ABSTRACT

Title of thesis: REAL-TIME UNATTENDED PACKAGE DETECTION USING A SYNTACTIC APPROACH

Gavin Rosenbush
Master of Science, 2007

Thesis directed by: Professor Rama Chellappa
Department of Electrical and Computer Engineering

Detecting unattended packages in video is a challenging surveillance problem. The goal is to detect packages and their owners, determine relationships between them and the environment, and recognize when an owner abandons a package. Concurrent events and complex interactions create problems for existing motion-based systems. Errors in target detection and tracking caused by shadows, noise, and occlusions create additional problems. We present a real-time system that addresses these issues and recognizes the unattended package event syntactically using a stochastic attribute grammar. Our system can detect events that occur concurrently, are corrupted by target and detection errors, or contain packages hidden behind other objects such as trash cans.

In this thesis, we review existing motion-based methods, then present our system and show results on unattended package detection in situations that would cause motion-based systems to fail.

REAL-TIME UNATTENDED PACKAGE DETECTION USING A SYNTACTIC APPROACH

by

Gavin Rosenbush

Thesis 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
Master of Science
2007

Advisory Committee:
Professor Rama Chellappa, Chair/Advisor
Professor Min Wu
Professor Ankur Srivastava

Acknowledgements

I would like to thank my advisory committee Dr. Rama Chellappa, Dr. Min Wu, and Dr. Ankur Srivastava for their guidance and support. For their technical support during this process, I would like to thank Seong-Wook Joo, Yang Ran, Amon Ducao, and Volkan Cevher.

This research was partially funded by a contract with the Maryland Industrial Partnerships Program (MIPS).

Table of Contents

List of Tables

List of Figures

List of Abbreviations

PETS Performance Evaluation of Tracking and Surveillance

Chapter 1

Introduction

Unattended package detection is a well studied problem [1, 12, 18]. Statistical and syntactic methods have been attempted to aid in the detection of packages and their attending status. Early methods simply sought to locate the packages [6, 19], while newer methods [18] assign ownership to the packages and track the owner's location to determine when a package is left behind. In this introduction section, we will first outline the existing methods for unattended package detection, and then give some background discussion on syntactic pattern recognition – the method used for this thesis. Lastly, we will present an outline of the thesis.

1.1 Unattended Package Detection

Unattended package detection is a problem that has been gaining attention in recent years due to security concerns in a variety of public areas. With the increase in camera surveillance of public areas, online intelligent systems are in high demand. We would like these systems to be able to identify suspicious events and alert security personnel to deal with them.

These systems have two advantages over human-monitored security systems. First, they can ease the workload of security personnel, allowing them to focus on vital tasks rather than monitoring innocuous video streams. Secondly, intelligent

systems can often detect scenarios in complex scenses that humans do not notice. Events unfold over many minutes, and the increased memory and processing power available on today's computers allows for tracking and detailed analysis.

1.1.1 Problem Formulation

The unattended package problem seeks to identify packages left behind by humans in public places and identify their former owners. In order to do this, we split the problem into several parts: detection and classification of humans and objects, tracking, establishing relationships among people and objects, checking for alarm conditions, and sounding an alert when they are satisfied. Previous methods differ in all but the last step.

Because video data contains many objects acting simultaneously, determining the relationships amongst them is a difficult task. Existing methods are detailed in chapter 2 and their disadvantages are described. These solutions are adequate when simple situations are being recognized, however when multiple events occur simultaneously involving complex relationships among objects and events, the use of stochastic attribute grammars can provide better solutions.

We focus primarily on unattended package detection in outdoor environments to demonstrate the improvements offered by syntactic pattern recognition, shadow removal, and human gait analysis. Some indoor scenarios are also presented to illustrate the improvements offered by stochastic syntactic pattern recognition in handling the noise of the input data.

1.1.2 Previous Work

Previous work is quite varied: many systems have been proposed to detect and classify human activity. The previous work that is drawn upon for this thesis can be divided into three categories: human activity classification, event detection, and video pre-processing. Several improvements are offered by these works, and their use in this work is detailed in chapter 3.

Human activity classification systems examine humans in video and classifying what activity is being performed – often for security applications. Work has been done in activity classification to identify people carrying objects by examining silhouettes in [7]. Additional work in [15] uses cues from the human gait to determine when a person is carrying an object.

Event detection systems examine multiple objects and infer relationships amongst them to sound alarms when certain events take place. A variety of techniques have been used. Finite-state machines were used in [1] to detect unattended packages. A stochastic context-free grammar and stochastic parser was used in [13] to detect high-level events in a blackjack game involving multiple players and a dealer. A stochastic attribute grammar was used for online event detection in a parking lot in [9].

Video pre-processing consists of methods to clean up input video from noise. Shadows are a problem in both indoor and outdoor video systems and several methods exist to alleviate the issue. Moving shadows are removed in [14] using the HSV color space to separate intensity and chromacity. Vector projections are used to

estimate the chrominance distortion for a pixel and threshold shadows in [1].

1.2 Proposed System

In this thesis, we propose a real-time system for recognizing the unattended package event. We develop a stochastic attribute grammar based on the framework in [9] and extend it with some original work. Primitive events are extracted from the video and are parsed for matches in real-time. Multiple event threads are maintained for events that may be unfolding simultaneously. Production probabilities are used to represent the certainty of the particular event's occurrence.

The system is extended to detect unattended packages when the package is not visible to the camera by analyzing the human gait based on [15] but extended to work in a real-time system. Video pre-processing algorithms are used to reduce the effect of moving shadows. Lastly, we expand the primitive extractor and utilize the expressivity of the attribute grammar to alleviate the affects of unavoidable tracker errors and noisy input.

Experiments demonstrating the capability of the system are shown in chapter 4.

The system is written in C++ and runs on multiple platforms.

1.3 Outline of Thesis

In Chapter 2 we describe motion-based methods for detecting unattended packages. In Chapter 3, the proposed system is described and finally in Chapter 4,

results are shown from experiments using the proposed system.

Chapter 2

Motion-Based Unattended Package Detection

2.1 Overview

Existing motion-based systems are quite varied in their capabilities. In this chapter we detail several motion-based approaches that were proposed at the 2006 PETS conference which focused on the unattended package detection problem and one finite-state machine approach that was proposed in 2003.

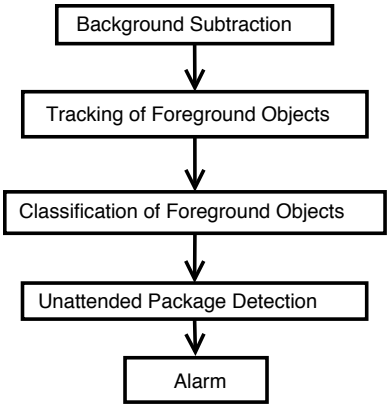


Figure 2.1: Block diagram showing the steps before and after the unattended package problem.

In our discussions, we do not pay much attention to the background subtraction or tracking methods used in these approaches, but rather on the anomaly detection mechanisms: the steps between tracking and sounding an alert, represented as the "classification" and "unattended package detection" boxes in Figure

2.1 and expanded in Figure 2.2. Additionally, several of these works use multiple-camera systems whereas ours is a single-camera system. Throughout we will point out the enhancements offered by using multiple cameras and refer to Chapter 3 which details how we accomplish similar goals in our single-camera system.

2.2 Motion-Based

The existing works will be compared based on how they accomplish the steps shown in Figure 2.2 in the unattended package problem. It is assumed that we get an input of the tracking data obtained by background subtraction and object tracking in the previous steps which aren't detailed in this thesis.

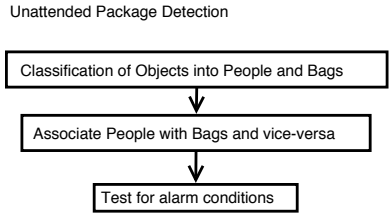


Figure 2.2: Steps in the Unattended Package Problem.

2.3 Target Classification

The first step in the process is to take the input targets from the tracker and classify them into people and packages. Most of the works do not place much emphasis on this step of the problem and use a simple pixel-area threshold as the classifier.

[1] uses a feature called compactness for classification where compactness

$C = \frac{area}{perimeter^2}$. A Bayesian classifier is used to combine measurements from video channels and classification is based on the Gaussian assumption and MAP quadratic discriminant functions.

In [18], the velocity and size of a blob is used to compute the likelihood that the blob is a package based on this distribution $P(B^i = 1 \mid X_{1:t}) \propto N(s_t^i, \mu_s, \sigma_s) exp(-\lambda v_t^i)$. Based on this likelihood function, objects with small size and low velocity are selected as packages. Candidates selected using this function must also not be on the border of the image nor on top of the positions of other candidates. These last two criteria eliminate tracker errors. After an object is selected as a package, subsequent frames are compared to the shape of the package from the segment containing the stationary package. This work determines the owner of packages (human objects) when certain tracker conditions are met. This is detailed in the next section on package ownership assignment.

[12] uses an area and height threshold to select package and human objects using scene calibration and single-view metrology.

2.4 Package Ownership Assignment

Once objects have been classified into humans and packages, we must assign ownership of the packages by the humans. Once we have assigned ownership to packages, the conditions of the unattended package event can then be tested in the next step. In this section we will detail how the various works determine ownership of packages.

A video history buffer is maintained for all cameras in [1]. When a new idle package is detected, this buffer is examined for all cameras. The authors examine the color histogram of certain areas of all human objects and select the owner based on the best match. This approach has the advantage that if a package drop is occluded, the owner can still be selected if the history buffer contains the owner with the package before the occlusion. However, this does not account for packages that are dropped behind other objects or packages that are too similar to the background.

In [18], the following observations are made. Before a person drops a package, the tracker is tracking the person carrying the package. As yet the package is unseen by the system. When the owner drops the package, one of two situations occur. Either the tracker remains on the person and a new tracker is created for the object or the tracker remains on the object and a new tracker is created for the person. Therefore, when a new package is detected, we check the tracker history for these situations to identify the owner.

[12] associates the closest human object to a package as its owner. Although this is a simple calculation, it does not scale well to situations with multiple people in the scene.

2.5 Alarm Conditions

The final step of the unattended package detection problem involves examining the relationships determined in the previous steps and testing for alarm conditions. Most of the existing works use a finite-state machine approach for determining when

to sound an alert for the unattended package event.

The finite-state machine approach in [1] uses a four-camera system with overlapping fields-of-view and combined processing. In this system, when a new package is detected, the system looks in its history to associate an owner with the package. The abandoned package state can be described as follows [1]:

$$abandoned = (v < V_0) \wedge (unowned \vee dist > D) \wedge (\Delta t > T_a)$$

The package must have a velocity under a certain threshold V_0 and the owner must be further than a distance D away from it or have no discernible owner. Once these conditions are met for a certain time period T_a, an alarm is set.

While this system solves the occlusion problem by employing multiple cameras, it is vulnerable to background subtraction and tracker errors: it will fail for any packages that are similar to the background. False positives will be generated when the ownership of a package cannot be deteremined. Ownership of packages is determined by examining a stored frame buffer which can cause problems when multiple people are in close proximity or in complex scenarios involving package handoffs. It is also unclear how concurrent events will be handled in such a system: false positives occurring first may distract the system from true positives.

The finite-state machine put forth in this work provides a good starting point for how unattended package detection would work in controlled scenarios, however it is clear that a framework that can express more complex relationships among events is needed.

Several works from the PETS 2006 conference also use a FSM approach for detecting unattended packages [12, 18]. Because the requirements for an unattended

package were fixed in the PETS 2006 conference, the works presented there have similar approaches. From the requirements of the conference, a package is considered unattended if its owner is greater than 3 meters away. A warning is announced if the owner is greater than 2 but less than 3 meters away from the package. [12, 18] and most of the other papers at PETS 2006 use a homographic transformation to map video coordinates onto the ground plane and calculate the true distances between objects.

Chapter 3

Unattended Package Detection using Stochastic Attributes

3.1 Overview

In this section we present the unattended package system. Relevant background material is given throughout. This chapter is organized as follows. First, we briefly discuss the detection and tracking method. Next, we explain primitive extraction, then pattern specification followed lastly by pattern recognition. A block diagram of the syntactic pattern recognition process as implemented in our system is shown in figure 3.1[9]. The system description in this chapter roughly follows this layout.

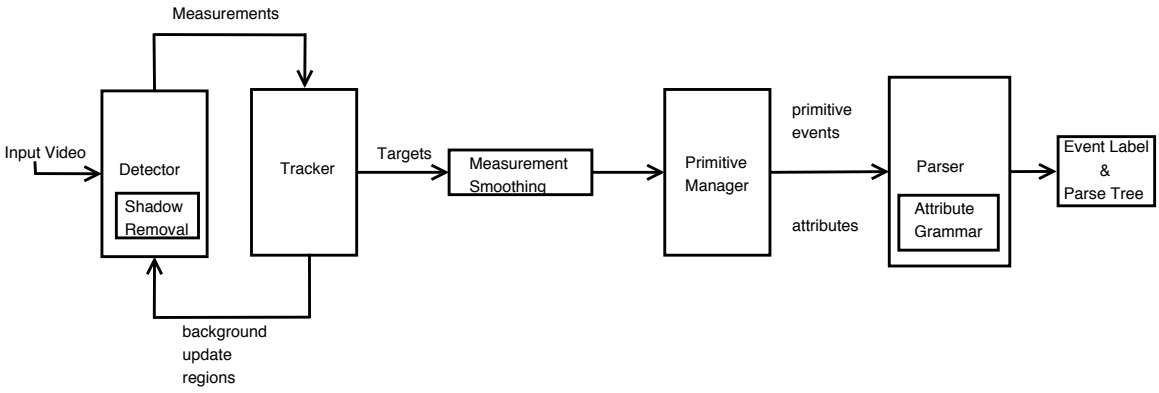


Figure 3.1: Block diagram of the recognition process.

3.2 Detection and Tracking

Detection and tracking is not the focus of this thesis, but we will go over it briefly. No tweaking has been done on the detector or tracker [9] for use on the data in our experiments. We demonstrate the power of the attribute grammar by using it to correct for detection and tracking errors we encounter.

The first step is to detect the foreground. Pixels are modeled individually using an adaptive Gaussian background model. Foreground candidates are selected using a color variance threshold. A timer is used for each pixel to delay foreground objects from becoming part of the background model. Connected components analysis is performed on the detected pixels to detect objects (measurements).

Existing targets from the previous frame are associated with the measurements of the current frame. A bipartite graph and a minimum weight edge covering algorithm [17] is used to associate measurements with targets. The weights in the graph are set to the Euclidean distance between the targets and the measurements. This is shown in Figure 3.2 [9]. A Kalman filter with constant velocity motion model is used to track the target state with the measurement bounding box as update input.

3.2.1 Video Pre-Processing

Once we have bounding boxes from the tracker, we do some video pre-processing in order to clean up the data for the other components of the system.

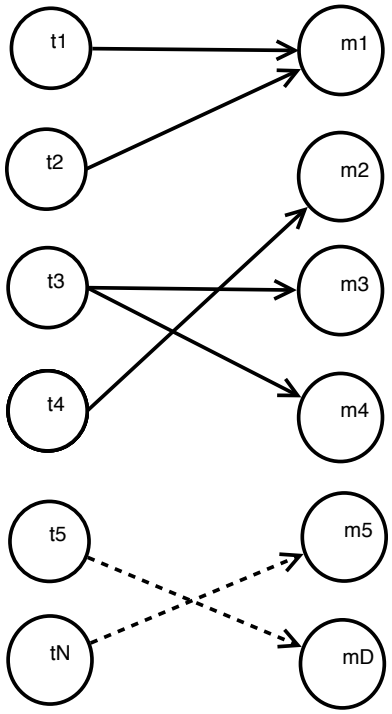


Figure 3.2: Tracking graph showing associations between measurements and targets. Dotted lines connect recently dissociated measurements and targets.

3.2.1.1 Measurement Smoothing

Before processing the target information we get from the tracker, we apply smoothing on much of it. We simply use a linear combination of the previous and current values for the next state value, eg. $X_n = \alpha X_{n-1} + (1 - \alpha)X_n$. This prevents outlying tracker data from having a large impact on the trajectory of the object. We also smooth the trunk width ratios which are described in section 3.4.1.1.

3.2.1.2 Shadow and Highlight Detection

In video systems, particularly those with human subjects, moving shadows and highlights (reflections) are often a problem because they differ from the background enough that they often show up as foreground objects. Because the background subtraction algorithm used in our system was not set up to handle shadows, we used a method that was presented in [1] and [8], tweaked it to our purposes and added it to the detection system.

The idea of this method is that a color pixel can be said to differ from its expected value (the background model) in terms of its *brightness distortion* and its *chromacity distortion*. Because moving shadows and reflections are cast by moving objects, namely people, they are similar in chromacity but different in brightness. Pixels that have a lower brightness are considered shadows and pixels with a higher brightness are considered highlights. In our system we simply check the absolute difference so that both shadows and highlights are removed from further consideration. This step was inserted into the system before the pixel variance is measured. We

simply classify the pixels as either shadow, highlight, or don't care. The pixels classified as shadows and highlights are removed from being considered as foreground pixels and all remaining pixels are classified as background or foreground using the model detailed previously.

To measure the pixel's distortion, we consider the geometric relationship between the expected pixel (background model) and the pixel under consideration, shown in Figure-3.3. The expected chromacity line is the line made by the background pixel in RGB space. The angle between the expected and measured pixel values is the *chromacity distortion* of the pixel. In order to compute the *brightness distortion* of the current pixel, we project it onto the expected chromacity line. The distance from where the project lies on the line to the expected value is the *brightness distortion*. If the chromacity distortion is below some threshold and the brightness distortion is greater than some threshold, then we classify the pixel as shadow or highlight.

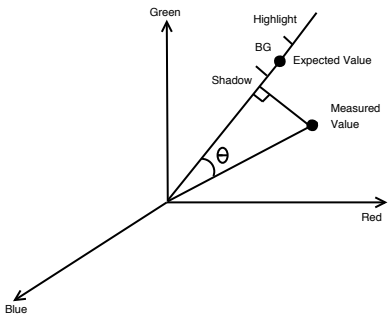


Figure 3.3: Shadow and highlight pixel classification model [1].

3.3 Pattern Specification

Pattern specification is the first step in any pattern recognition process. We must first specify what patterns we are trying to locate amongst the received signals. Once the pattern is specified, we can search the input signal for the pattern to identify its occurrence. This section will cover the various tools that we use to specify the patterns syntactically.

3.3.1 Grammars and Languages

Grammars are a fundamental construct of syntactic pattern recognition. A grammar with respect to a written language is a set of rules to indicate the proper formation of sentences. The grammar specifies what elements of speech (nouns, verbs, adjectives, etc.) make up a properly-formed sentence and in what order they may appear. Sentences that do not match the rules of the grammar are not considered correct and therefore are not members of the language. The language is the set of all strings that are generated by the grammar, and is referred to as L(G).

In syntactic pattern recognition, a common method to recognize patterns is to define one or more grammars, each for a pattern class we are trying to match. Signals that match the pattern class are members of the corresponding language.

Formally, a grammar is a set $G = (V_N, V_T, P, S)$ of non-terminal symbols, terminal symbols, productions, and a start symbol respectively. Symbols are the building blocks that make up the pattern. Non-terminal symbols are made up of other non-terminal symbols and terminal symbols, while terminal symbols are the

lowest building block in the pattern. In the case of written languages, clauses (eg. noun or verb clauses) are non-terminal symbols and are made up of terminal symbols (words, punctuation, etc). A start symbol is simply a symbol that can start off a rule in the grammar. The productions are the rules that specify how to construct a properly formed sentence. Beginning with the start symbol, non-terminals are expanded and when the sentence contains only terminal symbols, a complete well-formed sentence has been derived.

An example grammar is shown below [5]:

$< sentence > \rightarrow < nounphrase >< verbphrase >$

$< nounphrase > \rightarrow < article >< noun >$

$< verbphrase > \rightarrow < verb >< adverb >$

$< article > \rightarrow the$

$< noun > \rightarrow girl$

$< verb > \rightarrow walks$

$< adverb > \rightarrow gracefully$

3.3.2 Properties of Grammars

There are several properties of grammars that, if satisfied, make the grammar more useful for pattern recognition.

Context-free grammars have the property that all non-terminals can be replaced regardless of where they appear in the string. Restrictions are placed on the productions of context-free grammars. The right side of productions in a context-

free grammar must be a non-empty string of terminals and non-terminals [5]. The left side of the productions must be a single non-terminal.

Another type of grammar, which is a subset of the context-free grammar is the finite-state, or regular grammar. Productions in a regular grammar must be of the form: $A \rightarrow aB$ or $A \rightarrow b$ where A, B are non-terminals and a,b are terminals [5]. This form ensures that the strings generated by the grammar are finite.

As we will see in section 3.5, these properties are essential in the pattern recognition process.

3.3.3 Advanced Grammars

While grammars offer the ability to express complex patterns, sometimes additional expressivity is required. For example, the primitive extractor (detailed in section 3.4) can recognize when certain events occur, but it has no knowledge about other events that occur, either in the past or the future. Additionally, we may be only interested in a certain subset of events based on another attribute which the primitive extractor does not have access to–such as location of other objects or landmarks in the scene. In order to capture patterns that are based on complex relationships, we require a mechanism to record and examine them. This section describes two tools that allow more complex expressivity.

3.3.4 Attribute Grammars

One method to allow for more expressivity is the attribute grammar, an extension of the traditional grammar discussed in the last section. An attribute grammar specifies semantic rules for the productions which restrict the situations in which a match can occur to times when those rules are met. For example, we can place constraints on productions based on the spatial locations where the primitives occur (in a video).

We need to specify the following for the attribute grammar [9]:

- SD, the semantic domain consisting of a set of types (eg. integers or coordinates) and a set of functions oprating on the types (eg. distance, location proximity tests for coordinates)
- AD, a set of attributes associated with each symbol occuring in the productions.
- R, a set of attribute evaluation rules. These assign inherited attributes to non-terminal symbols.
- C, a set of semantic conditions associated with each production which are evaluated based on the terminals in a production.

Semantic rules are specified for each production in the form of attributes. There are two types of attributes: inherited and synthetic. Each terminal symbol gets an attribute value (synthetic) from the primitive extractor. As non-terminal symbols are expanded, they are assigned attributes (inherited) based on the at-

tributes of the terminals they contain and the rules specified in the grammar. These rules make up the set R in the list above.

The advantage of attribute grammars is that they allow us to represent more complex scenarios involving interactions between objects and properties.

3.3.4.1 Handling Concurrent Events

In video systems, multiple events may be unfolding simultaneously. If we are to capture these events, we need to make special allowances to handle them. Otherwise, the beginning of a second event will cause the first to go unnoticed.

The method, described in [9], defines an attribute called the *thread consistency id (tid)* for each symbol. This attribute serves to describe the relationships between symbols. For example, in our case of detecting a person leaving an unattended package, we want to ensure that the alert is only triggered when the person who drops the package leaves the scene–not when some other person leaves. When new symbols are seen, they can be placed in the correct event thread based on their *tid*. This will be discussed later in section 3.5.

An example production is $A \rightarrow B_0 C_1 D_N E_1$. The subscript of a symbol specifies which other symbol in the production must have a matching id. A subscript of N is a wildcard, indicating that there are no restrictions on the id of that symbol.

The left hand side of the production is index 0 and the right hand side begins with index 1. Thus, in the above production, the following relationships must hold for a match to be made:

- B.id=A.id
- C.id=B.id
- E.id=B.id

3.3.5 Stochastic Grammars

The grammars we have covered so far work well in ideal situations. However, when our input data is noisy, we can make use of statistics to pick out the patterns from the noise.

Stochastic grammars are meant to solve either of two problems:

- noisy data, when one string can be generated by multiple grammars (representing multiple pattern classes)
- some grammars may generate unwanted strings that aren't a part of the pattern class. Stochastic grammars allow us to assign these strings low probabilities.

Stochastic grammars are defined the same as traditional grammars, with changes only in the set of productions, P_S. P_S contains productions of the form $\alpha_i \stackrel{p_{ij}}{\rightarrow} \beta_{ij}$. where α_i and β_{ij} are non-terminals and non-terminals or terminals, respectively. The probabilities p_{ij} must satisfy the rule that for all $\alpha_i \stackrel{p_{ij}}{\rightarrow} \beta_{ij}$, $\sum_{j=1}^{n_i} p_{ij} = 1$. This insures that the sum of the probabilities of all productions with the same left hand side add up to 1.

As a string is generated using a series of productions, the overall probability

that a string is in the language is calculated by multiplying successive production probabilities p_{ij}.

The probabilities p_{ij} are determined using a-priori knowledge or observations of the probabilities of seeing certain strings and the above rule. Using this knowledge, we can create the stochastic grammar that will increase our likelihood of recognition in noisy data.

3.3.5.1 Generating Production Probabilities

Using a-priori knowledge of the output symbols, we can generate the production probabilities for a stochastic grammar that will generate the strings at the rates that are consistent with the evidence.

The simplest way to come up with the production probabilities is if we have a set of strings in L(G) and the probabilities that they occur. Using the strings we can create a grammar for the language. Knowing that production probabilities multiply and that all productions probabilities for a given non-terminal must add up to 1, we have a set of linear equations to solve for the production probabilities.

If the string probabilities of occurrence are not known, the production probabilities can be estimated using a sample of strings in the language and the theory of Markov processes[5]. Given a stochastic grammar with productions $A_i \rightarrow \eta_j$, and a set of strings $\overline{S_t} \subseteq L(G)$, we estimate p_{ij} as $\hat{p_{ij}} = \frac{n_{ij}}{\sum_j n_{ij}}$ where $n_{ij} = \sum_{x_k \in \overline{S_t}} f_k N_{ij}(x_k)$,$N_{ij}(x_k)$ is the number of times that the production $A_i \rightarrow \eta_j$ is used in parsing x_k and f_k is the probability of seeing string x_k. Therefore, n_{ij} is the expected number

of times that the production $A_i \rightarrow \eta_j$ is used in parsing all the sample strings in $\overline{S_t}$.

3.4 Primitive Extraction

To do syntactic pattern recognition [5] on video, we need a means of generating a string of terminal symbols (primitives) from a video signal. In our grammar, primitive symbols represent events such as "person appears", "person stops walking", and "object appears". We need a mechanism for extracting these events from the video and generating the symbols to be processed in the recognition step.

The tracker, which was detailed in section 3.2, sends the bounding boxes to the primitive extractor. There are two types of targets in the situations we are examining: humans and objects. A simple area threshold is used to distinguish between humans and objects. When the target is first seen, the appropriate "appear" symbol is generated.

Attributes of the tracked objects are measured and recorded and symbols are generated when these attributes change in pre-defined ways. For example, we track an object's velocity and position. Hysteresis thresholds are used for velocity so once a human crosses a certain high velocity threshold, we can generate the symbol "person starts". If the person later crosses a low velocity threshold, we generate the symbol "person stops".

We also use hysteresis thresholds for target area to classify targets as human or object. If an object crosses the opposite threshold, one of the target transition symbols is generated. Lastly, if gait analysis picks up periodicity changes, a drop or

pickup symbol is generated. This is described in detail in section 3.4.1.1.

A string of such primitive events can then be examined for the patterns we are interested in.

The primitive symbols used in our grammar are listed in Table 3.1. As part of the attribute grammar, attributes are stored along with each primitive symbol and are shown in the third column of the table. The attributes are:

- ID - Contains the target identification number of the object generating the symbol
- Location - Contains the spatial location in the video where the symbol was generated
- Related ID - Contains a target identification number of another object that is related to the one generating the symbol. This is used for objects to associate them with their owners. The ID attribute takes the value of the object and the Related ID attribute takes the value of the owner. In the case of an object pickup, the reverse is true for the human symbol.

When an object appears or disappears, we check the previous bounding boxes for overlap. If the box overlaps with a past box of a person, then we assign ownership of the object to the person in the related ID attribute above.

Several semantic conditions can be specified in the grammar. Semantic conditions are functions that are applied to attributes and if their conditions are met, then the production can be matched with the input string. This is an extension of the traditional grammar in which all information must be encoded in the symbols

Table 3.1: Table of Primitive Symbols and their attributes.

Target Type	**Primitive Symbol**	**Synthetic Attribute(s)**
Human	Appear	ID, Location
Human	Disappear	ID,Location
Object	Appear	ID, Location, Related ID
Object	Disappear	ID,Location, Related ID
Human	Starts	ID, Location
Human	Stops	ID, Location
Human	Human-Object Transition (Tracker Error)	ID, Location
Object	Object-Human Transition (Tracker Error)	ID, Location
Human	Drop (Gait Analysis)	ID, Location
Human	Pickup (Gait Analysis)	ID, Location

themselves. Parameters to these functions are primitive symbols or contextual objects. Contextual objects are pre-defined areas in the scene such as entrances, exits, and walkways. The conditions are:

- Equal - Check for equality of two attributes. This is used for package ownership assignment to ensure that the object dropped was left by the person we are tracking.
- Near - Checks for spatial proximity of two objects using the location attribute.
- Not Inside - Evaluates to true if one object is not inside another object. The second parameter is usually one of the pre-defined objects or areas (part of the scene).
- Inside - The opposite of the above condition.

3.4.1 Error Correction

Tracking and detection is not perfect, and errors can lead to missed or false detections in an unattended package system. The systems detailed in chapter 2 are vulnerable to these problems. Our system circumvents these errors by novel use of the attribute grammar.

Potential errors include:

- Detection - Moving shadows showing up as foreground objects. This can trick the system into classifying shadows as objects dropped by the human that is casting them.

- Detection - Dropped objects cannot always be seen by the detector. Lighting issues, small objects, or objects dropped out of the camera's view can all cause this problem. This can cause a problem with the system because the object is never a target.

- Tracking - Incorrect transitions when a package is dropped. Occurs when the human target ID number stays on the dropped object and a new target ID number is given to the person. This can cause confusion in the system because the human is appearing after the object (with a higher target ID number).

  Our methods to combat these unavoidable errors are detailed below.

3.4.1.1 Gait Analysis

Work in [15] used video to examine the human gait for patterns that indicate when someone is carrying an object. Locations at various heights are specified and the detection points at these heights in the bounding boxes are examined as the person walks across the camera's viewing area. This process is shown in Figure 3.4 for a person walking across the viewing area while carrying a briefcase in one hand. We tracked the detection at 10%, 20%, 80%, and 90% of the bounding box height. The patterns extracted from these heights, referred to as Human Gait DNA, are shown in Figure 3.5. The patterns go from top to bottom of the person as they go left to right in the diagram. The patterns on the left are from the trunk and arms, those on the right are from the legs. Asymmetry in any of the patterns, corresponding to limbs, may indicate that a person is carrying something. The

work in [15] was meant for walking trials, eg. a person walks across the screen and a classification is made.

(a) Bounding Box (b) Detection

Figure 3.4: Example Gait Analysis Trial.

Figure 3.5: Example Gait DNA Patterns.

In order to catch a person leaving an unattended package behind some occlusion in a real-time video system where trials do not occur, we used a similar idea and created a real-time algorithm for detecting the transition between carrying and not-carrying that indicates a package has been dropped. The first consideration is that this algorithm only works when the person is walking within 45 degrees of a perpendicular trajectory: it depends highly on seeing the side profile of the person. We compute the angle at which the person is walking and only proceed if they are walking within the acceptable range. This would be solved in a real system with multiple cameras providing more views.

In a package drop scenario in the ideal case the person's arm swing will remain

constant while holding the package and transition into a periodic motion after the package is dropped. To track this in a real-time system, we divide the blob in half and track the width of the torso as the person moves throughout the scene. Using a fixed buffer of past bounding box widths, we compute a variance on this data at each frame. We track the changes in this trunk width variance throughout the video. Ideally we would see a variance transition as shown in figure 3.6. The trunk widths have a low variance while the person is holding the object and a gradual transition to a higher variance occurs after they drop the object. Note that the width of the transition is dependent on the length of the video buffer used for storing past trunk width values. If the person bends down to drop the object, we expect to see a spike in the variance while they drop the object, and a gradual return to a steady variance higher than that of carrying while they walk away unencumbered. In reality, the data we get appears as follows. In Figure 3.7 we show the trunk widths as the person goes across the viewing area and drops a briefcase around frame 80. In Figure 3.8 we show how the variances of this sequence change as computed over a fixed-frame window. In order to detect the peak seen in the figure, we perform a linear regression on the variance sequence and threshold the slope. If the slope is over a certain value we generate a primitive event for package drop. This is shown in Figure 3.9.

3.4.1.2 Tracking Errors

As mentioned previously, incorrect transitions in the tracker may occur when a package is dropped. When the person drops the object, the human target ID

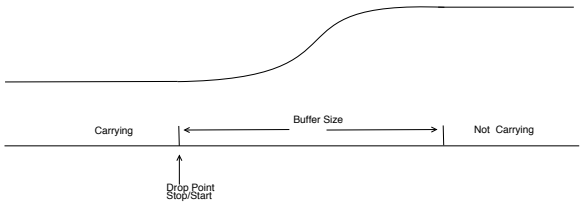


Figure 3.6: Ideal transition from carrying to not carrying after dropping an object. The drop point is shown in the figure.

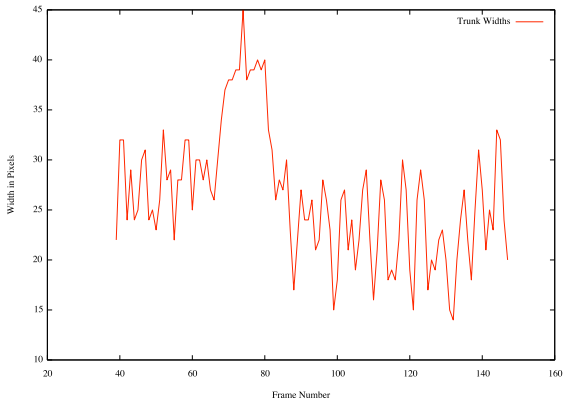


Figure 3.7: Trunk widths from a video with a package drop at frame 80.

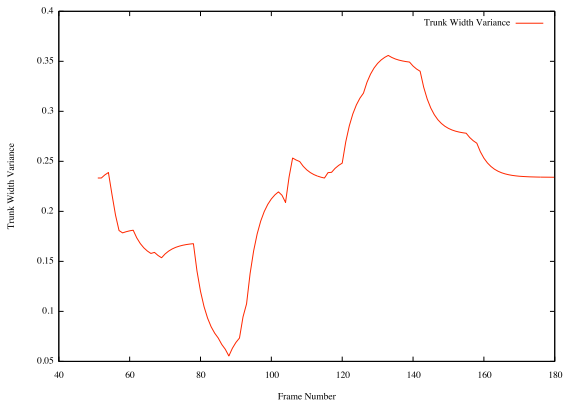


Figure 3.8: Trunk width variances across a video sequence containing a package drop at frame 80.

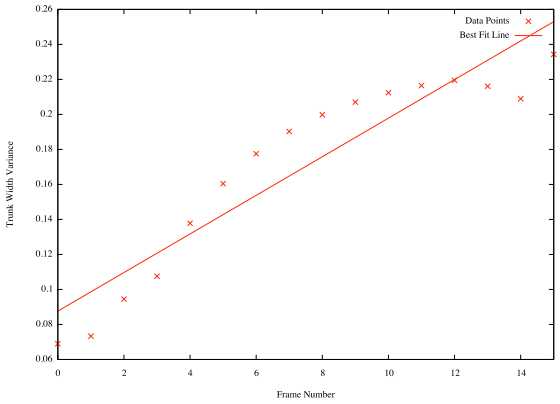
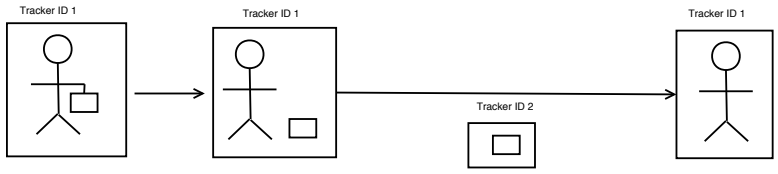
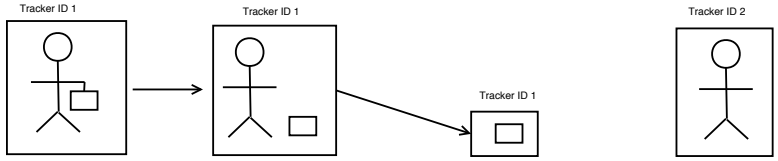


Figure 3.9: Trunk width variance transition during a package drop and best-fit line.

number stays on the dropped object and a new target ID number is given to the person. This can cause confusion in the system because the human is appearing after the object (with a higher target ID number). This situation is demonstrated in Figure 3.10.



(a) Normal Tracking



(b) Tracker ID 1 stays on the object, not the person

Figure 3.10: Showing the correct tracking and the tracking error when the bounding box stays on the object and not the person.

In order to combat this, we continuously classify the targets as human or object using hysteresis and allow for transitions in case the tracker has swapped two

targets. We have also encoded this into a rule in the attribute grammar to catch situations in which the tracker swaps two targets after a package drop. The primitive manager catches the swap and issues the correct classifications. This swap is also advantageous because it is a good indication that a package drop has occurred.

Below we show the rule that allows this transition to be recognized. We can see that a person must appear, followed by a person to object transition and the birth of a new object and person nearby. The results of this situation are seen in section 4.8.

$$PACKAGEDROP \rightarrow T\_PERAPP_0\ PERSTART_1\ T\_PEROBJ_1\ T\_OBJAPP_1\ T\_PERAPP_N\ PERSTART_5\ T\_DISAPPEAR_5$$

$$X_0.ID := X_1.ID, Near(X3.LOC, X5.LOC)$$

3.4.1.3 Shadows

Another tracking problem was shadows that are similar in size to objects. In order to filter out moving shadows, we applied a velocity threshold so that for a target to be considered an object, it must have a velocity below a certain amount.

Some shadows, particularly in indoor environments, appear with a low velocity when humans are stationary. These will not be filtered out by the above method. In order to remove these situations, we use the capabilities of stochastic attribute grammars to assign low probabilities to these situations.

Below we show the rule that allows this situation to be recognized. It is similar to the package drop rules except the object must disappear before the person and it must disappear inside the field of view. If this happens, we will assign it a low

probability and therefore not consider it a true package drop. If the final probability of an event is below a certain threshold, we consider it "unlikely". An example of this can be seen in section 4.5.

$PACKAGEDROP \rightarrow T\_PERAPP_0\ PERSTART_1\ T\_OBJAPP_N\ T\_DISAPPEAR_3\ T\_DISAPPEAR_1$

$X_0.ID := X_3.ID, Equal(X1.ID, X3.RID), NotInside(X5.LOC, FOV)$

3.4.1.4 Proposed System

The stochastic attribute grammar that we created for the unattended package scenario is presented in Table 3.2. Productions are shown along with their probabilities, attributes and required conditions. Inherited attributes are shown in the third column along with semantic conditions that must be met for the rule to be a match.

Using the grammar we detect two events, package drop and package pickup. More effort is devoted to package drop detection, but the basic functionality is there for detecting package pickup. The pickup event is recognized if an object appears, a person appears, and the object disappears with the related ID of the person.

There are several intermediate non-terminal states that reflect the status of the people in the scene which are used in the formulation of higher-level events. PERSTOP and PERSTART are finite-states which represent the current state of a person. If a person starts, stops, and starts again, they can still be considered in the PERSTART state due to its formulation below. With PERSTART we assign

the location of the symbol to be the most recent starting point. This is used in subsequent productions. PERDROP is also a non-terminal symbol that is similar to the PERSTART state. It allows a drop to occur if a person's gait flags a drop, pickup, and another drop.

There are five different productions that specify the package drop scenario. We will now describe the significance of each.

- The first production covers the simplest case in which a person appears, a package dropped by that person appears, and the person then disappears outside the field of view of the camera.

- The second production is the same as the first except the package disappears before the owner. This can be explained by one of two things: We have been fooled by a shadow object or the person who dropped the object picked it up and concealed it. Both cases are not package drops so we assign this production a low probability.

- The next two productions cover the situations in which the object cannot be segmented and gait analysis takes over. The first case applies the restriction that the drop should occur near the person's last stopping point and they must disappear outside the field of view. If these conditions are not met, the next production will be satisfied and we assign that a slightly lower probability.

- The last is the most complex of the productions. This covers the case when the tracker swaps target IDs on a package drop. This allows for the reclassification

by the primitive extractor and requires the misclassification to occur near the creation of the new person to assign the proper package ownership.

Table 3.2: Attribute Grammar for Unattended Package Detection.

Production	**P**	**Attributes**
$S \rightarrow PACKAGEDROP_N \mid PICKUP_N$		
$PERSTOP \rightarrow T\_PERSTOP_0$	1.0	$X_0.loc := X_1.loc$
$PERSTART \rightarrow T\_PERSTART_0\ T\_PERSTOP_1\ PERSTART_1$	1.0	$X_0.loc := X_3.loc$
$PERSTART \rightarrow T\_PERSTART_0$	1.0	$X_0.loc := X_1.loc$
$PERDROP \rightarrow T\_DROP_0\ T\_PICKUP_1\ PERDROP_1$	1.0	$X_0.loc := X_1.loc$
$PICKUP \rightarrow T\_OBJAPP_0\ T\_PERAPP_N\ T\_DISAPPEAR_1$	1.0	$Equal(X2.ID, X3.RID)$
$PACKAGEDROP \rightarrow T\_PERAPP_0\ PERSTART_1\ T\_OBJAPP_N\ T\_DISAPPEAR_1$	.5	$X_0.ID := X_3.ID, Equal(X1.ID, X3.RID)$
$PACKAGEDROP \rightarrow T\_PERAPP_0\ PERSTART_1\ T\_OBJAPP_N\ T\_DISAPPEAR_3\ T\_DISAPPEAR_1$	.05	$X_0.ID := X_3.ID, Equal(X1.ID, X3.RID), NotInside(X5.LOC, FOV)$
$PACKAGEDROP \rightarrow T\_PERAPP_0\ PERSTART_1\ PERDROP_1\ T\_DISAPPEAR_1$	.15	$X_0.ID := X_1.ID, Near(X2.LOC, X3.LOC), NotInside(X4.LOC, FOV)$
$PACKAGEDROP \rightarrow T\_PERAPP_0\ T\_PERSTART_1\ PERDROP_1\ T\_DISAPPEAR_1$	.15	$X_0.ID := X_1.ID$
$PACKAGEDROP \rightarrow T\_PERAPP_0\ PERSTART_1\ T\_PEROBJ_1\ T\_OBJAPP_1\ T\_PERAPP_N\ PERSTART_5\ T\_DISAPPEAR_5$	.15	$X_0.ID := X_1.ID, Near(X3.LOC, X5.LOC)$

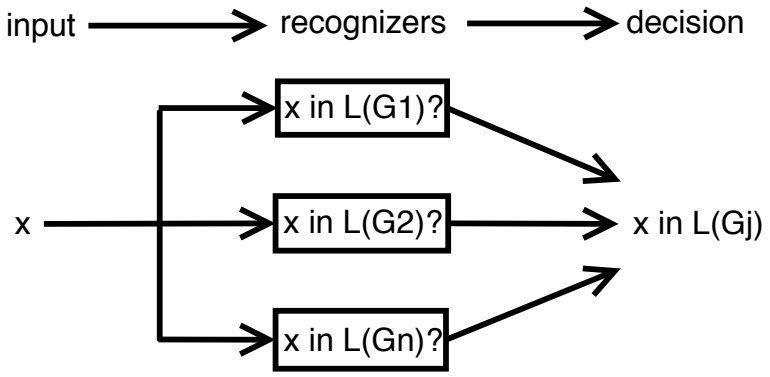


Figure 3.11: Block diagram of the recognition process.

3.5 Pattern Recognition

The first two sections covered pattern specification and primitive extraction. After the pattern has been specified and we have assembled a string of primitives, we can search for patterns using the methods detailed in this section. The most common method for syntactic pattern recognition is to construct a grammar for each pattern class and check the input string against each one. If the string is generated by the grammar, it is a member of the pattern class. In the case of a stochastic grammar, we select the grammar with highest probability, as described in section 3.5.5.

A block diagram of this process is shown in Figure 3.11.

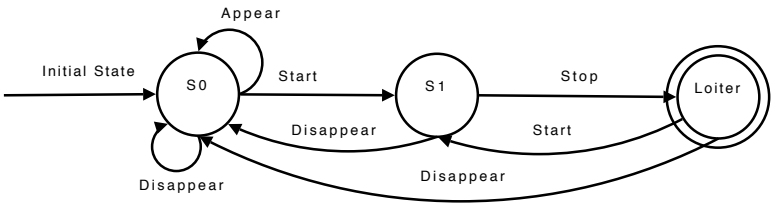


Figure 3.12: Discrete Finite Accepter for Loitering Scenario.

3.5.1 Previous Methods

3.5.1.1 Discrete Finite Accepters

In the case of regular or finite languages, discrete finite accepters (DFAs) may be used to test language-membership of a string. DFAs, formally (Q, Σ, S, M, F), consist of a finite number of states, an input alphabet, a start state, a set of state-transitions, and a set of accepted states (a subset of all states) respectively. Starting at the first symbol in the input string and beginning at the start-state, we trace through the automata. If at the end of the string we are in an accepted state, then the string is a member of the language.

By definition, A language L is called *regular* if and only if there exists some deterministic finite accepter M such that L=L(M) [11].

An example DFA is shown in Figure 3.12 that examines strings of events used to determine if a person is loitering. Input symbols are: Appear, Stop, Start, Disappear. The state with double circles represents the accepted "loitering" state.

3.5.1.2 Stochastic Automata

If we have a stochastic finite-state grammar we can create a stochastic finite-state automata that accepts strings that are members of the language [5].

We define the stochastic automata as follows: $A_s = (\Sigma, Q, M, \Pi_0, F)$ where Σ is the set of input symbols, Q is the set of internal states, M is the set of transition matrices (one for each input in Σ), Π_0 is the initial state distribution, $F \subseteq Q$ is the set of final states.

Given a stochastic grammar $G_s = (V_N, V_T, P_S, S)$ defined previously, we can define the elements of the stochastic automata A_s as follows:

- $\Sigma = V_T$
- $Q = V_N \cup T, R$ where T and R are terminating and rejection states respectively.
- $\Pi_{0,i} = \begin{cases} 1 & : \quad i = S \\ 0 & : \quad otherwise \end{cases}$
- $F = T$
- M is determined using the stochastic productions.

3.5.2 Parsing

DFAs are a sufficient solution when using a regular language without concurrent events. However, when examining more complex grammars like attribute grammars, or when events occur concurrently, DFAs are not sufficient. In these cases, we need a fast method to determine if the input string can be generated by

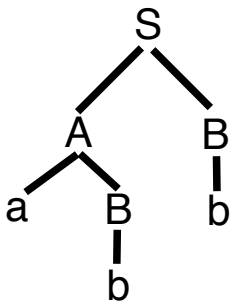


Figure 3.13: Derivation tree for a sentence in the grammar: $V_T = \{a, b\}, V_N = \{A, B\}, P = \{A \rightarrow aB, A \rightarrow a, B \rightarrow b\}$.

the pattern grammar. This process is called parsing, and it will also determine the sequence of primitives that was used to derive the input string or the "derivation tree" – an example of which is seen in Figure 3.13. There are multiple parsing algorithms with varying runtime-efficiencies that are detailed below.

3.5.2.1 Exhaustive-Search Parsing

The first, and most intuitive parsing algorithm is the top-down approach. This approach consists of an exhaustive-search derivation of the input string beginning with the start symbol. Each derivation possibility is traced, and paths are eliminated when they don't match the input string.

This method, although simple to implement, has several drawbacks. First, because it is an exhaustive search, it is computationally inefficient and is not suitable for real-time parsing. Secondly, the method is not guaranteed to terminate if the input string isn't in the language. Clearly, other methods need to be examined.

Equally exhuastive is the bottom-up approach in which derivation of the start symbol is attempted beginning with the input string. Like the top-down approach, this is also computationally inefficient.

3.5.2.2 Earley Parser

Earley proposed a parsing algorithm [3] that could be used to parse arbitrary context-free grammars with a runtime of $O(n^3)$ and $O(n^2)$ if the grammar is unambiguous [5].

Beginning with the input string, the algorithm maintains a dot location throughout the parsing process that represents the current status in the input string. At each iteration, *state sets* are generated which contain possible productions (states) that are being considered for the derivation of the input string.

A state is expressed as: $i :_k X(0) \rightarrow X(1)...X(j) \circ X(j+1)...X(n)$ where i represents the index of the state in the current set, k represents the state set from which the non-terminal X(0) was generated. The symbols to the left of the dot, X(1) through X(j), are symbols that have already been seen in the input string, and the symbols to the right of the dot are expected if the state is to match the input string.

The algorithm is divided into three stages: prediction, scanning, and completion. The states of the current state set $S(i)$ along with the next input symbol a are examined. X and Y are nonterminal symbols and α, β, γ are any sequence of symbols.

- Prediction: For each state of the form $i :_j X \rightarrow \alpha \circ Y\beta$, add $i :_i Y \rightarrow \circ\gamma$ to state set $S(i)$

- Scanning: For every state of the form $i :_j X \rightarrow \alpha \circ a\beta$ add $i+1 :_j X \rightarrow \alpha a \circ \beta$ to $S(i+1)$.

- Completion: For every state in $S(i)$ of the form $i :_j X \rightarrow \gamma\circ$ find states in $S(j)$ of the form $j :_l Y \rightarrow \alpha \circ X\beta$ and add $i :_l Y \rightarrow \alpha X \circ \beta$ to $S(i)$.

After these steps have been performed on all the state sets, if in any state set we have the state $P \rightarrow S\circ$ where $P \rightarrow S$ is a start state of the language, the sentence is generated by the grammar.

3.5.3 Parsing Attribute Grammars

To parse attribute grammars, we can modify the above algorithm as put forth in [9]. The main change is in storing and examining symbol's attributes as a part of the input string.

The changes for each step of the algorithm are, using the notation from the previous section:

- Prediction: Evaluate the inherited attributes $\{a_m\}$ for Y and assign them to Y in the added state.

- Scanning: Assign all attributes given with the next input symbol a to a in the added state.

- Completion: First check for all conditions on the attributes of γ. If these are satisfied, evaluate all synthesized attributes $\{a_m\}$. In the added state, assign $\{a_m\}$ to X.

3.5.4 Parsing Concurrent Events

To parse concurrent events, we modify the Earley algorithm as in [9]. As discussed in Section 3.3.4.1, the *thread consistency id (tid)* is used to separate event threads.

The following steps are taken, using the same notation as in the previous two sections, in addition to the above methods:

- Prediction: Check the subscript of Y, d. If it is a wildcard, assign Y a wildcard *tid* in the added state. Otherwise, assign Y a *tid* equal to the *tid* of the d^{th} symbol in the string. By the way we define these subscripts, the d^{th} symbol will be to the left of the dot.

- Scanning: In the scanning phase, the state is either scanned, skipped, or ignored. Scanning follows the same procedure as above, however if the state is skipped, we simply copy it directly to the next state set. If the state is ignored, we do not advance it further. One of the three actions on states is taken based on the following conditions:

  - If the input state is a wildcard, or it refers to an index that is a wildcard, we do one of two things. If the input symbol a matches the symbol after the dot, we skip and scan. If not, we simply skip. The skip occurs in

both conditions because in both cases we are dealing with a wildcard and do not know if the needed symbol will come later.

  - If the input symbol a's id matches with *tid* of the previous symbol, do a scan. This is an exact match.
  - Otherwise, ignore the state. This is not a match for the current event thread.

- Completion: The *tid* of the last symbol is passed to the completed nonterminal symbol.

3.5.5 Parsing Stochastic Grammars

The way we parse stochastic grammars depends on how the grammars are set up to do pattern recognition and what we'd like to accomplish. What is common in all these methods, however, is that the production probabilities multiply through the derivation of a string.

The methods as follows can be divided into two categories: those that exploit the production probabilities to reduce the parsing complexity and those that do not.

The methods that do not seek to reduce runtime simply seek to generate the probability that the input sentence is in the language generated by the grammar. An Earley parser that is modified simply to factor and record these probabilities is the simplest stochastic parser. When a string is parsed, its probability of membership in the grammar is immediately available. If there are multiple grammars representing multiple pattern classes, the simplest method is to parse the input string in each

grammar using the modified Earley parser and select the grammar with the highest probability. This is known as the stochastic maximum-likelihood recognizer.

However, in our system, because we used low probabilities to filter out undesirable events, we select the grammar with such a probability as the current pattern. In the absence of this negative likelihood, we select the maximum probability as in the maximum-likelihood recognizer. When these events occur, such as a shadow being tracked as an object which then disappears in the field of view, more than one production may match but this *negative* production should take precedence.

Because multiple productions with the same right side may exist in stochastic grammars, parsing becomes a non-determinstic task. Rather than take both paths, like the previous algorithms propose, the probability information can be used to construct the *syntax-controlling probability (scp)* for each production. The scp is a representation of the probability that the production is the correct one to use and is based on the expected number of times the production is used in all strings of the language. By choosing the path with the highest scp, we can reduce the number of steps needed in the parse and remain consistent with the probabilities of the language.

The scp, p_{ji}, is computed as follows:

$$p_{ji} = \frac{\sum_{x \in L} p(x) N_{ji}(x)}{\sum_{A_j \to \eta_i \in \Gamma_{\eta_i}} \sum_{x \in L} p(x) N_{ji}(x)}$$

where $N_{ji}(x)$ is the number of times that $A_j \to \eta_i$ is used in the derivation of the string x. Γ_{η_i} is the set of all productions with the right side of η_i.

3.5.6 Alarm Conditions

We recognize two activities in this system: package drop and package pickup. When an event is recognized, we are given the production probability along with a derivation tree of all the primitive symbols that make up the event. This allows us to associate the owner with the event, which in a real system could be used to locate the perpetrator.

In a real system, these two events would be monitored for each package and an alarm would be triggered when package drop is parsed by the grammar and would go off only if the same object is later involved in a package pickup. When the alarm goes off an image of the package owner can be displayed for security guards.

Chapter 4

Experimental Results

4.1 Overview

In this chapter we present experimental results from the attribute grammar system described in Chapter 3. Several scenarios are shown to emphasize the various strengths of the stochastic attribute grammar system. These coincide with the five productions for the package drop scenario and the one pickup scenario.

4.2 System Implementation

We developed this system on a standard PC platform using OpenCV and C++. We can feed videos to the system and get output streams showing the primitive events generated and the higher level events recognized.

When a primitive symbol is generated, the video pauses briefly and displays the event and a box around the target. When a high level event is recognized, the video pauses showing the event, its probability and all the primitive symbols that make up the event.

Recognition of the high level event does not interrupt detection of other events as multiple concurrent event threads are maintained.

4.3 Experiments

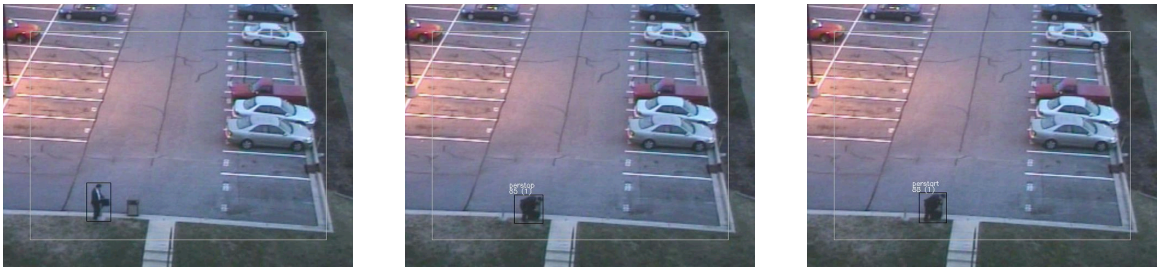
For the experiments we chose several scenarios that demonstrate the capabilities of our attribute grammar system. The data in sets 1,3, and 4 were recorded from surveillance cameras at the Army Research Laboratory. Data from sets 2 and 5 come from the PETS 2006 conference dataset. The descriptions of each data set are provided in the following sections along with still image output at key frames.

4.4 Data Set 1

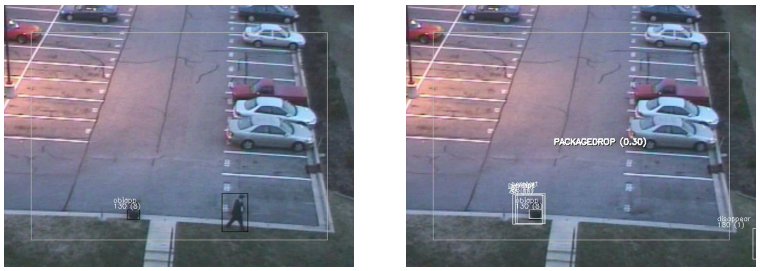
The first production covers the simplest case in which a person appears, a package appears which was dropped by that person, and the person then disappears outside the field of view of the camera. The results are shown in Figure 4.1.

4.5 Data Set 2

The second production is the same as the first except the package disappears before the owner. This can be explained by one of two things: We have been fooled by a shadow object or the person who dropped the object picked it up and concealed it. Both cases are not package drops so we assign this production a low probability. In this data set (Figure 4.2) we see a group of people that enter the scene, create a shadow which disappears, and then exit the scene. This is recognized by the grammar and assigned a low probability, indicating that it does not constitute a true unattended package drop situation.



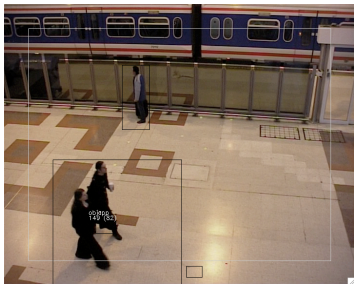
(a) Person appears (b) Person stops (c) Person starts



(d) Object detected (e) Person disappears

Figure 4.1: Test 1: Simple Package Drop Scenario.

(a) Person appears



(b) Shadow appears as an object

(c) Shadow disappears

(d) Person disappears; negative production recognized

Figure 4.2: Test 2: Showing a shadow which appears like an object and is removed from consideration by the negative logic in the grammar.

4.6 Data Set 3

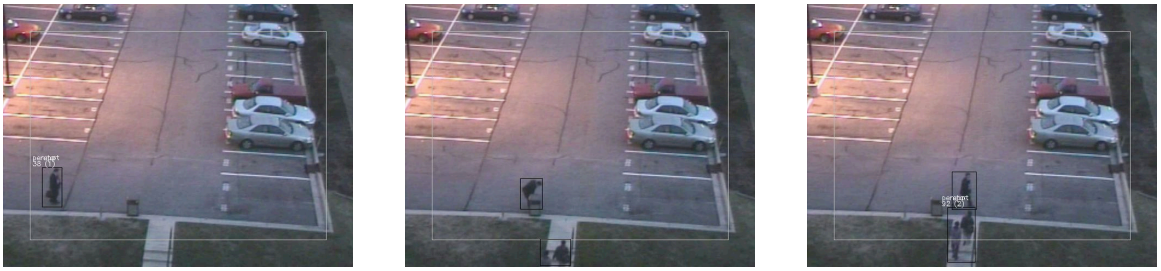
The next two productions cover the situation in which the object cannot be seen by the camera and gait analysis takes over. The first case applies the restriction that the drop should occur near the person's last stopping point and they must disappear outside the field of view. If these conditions are not met, the next production will be satisfied and we assign that a slightly lower probability. In this data set (Figure 4.3) we see a person dropping a package behind a trash can and the system recognizes the event.

4.7 Data Set 4

This next data set shows the shadow pre-processing in action (Figure 4.4). A person appears with a heavy shadow in a bright environment. The shadow is separated from the person and removed from the event thread of interest. The unattended package event is then successfully detected.

4.8 Data Set 5

The last is the most complex of the productions. This covers the case when the tracker swaps target IDs on a package drop. This allows for the reclassification by the primitive extractor and requires the misclassification to occur near the creation of the new person to assign the proper package ownership. In this test set, shown in figure 4.5, the unattended package event is successfully detected and the resulting string is made up of 37 primitive symbols.



(a) Person appears (b) Person drops bag (c) Person walks away

(d) Drop transition detected (e) Person disappears

Figure 4.3: Test 3: Person drops package behind a trash can.

(a) Person appears with shadow

(b) Shadow and person are separated

(c) Shadow disappears

(d) Object appears

(e) Person disappears; drop detected

Figure 4.4: Test 4: Showing shadow detection and removal.

(a) Person appears

(b) Person drops bag

(c) Person attends bag

(d) Person leaves bag; tracking error detected

(e) Object appears; tracking error corrected

(f) Person appears

(g) Package drop detected

Figure 4.5: Test 5: Complex scenario involving tracking errors which are corrected by the attribute grammar.

4.9 Data Set 6

The pickup event is recognized if we see an object appear, a person appear, and the object disappear with the related ID of the person. In this test set, shown in figure 4.6, we see a person leave a package, and then another group of people congregate around it. This shows how the pickup event can be used to indicate change in package ownership.

4.10 Conclusion

In conclusion, we have presented a system based on a syntactic representation of events for recognizing the unattended package event in video. The system can handle events that are corrupted by tracking and detection errors including occlusions, mis-tracked targets, and shadows. We model the event syntactically using a stochastic attribute grammar. Targets are tracked and primitive event symbols are generated which are then used in a multi-threaded recognition system based on the Earley parser. When an event is detected, we are given the probability of its membership in the language along with the derivation tree used. This allows us to judge the likelihood of the event, and also associate all targets that were involved in the event's unfolding.

The key advantage of the stochastic attribute grammar is in its expressivity. As we have demonstrated in this thesis, the expressivity of the attribute grammar can be used to combat many common problems in computer vision systems. If other events need to be detected, one can simply modify the rules of the grammar. The

(a) Object appears

(b) Object occluded

(c) Person appears

(d) Object changes ID

(e) Pickup recognized

Figure 4.6: Test 6: Showing ownership changes detected by the pickup event.

runtime of the system is another advantage. We can parse and maintain multiple event threads in real-time.

4.11 Future Work

There is a lot of work to do in this area: not only in improving the methods described in this thesis, but also extending them to other areas.

The syntactic approach can be used in real-time surveillance systems for monitoring public areas where we are interested in locating unattended packages and their owners. The error conditions corrected for in this thesis are likely to be concerns in a real-world system, so our framework is practical.

We also see this work extended to develop better trackers based on an ontological representation of targets and events. Using scene knowledge we can set up rules for what can and cannot physically happen in the scene and feed this information to the tracker to prevent occlusions, target swapping and other errors. This makes sense based on the human vision system in which we use our knowledge of what is going on and what is physically possible to model current events.

Bibliography

[1] Michael D. Beynon, Daniel J. Van Hook, Michael Seibert, and Alan Peacock. Detecting abandoned packages in a multi-camera video surveillance system. Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.

[2] Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. John Wiley & Sons, New York, 2001.

[3] J. Earley. An efficient context-free parsing algorithm. *ACM*, 13, 1970.

[4] King Sun Fu. *Syntactic Methods in Pattern Recognition*. Academic Press, New York, 1974.

[5] King Sun Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1982.

[6] D. Gibbons, G. Newsam, and M. Brooks. Detecting suspicious background changes in video surveillance of busy scenes. pages 22–26. Proceedings of Third IEEE Workshop on Applications of Computer Vision, 1996.

[7] Ismail Haritaoglu, Ross Cutler, David Harwood, and Larry S. Davis. Backpack: Detection of people carrying objects using silhouettes. International Conference on Computer Vision (ICCV), 1999.

[8] Thanarat Horprasert, David Harwood, and Larry S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. International Conference on Computer Vision (ICCV) Frame-rate Workshop, 1999.

[9] Seong-Wook Joo and Rama Chellappa. An automated visual event recognition system based on syntactic models of events. *Pending Publication*, 2007.

[10] David C. Lay. *Linear Algebra and its Applications*. Addison Wesley, New York, 2003.

[11] Peter Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett, 3rd edition, 2001.

[12] Jesús Martínez-del Rincón, J. Elías Herrero-Jaraba, Jorge Raúl Gómez, and Carlos Orrite-Uruñuela. Automatic left luggage detection and tracking using multi-camera UKF. IEEE International Workshop on PETS, 2006.

[13] Darnell Moore and Irfan Essa. Recognizing multitasked activities from video using stochastic context-free grammar. Proceedings of Eighteenth National Conference on Artificial Intelligence, 2002.

[14] A. Prati, I. Mikic, C. Grana, and M. Trivedi. Shadow detection algorithms for traffic flow analysis: A comparative study, 2001.

[15] Yang Ran, Rama Chellappa, and Qinfen Zheng. Finding gait in space and time. volume 4, pages 586–589. 18th International Conference on Pattern Recognition (ICPR), 2006.

[16] Robert Schalkoff. *Pattern Recognition: Statistical, Structural, and Neural Approaches*. John Wiley & Sons, New York, 1992.

[17] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, Berlin Hiedelberg, 2003.

[18] Kevin Smith, Pedro Quelhas, and Daniel Gatica-Perez. Detecting abandoned luggage items in a public space. IEEE International Workshop on PETS, 2006.

[19] Jia Wang and Wei-Tsang Ooi. Detecting static objects in busy scenes. Technical Report TR99-1730, 8, 1999.