* P2) *Or* (P1 *Meets* P2)

3.1.2 Results

Based on the detection and tracking results [1], our system is used to model and detect the set of events described on Table 3.1. The output of the system was converted into XML format, where for each event instance, the following attributes are computed:

- **Id**: Integer characterizing the event, unique for a video clip,
- **Name**: Name of the event (identification of the event). The list of event names to recognize in a sequence is delivered with the video,
- **Start and end time**: Integers corresponding to First and last frame of event detection,
- **Physical objects**: List of physical objects ID involved in this event (Id of objects used in the tracking phase),
- **Contextual objects**: List of contextual objects ID involved in this event (Id of objects described in the context, provided with the video data set),

The output was then evaluated by the ETISEO group and results are provided to each participant. Two metrics based on the number of detected events are provided, namely, precision and sensitivity. For each sequence we define:

- **The True Positive(TP)**: the system has detected a real event (exists in reference data and results).
- **The False Negative(FN)**: a real event has been missed by the system (exists only in reference data).

[1] We thank Son Dinh Tran for providing us with the detection and tracking results.

- **The False Positive(FP)**: the system has detected a situation that is not real (exists only in results).
- **Precision**: TP / (TP + FP).
- **Sensitivity**: TP / (TP + FN).

Table 3.2 shows the detection results for sequence ETI-VS2-AP-11, First Camera. Table 3.3 summarizes the detection results for all sequences. As shown in these tables, good recognition results are obtained. A precision value of 1.00 is obtained for all sequences - no false positives are detected. On the other hand, the sensitivity is not as good as precision - a value of 0.76 is obtained, which means that a large number of false negatives has been detected by the system. This can be explained by the high dependency of the event detection module on results provided by lower level vision modules, the background subtraction and tracking. We have also found that most of these false negatives are *Stops* and *Moves* events that have not taken place but detected by the event detection module. The thresholds set by the event detection module for the *Stops* and *Moves* events are domain-dependent. The current thresholds are learned by training on other longer videos and hence resulting in large number of false negatives when applied to the Etiseo videos.

3.2 Traffic Intersection Monitoring

We applied our system to monitor a traffic intersection. Our purpose is to analyze pedestrians and vehicles behaviors and detect and record traffic violations as they occur; traffic citations could then be issued to vehicle owners. This also can be used as a tool to analyze video archives to study pedestrians and vehicle behaviors in the intersection; based

Table 3.2: ETISEO Results for Sequence ETI-VS2-AP-11, C4

Results for scenario empty_area	
True Positives	2
False Positives	0
False Negatives	0
Precision	1.00
Sensitivity	1.00
Results for scenario enters_zone	
True Positives	2
False Positives	0
False Negatives	0
Precision	1.00
Sensitivity	1.00
Results for scenario inside_zone	
True Positives	2
False Positives	0
False Negatives	0
Precision	1.00
Sensitivity	1.00
Results for scenario stopped	
True Positives	1
False Positives	0
False Negatives	1
Precision	1.00
Sensitivity	0.50
Overall Performance for ETI-VS2-AP-11-C4	
Number of True Positives	**7**
Number of False Positives	**0**
Number of False Negatives	**1**
Precision	**1.00**
Sensitivity	**0.88**

Table 3.3: ETISEO Performance Results for All Sequences

Performance results for ETI-VS2-AP-11-C4.xml is:	
Number of True Positives	7
Number of False Positives	0
Number of False Negatives	1
Precision	1.00
Sensitivity	0.88
Performance results for ETI-VS2-AP-11-C7.xml is:	
Number of True Positives	7
Number of False Positives	0
Number of False Negatives	1
Precision	1.00
Sensitivity	0.88
Performance results for ETI-VS2-RD-6-C7.xml is:	
Number of True Positives	2
Number of False Positives	0
Number of False Negatives	0
Precision	1.00
Sensitivity	1.00
Performance results for ETI-VS2-BE-19-C1.xml is:	
Number of True Positives	5
Number of False Positives	0
Number of False Negatives	3
Precision	1.00
Sensitivity	0.62
Performance results for ETI-VS2-BE-19-C4.xml is:	
Number of True Positives	9
Number of False Positives	0
Number of False Negatives	3
Precision	1.00
Sensitivity	0.75
Performance results for ETI-VS2-MO-1-C1.xml is:	
Number of True Positives	4
Number of False Positives	0
Number of False Negatives	0
Precision	1.00
Sensitivity	1.00
Performance results for ETI-VS2-RD-10-C4.xml is:	
Number of True Positives	4
Number of False Positives	0
Number of False Negatives	4
Precision	1.00
Sensitivity	0.50
Overall Performance results:	
Number of True Positives	**38**
Number of False Positives	**0**
Number of False Negatives	**12**
Precision	**1.00**
Sensitivity	**0.76**

on this analysis, redesign steps can be taken to reduce the risk of accidents, for example. Here, we define and detect three types of safety violations and compare them to the expected normal behaviors. Specifically, we are interested in detecting:

- **Stop Sign Events**: Vehicles that stop at the stop sign and vehicles that don't stop
- **Road Crossing Events**: Pedestrians who cross the road using marked crosswalks and those who don't use the crosswalks
- **Right of Way Events**: Vehicles that yield to pedestrians in crosswalks and those that don't yield to pedestrians

In the following, we will define these events and show some experimental results.

3.2.1 Event Definitions

In this section, we will discuss the representation of the traffic intersection events in our Petri nets framework. The primitive library used has six primitive events, which are: "*Appears*, *Disappears*, *Moves*, *Stops*, *Enters_ROI*, *Exits_ROI*". Figure 3.2 shows the Petri net representations of the traffic intersection events. A place marked with * is the output place for the event.i.e. tokens in this place represent recognized instances of the event.

Stop Sign Events Figure 3.2a shows the Petri net representation of the event "A vehicle stops before the stop sign". It has two variables, R which represents the ROI where vehicles should stop before the stop sign and V which represents the vehicle. It also has 3 primitive events, *Enters_ROI*(R,V), *Stops*(V) and *Exits_ROI*(R,V). A vehicle entering the region R and stopping before leaving the region should satisfy this event model.

To detect vehicles that do not stop at the stop sign, we model this as a negative event whose limiting event is *Exits_ROI*(R,V). Normal behavior and violation can be modeled as follows:

- **Variables** Vehicle:V, ROI:R
- **Primitive Events** P1: *Enters_ROI*(V,R), P2: *Stops*(V), P3: *Exits_ROI*(V,R)
- **Temporal Relations** T1:P2 *Before* P3, T2:*NOT*(P2) *Before* P3, T3:P1 *Before* T1, T4:P1 *Before* T2
- **Normal Behavior** N1(V,R):T3
- **Violation**V1(V,R):T4

Road Crossing Events To cross the road legally, a pedestrian goes from one road side to another side using the crosswalk. If he does not use the crosswalk, it is considered a safety violation. This violation is modeled as a negative event whose limiting event is that the pedestrian changes the roadside without using the crosswalk. These events are modeled as follows:

- **Variables** Pedestrian: P, ROI: crosswalk, ROI: roadside1, ROI: roadside2
- **Primitive Events** P1: *Exits_ROI*(P,roadside1), P2: *Enters_ROI*(P,crosswalk), P3: *Enters_ROI*(P,roadside2)
- **Temporal Relations** T1: P2 *Before* P3, T2: *NOT*(P2) *Before* P3, T3: P1 *Before* T1, T4: P1 *Before* T2
- **Normal Behavior** N1(P,crosswalk,roadside1,roadside2): T3
- **Violation**V1(P,crosswalk,roadside1,roadside2): T4

Figure 3.2b shows the Petri net representations.

Right of Way Events Pedestrians have the right of way over vehicles in crosswalks. A vehicle arrives at the stop sign area after a pedestrian starts crossing the road using crosswalk should not move until he leaves the crosswalk (normal behavior). If the vehicle starts moving while the pedestrian is still in the crosswalk, this vehicle has committed a violation. These events are modeled as follows:

- **Variables** Pedestrian: P, Vehicle: V, ROI: crosswalk, ROI: stopsignarea
- **Primitive Events** P1: *Enters_ROI*(P,crosswalk), P2: *Exits_ROI*(P,crosswalk), P3: *Enters_ROI*(V,crosswalk), P4: *Exits_ROI*(V,crosswalk)
- **Composite Events** C1: N1(V,stopsignarea)
- **Temporal Relations** T1: P1 *Before* P2, T2: P3 *Before* P4, T3: T2 *During* T1
- **Normal Behavior** N3(P,V,crosswalk,stopsignarea):T1 *Overlaps* C1
- **Violation** V3(P,V,crosswalk,stopsignarea): T3

Figure 3.2c shows the Petri nets representations.

3.2.2 Preprocessing

The low level processing includes the background subtraction and the tracking. In our system, we used an adaptive background subtraction technique to segment foreground regions from the background. Adaptive background subtraction techniques, in general can adapt to slow changes of illumination by recursively updating the background model. We use the kernel density estimation method described in [47] to model background pixels. The model keeps a sample of intensity values for each pixel in the image

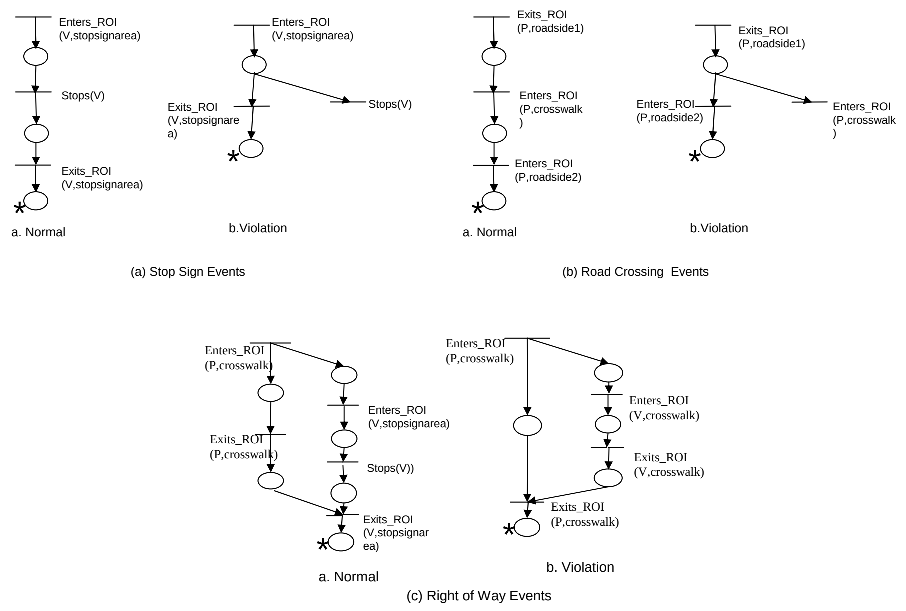


Figure 3.2: Traffic Monitoring Events

and uses this sample to estimate the probability density function of the pixel intensity using kernel density estimation. The model can handle situations where the background of the scene is cluttered and not completely static but contains small motion due to moving branches and bushes. The model is updated continuously and therefore adapts to changes in the scene background. Blobs are constructed by a connected component module that groups foreground pixels into corresponding blobs.

Tracking objects through the scene is done by finding correspondence between objects in two consecutive frames. These correspondences are determined by finding overlapping blobs in these two frames, assuming that the change between consecutive frames is limited. Occlusions and object interactions lead to blob merging and splitting making the tracking complex. In our system, we keep a list of the current entities. An entity is either a single object or a group of objects whose blobs are merged.

For the current frame, and for each blob, we find all entities that overlap with this blob in the previous frame. If the blob overlaps a single entity, then this entity's current position is updated to reflect this new data. If the blob overlaps more than one entity, this signals either an occlusion or objects becoming near to one another. Once this merge is detected, a new entity is created and added to the list of current entities. The overlapping entities are also removed from the list and added as children for the new entity. An appearance-based model is built for each entity before removing it and stored in the new entity to be used in matching entities when they split. If more than one blob overlap a single entity, there are two possibilities. If the entity represents a single object, this means that object fragmentation has occurred and in this case, we consider the new object bounding box as the union of these blobs' bounding boxes. Whereas if the entity represents more than one object, i.e. group split, we need to restore the identity of each object after the split. Since we are storing the appearance models before merging, we can match these models with the current blobs' appearance models so that each blob describes a single entity before merge.

Figure 3.3 shows an example of merging and splitting with objects being correctly matched before a merge and after a split. We ignore all blobs whose size is below a threshold. Two blobs are said to overlap if the overlap area is at least 50% of the smaller blob.

Generally, using only blob information to track objects has limitations. This can be enhanced by augmenting the tracker with object location and shape estimator such as Kalman filter to predict position and shape of the object.

Object classification is based on the geometry of the objects, mainly the aspect ratio of height and width of the object bounding box. Training data are provided to the classifier off-line and objects are classified into the following classes: pedestrian, vehicle, large vehicle (e.g. buses and trucks) and bicycle.

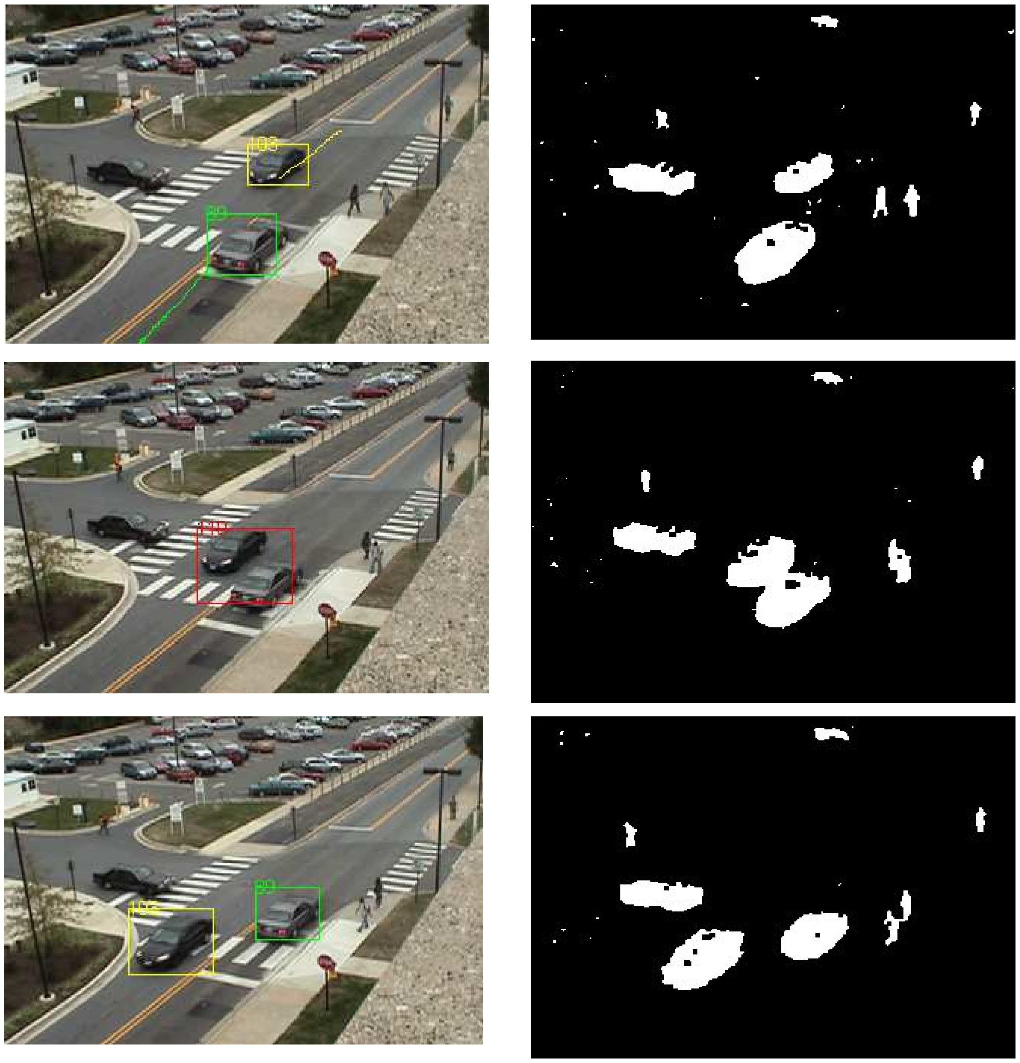


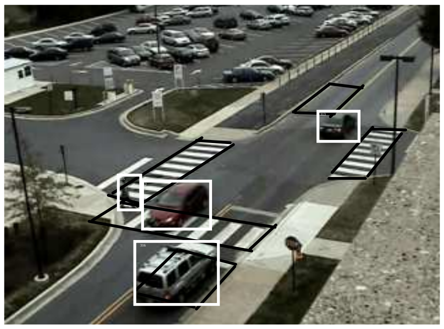
Figure 3.3: Tracking Results

3.2.3 Results

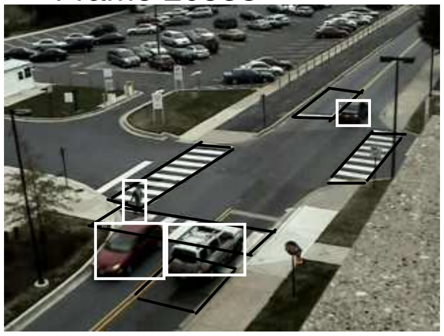
The system has been tested on a 14-minute video stream captured from a fixed camera monitoring a traffic intersection. The system is used to model and detect all types of normal behaviors and violations described in Section 3.2.1. Once a violation is detected, an alert is made by the system displaying the vehicle or the pedestrian committing the violation and the time when it happens.

Figure 3.4 shows 2 recognized instances of Right_of_Way event. In figure 3.4a, pedestrian 20 enters the crosswalk region in frame 344. Then vehicle 21 enters the stop sign marked region in frame 367, stops in frame 404 but does not leave until pedestrian 20 leaves the crosswalk region in frame 479 - normal behavior. In figure 3.4b, pedestrian 315 enters the crosswalk region in frame 20558. Then vehicle 316 enters the crosswalk in frame 20576. The vehicle does not give the right of way to the pedestrian and continues moving until it leaves the crosswalk in frame 20607 - a violation. Pedestrian 315 leaves the crosswalk in frame 20733.

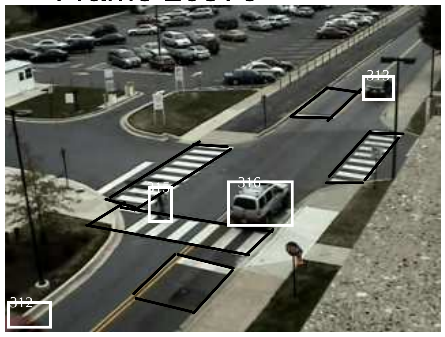
We compared the system results with ground truth for the 14-minutes video. The comparisons are shown in Table 3.4. From this table, we observe that more than 50% of the total number of vehicles do not obey the stop sign rule .The system was able to detect about 80% of these violations. Errors in object classifications and threshold selections explain the missed instances. Also, about 25% of the pedestrians do not use the crosswalks to cross the road. The system is able to detect all instances with a few false positives. Bicycles classified as pedestrians and dealing with a group of pedestrians as a single pedestrian explain these false positives.



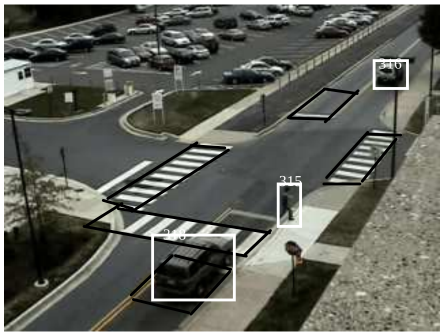
Frame 20558



Frame 20576



Frame 20607



Frame 20733

(b) Violation

Figure 3.4: Right of Way Events

Table 3.4: Traffic Intersection Event Detection Results

Event Name	True Positives	False Positives	False Negatives
Vehicles stop before stop sign	21	3	0
Vehicles don't stop before stop sign	22	5	0
Pedestrians use crosswalks	26	5	0
Pedestrians not using crosswalks	10	0	3
Vehicles not yielding to Pedestrians in crosswalks	2	0	1

3.3 Conclusion

We have applied the system to different natural videos to detect events with increasing level of complexity ranging from primitive events involving one object to more complex events involving multiple objects and multiple logical and temporal relations. In the first set of experiments, we applied our system on a dataset of video sequences whose length ranges from 800 frames to 3000 frames. In these sequences, the required events to be detected range from single primitives to simple sequences of two or three primitives. In the second set of experiments, we applied our system on a longer video sequence ($25200 frames$) to detect events and violations performed by pedestrians and vehicles in a traffic intersection. Events in this case are more complex and include multiple actors and multiple logical and temporal relations.

We have tested the system into 2 modes - offline mode, and online mode. For the first set of experiments, we have:

- The detection and tracking results are pre-computed.
- Event definitions and other contextual information are provided in XML format

to the system.

- The output of the event detection and recognition module has also been written to XML format. The video sequences are then annotated by these results.

For the second set of experiments (monitoring traffic intersection), we have:

- The detection and tracking are performed online by the system.
- Event definitions and other contextual information are provided through the GUI of the system.
- Whenever a violation is detected, an alert is displayed on the screen showing the vehicles and pedestrians involved in the violation.

.

We have also shown that the system performance depends heavily on lower level vision modules (e.g. detection and tracking). Any enhancement on these modules should also enhance the system performance. Independence of high level event detection and recognition modules from lower level vision modules makes applying these enhancements -once available- an easy task and does not require many changes in the code.

Chapter 4

Carried Object Detection

4.1 Introduction

One important problem in understanding human activities is to detect whether a person is carrying an object or not at different times. For example, if a person is carrying an object at time t1 and not carrying it at time t2, we can infer that the person has dropped the object or give it to another person between times t1 and t2. Another problem is detecting left packages in public places. Detection of left packages is among the goals of many visual surveillance systems of these places for security and safety concerns. In some cases, the left package can be detected by the background modeling component of the surveillance system [14, 15, 16, 17, 18]. On the other hand, if the package is left in an unseen place (e.g. behind a pillar or in a trash bin), then these methods will fail to detect it. In this case, we can infer that a package is being left if we detect that a person is carrying a package at one time and not carrying it at a later time.

Examples where the second approach can be useful include the following. A person enters a room carrying an object, deposits the object and then after a short time he exits wearing the same clothes but without the object. Another example is a person in a public place is observed by one or more cameras, then he disappears for a short period of time (e.g., behind a pillar) where he drops or picks up an object and he reappears. Even in cases where this person is continuously observed, he may drop or pickup objects in places not easily observed by surveillance cameras (e.g., in a trash bin). In these examples, direct detection of the left object itself is not possible, but could be inferred by deciding whether the owner is carrying an object at a given time but not carrying it at a later time.

In this part of the thesis, we present two approaches to the left package detection problem. We assume we have different instances of the same person within different cameras and at different times, and that the time separation between different instances of the same person is small, so that he does not change clothes between these instances. Both approaches poses the problem as a classification problem. The first approach, direct classification of silhouettes, classifies the subject's silhouettes for each instance directly to determine whether he is carrying an object or not. The second approach, appearance change detection, determines whether there is a significant change in human appearance between two different instances or not that might be due to an object being carried at one time but not the other. If there is a significant change in the human appearance, additional analysis is conducted to decide whether the person has dropped an object or acquired one from the scene.

For both approaches, we trained SVM classifiers [19] on a laboratory database that contains examples of people seen with and without two common objects, namely backpacks and suitcases. We used a boosting technique, AdaBoost [20], to select the most discriminative features used by the SVMs and to enhance the performance of the classifiers. We give recognition results for each approach and then compare both approaches and describe the advantages of each one.

The rest of the chapter is organized as follows. Section 4.2 provides background information about some theoretical methods used in this part of the thesis, namely Support Vector Machines, AdaBoost and integral images. We describe the database used in training the classifiers in section 4.3. The preprocessing step used by both approaches is discussed in section 4.4. We describe the first approach, direct classification of silhouettes, in section 4.5. The second approach, human appreence change detection, is discussed in section 4.6. Section 4.7 compares the results of both approaches.

4.2 Background

4.2.1 Support Vector Machines

SVMs were originally introduced by Vapnik and co-workers [19] and successfully extended by a number of other researchers. SVMs belong to the class of maximum margin classifiers. They perform pattern recognition between two classes by finding a decision surface that has maximum distance to the closest points in the training set which are termed support vectors [48]. We start with a training set of points $x_i \in IR^n$, $i = 1, 2, ...; N$ where each point x_i belongs to one of two classes identified by the label $y_i \in \{-1, 1\}$. Assuming linearly separable data, the goal of maximum margin classification is to separate the two classes by a hyperplane such that the distance to the support vectors is maximized. This hyperplane is called the optimal separating hyperplane (OSH). The OSH has the form:

$$f(x) = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i.\mathbf{x} + b, \tag{4.1}$$

The coefficients α_i and the b in Eq. (4.1) are the solutions of a quadratic programming problem. Classification of a new data point **x** is performed by computing the sign of the right side of Eq. (4.1). In the following we consider the equation

$$d(x) = \frac{\sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i.\mathbf{x} + b}{|| \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i ||} \tag{4.2}$$

where the sign of d is the classification result for x, and $|d|$ is the distance from x to the hyperplane. Intuitively, the farther away a point is from the decision surface, i.e. the larger $|d|$, the more reliable the classification result.

The entire construction can be extended to the case of nonlinear separating surfaces. Each point x in the input space is mapped to a point $z = \Phi(x)$ of a higher dimensional space, called the feature space, where the data are separated by a hyperplane. The key property in this construction is that the mapping $\Phi(.)$ is subject to the condition that the dot product of two points in the feature space $\Phi(x).\Phi(y)$ can be rewritten as a kernel function $K(x, y)$. The decision surface has the equation:

$$f(x) = \sum_{i=1}^{\ell} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b, \tag{4.3}$$

again, the coefficients α_i and b are the solutions of a quadratic programming problem. Note that $f(x)$ does not depend on the dimensionality of the feature space.

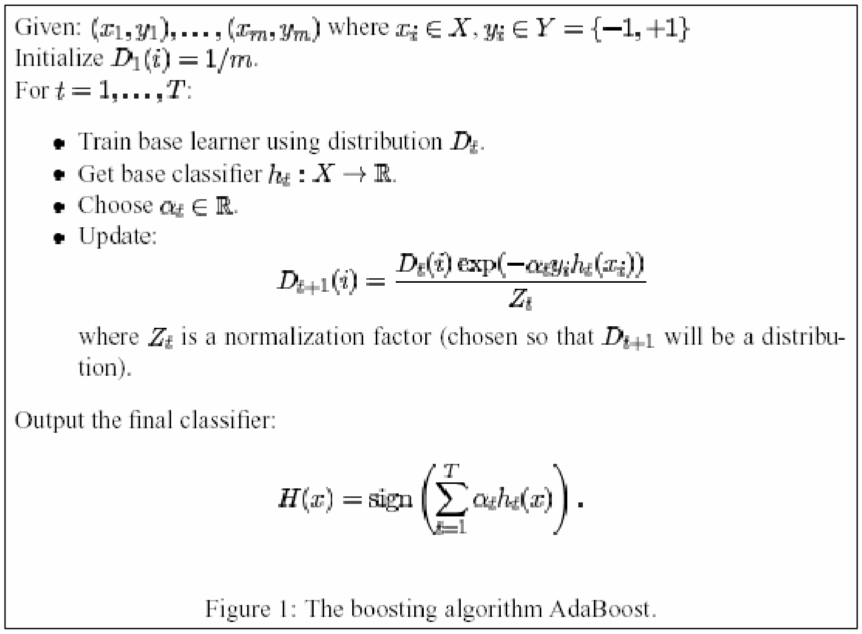
An important family of kernel functions is the polynomial kernel:

$$K(x, y) = (1 + x.y)^d, \tag{4.4}$$

where d is the degree of the polynomial. In this case, the components of the mapping $\Phi(x)$ are all the possible monomials of input components up to degree d.

4.2.2 Feature selection by AdaBoost

The AdaBoost classifier can be used to boost the classification performance of a simple learning algorithm (e.g. a simple perceptron) [20]. It does this by combining a collection of weak classifiers to form a stronger classifier. AdaBoost calls a weak classifier



```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:
    • Train base learner using distribution $D_t$.
    • Get base classifier $h_t : X \to \mathbb{R}$.
    • Choose $\alpha_t \in \mathbb{R}$.
    • Update:
          $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
      where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).
Output the final classifier:
    $H(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
```

Figure 1: The boosting algorithm AdaBoost.

Figure 4.1: The AdaBoost algorithm for the binary classification task

repeatedly in a series of rounds $t = 1, 2, ..., T$. For each call, a distribution of weights D_t is updated that indicates the importance of examples in the data set for the classification. On each round, the weights of each incorrectly classified examples are increased, so that the new classifier focuses more on those examples. The algorithm for the binary classification task is shown in figure 4.1 [20].

On the other hand, AdaBoost can also be used as a feature selection technique [49]. In this process, each feature is treated as a weak classifier. As a result each stage of the boosting process, which selects a new weak classifier, can be viewed as a feature selection process. On each round, AdaBoost chooses the feature with the best classification performance for the current boosting distribution. The weighting distribution for the training examples is updated to reflect how well every example was classified. The final strong classifier is a weighted combination of weak classifiers. The AdaBoost algorithm

```
• Given example images $(x_1,y_1)$ , … , $(x_n,y_n)$ where $y_i = 0, 1$ for negative and positive
  examples respectively.
• Initialize weights $w_{1,i} = 1/(2m), 1/(2l)$ for training example $i$, where $m$ and $l$ are the
  number of negatives and positives respectively.
For $t = 1 \dots T$
        1) Normalize weights so that $w_t$ is a distribution
        2) For each feature $j$ train a classifier $h_j$ and evaluate its error $\varepsilon_j$ with respect to $w_t$.
        3) Chose the classifier $h_j$ with lowest error.
        4) Update weights according to:
              $w_{t+1,i} = w_{t,i} \beta_t^{1-\varepsilon_i}$
              where $e_i = 0$ is $x_i$ is classified correctly, 1 otherwise, and
              $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
• The final strong classifier is:
      $h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \ge \frac{1}{2} \sum_{t=1}^{T} \alpha_t , \\ 0 & otherwise \end{cases}$        where        $\alpha_t = \log(\frac{1}{\beta_t})$
```

Figure 4.2: A variant of AdaBoost for aggressive feature selection

adapted for feature selection process is shown in figure 4.2 [49].

4.2.3 Integral Images

Rectangle features can be computed very rapidly using an intermediate representation for the image; the integral image [49]. The integral image at location x; y contains the sum of the pixels above and to the left of x; y, inclusive:

$$ii(x; y) = \sum_{x' \le x, y' \le y} i(x'; y'); \tag{4.5}$$

where $ii(x; y)$ is the integral image and $i(x; y)$ is the original image (see Figure 4.3.a). Using the following pair of recurrences:

$$s(x; y) = s(x; y - 1) + i(x; y) \tag{4.6}$$

$$ii(x;y) = ii(x-1;y) + s(x;y) \tag{4.7}$$

(where $s(x;y)$ is the cumulative row sum, $s(x;-1) = 0$, and $ii(-1;y) = 0$) the integral image can be computed in one pass over the original image. Using the integral image any rectangular sum can be computed in four array references, e.g. the sum of the pixels within rectangle D in Figure 4.3.b can be computed as: $ii(4) + ii(1) - ii(2) - ii(3)$. Clearly the difference between two rectangular sums can be computed in eight references. Since the two rectangle features defined above involve adjacent rectangular sums they can be computed in six array references, eight in the case of the three-rectangle features, and nine for four-rectangle features.

4.3 Database

Our database contains 180 training examples recorded in the keck lab [50] using 25 subjects, 2 bag types (backpack and suitcase). These examples are generated from sequences recorded using two cameras. We divided the dataset into into two subsets, the backpack dataset that contains 100 training examples and the suitcase dataset that contains 80 training examples. The combined dataset is the union of these two sets. We assume the default walking direction is fronto-parallel from right to left with possible small variations in the view angles. We have no restriction about how the subjects are carrying the bags. Figure 4.4 shows examples from the backpack dataset where subjects are carrying the backpack either on both shoulders or on only one shoulder. Figure 4.5 shows examples from the suitcase dataset where subjects are holding their bags with the hand facing the camera or the other hand. Both figures also show slightly different view angles.

We performed background subtraction for all data using the codebook BGS method

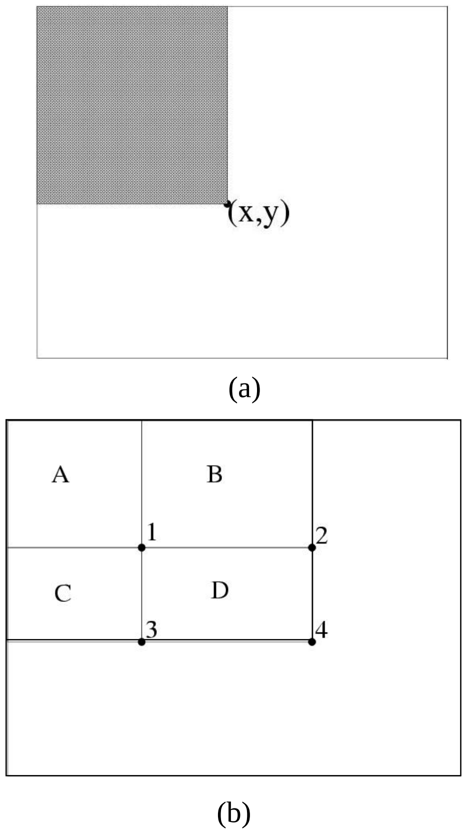
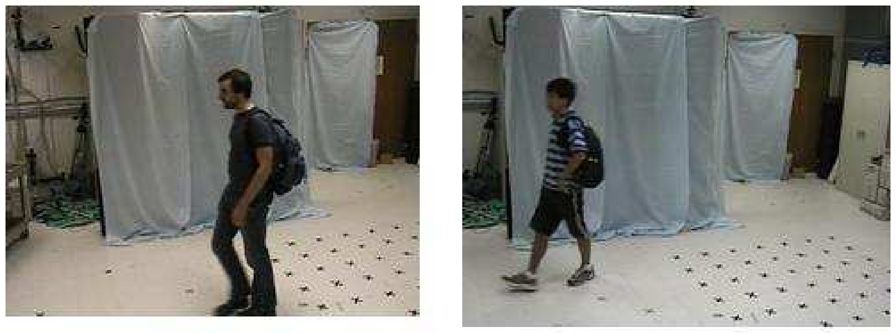
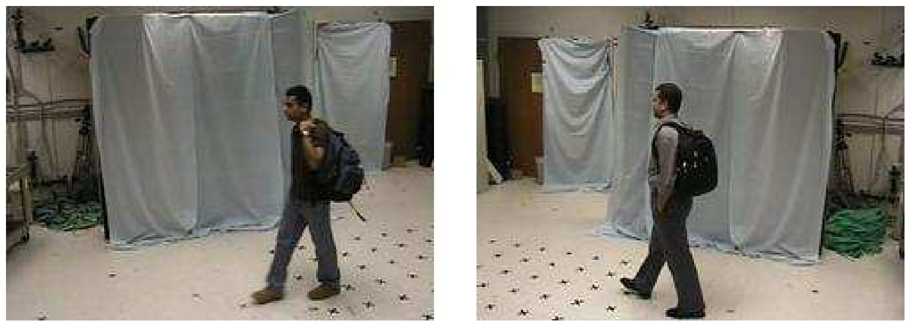


Figure 4.3: (a) The integral image at location (x; y) contains the sum of the pixels above and to the left of (x; y), inclusive. (b) The sum of the pixels within rectangle D is computed as: $ii(4) + ii(1) - ii(2) - ii(3)$


(a) Both Shoulders


(b) Left Shoulder

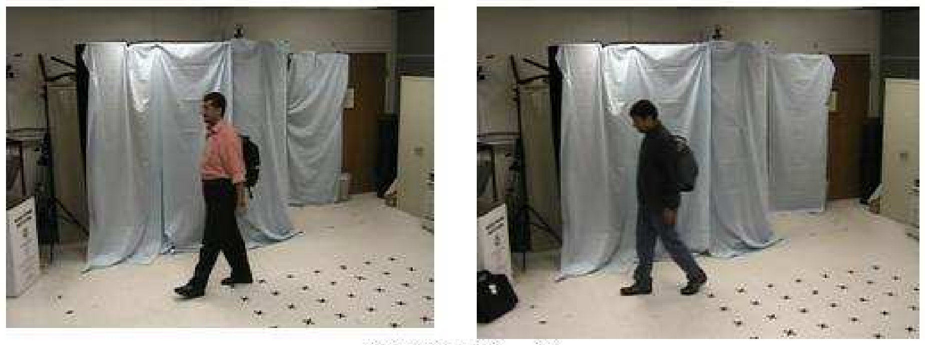
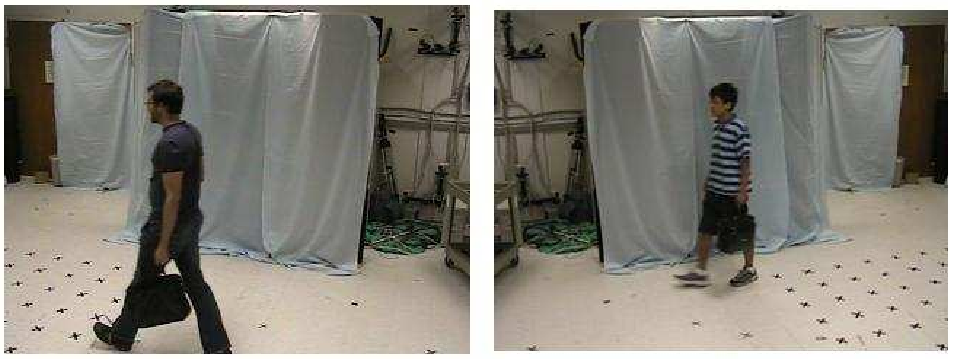
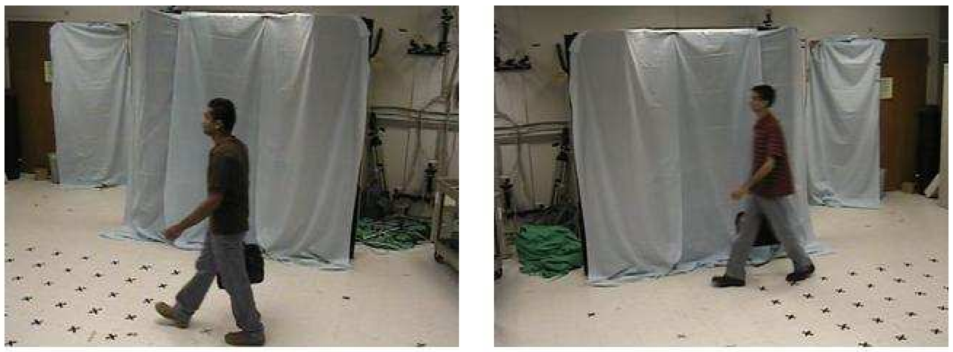

(c) Right Shoulder

Figure 4.4: Examples from Backpack Dataset Showing Different Ways of Holding the Backpack



(a) Left Hand



(b) Right Hand

Figure 4.5: Examples from Suitcase Dataset Showing Different Ways of Holding the Suitcase

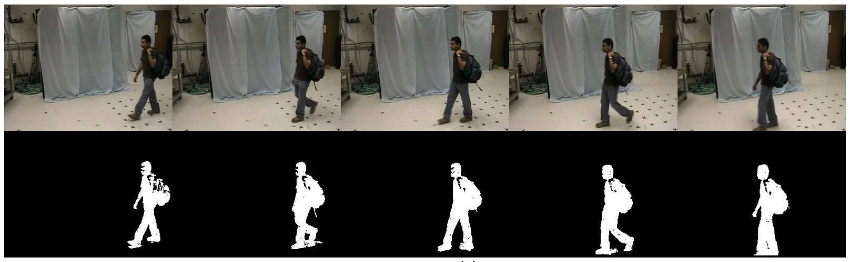
described in [51]. The frames in each sequence represent an integral number of walking cycles (one or two walking cycle depending on the available data). We applied the preprocessing described in section 4.4 to generate templates of size 64×48 pixels.

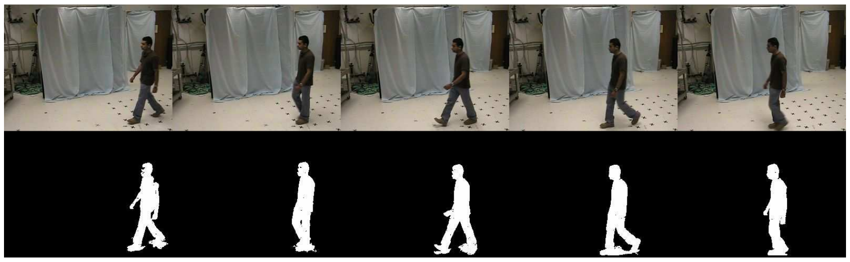
4.4 Preprocessing

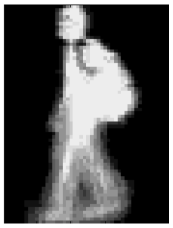
The purpose of the preprocessing step is to construct a template for each frame sequence that captures its features in both space and time. In this step, for each sequence, we use the codebook background subtraction method [51] to extract the silhouettes of the person in all frames. To construct a template of a predefined size $H \times W$, we resize all the silhouettes to that size, align them to the silhouettes' major axis and then superimpose these resized and aligned silhouettes to obtain the template. To resize a silhouette whose bounding box size is $h \times w$ pixels to the predefined template size of $H \times W$ pixels, we rescale the height to H pixels and maintaining the aspect ratio $h : w$ we calculate the scaling factor for the width. For each resized silhouette of height H and width $(H/h) \times w$, we calculate a major axis. We take the vertical line that passes through the median of the silhouette pixels as this major axis. Figure 4.6.a and 4.6.b show the background subtraction results for two sequences;the first sequence is for a person carrying a backpack and the second sequence is for the same person without the backpack. Figures 4.6.c and 4.6.d show the constructed templates for these sequences.

4.5 First Approach: Direct Classification of Silhouettes

In this approach, we classify the silhouettes of a given person's instance into two classes: person is carrying an object and person is not carrying an object. An instance of a person is represented by a set of frames captured at a given time by a given camera. We apply this approach in the framework of the event modeling and recognition system. The GUI of the system is used to mark regions in the scene, and the system is tasked to detect whether people accessing those regions are carrying an object. A person entering this area is tracked until he exits the area. We approximately align the detected silhouettes of the person and then generate features describing the aligned silhouette, as described in Section 4.6. The features in this case are an occupancy count map, ones' maximum run length map and zeros' maximum run length map. From the generated maps, we create the feature pool to be used by the classifiers.


(a)


(b)


(c)

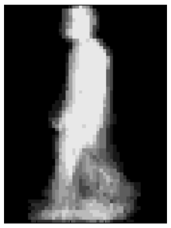

(d)

Figure 4.6: (a) Some frames of a sequence of person carrying a backpack with BGS results (b) Some frames of a sequence of person not carrying a backpack with BGS results (c) Template for sequence a (d) Template for sequence b

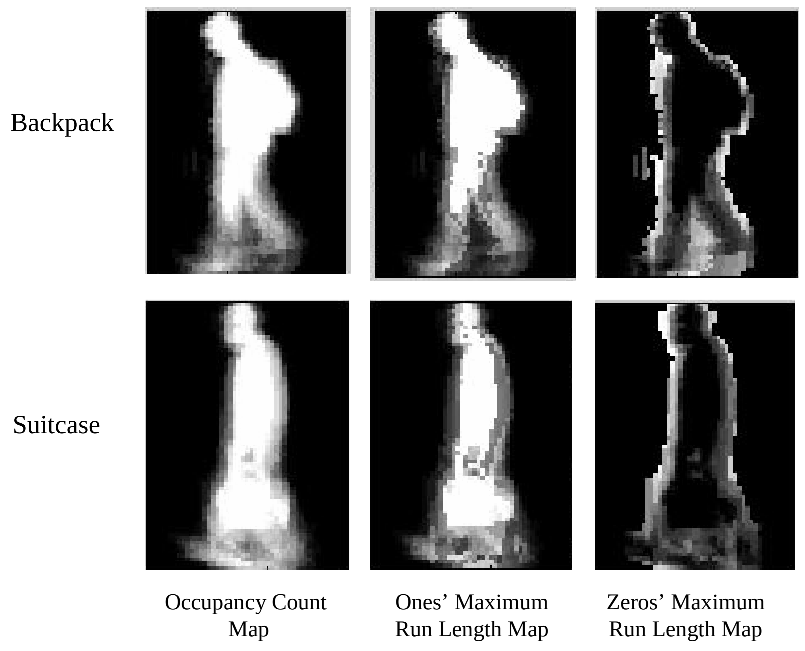


Figure 4.7: Feature Maps for First Approach

4.5.1 Features for Classifiers

Figure 4.7 shows the three feature maps used in this approach.

Occupancy Count Map

The first feature map is the occupancy count map, where the value of aligned pixel (x,y) represents the ratio between the number of frames where this pixel is foreground to the total number of frames.

$$OC_s(x,y) = \frac{\sum_i FG_i(x,y)}{N_s} \tag{4.8}$$

where s denotes the template name, N_s is the total number of frames in template s and

$$FG_i(x,y) = \begin{cases} 1;\text{ if } (x,y)\text{is foreground in frame i} \\ 0;\text{otherwise} \end{cases} \tag{4.9}$$

Ones' Maximum Run Length Map

For each pixel in the aligned silhouette, we compute the maximum run length of ones.

Zeros' Maximum Run Length Map

For each pixel in the aligned silhouette, we compute the maximum run length of zeros.

4.5.2 The Feature Pool

We create three feature maps for every instance of the person. Instead of using the values of these maps directly by the classifier, we divide each map into overlapping blocks of different sizes and different aspect ratios. The features for each block are the averages over these blocks. To speed up the computation of these rectangle features, we use the integral image representation proposed by [49] and described in section 4.2.3.

For each of the three feature maps, we compute the corresponding integral image. We use these integral images to compute features for different blocks.

To generate the feature vector for each training example, assuming template size 48x64, we used blocks of sizes ranging from 8×8 to 32×32, aspect ratios of $1 : 1$, $1 : 2$ and $2 : 1$ and step sizes of 4 and 8. This results in 915 blocks for each map.

4.5.3 Experiments

In this section, we have three test configurations: the backpack test, the suitcase test, and the combined backpack-suitcase test. We start by evaluating the performance of the support vector machine classifier on the carried object detection on each test configuration using two different sets of features:

- Features from the occupancy count map only; we call the classifier in this case **OC-classifier**.
- Features from the three feature maps; we call the classifier in this case **Extended-OC-classifier**..

Following that, we evaluate the performance of each classifier by performing feature selection through AdaBoost followed by the support vector machine classifier. We also discuss the best features selected by the AdaBoost feature selector.

We used 5-fold cross-validation to estimate the generalization error of the classifier [52]. In a k-fold cross-validation, the data set is divided into k subsets of (approximately) equal size. The classifier is trained k times, each time leaving out one of the subsets from training, but using only the omitted subset to compute the error criterion [52].

Support Vector Machine Classifiers

The results for the OC-Classifier and the Extended-OC-Classifier using support vector machines only are summarized in Table 4.5.3.

As shown from this table, the performance of both classifiers is significantly better for the suitcase case compared to the backpack case. This can be explained as follows. For the backpack case, the subject can hold the backpack in different ways, i.e. on one shoulder: near or away from the camera, or on both shoulders, the size of the backpack blob varies. This also occurs due to the slightly different view angle. For example, in Figure 4.8, the number of the pixels occupied by the backpack in Example 1 is much less than the case of Example 2. We can note that, for Example 1, it is difficult to distinguish between the person with or without the backpack.

	Backpack	Suitcase	Combined
OC-Classifier	82%	93%	88%
Extended-OC-Classifier	85%	90%	89%

Table 4.1: SVM Recognition Rates on Training Datasets for OC-Classifier and Extended-OC-Classifier.

The suitcase classification problem is the opposite of the backpack classification problem: Again, in the suitcase case, the subject can hold the suitcase in different ways, i.e. near or away from the camera. The shape of the suitcase can also change due to the variations of the view angle. Figure 4.9 shows two examples from the suitcase dataset with the occupancy count maps representing the person with and without the suitcase. The suitcase position is clear, regardless of the position of the suitcase relative to the camera or person.

On the other hand, there is a slight change in the performance of the OC-Classifier compared to the Extended-OC-Classifier. Using features from the ones' and zeros' maximum run length maps enhances the performance of the OC-classifier by 3% in the case of the backpack and worsens the performance of the OC-classifier by 3% in the case of the suitcase.

Feature Selection Classifiers

Table 4.5.3 summarize the results for selecting the best features using an AdaBoost classifier and applying a support vector machine classifier on the selected features. The table shows that the feature selection classifier significantly enhances the performance over the plain SVM classifier. The enhancement can be up to 9% as in the backpack test configuration.

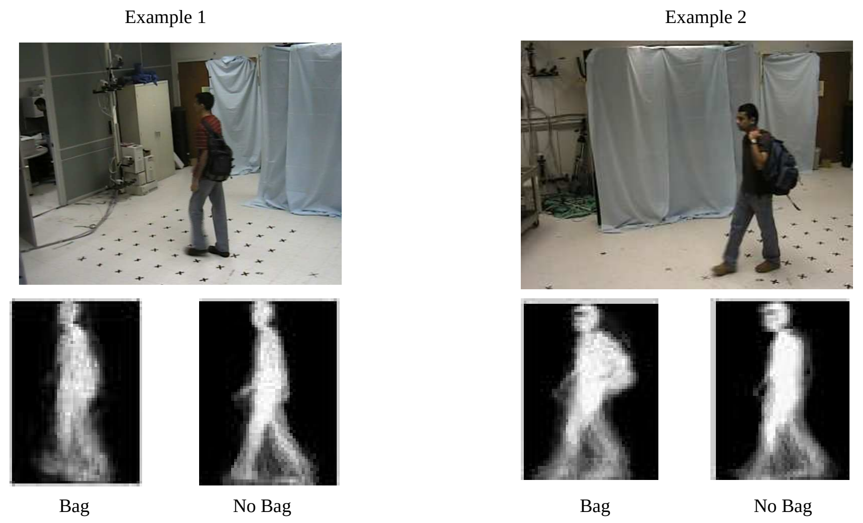


Figure 4.8: Examples from the backpack dataset with the occupancy maps

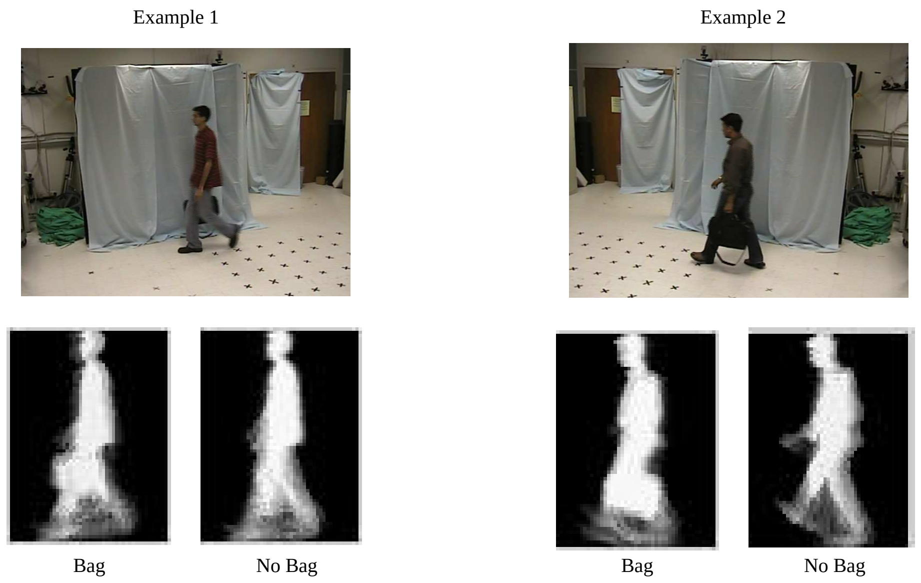


Figure 4.9: Examples from the suitcase dataset with the occupancy maps

	Backpack	Suitcase	Combined
OC-Classifier	90%	97%	93%
Extended-OC-Classifier	96%	93%	96%

Table 4.2: AdaBoost&SVM Recognition Rates on Training Datasets for OC-Classifier and Extended-OC-Classifier.

	Backpack	Suitcase	Combined
Number of features	58	1	128

Table 4.3: Number of Features Selected by AdaBoost for OC-Classifier.

Moreover, tables 4.5.3 and 4.5.3 show that the number of selected features is much less than the original number of features (915 for OC-Classifier and 2745 for Extended-OC-Classifier). The feature selection classifier reduces the number of features by more than an order of magnitude for the backpack and combined test configurations and selects only one feature for the suitcase test configuration corresponding to the location of the suitcase.

Figure 4.10 shows the location of the best selected features for the backpack and the suitcase test configurations for the OC-classifier. For the suitcase, the feature corresponding to the location of the suitcase is the best feature selected by AdaBoost.

4.6 Second Approach: Human Appearance Change Detection

We apply this approach in the framework of the event modeling and recognition system. The GUI of the system is used to mark critical regions in the scene (e.g. regions around

	`Backpack`	`Suitcase`	`Combined`
`OC`	27	1	44
`1`s run'	6	0	27
`0`s run'	10	0	43
Total Number	43	1	114

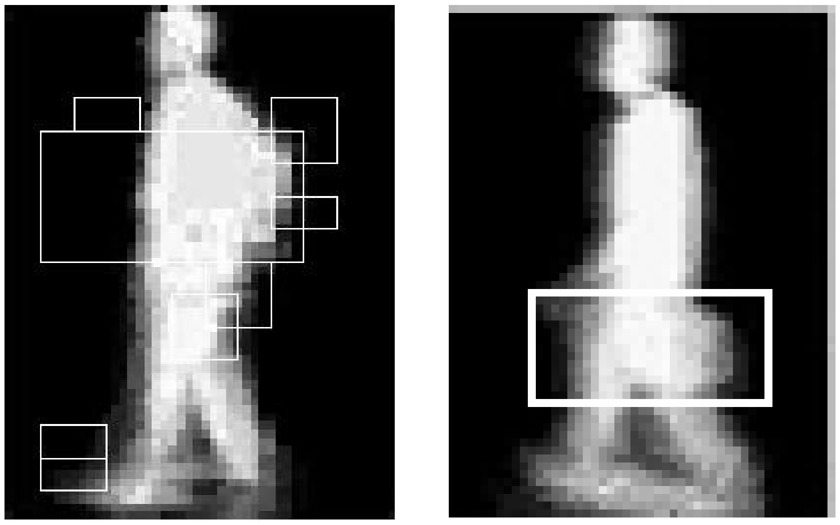


Figure 4.10: The OC-Classifier's best selected features for the (a) backpack test configuration (b) suitcase test configuration

trash bins, pillars or building entrances), and the system is tasked to detect significant appearance changes of people accessing those regions. To detect changes in a person's appearance, we track the person from the moment he enters the critical region and back in time to obtain a video sequence, which we will refer to as the 'Before' sequence. We also track the person from his exit from the critical region and forward to obtain a video sequence, which we will refer to as the 'After' sequence.

For each instance of the person, we approximately align the frames and then generate features that captures the shape and color information of the person's silhouette, as described in Section . The features in this case are an occupancy map and a color codebook (based on a vector quantization of the set of colors and frequencies) at each aligned pixel. To capture differences in shape and color between the 'Before' and 'After' sequences, we generate three maps, namely the occupancy difference map, the codeword frequency difference map and the histogram intersection map. Finally, from the generated maps, we create the feature pool to be used by the classifier.

4.6.1 Features for Classifiers

Occupancy Difference Map

To capture changes in the shape between the 'Before' and the 'After' sequence, we compute the occupancy difference map as the difference between the occupancy map representing the 'Before' sequence and the occupancy map representing the 'After' sequence.

$$OC_{Diff}(x,y) = OC_{Before}(x,y) - OC_{After}(x,y) \tag{4.10}$$

The top row of figure 4.11 shows the 'Before', the 'After' and the difference occupancy maps of a person carrying a backpack in the 'Before' sequence and not in the

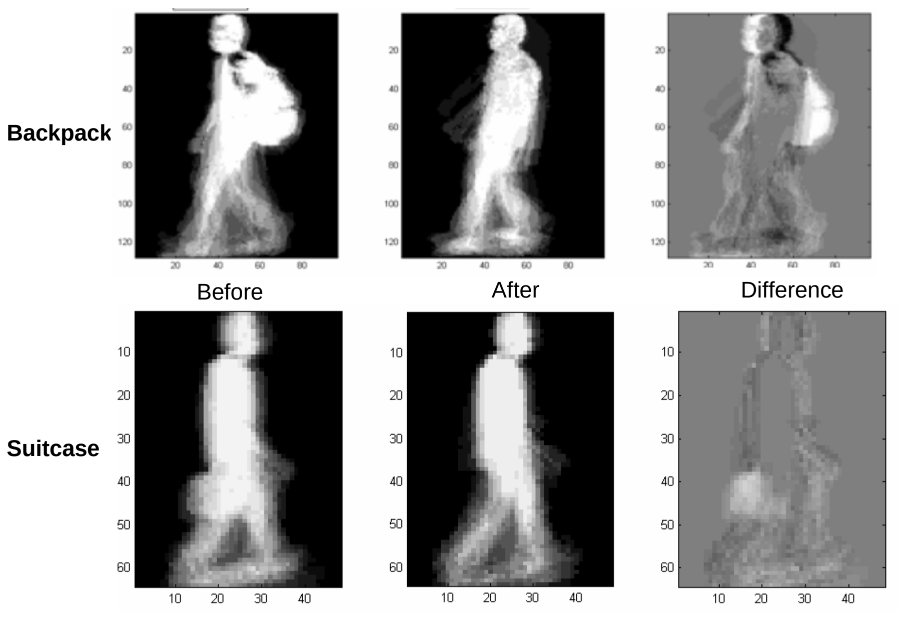


Figure 4.11: ‘Before’, ‘After’ and difference occupancy maps for (a) a backpack example ,(b) a suitcase example

‘After’ sequence. The bottom row of figure 4.11 shows the same for the suitcase luggage type. As can be seen in these two examples, dropping the bag leads to large change in the person’s appearance that appears as a white (or black) blob in the difference map.

Codeword Frequency Difference Map

For each aligned pixel (x, y), we compute a codeword frequency - the average number of codewords per occurrence.

$$CC_s(x, y) = \frac{\#codewords_s(x, y)}{OC_s(x, y)} \tag{4.11}$$

where s denotes the template name, $\#codewords_s$ is the number of codewords in the codebook of pixel (x, y) and $OC_s(x, y)$ is the occupancy map value at pixel (x, y).

One way to capture changes in the color between the ‘Before’ and the ‘After’ se-

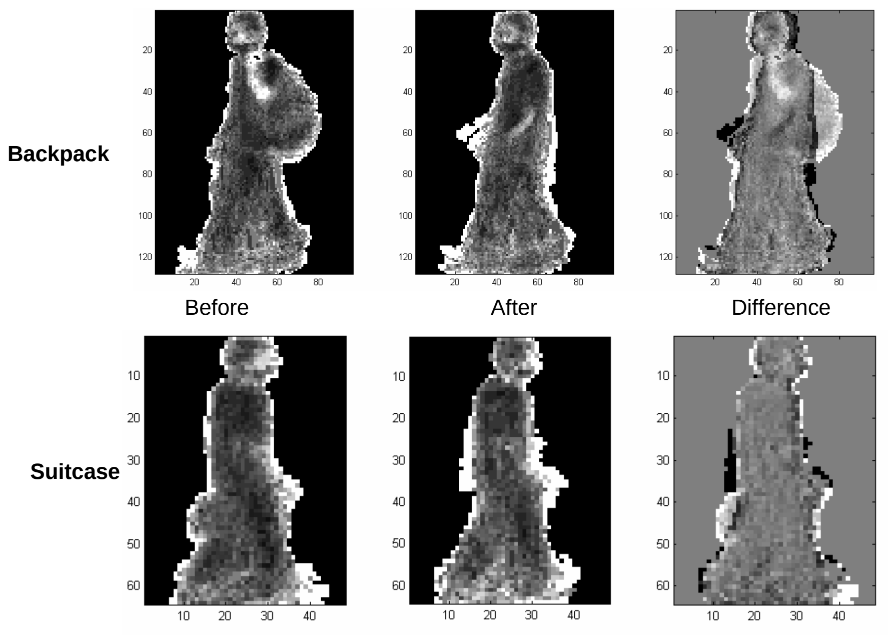


Figure 4.12: ('Before', 'After' and difference codeword frequency maps for (a) a back-pack example ,(b) a suitcase example

quence is to compute the codeword frequency difference map as the difference between the codeword frequency map representing the 'Before' sequence and the codeword frequency map representing the 'After' sequence.

$$CC_{Diff}(x,y) = CC_{Before}(x,y) - CC_{After}(x,y) \tag{4.12}$$

The top row of figure 4.13 shows the 'Before', the 'After' and the difference codeword frequency map of a person carrying a backpack in the 'Before' sequence and not in the 'After' sequence. The bottom row of figure 4.13 shows the same for the suitcase luggage type. As can be seen in these two examples, dropping the bag leads to a large change in the person's appearance that appears in the difference map.

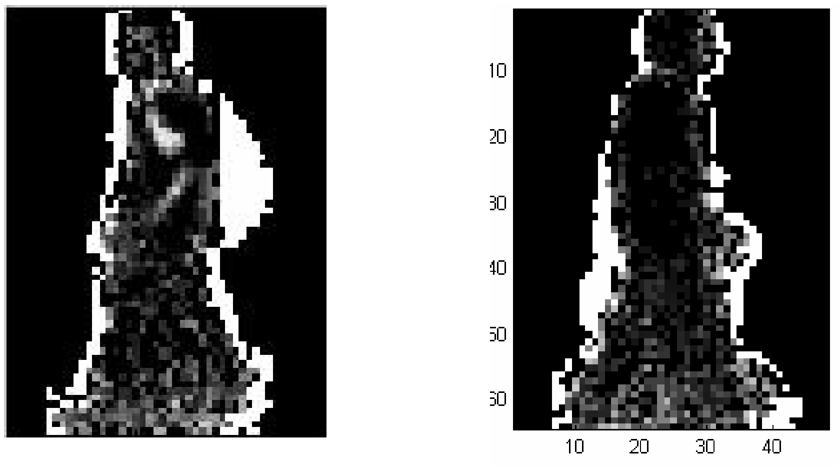


Figure 4.13: Histogram Intersection for backpack and suitcase examples

Histogram Intersection Map

Another way to represent the color changes between two templates is to measure the similarity between the codebooks representing corresponding pixels in the two templates. We use the color histogram intersection as a measure of similarity between these two codebooks. The color histogram intersection was proposed for color image retrieval in [53]. The intersection of histograms h and g is given by:

$$d(h,g) = \frac{\sum_A \sum_B \sum_C \min(h(a,b,c), g(a,b,c))}{\min(|h|, |g|)} \tag{4.13}$$

where $\mid h \mid$ and $\mid g \mid$ gives the magnitude of each histogram, which is equal to the occurrences of this pixel.

4.6.2 The Feature Pool

We have created three maps that describe the differences between two different image sequences of the same person. Instead of using the values of these maps directly by the classifier, we divide the image into overlapping blocks of different sizes and different aspect ratios. The features for each block are the averages over these blocks. To speed up the computation of these rectangle features, we use the integral image representation proposed by [49] and described in section 4.2.3.

For each of the three feature maps, we compute the corresponding integral image. We use these integral images to compute features for different blocks.

To generate the feature vector for each training example, assuming template size 48x64, we used blocks of sizes ranging from 8×8 to 32×32, aspect ratios of $1:1$, $1:2$ and $2:1$ and step sizes of 4 and 8. This results in 915 blocks. The features for each block are the averages taken over this block in the three feature maps computed using the integral images. The feature vector of each training example is the concatenation of the features of all blocks. This results in a feature vector of length 2745.

4.6.3 Experiments

In this section, we have three test configurations: the backpack test, the suitcase test, and the combined backpack-suitcase test. We start by evaluating the performance of the support vector machine classifier on the human appearance change detection on each test configuration. Following that, we evaluate the performance of the classifier that performs feature selection through AdaBoost followed by the support vector machine classifier. We also discuss the best features selected by the AdaBoost feature selector.

We used 5-fold cross-validation to estimate the generalization error of the classifier [52]. In a k-fold cross-validation, the data set is divided into k subsets of (approximately) equal size. The classifier is trained k times, each time leaving out one of the subsets from training, but using only the omitted subset to compute the error criterion [52].

Support Vector Machine Classifier

The results for the support vector machine classifier are summarized in Table 4.6.3. As shown from the table, the performance of the SVM classifier is significantly better for the backpack case compared to the suitcase case. This is due to a number of factors: (1) the backpack is always visible in all frames from at least one side-view camera. This is not true for the suitcase where the suitcase may be occluded behind the person in all frames in the side-view. (2) The location of the backpack behind the person's back makes its blob easily detectable from the person's body blob. For the suitcase, its location overlaps with the person's body and thus is harder to distinguish.

Feature Selection Classifier

Table 4.6.3 summarizes the results for selecting the best features using an AdaBoost classifier and applying a support vector machine classifier on the selected features. The table shows that the feature selection classifier significantly enhances the performance over the plain SVM classifier. The enhancement can be up to 10% as in the suitcase test configuration. Moreover, the table shows that the number of selected features is much less than the original number of features (2745). The feature selection classifier reduces the number of features by more than an order of magnitude for the suitcase and combined test configurations and by more than two order of magnitudes for the backpack test configuration. The next subsection discusses the selected features for the different test configurations.

Best Features for Human Appearance Change Detection

Table 4.6.3 summarizes the distribution of the best selected features among the three main features: OC_Diff, CC_Diff, and Histogram Intersection. For the backpack test

	Backpack	Suitcase	Combined
SVM	92%	78%	88%
AdaBoost and SVM	96%	88%	90%

Table 4.5: Recognition Rates on Training Datasets.

	Backpack	Suitcase	Combined
OC_Diff	0	9	28
CC_Diff	0	8	22
Histogram Intersection	9	26	50
Total Number	9	43	100

Table 4.6: Number of Features Selected by AdaBoost.

configuration, all of the nine best features belong to the Histogram Intersection. We believe that this is because the backpack case is easily classified due to the reasons described in the previous section. As the classification problem becomes harder, as in the suitcase and combined test configurations, the features that depend on the OC_Diff and CC_Diff features becomes more important. Moreover, the number of selected features increases.

Figure 4.14 shows the location of the best selected features for the backpack and the suitcase test configurations. It is interesting to see that for the backpack case, one of the best features is related to the person's head location. A person carrying a backpack will change his head position to accommodate the weight of the backpack. As expected, the features corresponding to the location of the backpack and the suitcase are among the best features.

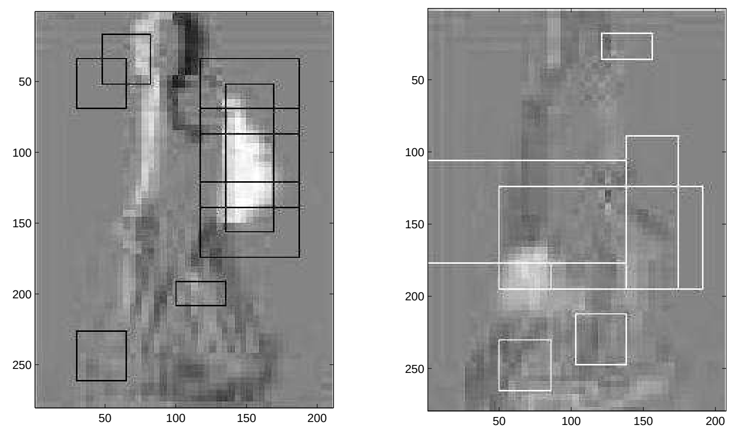


Figure 4.14: (a) Best selected features for the backpack test configuration (b) Best selected features for the suitcase test configuration

4.6.4 Effect of Changes in Camera Viewpoints on Performance

In order to evaluate the performance of the change detection method across large changes in camera viewpoint, we used the Keck multi-perspective lab [50] to capture sequences of walking people from multiple cameras at the same time. Assuming the horizontal direction going from right to left is the zero direction, Figure 4.15.a shows a subject carrying a backpack captured from angles $0, 15, 30, 45, -15, -30$. Figure 4.15.b shows a subject carrying a suitcase captured from angles $0, 15, 30, 45, -15, -30, -60$. Using 'Before' and 'After' sequences captured from the same view, we tested our classifier trained on the dataset described in section 4.5.1, to detect a package drop/pickup in each of the above directions.

The package drop/pickup was detected in the following directions: $0, 15, 30, -15$ for the backpack, $0, 15, 30, -15$ for the suitcase, but not for the remaining directions.

An intuitive explanation for this is the following. Most of the cases used in training the classifiers are for near-fronto-parallel views. So, for a small change in the view angle (between -15 and 30), the location of some features used by the classifiers does not change and hence the results are good. On the other hand, big changes in the view angles results in changes in the location of the backpack or suitcase with respect to the subject's silhouette and hence not detected by the classifiers.

Adding the view angle as a new feature to the classifiers, and training on a larger dataset containing different views of the subjects would make the approach scalable to handle larger changes in the view angle of the subject.

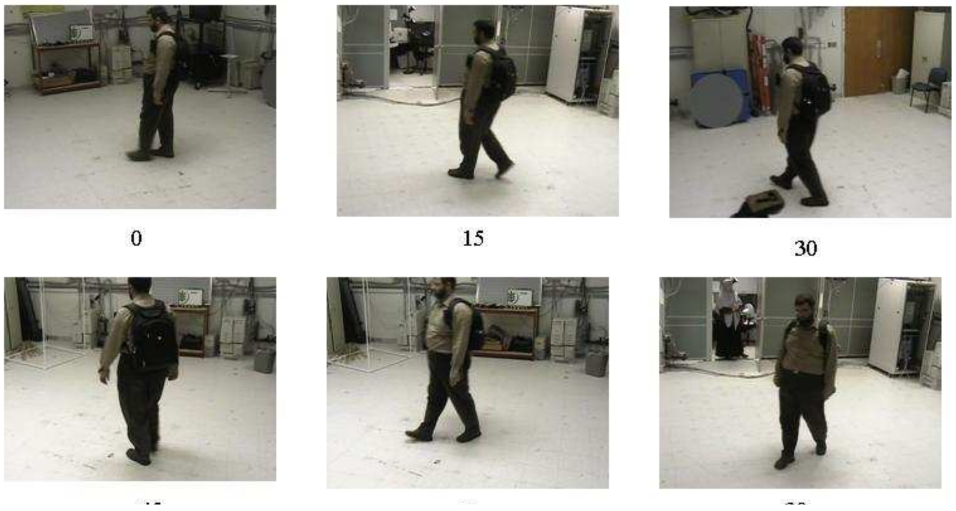
4.7 Discussion

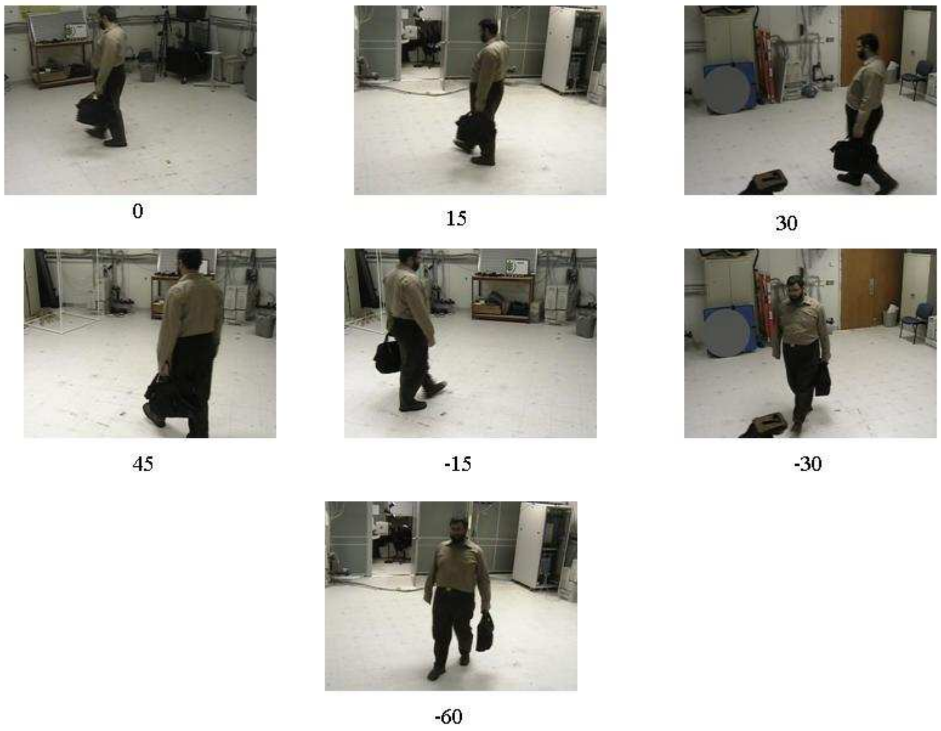
In this chapter, we have presented two approaches to the problem of carried object detection.

Comparing the results for both approaches, we can see that the same performance is obtained for Extended-OC-classifier and the change detection classifier- both of them outperform the OC-classifier for the backpack case by 6%. On the other hand, the OC-classifier is outperforming the Extended-OC-classifier by 4% and the change detection classifier by 9% for the suitcase bag type.

4.7.1 Backpack classification

Since the subject can hold the backpack in different ways, i.e. on one shoulder: near or away from the camera, or on both shoulders, the size of the backpack blob varies. This also occurs due to the slightly different view angle. For example, in Figure 4.16, the number of the pixels occupied by the backpack in Example 1 is much less than the case





(b) Suitcase

Figure 4.15: Different viewpoints of a subject carrying a backpack and a suitcase

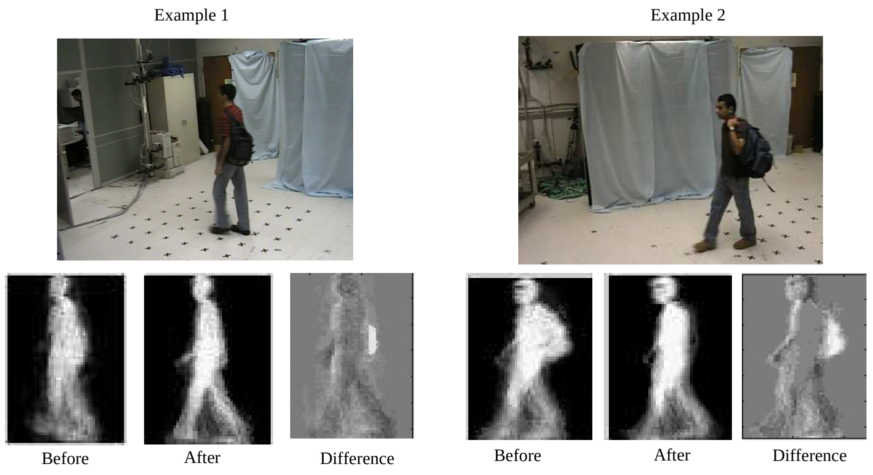
of Example 2.

We can note that, for Example 1, it is difficult to distinguish between the person with or without the backpack, corresponding to the Before and After sequences respectively. The backpack presence is more visible using the difference features. In other words, using the difference helps reduce the noise introduced by the person's body. This makes the job of the change detection classifier easier than the OC-classifier. In addition, the change detection classifier uses more features in the classification process, i.e. histogram intersection and the codeword count map. This further explains the better performance of the change detection classifier compared to the OC-classifier. The change detection classifier uses 9 features, all from the histogram intersection map. Although the feature pool contains all the features used by the OC-classifier, none of them were selected by the AdaBoost as important features for the change detection classifier.

On the other hand, the Extended-OC-classifier can reach the same performance of the change detection classifier by using about 40% of its features from the ones' and zeros' maximum run length maps. We believe that the Extended-OC-classifier reaches this high recognition rate since for the laboratory data the background subtraction results are usually good and hence features depending on pixel statistics can be computed efficiently. If the background subtraction results are noisy, these pixel statistics will not be meaningful.

4.7.2 Suitcase classification

The case here is the opposite of the backpack classification problem: Again, in the suitcase case, the subject can hold the suitcase in different ways, i.e. near or away from the camera. The shape of the suitcase can also change due to the variations of the view angle. Figure 4.17 shows two examples from the suitcase dataset with the occupancy



The OC-Classifier and Extended-OC-Classifier fail to detect the change.

The change detection classifier has detected the change.

All classifiers can detect the change.

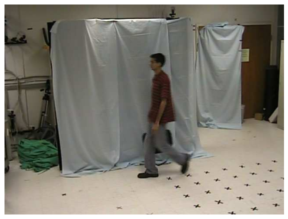
Figure 4.16: Examples from the backpack dataset

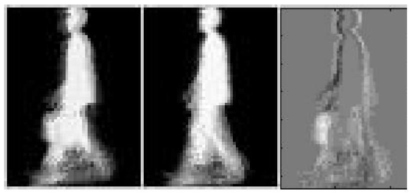
count maps representing the ’Before’ and the ’After’ sequences and the difference occupancy map.

The person’s legs usually interfere with a large number of pixels of the suitcase. This makes large parts of the suitcase treated as body parts in the case of the change detection classifier, as shown in Figure 4.17 in the Difference subfigures.

Since both the OC-classifier and the Extended-OC-classifier use the same selected feature from the occupancy count map and the OC-classifier outperforms the Extended-OC-classifier, we will compare only the OC-classifier to the change detection classifier. In the case of the OC-classifier, the suitcase position is clear, regardless of the position of the suitcase relative to the camera or person. The suitcase presence is more visible using the OC-classifier features. This makes the job of the OC-classifier easier than the change detection classifier. Actually, it use only one feature, which captures the position of the suitcase, to achieve its higher accuracy. On the other hand, the change detection classifier uses 43 features from the three feature maps, 9 of them is from the OC difference map. Even with the use of these 9 features, the noise introduced by the interference of the person’s legs cannot be handled by the change detection classifier.

Example 1



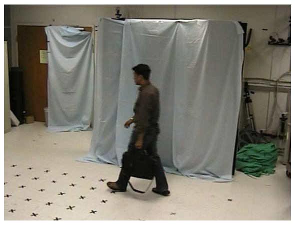


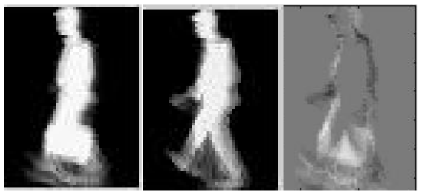
Before After Difference

OC-Classifier and Extended-OC-Classifier have detected the change.

The change detection classifier fails to detect the change.

Example 2





Before After Difference

All classifiers can detect the change.

Figure 4.17: Examples from the suitcase dataset

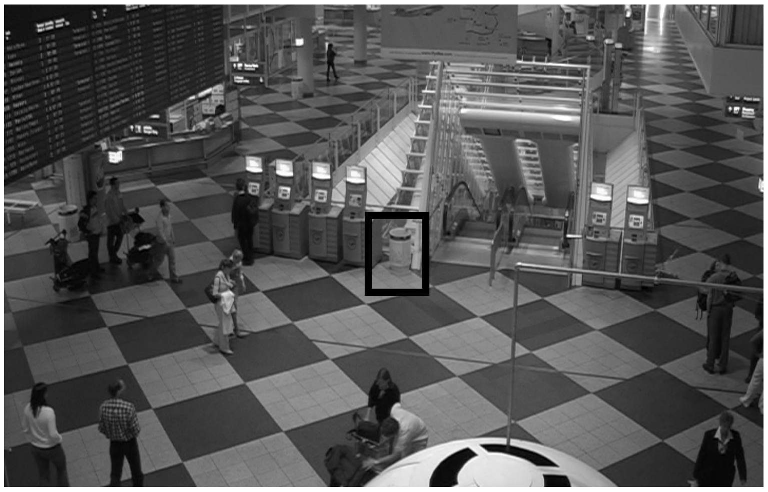
Chapter 5

Experiments II

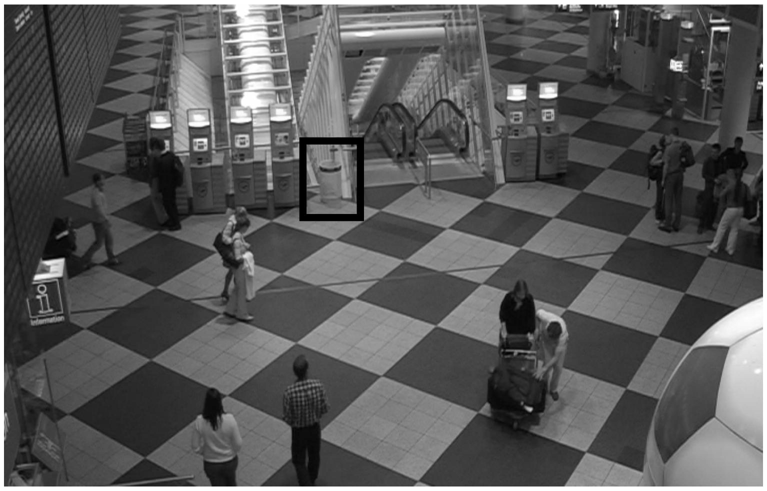
In this chapter, we test the carried object detection methods explained in Chapter 4 on videos captured at the Munich airport [1]. These videos were captured during the day where many people are moving around the scene and some scenarios are staged for event detection purposes. Figure 5.1 shows one scene taken by 2 cameras where we applied our analysis. The trash bin marked in the scene is used by subjects to drop their bags inside or behind, and then these bags are picked up by their owners or other persons.

We applied our analysis in the framework of the event modeling and detection system. Section 5.1 explains the Petri net event model for the left package detection problem. The alignment procedure used to solve some of the segmentation errors is discussed in section 5.3. We report results for the first approach in section 5.4 and the results for the second approach in section 5.5. We discuss these results and conclude in section 5.6.

[1]We thank Jan Neumann from Siemens for providing us with these videos



(a)



(b)

Figure 5.1: (a)First camera view. (b)Second camera view. Airport Scene Monitored by 2 Cameras

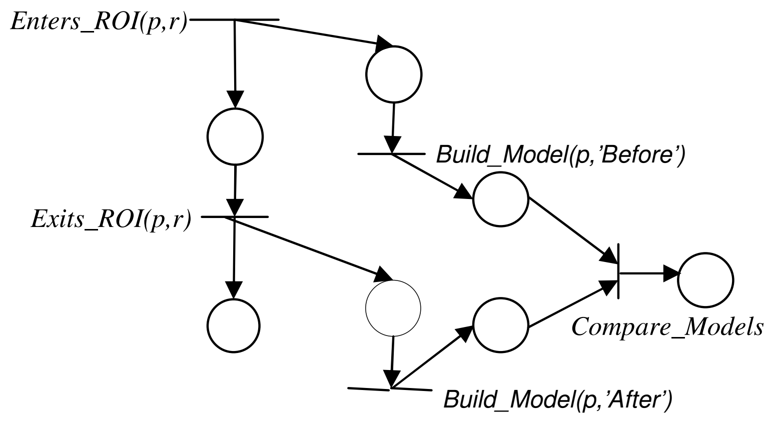


Figure 5.2: Petri Net for the Event 'Access_ROI'

5.1 Event Modeling

In our example application, we mark the region around the trash bin as a region of interest. Then we define the event template $Access\_ROI(p, r)$ from the two primitive events $P_1 = Enters\_ROI(p, r)$ and $P_2 = Leaves\_ROI(p, r)$ joined by the temporal relation $P_1 Before P_2$, where p and r are the person and the region variables that are bound at run time. Figure 5.2 shows the Petri net representation of the event $Access\_ROI(p, r)$. As can be seen in that figure, detecting the primitive $Enters\_ROI$ initiates the process of building a 'Before' template of the subject entering the ROI while detecting the primitive $Leaves\_ROI$ initiates the process of building an 'After' template of the subject leaving the ROI. Once both templates are built, we apply our methods (either classify each template independently by the direct classification methods or combine both templates and classify the resulting template by the change detection method) to detect whether the subject has abandoned an object in the ROI, acquired an existing one or left with no change.

5.2 An Example

We applied our carried object detection methods on many airport examples. We used the event modeling and detection system to detect people entering and leaving the marked ROI around the trash bin. Figure 5.3.a shows a frame from the first camera, where a person is detected entering the ROI and 5.3.b shows the same frame from the second camera. The person then drops the his backpack in the trash can and leaves the ROI. The tracking results are overlaid on these frames to show the person trajectories before and after accessing the ROI.

One problem is as figures 5.3.c and 5.3.d show, the segmentation results of the person before entering the ROI and after leaving contains many errors. As shown from these figures, in some frames the complete silhouette of the person is detected while in other frames some parts are not detected due to the complexity of the scene and the similarity between the clothes colors and the floor tiles colors. We apply an alignment procedure to align the detected parts in a fixed size template to reflect their actual position with respect to the whole body, as discussed in section 5.3.

5.3 Alignment Procedure

In the training data captured at the laboratory, the scene background was simply empty and only one subject is moving through the scene. This leads to good background subtraction results and simple tracking and hence good silhouettes can be obtained. In contrast to this simple setting, for the airport data, the background is more complex, there are no frames of the empty scene to initialize a background model, there are many people walking around the scene, and other conditions leading to bad segmentation and inaccurate tracking. Hence, the assumption that good silhouettes are available for ap-

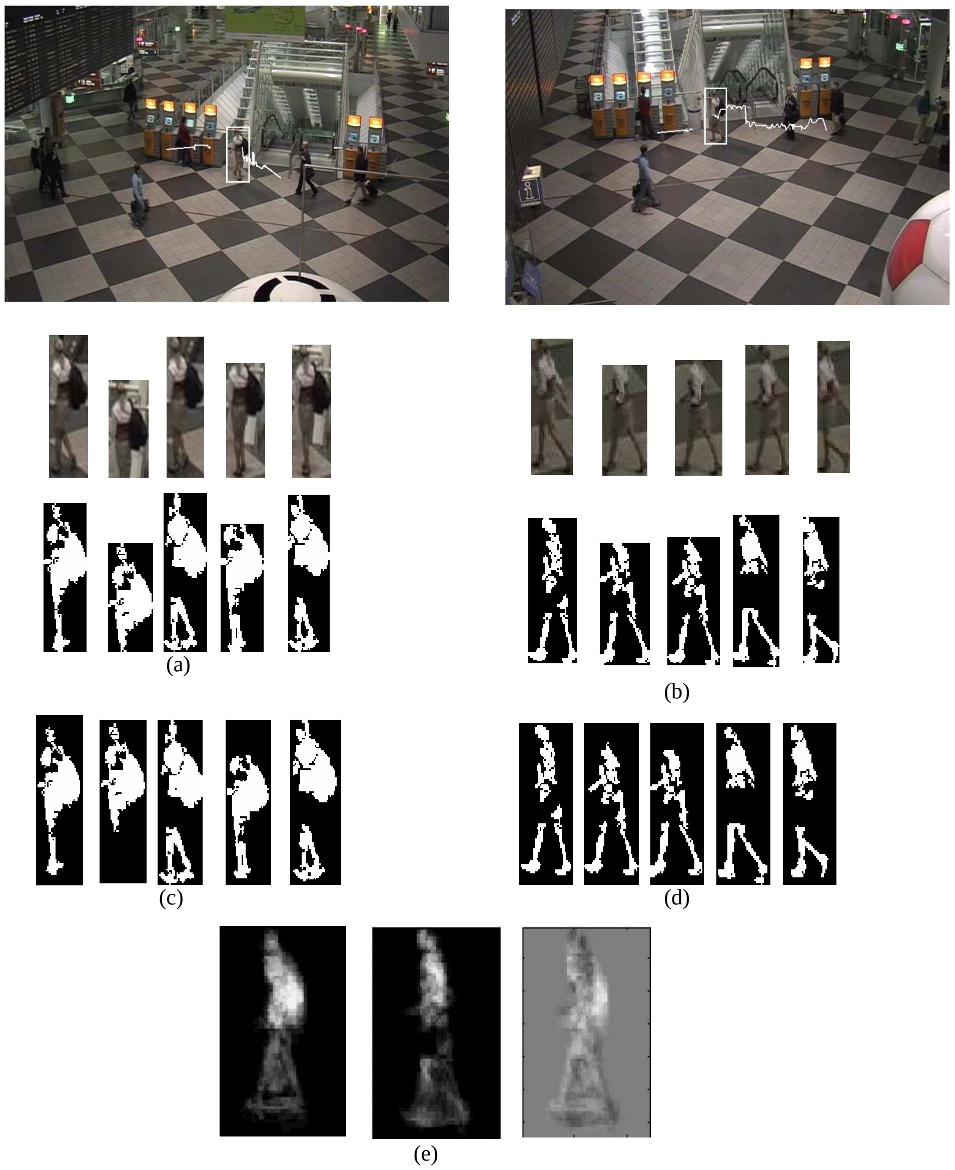


Figure 5.3: Airport Example

pearance change detection is not typically valid. Figure 5.4.a shows examples of bad segmentation where only parts of the silhouette are detected.

Figure 5.5 shows the procedure we use to align the detected parts of the silhouettes to reflect their actual position with respect to the complete body. Here we summarize the procedure. The BGS is applied to select foreground pixels on each frame. Blobs are extracted by a connected component analysis and tracked by computing overlapping bounding boxes.

From this tracking data, we compute the maximum height H_max and maximum width W_max of all tracked silhouette. We generate a template with height H_max and width W_max. We also estimate the walking direction of the subject as the direction of the subject trajectory detected by the tracking module. Using this computed walking direction, each tracked silhouette is positioned in the template image to reflect the actual position of the detected parts of the human body with respect to the whole body. Figure 5.4.b shows the silhouettes in 5.4.a after this alignment.

5.4 First Approach: Direct Classification of Silhouettes

We tested the direct classification of silhouettes on subjects from the airport video. Using the event detection GUI, we marked two regions. The first is around the trash bin, as discussed in section 5.2. The second is a randomly selected polygon on the empty space of the airport floor. Detecting 15 people entering and leaving the specified ROIs, we applied our analysis to detect whether each subject is carrying a bag or nor before he enters the ROI and after leaving it to identify people dropping or picking objects. Here, we show the results for both the OC-classifier and the Extended-OC-classifier and discuss these results. For each classifier, we show the results in two ways. First, we alignment.bmp

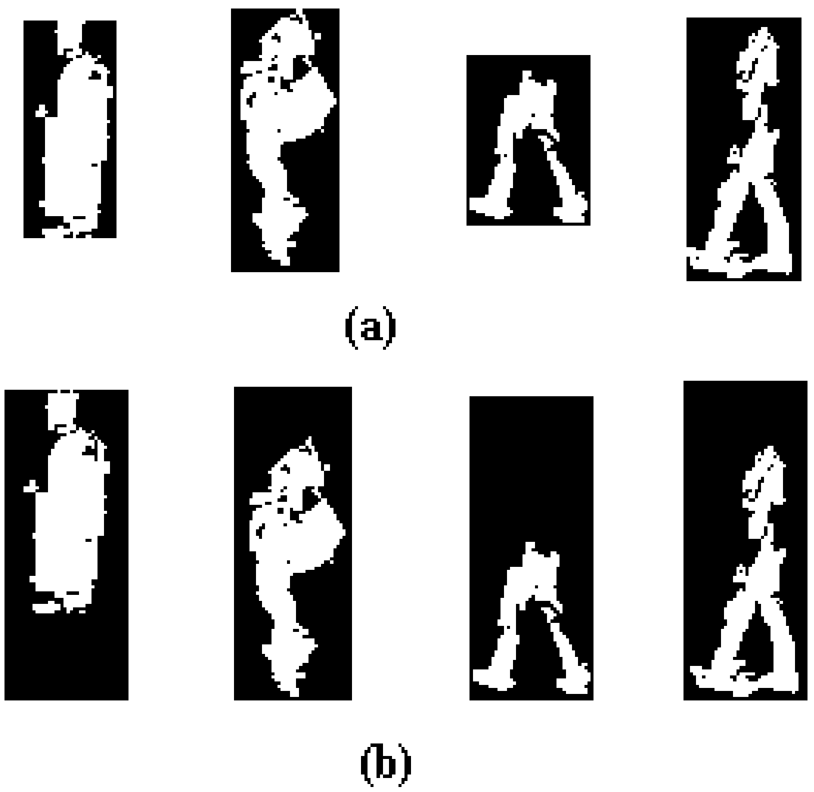


Figure 5.4: (a) Sample Blobs Tracked by Blob Tracking Module (b) Aligned Blobs to

be

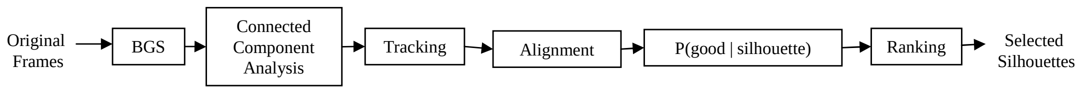


Figure 5.5: Alignment Procedure

show the classifier performance in classifying each subject instance as carrying a bag or not. Second, we will show the results in terms of detecting the change in the subject resulting from dropping or picking up a bag while accessing the ROI.

Tables 5.4 and 5.4 show the recognition rates for the ‘Bag/No Bag’ case for both classifiers. The performance of both classifiers is comparable. Both classifiers detect only 5 out of 18 cases where the subject has a bag (backpack or suitcase)- (i.e. the number of true positives is 5 and false positives is 13). Both classifiers also classify almost all ‘No Bag’ cases correctly. For the OC-Classifier, the recognition rate is 57%. For the Extended-OC-Classifier, the recognition rate is 53%. Comparing these results to the laboratory data results in Chapter 4, we can see the great drop in the recognition rates. This can be explained by the bad segmentation results obtained from the airport data compared to the good ones obtained from the laboratory data.

	Bag	No Bag
Bag	5	13
Nobag	0	12

Table 5.1: Recognition Results for Airport Dataset for OC-Classifier - Format 1.

	Bag	No Bag
Bag	5	13
Nobag	1	11

Table 5.2: Recognition Results for Airport Dataset for Extended-OC-Classifier - Format 1.

	No. of cases	Correctly Classified	Incorrectly Classified
Nobag-Nobag	4	4	0
Bag-Bag	7	6	1
Bag-Nobag	3	1	2
Nobag-Bag	1	0	1

Table 5.3: Recognition Results for Airport Dataset for OC-Classifier - Format 2.

Tables 5.4 and 5.4 show the recognition rates for the ‘Change/No Change’ case for both classifiers. For the OC-Classifier, the overall recognition rate is 73%. When there is no change, 10 out of 11 cases are correctly classified, recognition rate is 91%. When there is a change, only 1 out of 4 cases is correctly classified, recognition rate is 25%. For the Extended-OC-Classifier, the overall recognition rate is 66%. When there is no change, 9 out of 11 cases are correctly classified, recognition rate is 82%. When there is a change, only 1 out of 4 cases is correctly classified, recognition rate is 25%. Although the recognition rates is not good enough, we have found that some cases are classified correctly but for the wrong reason. Many cases where the subject is carrying a bag in the before and the after sequences, the bag is not detected in both sequences - so the classifier decision is that there is no change.

5.5 Second Approach: Human Appearance Change Detection

For the example in section 5.2, we can see that the walking direction of the person is not the same in the before and the after sequences. In the first camera, the person is walking

	No. of cases	Correctly Classified	Incorrectly Classified
Nobag-Nobag	4	3	1
Bag-Bag	7	6	1
Bag-Nobag	3	1	2
Nobag-Bag	1	0	1

Table 5.4: Recognition Results for Airport Dataset for Extended-OC-Classifier - Format 2.

in a near-diagonal direction in the 'Before' sequence and he is walking fronto-parallel in the after sequence. In the second camera, the person is walking fronto-parallel in both the 'Before' and the 'After' sequences. To apply the appearance change detection procedure, we need both sequences to represent a similar view, preferably, the fronto-parallel view. We discuss the view selection problem in section5.5.1.

Assuming the tracker is able to track the person for large number of frames, we need to select around 20 good silhouettes to apply the appearance change detection procedure. We discuss the frame selection problem in section 5.5.2.

5.5.1 View Selection Problem

The performance of any image based appearance analysis is inherently view-dependent. Since we are detecting appearance changes resulting from dropping or picking objects like a backpack or a suitcase, these objects are most visible when the person is walking fronto-parallel to the camera plane. Detecting these appearance changes also compares the 'Before' sequence and the 'After' sequence and tests whether the difference between them is significant or not. For this comparisons to be meaningful, the 'Before' sequence and the ‘After’ sequence should represent the same or almost the same view of the person, preferably fronto-parallel views.

We achieve this view-selection in the multi-camera system by dynamically selecting the camera (or cameras) that capture the subject from the most similar views. For each camera, and for both the ‘Before’ and the ‘After’ sequences, we estimate the walking direction of the subject as the direction of the subject trajectory detected by the tracking module. To perform the comparisons required to detect changes in the appearance, we compare only the ‘Before’ and the ‘After’ sequences with the most similar walking direction.

For the example in section 4.2, the person is walking in a near-diagonal direction in the ‘Before’ sequence and fronto-parallel in the after sequence, for the first camera. In the second camera, the person is walking fronto-parallel in both the ‘Before’ and the ‘After’ sequences. Thus, we compared the ‘Before’ sequence captured by the second camera and the ‘After’ sequence captured by both cameras. The change detection module has detected the backpack drop in both cases.

5.5.2 Frame Selection Problem

Assuming we have applied the alignment procedure discussed in section 5.3 to all silhouettes produced by the tracker, then we need to select a small number of these frames where the person’s silhouette is as complete as possible to use these frames for the change detection procedure. For each aligned silhouette, a probability measure that estimates how good the silhouette represents a good one is calculated. This measure is calculated as follows. We build a database of about 600 silhouettes extracted from the airport data. Each silhouette is labeled manually as good or bad. For each silhouette, we divide the silhouette into k horizontal strips of equal length and compute a feature vector of length k, where the value of the j^{th} feature is the percentage of foreground pixels in the j^{th} horizontal strip. We train a classifier on these two classes. Then for each test silhouette, the classifier outputs the most probable class for that silhouette. For each class, the classifier also estimates the posterior probabilities that this class was the source of that silhouette, i.e. $p(good|silhouette)$ and $p(bad|silhouette)$. We use $p(good|silhouette)$ as the probability measure on which we rank the silhouettes. Then we select the top k silhouette to be used for change detection analysis.

5.5.3 Results

In section 5.2, we provided an example from the airport data and discussed the view selection problem and the frame selection problem. For frame selection, we applied the alignment process described in section 5.3 to both the 'Before' and 'After' sequences, ranked the frames and selected the top 20 frames in each sequence to perform the analysis for appearance change detection. Figures 5.3.e and 5.3.f shows the silhouettes in figures 5.3.c (from the second camera) and 5.3.d (from the first camera) after applying this alignment process. We compared the 'Before' sequence captured by the second camera and the 'After' sequence captured by both cameras. Generating the features and providing them to the SVM classifier, the change detection module has detected the backpack drop in both cases. We tested the appearance change detection on other subjects from the airport video. Using the event detection GUI, we marked two regions. The first is around the trash bin, as in the previous example. The second is a randomly selected polygon on the empty space of the airport floor. Detecting 15 people entering and leaving the specified ROIs, we applied the same analysis to compare the appearance of each person before and after accessing the ROI to identify people dropping or picking objects. Figure 5.6 shows a drop example and a pickup example. Table 5.5.3 summa-

	No. of cases	Correctly Classified	Incorrectly Classified
Nobag-Nobag	4	4	0
Bag-Bag	7	7	0
Bag-Nobag	3	2	1
Nobag-Bag	1	1	0

Table 5.5: Recognition Results for Airport Dataset for Change Detection Classifier.

rizes the classification results. The SVM classifier classifies all cases where there is no change (Nobag-Nobag or Bag-Bag) correctly. The classifier also detects 3 out of 4 cases where a change takes place (Bag-Nobag or Nobag-Bag). The only change that was not detected by the classifier was due to very bad segmentation results.

5.6 Discussion

Moving to real world video required addressing problems such as bad segmented silhouettes. Our frame alignment process tries to reduce these bad results.

Using these real world data, we have compared the performance of the first approach, where we classify features computed using silhouettes of a given instance directly to determine if the person is carrying a bag or not to the performance of the second approach, where we classify features computed using silhouettes of 2 instances of the same person to determine if there is a significant change in the person's appearance due to bag drop or pickup or not. It has been shown that the change detection classifier outperforms both the OC-Classifier and the Extended-OC-Classifier. This can be explained by the robustness of the change detection classifier to segmentation errors. Using the difference

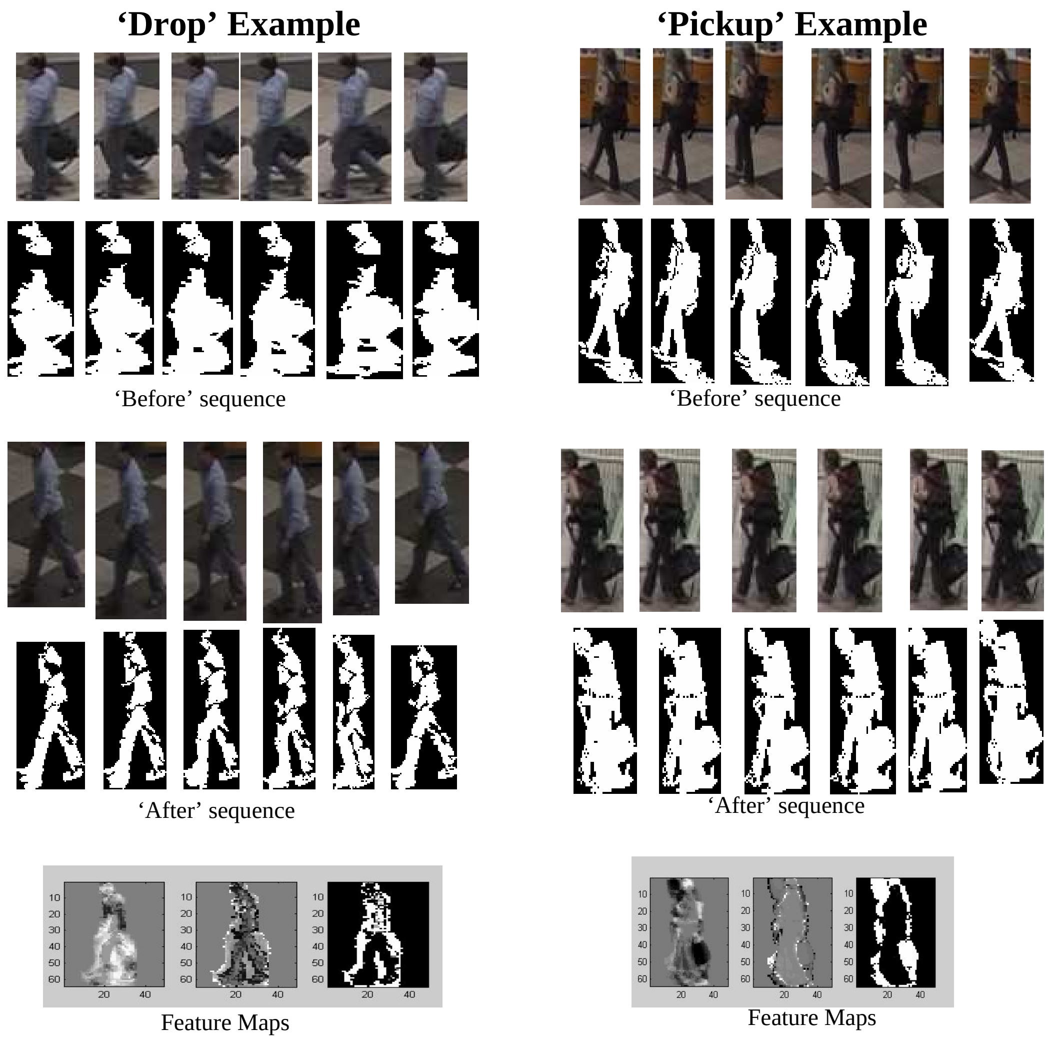
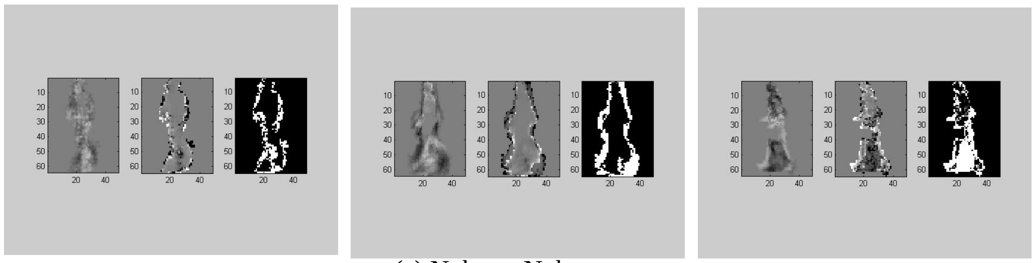
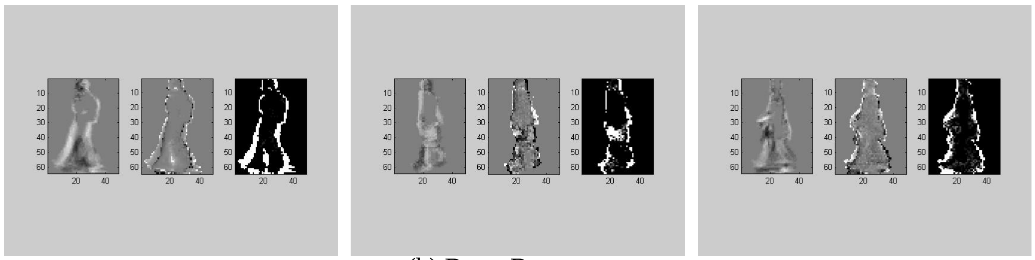
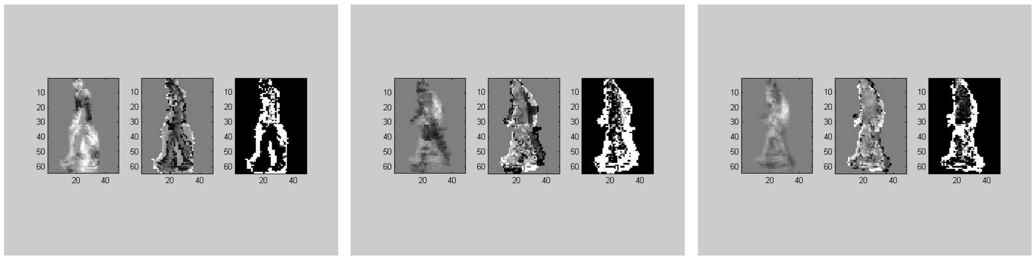


Figure 5.6: Examples from Airport Dataset


(a) Nobag - Nobag


(b) Bag - Bag


(c) Bag - Nobag

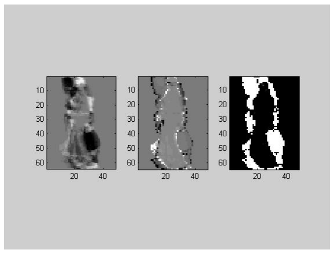

(d) Nobag - Bag

Figure 5.7: Feature Maps from Airport Dataset

removes the noise introduced by the segmentation errors.

Chapter 6

Conclusions and Future Work

6.1 Summary

The first part of the thesis describes a framework for modeling and recognition of events from surveillance video. Our framework is based on deterministic inference using Petri nets. Events are composed by combining primitive events and previously defined events by spatial, temporal and logical relations. We described the system's graphical user interface (GUI) where such event models can be formulated. An automatic mapping mechanism is devised to map each event structure into a set of Petri net models that represent the components of the event. Lower-level video processing modules (background subtraction, tracking, etc.) are used to detect the occurrence of primitive events. These primitive events are then filtered by the Petri nets models to recognize composite events of interest.

We have evaluated the performance of the system across different natural scenes to detect events with increasing level of complexity ranging from primitive events involving one object to more complex events involving multiple objects and multiple logical and temporal relations. In the first set of experiments, we have applied our system on a dataset of video sequences provided as a part of ETISEO project. The length of these sequences range from 800 frames to 3000 frames. In these sequences, required events to be detected range from single primitives to simple sequences of two or three primitives. In the second set of experiments, we applied our system on a longer video sequence (25200 frames) to detect and recognize events and violations performed by pedestrians and vehicles in a traffic intersection. Events in this case are more complex and include multiple actors and multiple logical and temporal relations.

In the second part of the thesis, we have addressed the problem of detecting carried objects. We have presented two machine learning approach to detect whether the subject has an object at one time and not at another time. We applied both approaches to the problem of left package detection. We trained SVM classifiers on a laboratory database that contains 180 examples of people seen with and without two common objects, namely backpacks and suitcases. A recognition rate of 96% was obtained in the case of backpacks, 93% for suitcases and 90% for the combined case. Using a boosting technique, AdaBoost, to select the most discriminative features, we reduced the number of features used by the SVM classifier to less than 4% of the total original number of features. We have also tested both approaches on videos captured at the Munich airport.

We plan to refine and extend our work in the following ways:

6.2 Directions for Future Work

6.2.1 Event Modeling and Recognition

- Since uncertainty is inherit in video data, we plan to extend our framework to deal with these uncertainties. For Petri nets, there is a great deal of research handling uncertainty in inference using them. Examples include fuzzy Petri nets, possibilistic Petri nets and stochastic Petri nets. As probabilities are the most intuitive and accepted measures of uncertainty, the same inference capabilities of Bayesian networks can be applied to Petri nets.

- Extend our event detection framework to deal with multiple cameras simultaneously as multiple camera-based visual surveillance systems can be extremely helpful because the surveillance area is expanded and multiple view information can overcome occlusion and reduce uncertainties .

6.2.2 Left Object Detection

- Since any human appearance analysis method is inherently view-dependent, we need to extend our approach by extracting other features that are independent of the camera view.
- Apply a similar classification approach to interpret a subject's body movement, e.g. to determine whether his movement in a critical region can be interpreted as a drop or pickup action.

BIBLIOGRAPHY

[1] T. Starner, A.Pentland. Real-time American Sign Language Recognition from Video Using Hidden Markov Models. *Proceedings of International Symposium on Computer Vision*, pages 265 –270, November 1995.

[2] Nuria Oliver, Barbara Rosario and Alex Pentland. A Bayesian Computer Vision System for Modeling Human Interactions. *Proceedings of Intl. Conference on Vision Systems ICVS99. Gran Canaria. Spain.*, January 1999.

[3] Ivanov, Y.A.; Bobick, A.F. Recognition of Visual Activities and Interactions by Stochastic Parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:852–872, 2000.

[4] P. Remagnino, T. Tan, K. Baker. Agent Oriented Annotation in Model Based Visual Surveillance. *ICCV, 4-7 January 1998,Bombay, India*, pages 857–862, January 1998.

[5] Somboon Hongeng, Ramakant Nevatia. Multi-Agent Event Recognition. *ICCV*, pages 84–93, 2001.

[6] Nicolas Moenne-Loccoz, Francois Bremond, Monique Thonnat. Recurrent Bayesian Network for the Recognition of Human Behaviors from Video. *ICVS*, pages 68–77, 2003.

[7] Claudio S. Pinhanez, Aaron F. Bobick. Human Action Detection Using PNF Propagation of Temporal Constraints. *CVPR*, January 1998.

[8] Nathanael Rota, Monique Thonnat. Activity Recognition from Video Sequences using Declarative Models. *ECAI 2000*, pages 673–680, 2000.

[9] Van-Thinh Vu, Francois Bremond, Monique Thonnat. Automatic Video Interpretation: A Recognition Algorithm for Temporal Scenarios Based on Pre-compiled Scenario Models. *ICVS*, pages 523–533, 2003.

[10] Charles Castel, Laurent Chaudron and Catherine Tessier. What Is Going On? A High Level Interpretation of Sequences of Images. *4th European conference on computer vision, Workshop on conceptual descriptions from images, Cambridge UK*, 1996.

[11] J.L. Peterson. Petri Nets. *ACM Computer Surveys*, 9:223–252, 1977.

[12] C. L. Forgy. RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.

[13] D.D. Burdescu, M. Brezovan. High Level Petri Nets and Rule Based Systems for Discrete Event System Modelling. *International Journal of Smart Engineering System Design*, 3:81–97, 2001.

[14] N. Bird, S. Atev, N. Caramelli, R. Martin, O. Masoud, and N. Papanikolopoulos. Real time, online detection of abandoned objects in public areas. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006*, pages 3775 – 3780, 2006.

[15] G. L. Foresti, L. Marcenaro, and C. S. Regazzoni. Automatic detection and indexing of video-event shots for surveillance applications. *IEEE Transactions on Multimedia 2002*, 4(4):459–471, December 2002.

[16] Fengjun Lv, Xuefeng Song, Bo Wu, Vivek Kumar Singh, and Ramakant Nevatia. Left-luggage detection using bayesian inference. *Proceedings 9th IEEE International Workshop on PETS, New York, June 18, 2006*, pages 83–90, 2006.

[17] M. Spengler and B. Schiele. Automatic detection and tracking of abandoned objects. *Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance 2003*, 2003.

[18] D. Thirde, M. Borg, J. Ferryman, J.Aguilera, M. Kampel, and G. Fernandez. Multi-camera tracking for visual surveillance applications. *11th Computer Vision Winter Workshop 2006*, 2006.

[19] V.N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988– 999, 1999.

[20] R. E. Schapire. A brief introduction to boosting. *IJCAI*, pages 1401–1406, 1999.

[21] Hilary Buxton, Shaogang Gong. Visual Surveillance in a Dynamic and Uncertain World. *Artificial Intelligence*, 78(1-2):431–459, 1995.

[22] Stephen S. Intille, Aaron F. Bobick. A Framework for Recognizing Multi-Agent Action from Visual Evidence. *AAAI/IAAI*, pages 518–525, 1999.

[23] Douglas Ayers, Rama Chellappa. Scenario Recognition from Video Using a Hierarchy of Dynamic Belief Networks. *ICPR*, pages 1835–1838, 2000.

[24] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[25] Tadao Murata, Du Zhang. A Predicate-Transition Net Model for Parallel Interpretation of Logic Programs. *IEEE Transactions on Software Engineering*, 14(4):481–497, 1988.

[26] Hura, G.S. Representation and Processing of Rule-Based Expert System Using Petri Nets: A Viable Framework . *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, 2:934 –937, 1993.

[27] Liwu Li. High-level Petri Net Model of Logic Program with Negation . *IEEE Transactions on Knowledge and Data Engineering*, 6:382 –395, 1994.

[28] Murata, T.; Jaegeol Yim;. Petri net Methods for Reasoning in Real-Time Control Systems. *1995 IEEE International Symposium on Circuits and Systems*, 1:517 –520, 1995.

[29] P. Kemper. Transient Analysis of Superposed GSPNs. *IEEE Transactions on Software Engineering*, 25:182 –193, March-April 1999.

[30] Lin, C.; Marinescu, D.C. Stochastic High-Level Petri Nets and Applications. *IEEE Transactions on Computers*, 37:815 –825, July 1988.

[31] C.G. Looney. Fuzzy Petri Nets for Rule-Based Decisionmaking. *IEEE Transactions on Systems, Man and Cybernetics*, 18:178 –183, Jan.-Feb. 1988.

[32] Shyi-Ming Chen; Jyh-Sheng Ke; Jin-Fu Chang. Knowledge Representation Using Fuzzy Petri Nets . *IEEE Transactions on Knowledge and Data Engineering*, 2:311 –319, September 1990.

[33] Konar, A.; Mandal, A.K. Uncertainty Management in Expert Systems Using Fuzzy Petri Nets. *IEEE Transactions on Knowledge and Data Engineering*, 8:96 –105, Febraury 1996.

[34] Scarpelli H., Gomide F., Yager R.R. Reasoning Algorithm for High-Level Fuzzy Petri Nets. *IEEE Transactions on Fuzzy Systems*, 4:282 –294, August 1996.

[35] J. Cardoso, R. Valette and D. Dubois. Possibilistic Petri Nets . *IEEE Transactions on Systems, Man and Cybernetics B*, pages 573 –582, October 1999.

[36] P. Muro-Medrano and J. Banares and J. Villarroel. Knowledge Representation-Oriented Nets for Discrete Event System Applications.

[37] I. Haritaoglu, R. Cutler, D. Harwood, and L. S. Davis. Backpack: Detection of people carrying objects using silhouettes. *IEEE International Conference on Computer Vision 1999*, 1:102–107, 1999.

[38] C. Sacchi and C. S. Regazzoni. A distributed surveillance system for detection of abandoned objects in unmanned railway environments. *IEEE Transactions on Vehicular Technology 2000*, 49(5):2013–2026, September 2000.

[39] Jesus Martinez del Rincon, J. Elias Herrero-Jaraba, Jorge Raul Gomez, and Carlos Orrite-Urunuela. Automatic left luggage detection and tracking using multi-camera ukf. *Proceedings 9th IEEE International Workshop on PETS, New York, June 18, 2006*, pages 59–66, 2006.

[40] Kevin Smith, Pedro Quelhas, and Daniel Gatica-Perez. Detecting abandoned luggage items in a public space. *Proceedings 9th IEEE International Workshop on PETS, New York, June 18, 2006*, pages 75–82, 2006.

[41] High-level Petri Nets - Concepts, Definitions and Graphical Notation. *Final Draft International Standard ISO/IEC 15909*, October 2000.

[42] Abbas K. Zaidi. On Temporal Logic Programming Using Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics A*, 29(3):245 –254, May 1999.

[43] Egenhofer,M. J. and Herring, J. Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. *Technical Report, Department of Surveying Engineering, University of Maine.*, 1991.

[44] W. Hu, T. Tan, L. Wang, and S. Maybank. A Survey on Visual Surveillance of Object Motion and Behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(3), August 2004.

[45] Fatih Porikli. Achieving real-time object detection and tracking under extreme conditions. *Journal of Real-Time Image Processing*, 1(1):33–40, October 2006.

[46] Etiseo project. http://www.silogic.fr/etiseo/.

[47] A. Elgammal, R. Duraiswami, and L. S. Davis. Background and Foreground Modeling using Non-parametric Kernel Density Estimation for Visual Surveillance. *Proceedings of the IEEE*, July 2002.

[48] B. Heiseley, P. Hoz, and T. Poggio. Face recognition with support vector machines:global versus component-based approach. *International Conference on Computer Vision, ICCV2001*, 2001.

[49] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 1(2), 2002.

[50] E. Borovikov, R. Cutler, T. Horprasert, and L. Davis. Multi-perspective analysis of human actions. *Third International Workshop on Cooperative Distributed Vision*, 1999.

[51] K. Kim, T.H. Chalidabhongse, D. Harwood, , and L. Davis. Background modeling and subtraction by codebook construction. *International Conference on Image Processing*, pages 3061– 3064, 2004.

[52] J.S.U. Hjorth. *Computer Intensive Statistical Methods Validation, Model Selection, and Bootstrap*. London: Chapman and Hall, 1994.

[53] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1), 1991.