School of Computing

Incremental Learning for Querying Multimodal Symbolic Data

Mihai Lazarescu Mugurel

This thesis is presented as part of the requirements for

the award of the Degree of Doctor of Philosophy

of the

Curtin University of Technology

February, 2000

Abstract

In this thesis we present an incremental learning algorithm for learning and classifying the pattern of movement of multiple objects in a dynamic scene. The method that we describe is based on symbolic representations of the patterns. The typical representation has a spatial component that describes the relationships of the objects and a temporal component that describes the ordering of the actions of the objects in the scene. The incremental learning algorithm (ILF) uses evidence based forgetting, generates compact concept structures and can track concept drift.

We also present two novel algorithms that combine incremental learning and image analysis. The first algorithm is used in an American Football application and shows how natural language parsing can be combined with image processing and expert background knowledge to address the difficult problem of classifying and learning American Football plays. We present in detail the model developed to represent American Football plays, the parser used to process the transcript of the American Football commentary and the algorithms developed to label the players and classify the queries. The second algorithm is used in a cricket application. It combines incremental machine learning and camera motion estimation to classify and learn common cricket shots. We describe the method used to extract and convert the the camera motion parameter values to symbolic form and the processing involved in learning the shots.

Finally, we explore the issues that arise from combining incremental learning with incremental recognition. Two methods that combine incremental recognition and incremental learning are presented along with a comparison between the two algorithms.

Acknowledgement

I would like to thank the many people who have helped me throughout my doctoral studies. My warmest thanks go to my supervisor Professor Svetha Venkatesh. She always encouraged me and kept me focused on the objectives of the thesis. Many, many thanks go to my co-supervisor Professor Geoff West who gave me fantastic support during my studies. I would like to thank Professor Terry Caelli for his help and advice during the first year of my PhD. Without their guidance, *divine* wisdom and support I doubt I would have been able to complete the work for the thesis. I would also like to thank Adrian Pearce for providing me with priceless information about American Football and for the discussions on machine learning. Thanks also go to Kim Shearer and Ak Tuah for their expertise on Latex and to the Computing Science staff for their help during my studies.

Thank you all.

Finally I would like to thank my family for doing the both the possible and the impossible to make sure that I can concentrate on my PhD.

Mihai

Preface

The original work presented in chapters 3, 4, 5, 6, 7 and 8 of this thesis has previously been published in various forms including refereed conference papers and technical reports.

Chapters 4, 5 and 6 present research on the American Football application and has been published in Proceedings of IAPR '98 [71], Proceedings of ICAPR'98 [72], Proceedings of ICMCS'99 [74] and in Technical Report 2, Curtin University [67].

Chapters 3, 6 and 7 present research on the incremental learning algorithm and has been published in Proceedings of ICML-99 Workshop on Machine Learning in Computer Vision [69], Proceedings of the Bad Homeburg Workshop on Computer Vision [70] and will also appear in a special issue of Computer Vision. Work described in these two chapters has also been submitted (under review) to the Journal of Machine Learning.

The work presented in chapters 4, 5 and 8 has been accepted by AI99 [68] and is under review for DICTA99 [73].

Contents

List of Figures

Chapter 1

Introduction

For over three decades researchers in computer vision have been investigating image processing and image understanding. The focus of the research has been mainly on single frame analysis due to the complex and time consuming processing involved. Continuing advances in computer hardware have reduced previous limitations in image processing and allowed research on entire video sequences. One computer vision field that involves processing of video sequences is video indexing in which the goal is to develop systems that can automatically understand and index video data. The problem with video processing is that it is, in most cases, processed as a simple sequence of images and so far few applications use context knowledge to help in the video data analysis and other sources. Furthermore once the video data is processed, no attempt is made to learn from it. One of the greatest challenges in video indexing is to develop systems that can automatically annotate footage of complex situations such as team sports, for example American Football. This is a very complex problem as it involves understanding the movements of multiple objects in a dynamic scene in situations where the objects can be occluded and have erratic movement. Though numerous methods have been developed, none of the methods can accomplish this task consistently.

To address the problem of video indexing of footage from sport it is necessary to use all the information available from the video data. For example the information extracted from the low level image processing, the clues extracted from the audio data corresponding to the video segment and the context information. Finally, it is desirable to learn from the current data to improve the recognition ability of the system. Using this approach, the work described in this thesis attempts to demonstrate the advantages of combining data from several sources of information and machine learning to index video data.

1.1 Aims and Approaches

This thesis has four major aims. The first aim is to develop algorithms that can classify and learn complex spatio-temporal patterns. The second aim is to show that a complex problem such as classifying American Football plays can be solved by combining image processing, text processing and learning. The third aim is to demonstrate that camera parameter estimation and learning can be used to recognise and learn cricket shots. Finally, the fourth aim of this thesis is to investigate combining incremental learning and incremental recognition.

The work on incremental learning presented in this thesis addresses three important issues in machine learning: memory size, forgetting and concept drift. In this thesis we do not treat the three issues independently (as is the general rule) but rather in combination and the reason for selecting these particular issues is that the patterns that we attempt to recognise and learn are complex and evolve over time. The aim is to develop an algorithm that generates compact concept hierarchies, uses forgetting to keep the concepts consistent with the stream of information and tracks concept drift making it suitable for real world applications. To this end, the algorithm was evaluated in three applications: American Football, cricket and fighter combat.

The work on American Football combines image analysis, natural language processing and learning. The application does not treat video as a simple collection of images. Instead it uses background knowledge to select the low-level processing required and complements the information extracted from the video with clues obtained from the transcript of the audio track associated with the video segment. The transcript of the commentary is essentially an expert description of the dynamic scene in the video segment and therefore a great deal of information can be extracted through simple text parsing. The aim of using a combination of video and text information is to reduce the complexity of the play classification task and to prevent the system being dependent on only one source of information. Incremental learning is also used to learn from previous examples in order to improve the classification performance of the system and to handle cases involving data that has not been encountered previously. To demonstrate the benefits of combining image analysis, natural language processing and learning, both simulation and real-world data are used.

The work on cricket combines image analysis and learning to classify cricket shots. The image analysis in this application uses camera motion estimation and does not involve any background knowledge. Learning is again used to improve the classification performance of the system and to better handle cases involving shots that have not been encountered previously.

The fourth aim of this thesis is to investigate combining incremental learning and

incremental recognition. This is necessary to deal with patterns and situations that cannot be isolated form the data stream and global features determined.

Two methods of incremental learning and incremental recognition are presented. Each method uses a different procedure to update the concepts in the hierarchy. The aim is to resolve the issues that need to be considered when attempting to dynamically update concepts when using these two methods.

1.2 Contributions

The major contributions of this thesis include the development of an incremental learning algorithm, the production of novel algorithms for image processing, text processing and machine learning as applied to video indexing and for the recognition and learning of the pattern of movement of multiple labelled objects in a dynamic scene.

First, an incremental learning algorithm is proposed to recognise and learn the pattern of movement of multiple labelled objects in a dynamic scene. The algorithm addresses three important issues in incremental learning, specifically, memory size, forgetting and concept drift. The algorithm generates compact conceptual hierarchies that are particularly useful in data intensive domains. Forgetting is an important part of the algorithm as it is used to keep the concept structure compact by removing irrelevant or out of date information. The incremental learning algorithm also uses forgetting to help in tracking concept drift by keeping existing concepts consistent with the incoming stream of data.

Second, it is shown that relatively simple text and novel image processing can be used to classify American Football plays. The text processing involves parsing for clues about the plays and can be used to determine the type of play, the action (such as pass, run or punt) of the play, the players and their positions (such as quarterback or tail-back) involved and whether the play was successful or not. Novel image processing algorithms are used to determine the player formation (initial setup) and the movement of the players over the duration of the play.

Third, a method that uses camera motion estimation and incremental learning to classify cricket shots is presented. The image analysis used in the application does not involve tracking any specific objects in the shots (such as the players or the ball) but rather uses camera motion and reasoning to determine the most probable cricket shot in the video segment.

The final contribution is an investigation into and novel algorithms for combining incremental learning and incremental recognition. We describe two algorithms that use incremental learning and incremental recognition. The first algorithm

updates the concepts in the hierarchy dynamically (as the data is analysed). The second algorithm updates the concepts only after all data is available.

1.3 Outline of the thesis

This thesis is organised as follows:

Chapter 2 presents a survey of the machine learning literature relevant to the work presented in this thesis.

Chapter 3 presents the incremental learning algorithm — ILF (incremental learning with forgetting). The algorithm generates compact concept structures, uses forgetting to keep known concepts consistent with the observed data and can track concept drift.

Chapter 4 describes in detail the knowledge structure used to represent American Football plays. The play model combines expert knowledge, domain knowledge, spatial knowledge and temporal knowledge.

Chapter 5 describes the procedure used to label players in American Football plays. The labelling method uses information extracted from video combined with context knowledge to determine the label of the players in the initial setup.

Chapter 6 presents the algorithm used to answer queries about American Football plays. The algorithm uses clues from the text of the transcript of the natural language commentary of the game and information extracted from the video to compare a query to a model play.

Chapter 7 describes the algorithm used by ILF to generalise the spatial and temporal information.

Chapter 8 presents the application of ILF to cricket where the objective is to classify and learn common cricket shots.

Chapter 9 presents an augmented version of ILF that can perform incremental recognition and learning. Two methods of learning are described: a model based method and time instance based method. This is applied to 3D trajectories that do not have well defined start and end points in time like American Football and cricket.

Chapter 10 presents a summary of the main contributions of this thesis.

In addition to the previous chapters, three additional Appendices are included:

Appendix A presents more results obtained from classifying American Football plays.

Appendix B presents the keywords used by the parser used to extract information from the transcript of the natural commentary of American Football games.

Appendix C presents the patterns used by the parser to generate information frames about American Football plays and players.

Chapter 2

Related Work

2.1 Introduction

The work described in this thesis is of relevance to the research areas of machine learning, object tracking, camera motion parameter estimation and systems that combine natural language processing with image analysis. The machine learning component of the work in this thesis is used in applications that combine learning with object tracking, camera parameter estimation and images analysis.

In this chapter we present past work in the area of machine learning. The background work done in object tracking and systems that combine natural language processing and image analysis is presented in chapter 4, whilst in chapter 8 we present previous work done in the field of camera parameter estimation.

The layout of the chapter is as follows: Section 2 describes incremental and one-step learning. In Section 3 we present the issues of importance in incremental learning. Section 4 describes some of the best known incremental learning systems.

2.2 Machine Learning

> *Learning is a many-faceted phenomenon. Learning processes include the acquisition of new declarative knowledge, the development of motor and cognitive skills through instruction or practice, the organisation of new knowledge into general, effective representations, and the discovery of new facts and theories through observations and experimentation [19].*

One can classify machine learning systems using different dimensions. We have chosen three dimensions as particularly meaningful [19]:

- Classification on the basis of the underlying learning strategies used.
- Classification on the basis of the representation of knowledge or skill acquired by the learner.
- Classification in terms of the application domain for which the knowledge is acquired.

Of relevance to the work described in this thesis is the first dimension which includes *learning from examples.* Learning can be a one-step process (or one-trial) or incremental. In the case of one-step learning the general procedure is as follows: all the training examples are presented to the system at the beginning of the learning process and the system develops, from the input data, a set of rules [90, 91] or a decision tree/lattice [92, 87] that best classify the instances present in the training set [20], [17]. This type of learning has been shown to produce effective, efficient and good concept descriptions from a given set of examples and is often used in a wide number of applications such as medical diagnosis [78]. One-step learning is well suited to batch learning as long as the data is relatively small.

Two of the most successful non-incremental learning algorithms are ID3 [90] and C4.5 [93]. The ID3 algorithm creates a decision tree by performing a recursive selection of attributes from a given set. ID3 was designed to handle large induction tasks and is based on an information gain function. Partitioning at each level of the tree is achieved by finding the attribute that maximises the information gained in selecting that attribute. A test partitioning the objects into separate values of the attribute then becomes the root of the tree, with the leaf nodes being the partitioned subsets. For example consider the case where attribute A has values A_1, A_2, A_3, ..., A_n and attribute A is selected as a decision attribute at the root of the tree. The training set B will be partitioned into subsets B_1, B_2, B_3, ..., B_n where B_i contains those objects in B that have value A_i for attribute A. However, these types of systems are limited since they do not have the ability to modify concept descriptions that are contradicted by new examples, and must rebuild the concept completely in order to accommodate new facts.

Incremental learning does not have this limitation as it allows the concept descriptions to be modified to reflect new learning events. Incremental learning is also useful when dealing with data intensive domains (millions of examples). For this reason incremental learning is also more suited to real-world situations. Human learning is incremental. A human develops concept descriptions based on the facts available at a given time and incrementally updates those descriptions as

new facts become available. There are essentially two reasons why human beings learn incrementally: the facts are generally received in the form of a sequential flow of information (typically the information received comes in steps and the human has to learn how to deal with a situation long before all the facts are available). Humans have limited memory and processing power (humans do not store everything they are exposed to, but rather only what they perceive as the most significant facts and generalisations) [97].

Several issues have been identified in incremental learning. They are described in detail below.

2.2.1 Bias

Bias is an important issue in incremental learning because it has impact on both the process of choosing hypotheses and on the ordering of the data used in the training period. The first definition of bias was given by [82] as being *any basis for choosing one generalisation over another, other than the strict consistency with the instances.* Gordon and desJardins [40] expanded Mitchell's definition *to include any factor (including consistency with the instances) that influences the definition or selection of inductive hypotheses.* Essentially there are two types of bias: **representational** and **procedural**. The representational bias defines the states in a search space. The representational bias also specifies:

- a language — disjunctive normal form (DNF),
- an implementation for the language, and
- a set of primitive terms that describe the allowable features, their types and their values which thereby defines the set of possible states [40, 110].

The procedural bias determines the order in which the states present in the space defined by the representational bias are traversed. A simple example of a procedural bias is the preference for a simple or specific hypothesis [110].

There are two main attributes that are typically associated with a representational bias. The first attribute is **strength**. A **strong** representational bias defines a small hypothesis space while a **weak** representational bias defines a large hypothesis space [119]. The second attribute is **correctness**. A **correct** bias defines a space that includes the target concept. A strong correct bias is usually desirable because it restricts the number of hypotheses from which a choice has to be made and, therefore, the system can converge more quickly to the target concept. The procedural bias determines the order in which the states present in

the space defined by the representational bias are traversed. A simple example of a procedural bias is the preference for a simple or specific hypothesis [40].

Both the procedural and representational biases can be evaluated empirically or analytically by determining what effect they have, or are expected to have, on the learning performance. Bias selection essentially involves the use of the results generated by the evaluation process to determine what is the appropriate bias or in some instances the appropriate sequence of biases, that should be used during the learning process. **Shifting** bias refers to the special case where bias selection occurs again after the learning process has already begun. In this case, the system may simply choose the next bias in a bias sequence that was established before the learning process started, or may use the results of the learning performed so far to evaluate potential alternative biases. In both instances there is a need to incorporate the knowledge that has already been learned to initialise and guide the search through the space defined by the new bias [40].

To demonstrate the importance of choosing the appropriate representational bias, consider the following example in which a set of features must be selected for expressing hypotheses. Let us say that the target concept which we desire is a description of blocks that are stable when placed on a table, and that the stability of the blocks depends on the shape of the blocks (the system has been told of this dependency). The blocks have three main attributes: *size*, *colour* and *shape*. Two examples are given to the system: a small, blue cube that is a positive instance and a small, red sphere that is a negative instance since it is not stable (it rolls off the table). If the system chooses a bias that prefers *size* and *colour* over *shape* in constructing concept definitions, the system may form the following hypothesis: *Small, blue blocks are positive and small red blocks are negative.* However, if the system were to choose a bias that prefers *shape* over *size* and *colour*, the system might form the following hypothesis: *Cubes are positive and spheres are negative.* The hypotheses generated by the system in the two instances are consistent with the training examples described so far. Consider the case in which a new example is given to the system that says that a small, blue sphere has been observed. The first hypothesis developed will predict that it will be stable while the second hypothesis will predict the contrary. Of the two hypotheses, the second one is clearly the better predictor of the concept of stability, thus proving that by choosing the right bias the predictive accuracy of the learner can be improved. Choosing the right bias also has the benefits of improved efficiency and readability.

Basically there are two types of bias evaluation methods: **generate-and-test (the methods are online and empirical)** and **theoretical studies (the processing is done off-line and is analytical)** [110]. The **generate-and-test** methods are important in the knowledge-poor situations where it is necessary to gather knowledge. The predictive theoretical analyses are important when they do not make too many simplifying assumptions [110].

2.2.2 Concept Drift

In many real-world domains, the context in which some concepts of interest depend may change, resulting in more or less abrupt and radical changes in the definition of the target concept [126, 123].

Consider the example of climate conditions in different parts of the globe. If one examines the climate at the Equator, one can identify climate conditions which are specific to the region. For example an average temperature of 28 degrees, high humidity and so on. Therefore it is possible to build a concept about the climate at the Equator. However, as one examines the change in climate conditions from the Equator as one travels to the North Pole, one can observe that the conditions change. The average temperature drops, the humidity changes and consequently the concept built based on the conditions encountered at the Equator are no longer valid. The concept needs to be changed in order to reflect the new conditions.

As another example, consider measuring devices or sensors which may alter their characteristics over long periods of time, resulting in a perceived change of the world and the necessity to modify prediction rules that rely on these measurements. In such situations it is necessary for the system to be able to track and adapt to these changes in the environment. The problem of tracking these environment changes is known as *concept drift.*

The major problem in incremental learning is how to distinguish between a *real* concept drift and a *virtual* concept drift which typically results from slight irregularities of the data. If the learning method is designed to react quickly to the first signs of concept drift, it may be misled into over-reacting to noise, erroneously interpreting it as concept drift. This leads to unstable behaviour and low predictive accuracy in noisy environments [126].

On the other hand, an incremental learner that is designed primarily to be highly robust against noise runs the risk of not recognising real changes in the target concepts and may thus adjust to changing conditions very slowly, or only when the concepts change radically. The ideal incremental learner needs to trade stability and robustness against noise with flexible and effective context tracking capabilities, but unfortunately these two requirements seem to oppose each other [126].

Real concept drift reflects changes in world knowledge, while virtual concept drift occurs as a function of the representation and procedures used to generate concepts (virtual concept drift indicates only temporary changes in the conditions and the problem can be addressed by having an in-built reluctance to modify known concepts).

Incremental learning is usually defined in terms of how many contexts exist in the world, how to discern them, what is their ordering, and what impact they exercise on the concept drift. Sometimes, the drift consists of changed values of some variables, but sometimes the relevance of individual variables or predicates can dramatically change. Moreover, the transition is usually only gradual with rather fuzzy boundaries between two different concept interpretations [125].

There are many different kinds of concept drift in the real world. Characteristic attribute values may change, the value domains of attributes may evolve over time, attributes once important may become meaningless, new ones may emerge; differences between successive concepts may be drastic or affect only some small facet; some concepts change only gradually, creating phases of ambiguity and uncertainty between periods of stability, other concepts may change overnight [124].

The notion of concept drift has received some attention in the literature on computational learning theory in recent years. For instance, Helmbold and Long [125] have explicitly investigated various conditions under which effective drift tracking is possible. They start from the observation that drift tracking is strictly impossible if there are no restrictions on the type of concept changes allowed (as an extreme example, consider a sequence of concepts that randomly alternates between the constant function 1 and the constant function 0 after every example). They then go on to study various restrictions on the severity (*extent*) or the frequency (*rate*) of concept changes.

In particular, Helmbold and Long [48] assumed a permanent (possibly with each example) but very slight drift. Their main results were:

- a general algorithm that tolerates concept drift of extent up to $\Delta \leq C_1\epsilon/(d\ln(1/\epsilon))$;
- a randomized version of this algorithm which is potentially more efficient computationally but tolerates drift of lower extent ($\Delta \leq C_2\epsilon^2/d^2\ln(1/\epsilon)$));
- upper bounds on the tolerable amount of drift for two particular concept classes (halfspaces and axis-aligned rectangles), which essentially say that no algorithm can track concept drift greater than $C_3\epsilon^2/n$ (where n is the dimension of the example space), if the prediction error is to stay below ϵ.

Here, the C_i's are positive constants, ϵ is the maximum allowed probability of misclassifying the next incoming example, and d is the Vapnik-Chervonenkis dimension of the target class. The extent of concept drift ϵ is measured in terms of the relative error of two successive concepts (i.e. the probability that they will disagree on a randomly drawn example).

The problem of tracking the concept drift is primarily caused by not knowing

what to forget and when to forget. Forgetting was implemented in the original FLORA system [125, 124, 122, 127, 65] in the form of a window of fixed size thus delimiting the number of samples in "*working memory*". However, the method proved to be incapable of dealing with some types of concept drift. A heuristic routine that controlled the size of the window was introduced in the later versions of the FLORA system to increase the system's ability to better track concept drift.

An alternative to a time window as a means of controlling forgetting is ageing of knowledge. This method was used in the FAVORIT system [66], where each exemplar is assigned a weight which slowly decays with time. If the same exemplar reappears, the weight is incremented. Exemplars whose weight drops below some threshold are forgotten.

2.2.3 Context

The context also plays a major part in incremental learning and in detecting *real* concept drifts.

Context is essential because it determines which attributes to use for a given problem out of many available, and consequently, as the context of a problem changes, there is a corresponding change in the set of attributes which are considered relevant [26, 118, 28].

To understand the role of context, one only has to consider the reasoning used by doctors when diagnosing a patient. Doctors seldom use all the information that is necessary to specify a diagnosis of a flu or yellow fever. Rather, they are able to exploit the information obtained from a small number of basic attributes and then supplement them with contextual information — yellow fever is uncommon in the Ottawa region so it can probably be *excluded* as a possible cause.

For another example, consider the case in which a system has to learn to detect oil spills on the sea surface by scanning satellite radar images. In this task, the only reasonable approach to assess the learner is to train on one set of images, and test on a different set of images. The learning task involves a large number of attributes which are used to describe the objects in the images in terms of their characteristics produced by a set of vision modules. These vision characteristics describe the average brightness of the pixels in the object, the jaggedness of the object's contour, the sharpness of its edges, etc. Early experimentation has shown that these vision attributes are not sufficient to classify objects into spills and non-spills with adequate accuracy. The domain experts pointed out that other data such as meteorological conditions, the angle of the radar beam that produces the image and even the proximity of the object to land can improve predictive accuracy. This is a typical example where the context information has

to be taken into account. Context is important in the learning process and that in many cases the learning task may not be satisfactorily resolved if the context is ignored [77].

Matwin and Kubat [77] defined three classes of concepts in terms of the concept's dependency on the context.

- The absolute concept class. The concepts belonging to this class do not depend on any context and are unambiguosly defined, such as: even number, abelian group or quadratic equation.
- The relative concept class. The concepts belonging to this class do not possess any property that is present under all circumstances. The meaning can be totally different in different contexts. Poverty, beauty, and high speed seem to represent relative concepts.
- The partially relative concept class. The concepts belonging to this class are characterized by a set of some context-sensitive properties that vary in different contexts. For instance, a swimming suit will always consist of the same basic components, but its color, shape, size and material will follow the dictates of fashion.

The learning process can become considerably harder if the concepts of interest depend on some *hidden context*. Mild weather means something different in Siberia and in Central Africa; Beatles fans had a different idea of fashionable hair-cut than the Depeche-Mode generation. Or consider weather prediction rules which may vary radically depending on the season. Changes in the hidden context can induce more or less radical changes in the target concepts. Effective learning in environments with hidden contexts and concept drift requires that the learning algorithm be able to:

1. detect context changes without being explicitly informed about them,
2. quickly recover from a context change and adjust hypotheses to a new context, and
3. make use of previous experience in situations where old contexts and corresponding concepts reappear.

The most common approach used to deal with the problem of hidden contexts has been suggested by Widmer and Kubat (and also implemented in their FLORA family of algorithms). The method uses a window which moves over recent past instances and uses the learned concepts for prediction only in the immediate future.

The rationale for windowing approaches is that the window is likely to contain mainly instances from the most recent context, and thus will allow the most current concept to be correctly learned. The assumption made is that instances of a context are contiguous in time and hence the intervals of time can be used to delineate contexts. As the context is known to vary in time, the learner trusts only the latest examples — their set is referred to as the window [43]. Newly arrived examples are added to the window and the oldest ones tend to be deleted. Both of these actions (addition and deletion) trigger modifications to the current concept hypothesis to keep it consistent with the examples in the window. In the simplest case, the window is of fixed size, and the oldest example will be dropped whenever a new one comes in (this is also known as time based forgetting). The system also maintains a store of concept descriptions or hypotheses pertaining to previously encountered contexts. When the learner suspects a context change, it will examine the potential of previously stored descriptions to see if there is a match for the current situation, or it will start developing an entirely new one [124].

Another approach is used by Widmer (implemented in the MetaL(B) system) [122]. The approach is based on the assumption that data may in fact contain explicit clues that would allow one to identify the current context, if one knew what these clues are. Technically, such clues would be attributes or combinations of attributes whose values are characteristic of the current context; more or less systematic changes in their values might then indicate a context change. As a simple example, consider the license plates attached to vehicles in a particular country. An agent crossing the border between, say, Austria and Germany might notice that all of a sudden the license plates look different, in a systematic way, and that might lead it to suspect that it is now in a different environment where some of the rules it has learned before may not be valid any more. Other example of contextual clues include climate or season in weather prediction and speaker nationality in speech processing.

2.2.4 Forgetting

Forgetting is another important aspect of incremental learning [66, 116, 22]. Due to the very nature of the learning, some of the data aquired previously becomes outdated or irrelevant and therefore it can be *forgotten* (discarded). The several advantages that forgetting offers include [66]:

- It helps to restructure aquired knowledge.
- It helps to prune the noise and irrelevant knowledge.
- It helps to reduce the amount of memory necessary for processing.

- It helps shorten the retrieval time.

2.2.5 Memory Size

There are three major models of system used in incremental learning systems.

Full Memory — In this model the system remembers all the training examples. It is the most popular model since, in general, it has the highest probability of generating an accurate hypothesis. The problem is that the system needs large amounts of space and memory to *remember* and process the training data. In some cases the system needs special routines to reduce the time necessary to search the stored information. There is also the question of whether it is really necessary to remember everything. In a constantly changing world, events and facts recorded in the past can simply become irrelevant and hence should be forgotten.

Partial Memory — In this model, as the name suggests, the system remembers only a part of the training examples. The most common approach used in partial memory systems is to build their concept definition from a window that moves over the stream of examples. The rationale for the window approach is that the window is likely to contain mainly instances from the most recent context, and thus will allow the most current concept to be correctly learnt. The main assumption made is that instances of a context are contiguous in time and hence, intervals can be used to delineate contexts. In the simplest case the size of the window is fixed and the oldest example is dropped from the window whenever a new example comes in. This method is also known as *time-based* forgetting. In more complex approaches the window can have a variable size which is controlled by special heuristic procedures/statistical methods. The major problem is what to forget. It is very easy to dismiss a fact as useless simply because it did not *seem* to be of any use for a given period of time. However, this fact may possibly be part of a bigger picture and by dismissing it as irrelevant we may lose important information.

Selective Memory + Summary Statistics — This model is probably the closest to emulating the way humans remember data. The system *remembers* important training instances and keeps statistics of the data thought to be *less* important.

2.2.6 Incremental Learning Systems

Several incremental learning systems have been developed to deal with the issues presented above. Below we describe some the best known systems.

GEM — GEM (Generalization of Examples by Machine) is an extended version of the AQ covering algorithm that allowed incremental formation of concept descriptions. The system was full memory, which meant that while the system allowed concept descriptions to be modified to accommodate new facts, none of the facts seen were ever forgotten which in turn required increasing amounts of memory over time [97].

COBWEB — Conceptual clustering, introduced by Michalski [81, 18, 80] is used in COBWEB. The system is full memory — no forgetting is possible (just as is the case with GEM). Unlike the GEM algorithm, COBWEB uses a probabilistic measure of the features in the instances to build its concept hierarchy. Another characteristic of COBWEB is that all the nodes in the conceptual hierarchy share the same features and therefore the system is able to perform better in cases of missing data [97]. The original algorithm has the limitation that it is restricted to only symbolic data, but this limitation has been addressed in the COBWEB variant CLASSIT, which is able to handle symbolic and real valued attributes [37].

UNIMEM — uses an incremental concept clustering algorithm that allows forgetting. The system organises the concepts in a hierarchy that permits multiple tests to be performed on new instances at each node and which simulates forgetting through the use of feature indexing. Each time a feature is matched, its index is incremented (the feature value is strengthened) while each time a feature is not matched, its index is decremented. If a feature's index reaches a low threshold (set by the user) then it is removed from the tree, and is effectively forgotten [37].

FLORA — [125, 124, 122, 65] assumes an incremental concept learning scenario, where a stream of training examples (positive and negative of a target concept) is input. Examples are processed one by one and the system updates its concept hypotheses after each instance. The representational language is propositional: examples are described by attribute-value pairs, and generalizations/hypotheses are sets or disjunctions of conjunctive expressions. The system is specifically targeted at learning problems exhibiting concept drift. The main components of the FLORA3 method are [126, 127]:

- concept representation in the form of three description sets,
- a forgetting operator and a time window over the incoming examples to control forgetting,
- a heuristic algorithm that automatically and dynamically adjusts the size of this window during learning, and
- a method to store concepts and re-use them in new contexts.

The concept hypothesis is represented by three description sets called ADES, PDES and NDES. ADES contains generalisations that are consistent with the examples, NDES contains generalizations that consistently describe negative instances while PDES contains generalizations that were once useful and might become relevant again. The description items in ADES and PDES are generated by incremental generalization in response to positive and negative instances. When new instances are added to or deleted from the current window, some items will be moved from one set to another. In particular, the set PDES of 'potential hypotheses' contains items that were once in ADES or NDES, but are contradicted by some examples. They are kept in PDES in the hope that they may become relevant again when old instances are dropped from the window. PDES acts as a reservoir of potentially useful hypotheses. More precisely, modifications to the window can affect the contents of the description sets in the following ways:

- Adding a positive example to the window may cause a new description item to be included in ADES, or some existing items to be either 'confirmed' or generalized to accommodate the new instance, and/or existing items to be transferred from NDES to PDES.
- Adding a negative example to the window may cause a new description item to be included in NDES, or some existing items to be 'reinforced' or generalized, or existing items to be transferred from ADES to PDES.
- Forgetting an example (dropping it from the window) can cause a new description item to be weakened (the corresponding counters are decremented), or even deleted from the current description set (if the counter drops to zero), or moved from PDES to ADES (if the example was the only negative instance covered) or to NDES (if the example was the only positive one).

STAGGER — The main idea behind STAGGER's learning method is a distributed concept representation composed of a set of dually weighted, symbolic characterizations. As each new instance is processed, a cumulative expectation of its identity is formed by using the pair of weights and generation of new Boolean characterizations. This latter process constructs more general, more specific, and inverted versions of existing concept description elements. These characterizations compete for inclusion in the concept description with the elements that were combined to form them [104, 105].

Concepts are represented in STAGGER as a set of dually weighted, symbolic characterizations. Each element of the concept description is a Boolean function of attribute-value pairs represented by a disjunct of conjuncts. An example element matching either small blue figures or square ones would be represented as *(size small and color blue)* or *shape square.* These characterizations are dually

weighted in order to capture positive and negative implication. One weight represents the sufficiency of a characterization for prediction and the other represents its necessity [105].

STAGGER uses logical sufficiency **(LS)**, or positive likelihood ratio, as measure of sufficiency. Similarly, logical necessity **(LN)**, or negative likelihood ratio, serves to measure necessity. The dual weights associated with each characterization are used together with estimated prior odds to calculate the odds that a given instance is positive. Expectation is the product of the prior odds of a positive instance and the **LS** values of all matched characterizations and the **LN** values of all unmatched ones. The resulting number represents the odds in favour of a positive instance [105].

In addition to representing concepts in a distributed manner and using Bayesian measures to compute an expectation value, STAGGER incrementally modifies both the weights associated with individual characterizations and the structure of the characterizations themselves. These two latter abilities allow STAGGER to adapt its concept description to better reflect the concept [105].

STAGGER's incremental learning method tolerates systematic noise and concept drift. It begins with simple characterizations and can learn complex characterizations by conducting a middle-out beam search through the space of possible conjuctive, disjunctive, and negated characterizations [105].

AQ15 — This system uses full memory of past examples [13]. Each time a new example arrives, the system considers all the previously seen examples before it modifies the current hypothesis. AQ15 is based on the star methodology that includes numerous generalization, specialization, and reformulation rules. These rules are used to enable the system to perform a chain of inferences, in order to derive new descriptors for inclusion in concept descriptions. The generalization rules transform a description into a more general description, one that tautologically implies the initial description, whereas specialization rules make the opposite transformation: given a description, they generate logical consequences from it. Reformulation rules transform a description into another logically equivalent description [17].

AQ15 essentially attempts to find the most general rule in the rule space that discriminates the training instances of one class from training instances in all other classes. It has a generality parameter that allows it to develop descriptions ranging from maximally general to maximally specific. It is also capable of learning concepts incrementally from data that is either erroneous or inconsistent [17].

YAILS — This system is based on a search algorithm which involves two major steps. Given a new example to learn, the first step consists of modifying the current concept in order to adapt it to the new example. If it does not succeed,

it starts the second step which tries to invent a new rule that covers the example. Essentially, the learning can be regarded as a search over the space of all possible conjunctions within the language of the problem. The search is guided by an evaluation function and it employs two types of search transformations: specialization and generalization. Consequently, the system has two specialization (and generalization) operators. The first is adding (removing) one condition to a rule. The second is the restriction (enlargement) of a numerical interval within the rule. The purpose of the evaluation function is to assess the *quality* of some tentative rule. YAILS uses an evaluation function which relates two properties of a conjunction of conditions: consistency and completeness [115].

YAILS is able to deal with two types of unknown information. The first arises when the value of some attribute is *unknown* and the second when the value is irrelevant. While the first case is interpreted as a kind of noise, the second one is treated as a *don't care* situation [115].

Systems such as AQ use a covering strategy during learning. This means that the system attempts to cover all known examples and that whenever some example has been covered it is removed. The system would consider a rule useless if it covered examples that are already covered by other rules and as a result it removes the rule. *YAILS uses a different approach.* Whenever the current theory does not cover a new example, new rules are created. The goal of this procedure is to find a "good" rule that covers the example. However, the introduced rule may cover examples already covered by other examples. The only criterion used to consider a rule is its quality. Therefore, the system usually generates more rules than other systems. The advantage of the greater number of rules is that some of the redundant rules may become useful. This means that by not discarding a redundant rule the system can save learning time. In addition the redundant rules may help deal with some of the uncertainty that arises during the classification. Consider the case in which a rule cannot be used to classify an example because it tests attributes whose values are unknown in the example. If redundant rules are admitted, it is possible that one such rule can be found to classify the example. YAILS uses a simple mechanism to control redundancy. The goal is to obtain the advantages of redundancy (better efficiency and higher accuracy) while at the same time minimising the number of rules used in classification. This mechanism consists of splitting the learned rules in two sets: the foreground rules and the background rules. This split is guided by a user-definable parameter (minimal utility) which acts as a way of controlling redundancy. The utility of one rule is calculated as the ratio of the number of examples uniquely covered by the rule, divided by the total number of examples covered by the rule. The higher the minimal utility threshold, the less redundant is the theory in the foreground. The redundancy present in the foreground set of rules is called static redundancy. YAILS uses only the foreground set of rules during the classification. Only when it is not able to classify an example does it try to find a rule in the background

set. If such a rule is found, the system transfers it from the background set to the foreground set so that in the end the foreground set contains the rules used during the classification of the examples. This latter type of redundancy is called dynamic redundancy. The advantage of this strategy is the minimisation of the introduction of the redundant rules [115].

AQUA — This is a story understanding program that learns about terrorism by reading newspaper stories about unusual terrorist incidents. The system uses case-based reasoning to understand and learn from novel situations. This is done according to the following process:

- Use the problem description to get reminded of an old case.
- Retrieve the results (lessons, explanations, plans) of processing the old case and give them to the understander, planner or problem-solver.
- Adapt the results from the old case to the specifics of the new situation.
- Apply the adapted results to the new situation.

The intent behind case-based reasoning is to avoid the effort involved in re-deriving these lessons, explanations or plans by simply reusing the results from previous cases. However, this process assumes that past cases are well understood and provide good lessons to be used for future situations, since it is these very cases that determine the performance of the system in new situations. This assumption is usually false when one is learning about a novel domain since cases encountered previously in the domain might not have been understood completely. Instead, it is reasonable to assume that the reasoner would have gaps in the knowledge represented by these cases. Even if past cases are not well understood, they can still be used to guide processing in new situations. However, in addition to using the past case to understand the new situation, a reasoner can also learn more about the old case itself, and thus improve its understanding of the domain. The learning process is therefore incremental, since the gaps are filled through experience. AQUA learns exactly the same way. The system retrieves past explanations from situations already in memory, and uses them to build explanations to understand novel situations encountered in newspaper stories about terrorism. Therefore, it fills the gaps in the retrieved explanation that is being used as a precedent in understanding the new situation. What is done with the newly learned information depends on the kind of knowledge gap the system is trying to fill. The new piece of knowledge could [95, 94, 27, 22]:

- result in a new explanation in memory,
- it could be used to fill in a gap in an existing explanation,

- it could be used to elaborate an existing explanation, if that explanation was not detailed enough to deal with the new situation, or
- it could be used to reorganize or re-index knowledge in the memory.

In doing so, the system refines its understanding of the domain by filling gaps in these explanations, by elaborating explanations, by learning new indices for the explanations, or by specializing abstract explanations to form new explanations for specific situations. The system learns incrementally since it improves its explanatory knowledge of the domain in an incremental fashion rather than by learning complete new explanations from scratch [95, 94].

PREDICTOR — This system uses a method that consists of a bias tester and an adjuster to address the issue of bias. The approach used to perform the bias testing uses formal definitions of assumptions about the bias which are used as a guide in the analysis of why the bias is inappropriate for learning the target concept. The bias tester is active since it tests the bias using queries to an oracle. The bias adjuster then records the analysis results and adjusts the bias accordingly. The main advantage of this approach is that it enables the system to minimally weaken the bias while correcting it [39].

2.3 Summary

In this chapter we have described previous work done in the area of machine learning that is of great relevance to one of the problems addressed in this thesis, specifically incremental machine learning. There are three other areas of research that are relevant to this the work presented in this thesis: systems that combine natural language processing and image analysis, object tracking and camera motion parameter estimation. The research done in these area is covered in chapters 4 and 8 where we describe two applications that use the incremental learning algorithm described in the next chapter.

We have covered, in this chapter, the known incremental learning algorithms and described in detail the important issues that need to be considered when developing an incremental learning algorithm. The incremental learning algorithms have been designed to address one (memory size — UNIMEM, bias — PREDICTOR) and in some cases two (concept drift and context — FLORA) of the important issues in incremental learning. The problems we attempted to address in this thesis required an incremental learning algorithm to address the issues of memory size, forgetting and concept drift but at present there is no algorithm available that can address all three issues. In the following chapter we describe a new incremental learning algorithm — ILF — that address the issues of memory size, forgetting and concept drift.

Chapter 3

ILF - Incremental Learning with Forgetting

3.1 Introduction

A goal of this thesis is to develop an incremental algorithm that can be used to learn the pattern of movement of multiple labelled objects in a dynamic scene. The application domains are assumed to be data intensive with many attributes and attribute values. There are three issues which we consider to be most relevant to our work, specifically *memory size (data related)*, *data pruning (data related)* and *concept drift (dynamic environment related)*.

The algorithm developed (called ILF: incremental learning with forgetting) has three main aims:

- To generate *concept hierarchies* — To generate concept hierarchies which are compact and do not store instances;
- To *forget* — To use an advanced evidence based forgetting, and
- To *track concept drift* — To generate concepts which use generalizations of the instances observed rather than storing the instances allowing a more accurate tracking of concept drift.

The first two issues deal with the way in which the data observed is stored and processed. The third issue deals with the way in which the concepts can change over time. In our work we have considered concept drift to be the way a concept evolves as new data is observed and integrated into the concept.

A concept is a collection of attributes or features and their associated values and in [2] it was pointed out that in fact if the data stored in the concept changes then it is simply attribute drift. While it is true that in fact it is the attribute data that changes, we believe that once the concept has undergone a certain amount of change that concept itself changes — essentially a different concept has evolved from the original concept — hence the concept drifts.

To demonstrate the advantages of ILF over UNIMEM and COBWEB we will describe each of the algorithms in detail and then we will analyse their differences.

The layout of this chapter is as follows: Section 1 presents the incremental learning systems UNIMEM and COBWEB. In Section 2, we present the incremental learning with forgetting algorithm — ILF. We compare ILF with UNIMEM and COBWEB in Section 3, whilst in Section 4 we discuss the differences between ILF, UNIMEM and COBWEB.

3.2 UNIMEM and COBWEB

3.2.1 UNIMEM

UNIMEM is an incremental conceptual clustering system which was designed to handle complex tasks such as natural language understanding and inference. The instances are represented in UNIMEM as conjunctions of features that can be numeric and symbolic.

UNIMEM organises its knowledge into a concept hierarchy through which it sorts new instances. The nodes high in the concept hierarchy represent more general concepts while their children represent more specific variants of the concepts. Each concept has an associated set of instances and they represent the leaf nodes in the concept hierarchy. Each instance can be stored in several feature nodes [37]. The nodes can specify the results of multiple tests which allow the system to handle instances with missing attributes. Each feature on a link has an associated integer score that specifies the number of links on which that feature occurs [37].

UNIMEM combines learning and classification as it sorts a new instance through its concept hierarchy, modifying the hierarchy in the process. The full UNIMEM algorithm is shown in Figures 3.1 and 3.2.

The algorithm has three functions: *UNIMEM*, *GENERALIZE* and *EVALUATE*. The *UNIMEM* function controls the processing of the instances. The *EVALUATE* function updates the predictability values of the attributes in the concepts, while the *GENERALIZE* function adds, removes or updates the attributes in

```
Input: The current node N of the concept hierarchy.
       The name of an unclassified instance I.
       The set of I's unaccounted features F.
Results: The concept hierarchy that classifies the instance.
Top-level call: Unimem(Top-node,I,F).
Variables: N and C are nodes in the hierarchy.
           G, H and K are sets of features (attribute values).
           J is an instance stored on a node.
           S is a list of nodes.
-------------
UNIMEM(N,I,F).
-------------
  Let G be the set of features stored on N.
  Let H be the features in F that match features in G.
  Let K be the features that do not match features in G.

If N is not the root node,
  Then If EVALUATE(N,H,K) returns TRUE or there are too few features in H,
         Then return the empty list.
Let S be the empty list.
For each child C of node N,
    If C is indexed by a feature in K,
       Then let S be Union(S,Unimem(C,I,K)).
If S is the empty list,
   Then for each instance J of node N,
        Let S be Union(S,GENERALIZE(N,J,I,F)).
If S is the empty list,
   Then store I as instance of node N with features K,
        For each feature J in F serving as in index to N,
            Increment the predictivness score R of J by 1.
            If R is high enough,
               Then remove J as an index leading to N.
Return N.
```

Figure 3.1: The UNIMEM algorithm – copied from [37].

concepts as required.

As UNIMEM checks the nodes in its hierarchy, it uses features on each node to match the instance. If the instance matches the description on the node closely enough, then it sends the instance down those links that contain features in the instance, and continues the process with the relevant children nodes until it gets to a leaf node (or image node). Successful recognition occurs if the instance matches a node in the tree. The number of features necessary for a match and the threshold used to determine whether two values match, are system parameters. *Whether or not the instance successfully matches, UNIMEM uses an EVALUATE procedure to update the node's scores.* Therefore each time a new training instance is seen, all potential matches in the UNIMEM hierarchy will undergo a feature value update (either increment or decrement — forgetting). Essentially each new instance seen affects all concepts so the effect is not localised [37].

When UNIMEM reaches a node that matches the instance but none of whose children match, it compares all the instances stored in that node with the new instance. If an old instance stored in the node matches the new instance closely

```
Variables: N and C are nodes in the hierarchy.
           F,G,H and K are sets of features (attribute values).
           I and J are the names of instances.
           S and T are predictivness score of a node's feature.
-------------------
GENERALIZE(N,J,I,F).
-------------------
 Let G be the features in instance J.
 Let H be the features in F that match features in G.
 If H contains enough features,
    Then create a new child C of node N.
         Index and describe C by the features in H.
         For each feature K serving as index to C,
             Increment the predictivness score R of K by 1.
             If R is high enough,
                Then remove K as an index.
         Remove J as an instance of N.
         Let G' be the features in G that are not in H.
         Let F' be the features in F that are not in H.
         Store J as an instance of C with features G'.
         Store I as an instance of C with features F'.
         Return C.
---------------
EVALUATE(N,H,K).
---------------
 For each nonpermanent feature F in H,
     Raise the predictability score S for F on N.
     If S is high enough,
        Then make F a permanent feature of N.
 For each nonpermanent feature G in K,
     Lower the predictability score T for G on N.
     If T is low enough,
        Then remove the features G from N.
             If N has too few features,
                Then remove N from its parent's list of children.
                Return TRUE.
Return FALSE.
```

Figure 3.2: The UNIMEM algorithm (continued) – copied from [37].

enough, then UNIMEM builds a new node based on the common features between the two instances and stores the instances (the new instance plus the instance matched in the node) with the newly created node. When this happens, the system also updates the predictiveness value for each feature indexing the newly built node. UNIMEM compares the new instance to *each* one of the instances stored with the node, and as a result it can build multiple nodes. If the new instance does not match any of the instances in the node, then the system stores it with the current node [37].

UNIMEM is an incremental learning system that has the advantage that it does not use full memory — it forgets irrelevant instances. However UNIMEM does have a significant disadvantage as it requires the user to set the system parameters to make clustering decisions [37].

3.2.2 COBWEB

COBWEB is an incremental system for hierarchical conceptual clustering. The system carries out a hill-climbing search through a space of hierarchical classification schemes using operators that enable bidirectional travel through this space [32, 31, 96].

COBWEB represents input instances as a set of attribute-value pairs. Each attribute can have only one value of symbolic type. The concepts built by COBWEB can be described in terms of their attributes, values and associated weights. Each concept has a probability value associated with it. All nodes in the hierarchy from top to bottom share every attribute observed in the training instances. Also associated with each attribute is every possible value for that attribute. COBWEB's concept hierarchy has more general nodes at the top and more specific nodes at the bottom. The leaf nodes represent the specific instances that it has encountered. *COBWEB never deletes instances and therefore it* ***does not forget*** *any of the data seen* [37, 108].

There are four components that make up the COBWEB system [33]. The first component is a heuristic evaluation measure called *category utility*, which is used to guide the search. The method was originally developed as a means of predicting the basic level in human classification hierarchies. Research suggests that some categories are more basic than others and are retrieved more rapidly and more frequently.

COBWEB incrementally incorporates objects into a classification tree, where each node is a probabilistic concept that represents a class. The incorporation of an object is the process of classifying the object by descending the tree along an appropriate path, updating counts along the way, and performing one of several operators at each level. There are four operators:

Operator 1 — classifying the object with respect to an existing class. Perhaps the most natural way of updating a set of classes is to simply place a new object in an existing class. In order to determine which category best hosts a new object, COBWEB tentatively places the object in each category. The partition that results from adding the object to a given node is evaluated using a category utility. The node that results in the best partition is identified as the best existing host for the new object [33].

Operator 2 — creating a new class. In addition to placing objects in existing classes, there is a way to create new classes. Specifically, the quality of the partition resulting from placing the object in the best existing host is compared to the partition resulting from creating a new singleton class containing the object. Depending on which partition is best with respect to category utility, the object is placed in the best existing class or a new class is created. This operator allows

COBWEB to automatically adjust the number of classes in a partition [33].

Operator 3 — combining two classes into a single class. COBWEB uses this operator (combined with the split operator) to deal with the effects of skewed data. The operator merges two nodes in the hope that the relevant partition is of a better quality. Merging the two nodes involves creating a new node and summing the attribute-value counts of the nodes being merged. The two original nodes are made children of the newly created node [33].

Operator 4 — dividing a class into several classes. As in the case of node merging, splitting may also increase partition quality. In the split process, a node of a partition (containing **n** nodes) may be deleted and its children promoted, resulting in a partition of **n + m - 1** nodes, where the deleted node had **m** children nodes [33].

The last component of COBWEB is the control function which is shown in Figure 3.3. For each new instance, COBWEB uses the control function to compute four probability scores (one score for each one of the operators described above). The highest score determines if a new node is built, an existing node is modified or an existing node is removed from the hierarchy.

```
FUNCTION COBWEB(Object, Root [of a classification tree])
1)Update counts of the Root
2)IF the root is a leaf
  THEN Return the expanded leaf to accommodate Object
  ELSE Find that child of Root that best hosts Object
       and perform one of the following
     a) Consider creating a new class and do so if
        appropriate
     b) Consider node merging and do so if appropriate
        and call COBWEB(Object, Merged Node)
     c) Consider node splitting and do so if appropriate
        and call COBWEB(Object, Root)
     d) IF none of the above (a, b or c) were performed
        THEN call COBWEB(Object, Best child of Root)
```

Figure 3.3: The COBWEB control function.

The advantage offered by COBWEB is that it uses a well defined evaluation function to guide it through the classification (unlike UNIMEM which has a user defined parameter) and the incremental learning process. However COBWEB has the disadvantage that it does not delete any of the instances seen and that it assumes that each instance consists of a single object (thus it avoids issues of finding mappings between analogous components) [33].

3.3 Incremental Learning with Forgetting (ILF)

ILF is designed to deal with complex (data intensive) spatio-temporal patterns and all the data is assumed to be symbolic (it is possible to have numeric data as long as some form of indexing for the numeric data is provided). Hence our algorithm was developed for data with a high number of symbolic attributes. Another major assumption we make is that each concept has several distinct variants and that it can drift over time.

Since the data has a high number of attributes and the concept undergoes changes over time, an important part in our algorithm is forgetting. The problem with UNIMEM is that it has no concept of different types of memory; i.e. there should be a method by which a common concept is treated differently from a concept that occurs infrequently.

I.L.F uses a representation similar to COBWEB's, where all concepts consist of the same number of features (which is equal to the number of features observed in the instances). This is because this type of representation allows an easier handling of missing data.

Finally, unlike UNIMEM or COBWEB, our algorithm does not store training instances and always generalises.

3.3.1 Data Structure

The concepts developed by our algorithm are stored in a hierarchy in which all nodes share all the features observed in the training instances. For example if the instance observed has six features, then all nodes in the hierarchy will have six features. The nodes in our structure do not store the individual instances as is the case with UNIMEM or COBWEB. Instead any feature's range of values is defined with the help of a set which covers all the values encountered in the training instances which were used in the generalisation process. There are several reasons for choosing this type of representation. The first reason is that if a learner retains some of the early instances, the concept tends to become fragmented when dealing with several distinct variants and so the system is no longer able to recognise the actual concept or track its drift (one of the problems with UNIMEM). The second reason is that by using all observed features to build the concepts, the system is able to handle cases of missing or noisy data. The third reason is that it substantially reduces the size of the concept and as a result the amount of memory required to store the hierarchy is reduced.

The system sorts the instances in a similar manner to UNIMEM. Each new instance is compared with the current node in the hierarchy to determine if there is

enough evidence to justify the update of the current node. Each node produces an evidence score which determines whether the instance does or does not match the current node. The score is computed as a function of age. In our concept representation, each feature has a set of values associated with it and each one of the values in the set has an age associated with it. An example of a concept with two features (first feature has two values, the second feature has only one value) is shown below:

```
Concept(Age) = {
                  Feature 1 = {(Value1,Age), (Value2,Age)}
                  Feature 2 = {(Value3,Age)}
                }
```

Consider the case in which the system contains the concept shown above and is attempting to match it against the instance given below:

```
Instance = {
             Feature 1 = {(Value1,Age)}
             Feature 2 = {(Value4,Age)}
           }
```

Let us assume that there is enough evidence to update the concept. In this case the feature values and their associated age values can be updated in three ways depending on whether the feature values match the corresponding values in the new instance. If the value of a feature is *matched* by its corresponding value in the instance, then its *age is decremented.* For example the first value of Feature 1 (Value1) in the concept matches the instance. Hence its age is incremented. If the value of the feature is *not matched* by the value in the instance *but the value has been observed before* then its *age is incremented.* In the example, the second value of Feature 1 does not match the value in the instance. However, the value in the instance has been observed before since it is included in the set of values for Feature 1. Hence the age of the second value for Feature 1 is decremented. Finally if the value of the feature is *not matched* by the value in the instance *and the value has not been observed before* then the *new value is added to the feature value set.* Also the other values in the set have their respective age values decremented. In our example, the value of Feature 2 in the concept does not match the value in the instance and has not been encountered before. Therefore the new value in the instance is added to the feature value set (Feature 2 becomes {(Value3,Age), (Value4,Age)}), and the age of the previous value in the set (Value3) is decremented.

If the system does not find enough evidence to modify the current node, then the node is left *unchanged* and the next node in the hierarchy is considered. The next node is also considered even if the node is modified.

In this way multiple concepts/nodes can be updated (provided enough evidence was found) by the same instance. While this procedure results in the system updating nodes which should not be updated when one considers the overall set of instances, results show that over time the unnecessary modifications are "aged out" of the concepts. The main reason for choosing to update multiple nodes is that it is a simple way of representing multiple matches between the concept and the instance which is more appropriate than a single match and it allows for ambiguity.

Finally, it is possible to merge two similar concepts into a single more general concept. The way in which this is achieved is by tracking the similarity scores computed by the system when analysing new instances. If the scores computed for concepts over a set number of instances is higher than a set threshold, then the system checks to see whether there is enough evidence to combine the two nodes into a single node (similar to the way in which COBWEB merges nodes). If evidence is found to justify the merger, the system builds a new node which combines the feature values along with their *updated* associated age, computes a new age for the new concept and finally, removes the two concepts that have been merged.

The algorithm for ILF is shown in Figures 3.4 and 3.5. The processing is controlled by the *ILF* function. For each new instance a score is computed by applying function *Match*. If the function *Match* indicates a match then *ILF* attempts to update the current node. The system computes a *Similarity_Score* and if the score is greater than or equal to the *Similarity_Threshold* (which is a system parameter and is set at 75%) If the similarity conditions is satisfied the two nodes in the hierarchy are merged by applying the *Merge* function.

3.3.2 Forgetting

Forgetting offers the following advantages:

- It helps prune out old or irrelevant data.
- It helps restructure the acquired knowledge.
- It helps in reducing the memory size necessary for processing.
- It helps shorten the retrieval time.

In our algorithm we have essentially taken the UNIMEM type of forgetting and changed it in a way that is more similar to the human style of forgetting. Research in cognitive psychology [114, 130, 41, 29, 121] suggests that human beings have

```
Input: Node N of the concept hierarchy.
       C1, C2, ..., Cm children nodes of N.
       Instance I.

High_Sim1 = 0; High_Sim2 = 0; Node_Id1 = -1; Node_Id2 = -1;


-----------
ILF(N,I).
-----------


  if N = Root then
   for i = 1 to m
     ILF(Ci,I).
   endfor
  else
    if Match(N,I) = True then
       Update(N,I).
    endif
  endif
  if High_Sim1 > 0 && High_Sim2 > 0 then
    Merge(Node_Id1, Node_Id2);
  endif

Let A1,A2, ..., An be features of N and
    B1,B2, ..., Bn be features of I.


----------
Match(N,I).
---------


Similarity_Score = 0;
 for i = 1 to n
   if N(Value(A1)) = I(Value(Bi)) then
    Similarity_Score++;
   endif
 endfor

 if Similarity_Score > Similarity_Threshold then
    if High_Sim1 = 0 then
       High_Sim1 = Similarity_Score; Node_Id1 = N;
    else if High_Sim2 = 0 then
            High_Sim2 = Similarity_Score;  Node_Id2 = N;
         else if High_Sim1 < Similarity_Score then
                 High_Sim1 = Similarity_Score;  Node_Id1 = N;
              else if High_Sim2 < Similarity_Score then
                      High_Sim2 = Similarity_Score;  Node_Id2 = N;
   return True;
  else
   return False;
  endif
```

Figure 3.4: The ILF algorithm.

```
Let A1,A2, ..., An be features of N,
    B1,B2, ..., Bn be features of I,
    NValue-1(A1), NValue-2(A1), ..., NValue-p(A1) be the values of feature A1 in N,
    IV(B1), IV(B2), ..., IV(B) be the values of features B1, B2, ..., Bn and
    Age(Value-1(A1)), Age(Value-2(A1)), ..., Age(Value-k(A1)) be the ages of values
    1, 2, ..., k of feature A1.


-----------
Update(N,I).
-----------


 for i = 1 to n
  for j = 1 to p
     if IV(Bi) = NValue-j(Ai) then
       Update_Age(Age(Value-j(Ai)),True);
     else
       Update_Age(Age(Value-j(Ai)),False);
  endfor
 endfor

Let A1,A2, ..., An be features of Node_Id1,
    B1,B2, ..., Bn be features of Node_Id2,
    N1Value-1(A1), N1Value-2(A1), ..., N1Value-p(A1) be the values of feature A1 in
    Node_Id1,
    N2Value-1(B1), N2Value-2(B1), ..., NValue-s(B1) be the values of feature B1 in
    Node_Id1,
    Age(Value-1(A1)), Age(Value-2(A1)), ..., Age(Value-m(A1)) be the ages of values
    1, 2, ..., m of feature A1.


-------------------------
Merge(Node_Id1, Node_Id2).
-------------------------


Merge_Score = 0;

  for i = 1 to n
   for j = 1 to p
    for k = 1 to s
     if N1Value-j(Ai) = N2Value-k(Bi) then
        Merge_Score++;
     endif
    endfor
   endfor
  endfor

 if Merge_Score > Threshold then
  for i = 1 to n
   for j = 1 to p
    for k = 1 to s
     if N1Value-j(Ai) = N2Value-k(Bi) then
        Update_Age(Age(Value-j(Ai)),True);
     endif
    endfor
   endfor
  endfor
```

Figure 3.5: The ILF algorithm continued. The *Update_Age* function is shown in 3.6.

an *immediate memory* used to store information for only a couple of seconds, a *short term memory* which is used to store information (in a ordered manner) for time intervals ranging from minutes to hours and a *long term memory* which is used to store information in an associative and highly structured manner for very long periods of time (years). Information is passed (after some processing) from the immediate memory to the short term memory and eventually to the long term memory depending on the level of reinforcement or in some cases on the nature of the information (emotion plays an important factor in the way in which the same fact is dealt with — for example narrowly escaping a car crash is generally remembered well without reinforcement).

We have implemented an ageing procedure which simulates short-term memory (set of facts just observed and which can be discarded if not reinforced), medium-term memory (concepts already reinforced but not up to the level justifying permanent storage and which can still be forgotten) and permanent memory (concepts which have been reinforced so many times that it justifies permanent storage and which cannot be forgotten).

The algorithm uses ageing at two levels: the *data level* and the *concept level.* The data level refers to the data which is contained in the concept. Each attribute in the model has one or more generalised symbolic descriptions and each description has an age value associated with it. Similarly the concepts in the hierarchy have an age value associated with each of them. The age value of the concept is computed by calculating the average age of the attributes in the concept.

There are three stages at the data level: *system-seen* (equivalent to the human *immediate memory*), *system-learned* (equivalent to the human *short-term memory*) and *expert-learned* (equivalent to the human *long-term memory*). The difference between the three stages is the way in which the age of the data is updated. Any data that has reached the *expert-learned* stage no longer ages and therefore it cannot be forgotten. The difference between *system-seen* and *system-learned* is the amount by which the age of the data increases or decreases.

The age of the data in a given concept is updated only when the condition, that the system evaluates that there is enough evidence for the given concept to be modified, is satisfied.

If the new instance is found to be similar to the model, then the age of each of the attribute values (provided it is at the *system-seen* or *system-learned* stage) is reassessed. If there is an attribute value in the concept which matches its corresponding value in the new instance, then that attribute value is reinforced and its age is decreased. If the attribute value in the concept does not match its corresponding value in the instance, then the age of the value is increased. The basic algorithm for updating the age of the feature values is shown in Figure 3.6: There are six user set parameter values. The six values are:

```
Vf - feature value   F - feature
Data age levels - System_Seen, System_Learned, Expert_Learned
Match(x,y) - compare values x and y and returns TRUE if x matches (has the same
             symbolic value as) y
Age_Level(F,x) -  returns the age level of value x for feature F
Age_Update(F,x,y) - updates the value x of feature F by amount y
IncreVal1 - Increment value at the System_Seen level
IncreVal2 - Increment value at the System_Learned level
DecreVal - Decrement value at the System_Seen/System_Learned level

if (Match = True)
 if (Age_Level(F,Vf) = System_Seen)
    Age_Update(F,Vf,IncreVal1);
 if (Age_Level(F,Vf) = System_Learned)
    Age_Update(F,Vf,IncreVal2);
 if (Age_Level(F,Vf) = Expert_Learned)
             ;
if (Match = False)
 if ((Age_Level(F,Vf) = System_Seen) || (Age_Level(F,Vf) = System_Learned))
    Age_Update(F,Vf,DecreVal);
 if (Age_Level(F,Vf) = Expert_Learned)
             ;
```

Figure 3.6: Ageing algorithm used by ILF.

- *Forget/Prune Threshold.*
- *System_Seen — System_Learned Age Threshold (SST).*
- *System_Learned — Expert_Learned Age Threshold (SLT).*
- *Feature/Attribute Initial Age (FAIA).*
- *Age Increment Rate at the System_Seen Level — IncreVal1.*
- *Age Increment Rate at the System_Learned Level — IncreVal2.*
- *Age Decrement Rate at the System_Seen Level/System_Learned Level — DecreVal.*

To demonstrate how the ageing algorithm works consider the following case in which the seven values are set as follows: *Forget Threshold (-200), SST (-100), SLT (0), FAIA (-190), IncreVal1 (+2.5), IncreVal2 (+5)* and *DecreVal (-2.5).* These are the values that were used in the N.F.L. application (described in the next three chapters).

Data seen by the system for the first time is given the age of -190. Every time that data is seen again, the age is decreased by 2.5. Similarly if the data is not seen the age is increased by 2.5. Once the data seen reaches the age of -100, it reaches the stage of *system-learned* and we increment/decrement the age by different amounts (+5 and -2.5). If the data reaches the value of 0, it reaches the stage of *expert-learned* and cannot be forgotten. If the age value falls below -100,

it gets to the stage of *system-seen*. Finally, if the age falls below -200 the data is no longer considered relevant and it is discarded.

The concept age is computed using the age of the data contained in the concept (average of the features). When the age of a concept reaches the value of -200, the concept is removed from the hierarchy.

Notice that while there is a difference in the increment applied at the System-Seen and System-Learned stages (+5 vs +2.5), there is no difference in the amount of decrement applied (-2.5 vs -2.5). This is to indicate that once a bit of data has reached the System-Learned stage, then there is more evidence to strengthen (increment) its age value then there is evidence to weaken (decrement) it.

Using expert set parameters offers advantages but also has some disadvantages. The advantage is that ILF benefits from the user expertise since it is given a good basis for learning — none of the parameters need to be determined by the system.

The disadvantage is that the system will not work on new domains without the user setting the parameter values — user input is required. This may be considered a significant disadvantage from the point of view of applications in previously unseen domains since ILF lacks a mechanism to determine the parameter values by itself. The reason for not developing such a mechanism is that we believe that generally all learning is at some stage supervised. Consider the case of human beings. In general humans rarely learn without supervision — even learning basic skills such as to walk requires a teacher and a significant amount of time and practice. Humans are generally given clues as to what they need to do.

In the following section we show the differences between ILF, UNIMEM and COBWEB for a particular example. This will highlight the differences in the concept hierarchy, forgetting and concept drift.

3.4 ILF vs UNIMEM and COBWEB

3.4.1 Example

The data set for this example consists of five instances (with eight features) of which three belong to class (or concept) A and two belong to class B. For this example as well as all examples in the following chapters, we will use the parameter values described in the previous section. The values are shown below: *Forget Threshold (-200), SST (-100), SLT (0), FAIA (-190), IncreVal1 (+2.5), IncreVal2 (+5)* and *DecreVal (-2.5).*

At the start of the processing, the same threshold was set for a match between a concept and an instance for both our system and UNIMEM. The threshold value was 75% of the features (six out of eight features). In addition, to remove a feature from the UNIMEM concept, the weight of the feature value would have to be decremented four times. In the case of ILF similarly if a feature is decremented four times then it is removed.

INDEX	CLASS	A1	A2	A3	A4	A5	A6	A7	A8
1	A	1	1	1	1	1	1	1	1
2	B	4	4	1	3	2	4	3	3
3	A	1	2	1	1	1	2	1	1
4	A	1	2	2	1	1	1	1	1
5	B	4	1	1	3	2	4	3	3

Table 3.1: The five instances along with the class to which they belong.

Comparing the attributes of the two classes (see Table 3.1), it is apparent that the attributes of class A are generally 1's and 2's, while of those of class B contain 3's and 4's. Note that no two examples have exactly the same attribute values although many do match. Also note the discrete (symbolic) makeup of the data.

When the first instance is seen by UNIMEM it builds a node for it with an associated instance. COBWEB builds only a single class node that contains the first instance and sets the probability of the class to 1.0. Our algorithm (ILF) also builds a single node for the first instance. The hierarchies for the algorithms after the first instance is processed are shown in Figures 3.7a, 3.7b and 3.8a. ILF sets the initial age for the observed feature values to -190.

The second instance is distinctly different from the first one (less than 75% of attributes match) so UNIMEM *updates* the values of the features in the existing node in the hierarchy and builds a new node for the new instance. COBWEB uses the category utility described previously to determine if a new class node should be built for the second instance or if an existing class can be merged with the instance. Creating a new class node is determined to be the best option, so COBWEB generates a new class node. Both class nodes have the same probability of occurrence: 0.5.

Our system simply builds a new node for the new instance. The hierarchies after the second instance has been analysed are shown in Figures 3.9a, 3.9b and 3.8b. In all cases the second class uses a new leaf each.

UNIMEM analyses the third instance and determines that it matches the node built for the first instance (more than 75% of the attributes match). Therefore it builds a new more general node that consists of the six features common to the node and the new instance. It also stores both instances as its children. The

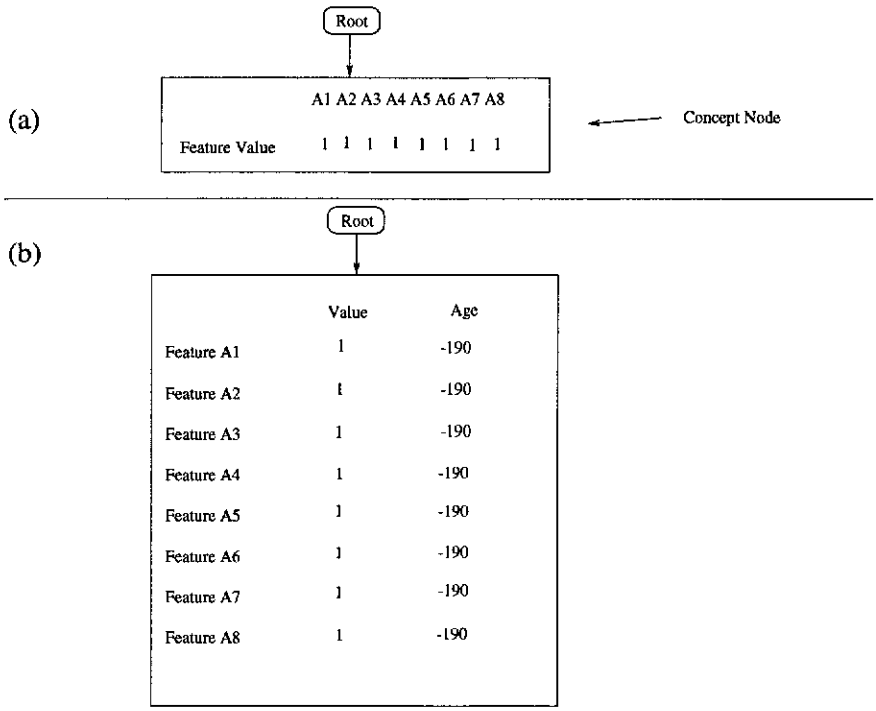


Figure 3.7: Part (a) shows the UNIMEM structure while part (b) shows the structure built by our algorithm.

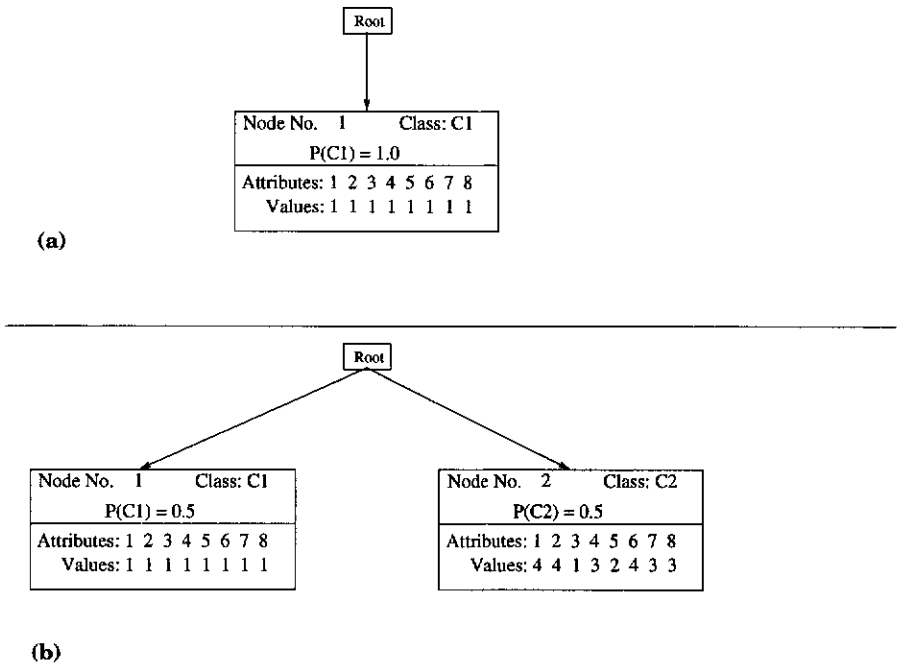


Figure 3.8: Part (a) shows the COBWEB hierarchy after 1 instance while part (b) shows the COBWEB structure after 2 instances.

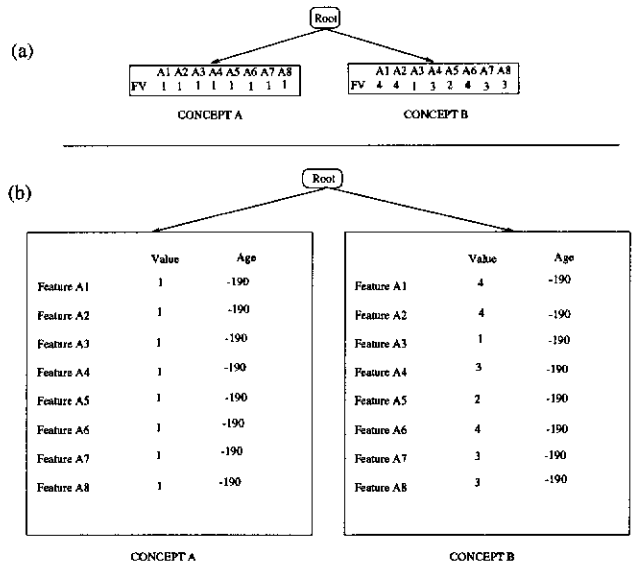


Figure 3.9: Part (a) shows the UNIMEM structure after instance number 2 was processed while part (b) shows the structure built by ILF after instance 2 was analysed.

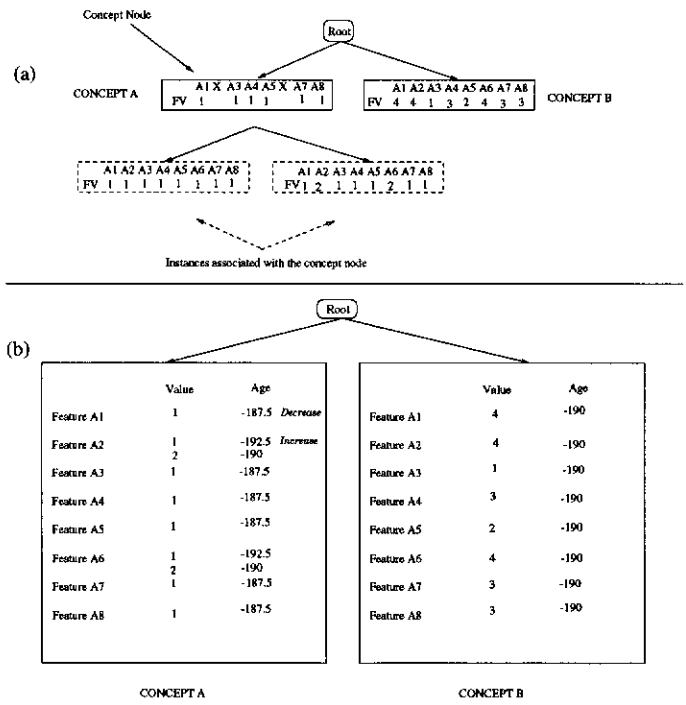


Figure 3.10: Part (a) shows the UNIMEM structure after instance number 3 was processed while part (b) shows the structure built by ILF after instance 3 was analysed.

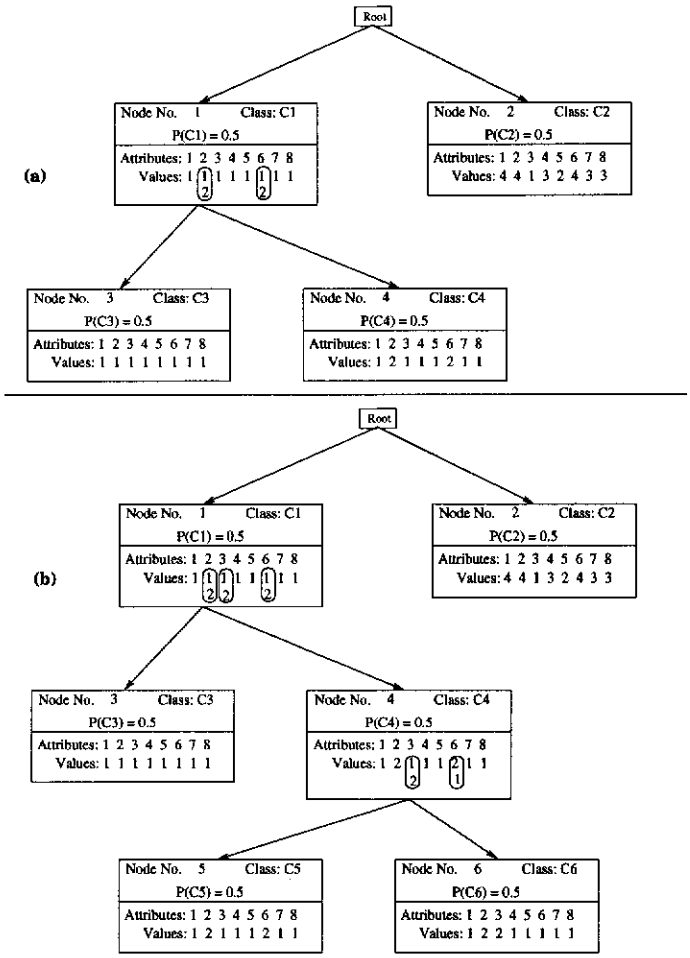


Figure 3.11: Part (a) shows the COBWEB hierarchy after instance number 3 was processed while part (b) shows the COBWEB structure after instance number 4 was analysed.

new instance does not match the node built for the instance belonging to class B so UNIMEM simply updates the values for the features in the node (in a sense ageing them).

COBWEB uses the category utility to determine which one of the two existing classes best matches the instance. Class 1 is the best match and COBWEB checks to see if the new instance should be merged with class 1 or if a new partition (new nodes) should be generated. COBWEB determines that it is best to create a new partition so it updates the node containing class 1 making it more general. The node's probability remains 0.5. Two new nodes are created, one for the old instance in class 1 and one for the new instance. Both nodes have a probability of 0.5 and are the children nodes of the node containing class 1.

When our system analyses the third instance, it determines that the instance matches the first concept in the hierarchy and it proceeds to update the concept.

Each feature whose value is matched by the corresponding value in the new instance has its age decremented. If the feature's value is not matched then its age is incremented. If the feature's value is not matched and the feature's range of values does not contain the value in the new instance, the new values from the instance are added (the new value is added to the feature's value set) along with an initial age of -190. No new nodes are built. The updated hierarchies are shown in Figures 3.10a, 3.10b and 3.11a.

The fourth instance is compared by UNIMEM against both child nodes of the root of the hierarchy. The first child node is the more general node built after analysing the third instance. It has 6 features and only 5 of those features (1,4,5,7 and 8) match the features in the new instance. Hence only 63% of the features in the new instance are matched which is not enough to send the instance down this branch of the hierarchy. The matching process fails and UNIMEM updates the values associated with the general node. The fourth instance also fails to match the second child node of the root. The node is updated and since both child nodes of the root node failed to match the new instance, a new node is built which contains the instance.

COBWEB analyses the fourth instance and after applying the category utility it determines that the best match is class 4. It updates the node containing class 4, making it more general while retaining the original probability of the node of 0.5. Two new nodes are created and each node has a probability of 0.5.

ILF matches successfully the fourth instance against the first concept in the hierarchy. The concept's feature values and their associated age values are updated depending on whether or not they match the values in the new instance. The resulting hierarchies after the fourth instance is processed are shown in Figures 3.12a, 3.12b and 3.11b.

Finally, UNIMEM compares the fifth instance against the three child nodes of the root in the UNIMEM hierarchy. The child node representing the two instances of class A fails to match the new instance and its feature values are updated accordingly. The same process is applied to the node created to accommodate the fourth instance (also belonging to class A). The third child node which represents the second instance *does match* the new instance. Hence the common features are used to build a new more general node. The two instances belonging to class B are stored as child nodes of the general node.

Just as in the case of the previous instance, COBWEB attempts to determine whether to merge the fifth instance with one of the existing classes or whether a new class should be generated. In this case, *class 2* is updated to be more general and two nodes are built, one for the old instance in *class 4* and one for instance five.

In the case of ILF, the fifth instance matches the second concept in the hierarchy. Again the concept's feature values and their associated age values are updated depending on whether or not they match the values in the new instance. The final hierarchies at the end of the processing are shown in Figures 3.13a. 3.13b and 3.14.

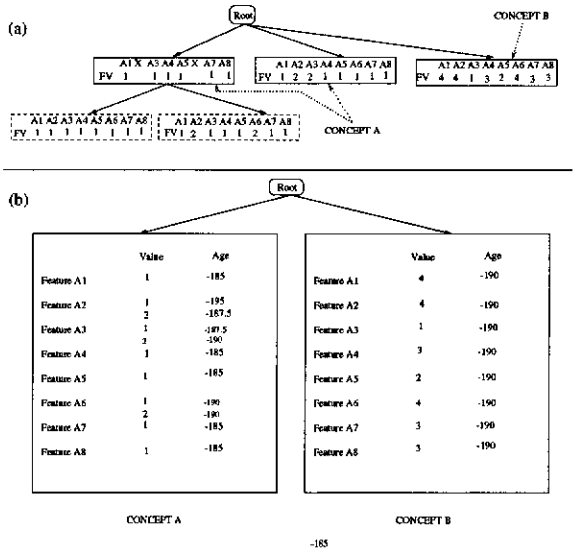


Figure 3.12: Part (a) shows the UNIMEM structure after instance number 4 was processed while part (b) shows the structure built by ILF after instance 4 was analysed.

The main difference between the UNIMEM hierarchy and that of ILF is that UNIMEM has divided the concept A over several nodes and it essentially updates the other existing concept feature values regardless of whether they match the new data or not. The major disadvantage in dividing the concept over several nodes is that the features are not strengthened as much as they would be if the concept covered just one node. Hence data which might still be relevant could be deleted since it was not reinforced.

Notice also that the number of nodes created by ILF is smaller than UNIMEM's (2 vs 7). COBWEB builds a total of 8 nodes (5 instance nodes) and therefore uses the largest amount of memory when compared with ILF and UNIMEM. Note also that it splits concept A just like UNIMEM.

In this example ILF is better than COBWEB as it generates only 2 nodes to represent the same set of instances and it also has the capability to forget or prune old and irrelevant data.

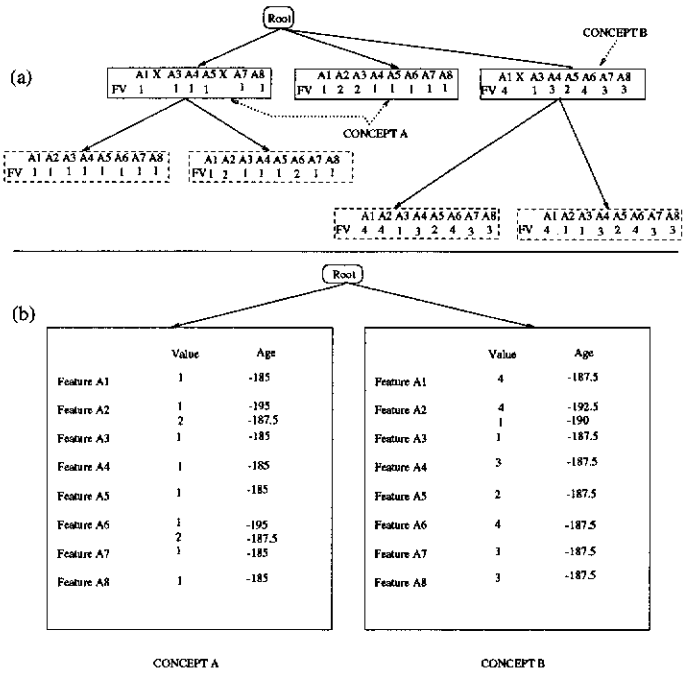


Figure 3.13: Part (a) shows the structure build by our algorithm after instance number 5 was processed while part (b) shows the structure built after instance 5 was analysed.

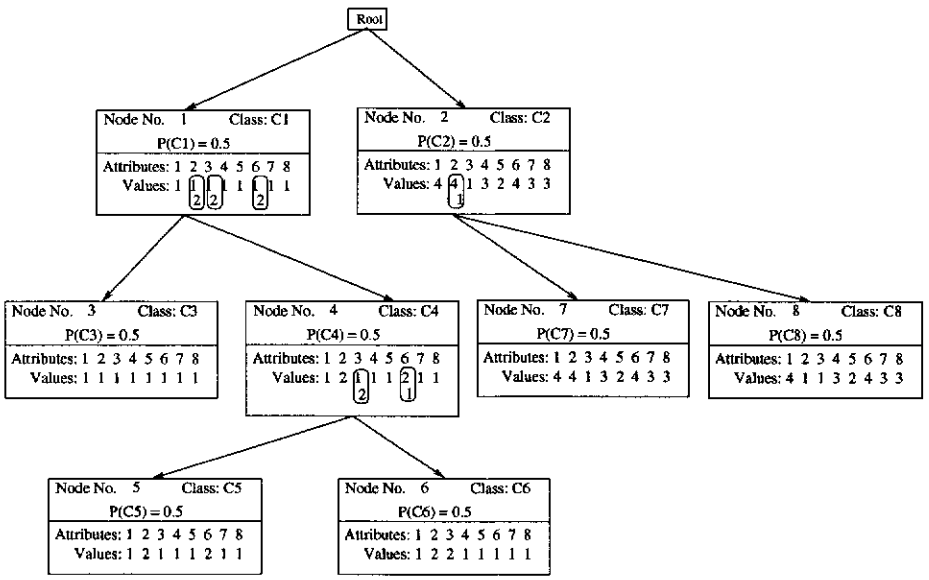


Figure 3.14: The COBWEB structure after 5 instances.

3.4.2 Results

We compared ILF with UNIMEM on several sets of data that we have generated to determine our system's performance in building concepts and tracking concept

drift. The reasons why we compare our system with UNIMEM only (though our system is in some ways similar to COBWEB) is because UNIMEM uses some form of generalising and forgetting whereas COBWEB does not. Hence we consider it to be the more appropriate system to deal with concept drift (needs forgetting) as well as with data consisting of a large number of features (needs generalisation).

We conducted tests on seven sets of data generated based on simple spatio-temporal data of four labelled objects positions over two frames. The data was generated based on the typical queries encountered in the American Football application that motivated the development of ILF.

The sets contain 120 to 160 instances in total and represent the relationships between three labelled objects at two different time instances. Each set contains instances which belong to one of three concepts/classes and each concept has between four and ten variants. Figures 3.15 and 3.16 show for class 7 the dominant concept, 5 variants and 4 new variants. The sets of instances consist of two training stages: a training stage with no concept drift (60 instances) and a training stage with concept drift (60-85 instances). In the first stage only the dominant concept and the variants are used. For example for class 7 only the dominant concept and the 5 variants were used. In the second training stage the new variants are used to generate the concept drift. Hence in the case of class 7, 4 new variants were used. The new concept developed at the end of the training is shown as last entry in Figure 3.16.

As detailed in the previous section, given concepts A and B, concept A is defined to be a variant of concept B is the difference between concept A and B is smaller than or equal to a set similarity threshold. The difference is determined by performing a match between the attribute values of the two concepts.

We set the same threshold for a match between a concept and an instance for both our system and UNIMEM at 75% of the features. To remove a feature from the UNIMEM concept, the weight of the feature value would have to be decremented 4 times. For a true comparison, to remove a feature value from ILF the value has also to be decremented 4 times. We compared the concept hierarchies generated by ILF and UNIMEM after 60 instances have been examined (when all three concepts should be well established) and at the end of the set of instances (when one of the concepts should be modified to accommodate the concept drift, while the other two concepts should remain largely unchanged). The results are shown in Tables 3.2 and 3.3.

Each row shows results after training on the corresponding data set with the number of nodes produced by ILF and UNIMEM. The first 7 rows in Tables 3.2 and 3.3 show the number of nodes at the end of the two stages of training. The numbers in the parentheses indicate the number of *image nodes (instance nodes)* built by UNIMEM. The next 7 rows show the number of nodes after the concept

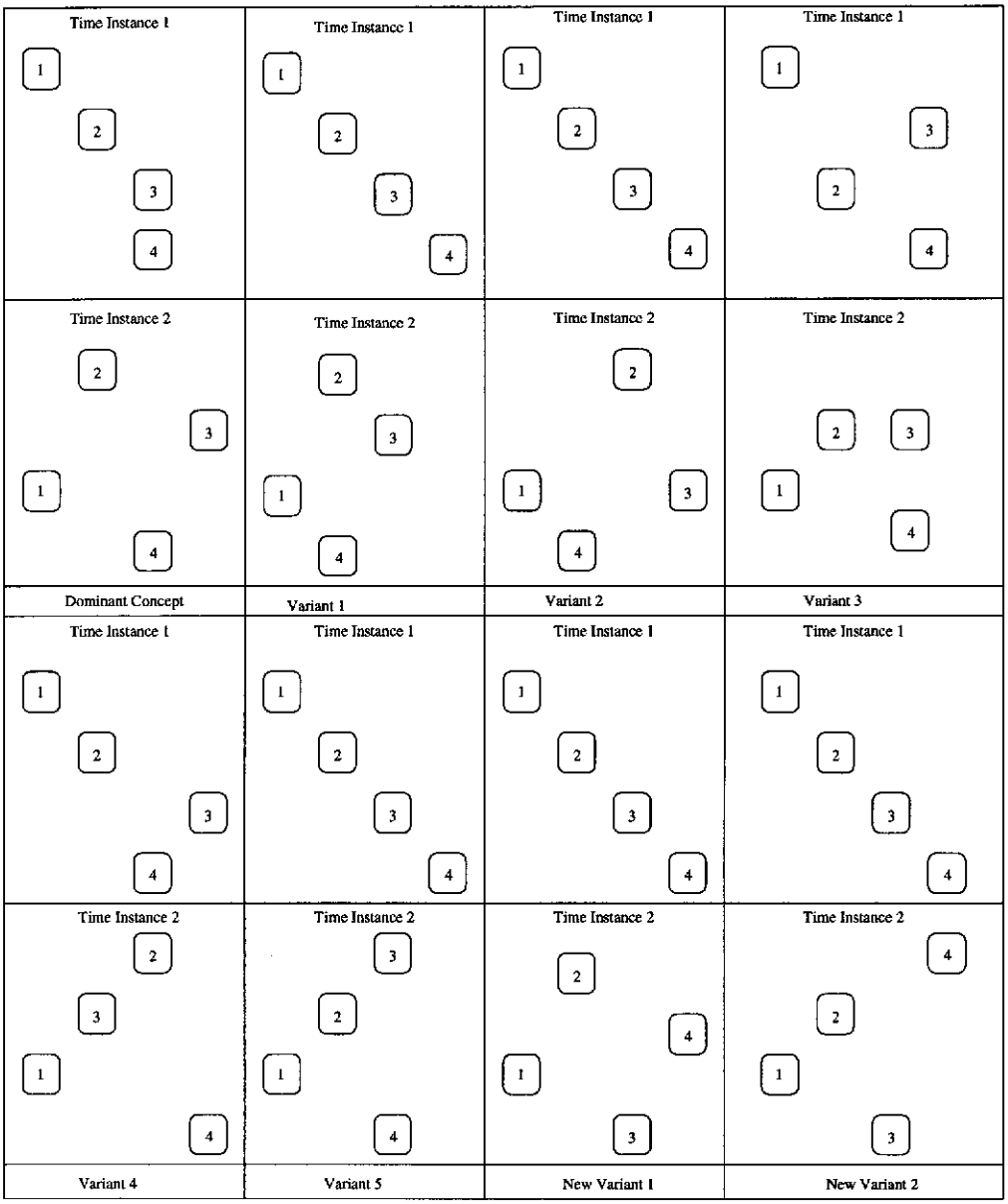


Figure 3.15: Class 7 and its variants.

drift data have been processed. Note that at the end of the training data for the 6th data set, UNIMEM completely removed one of the classes from memory (hence only two were left at the end of the training run).

At the end of the training, tests were conducted using the new variants as well as the new concepts. The results obtained are shown in Table 3.4.

The results show that UNIMEM in general tends to build more nodes and discards data faster than ILF. This results in UNIMEM performing less accurately as is illustrated in Table 3.3, especially for the 6th data set.

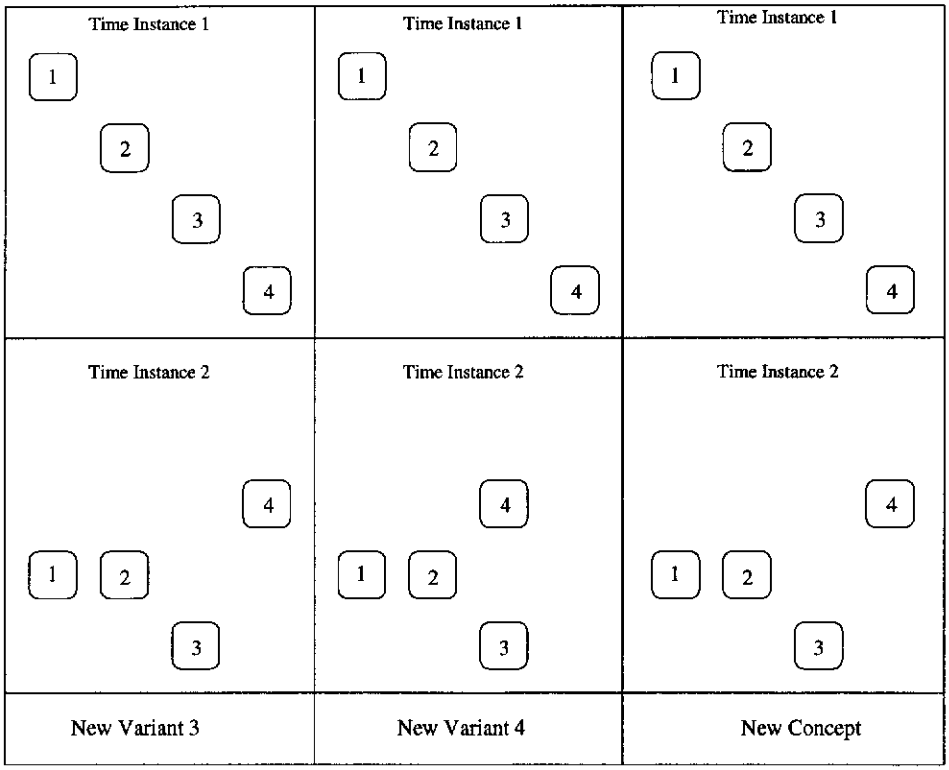


Figure 3.16: Two more variants of class 7 and the new version of class 7.

Data Set	ILF	UNIMEM
1	3	13 (+14)
2	3	5 (+5)
3	3	7 (+6)
4	3	6 (+6)
5	3	8 (+8)
6	3	2 (+2)*
7	3	7 (+8)

Table 3.2: The number of nodes created by ILF and UNIMEM for the 7 datasets after 60 instances.

Data Set	ILF	UNIMEM
1	3	3 (+3)
2	3	4 (+4)
3	3	5 (+5)
4	3	13 (+13)
5	3	18 (+22)
6	3	7 (+6)
7	3	6 (+7)

Table 3.3: The number of node created by ILF and UNIMEM for the 7 datasets at the end of the training sets.

Data Set	ILF	UNIMEM
1	90	38
2	100	36
3	100	48
4	90	88
5	84	41
6	71	20*
7	74	57

Table 3.4: The percentage of correct classifications.

Examples of the hierarchies developed by ILF and UNIMEM for the seventh data set are shown in Figures 3.17 and 3.18. UNIMEM develops 7 concept nodes and 8 image nodes after 60 instances. ILF develops only 3 concept nodes — one for each class (the classes are 2, 5 and 7). When the second part of the data set is processed, UNIMEM removes the original concept developed for class 7 and splits the new variants of class 7 into two groups, new variants 2 – 3 in one group and 1 – 2 into a separate group. A total of 6 concept nodes and 7 image nodes are developed by UNIMEM. ILF on the other hand updates the original concept with data from the new variants and maintains just 3 concept nodes.

The results we have obtained indicate that our system was able to accurately track the changes in the concepts in the sets of data presented to it while producing a very compact hierarchical structure. UNIMEM did not track the concept changes in three of the seven sets because its concept descriptions were split. This resulted in UNIMEM deleting significant data and in developing weaker and inaccurate concepts.

3.5 Discussion

The algorithm we have developed shares the advantages of UNIMEM. It allows multiple nodes to be modified, it allows concept overlapping (showing similar attributes) and it allows data to be pruned from the conceptual hierarchy.

ILF builds concepts that are made up of all observed features as COBWEB does. It also merges concepts similarly to COBWEB (find the best two candidates and if there is enough evidence then merge the concepts).

The algorithm we have developed has several advantages over both UNIMEM and COBWEB. The first advantage is that we use a more refined (and we believe more appropriate) type of forgetting which is applied locally and only when there is enough evidence to justify the update process.

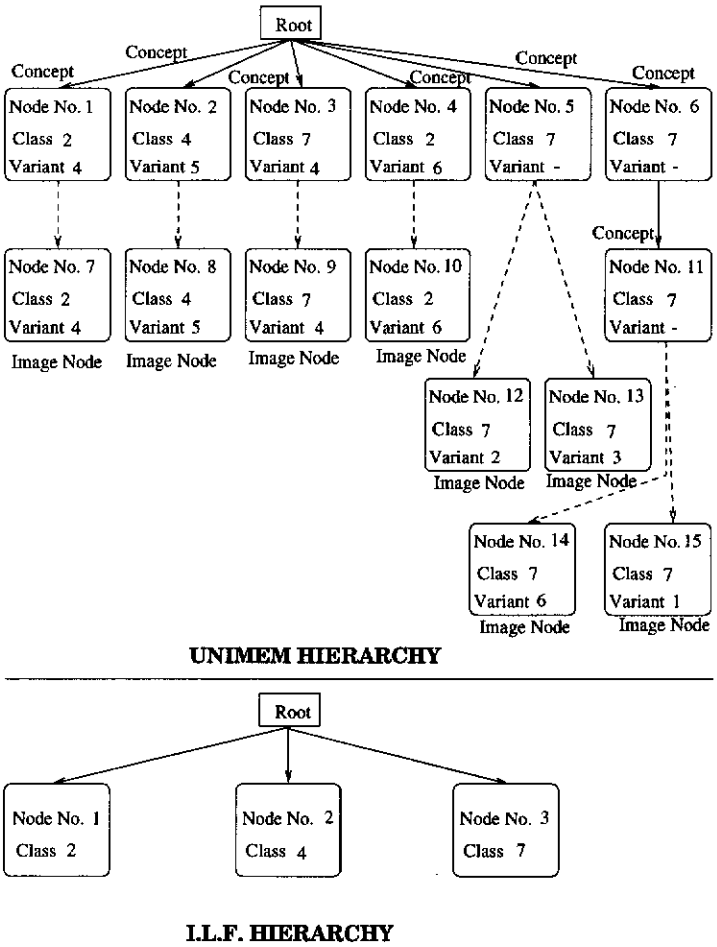


Figure 3.17: The conceptual hierarchies developed by UNIMEM and ILF after 60 instances.

UNIMEM *forgets* in a very simple manner. To do this it uses a feature value increment-decrement mechanism. Each time a new training instance is seen, the concepts at each level in the UNIMEM hierarchy will undergo a feature value update (either increment or decrement). Essentially each new instance seen affects all concepts so the effect is not localised.

COBWEB is a full memory system (stores all the instances) and therefore it *does not forget* any of the instances given as input.

Unlike UNIMEM and COBWEB, our algorithm uses an advanced type of *localised* forgetting which updates the feature values at several levels using the *evidence gathered from the training data.* We have implemented an aging procedure involving three levels of data values which simulate short-term memory (set of facts just observed and which can be discarded if not reinforced), medium-term memory (concepts already reinforced but not up to the level justifying permanent storage and which can still be forgotten) and permanent memory (concepts which

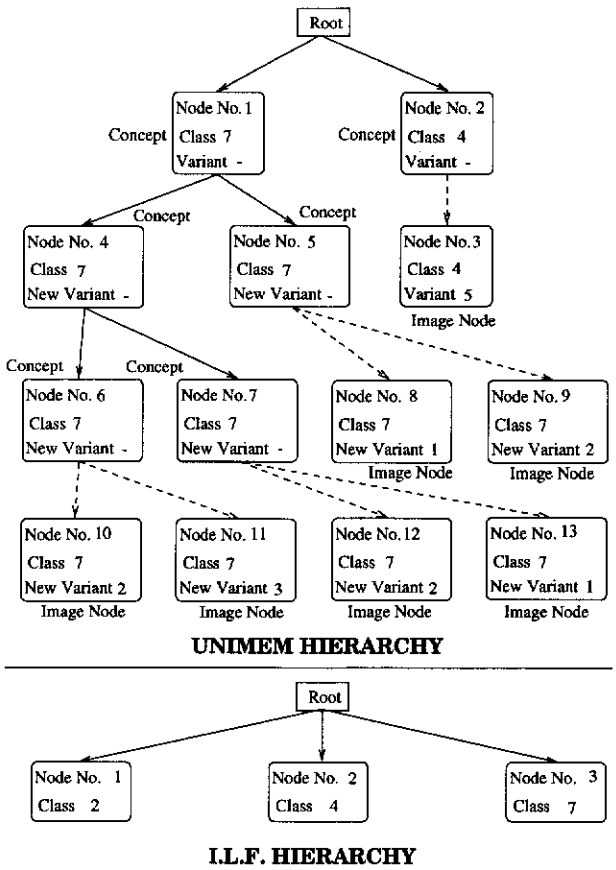


Figure 3.18: The conceptual hierarchies developed by UNIMEM and ILF at the end of the data set.

have been reinforced so many times justifying permanent storage and which cannot be forgotten). Each level of data values involves a different type of value re-enforcement. Also the concepts are updated *only* if there is evidence to do so. One training instance affects *only* the concepts which are determined to be similar to it so the update is done locally.

The second advantage is that concepts developed by ILF are *compact* since *no instances are stored, do not become fragmented* and, are well suited for domains involving large numbers of attributes and training instances. This also enables fast and efficient access to the concepts developed. Unlike our algorithm, both UNIMEM and COBWEB *store instances* in their concepts and as a result the concepts developed require more memory and consequently are less well suited to deal with data intensive domains. Also by storing instances, the concepts built by UNIMEM and COBWEB, in many cases tend to become fragmented. Furthermore since the hierarchy is more compact, the system requires less time to access and update the concepts (when necessary).

ILF performs better than UNIMEM and COBWEB when attempting to track

concept drift. UNIMEM and COBWEB develop concept hierarchies which *store* training instances. In domains where a concept has several similar variants this leads to the concept becoming fragmented and this makes the task of tracking the concept drift more difficult. UNIMEM implements forgetting by updating the concepts regardless of whether they match or not and, this leads to situations where some concepts are modified when there is no need to do so. Hence relevant data is pruned from the tree. COBWEB does not forget so the system updates its hierarchy based on all the data it has seen. The problem of concept drift involves tracking the changes in the target concept and performing the required concept modifications to keep the concept consistent with the incoming data. When considering all the data as COBWEB does, the changes in the target concept become less obvious (essentially a window based approach is more appropriate) and so keeping track of the concept drift becomes a more complex task.

3.6 Summary

In this chapter we have presented the incremental learning algorithm: ILF. The algorithm generates compact conceptual hierarchies, uses an evidence based forgetting method to prune old or irrelevant data and tracks concept drift. When comparing ILF with two similar algorithms: UNIMEM and COBWEB, ILF performs better in terms of classification performance, memory usage and tracking concept drift. The algorithm has been used in two applications: an American Football querying system and a cricket learning/classification system. In the next chapter we cover the first application — American Football.

Chapter 4

N.F.L. Application — Play Model

4.1 Introduction

The first application of ILF is American Football. American Football was chosen because it is a highly structured game and thus it allows accurate models of plays to be built using a combination of spatio-temporal and background knowledge. This is not possible for other sports such as soccer and rugby as they are unstructured — no predefined pattern exists that can be modelled.

This application involves the recognition and learning of American Football plays using information from two sources: natural language commentary and video. The main benefit of combining image information from the American Football tapes and natural language commentary is that it allows for information to be fused from both sources. Thus, the play classification is not dependent on a single source of information. Furthermore it allows semantic understanding of video at a higher level than would be possible by looking at video images alone.

There are many levels of complexity at which this can be done. At the simplest level, different types of information are extracted from the text and the video, and these are combined in establishing the meaning. At more complex levels, the information extracted from the text can be used to constrain and improve image analysis, and vice-versa, analysis of events in the visual data can be used to better analyse the text. In other words, the second system is more complex as it examines the interconnection of two complex subsystems: text and image analysis for understanding semantic content of images.

Combining natural language processing with video data analysis offers great ben-

efits in determining the content of video. These benefits can be further enhanced by combining them with learning. This is because, once the combined processing of the types of data has been optimised the performance of the system cannot be improved further. Moreover, the system does not have the ability to deal with completely new situations or with plays in which the attribute values change over time.

Learning has additional benefits. If the system has the ability to learn then it is capable of continuously improving its classification performance as well as dealing with new examples. Several methods of learning have been developed but the one that is best suited for real world situations is *incremental learning* which has been used in the ILF (incremental learning with forgetting) algorithm (see chapter 3).

In this application, ILF combines context knowledge, attribute selection and the ageing of knowledge to keep the model representation of the plays consistent with the information extracted from American Football video tapes.

In this chapter we cover the data structure used to represent and store the *symbolic descriptions* of the American Football plays. The layout of this chapter is as follows: in Section 2 we describe a number of systems that use natural language processing and image analysis which are relevant to the work described in this thesis. Section 3 presents work relevant to the American Football application in the field of object tracking. The model used to represent American Football plays is shown in Section 4, whilst in Section 5 we describe the hierarchy used to store the play models. The work described in this chapter is only a part of the American Football application of ILF. The following two chapters cover the labelling of the players in the initial setup and the play model recognition algorithm.

4.2 Systems that Combine Natural Language and Image Analysis

Systems that have connected natural language and image analysis are relevant to the work in this thesis as we combine natural language processing with image processing in the American Football application. A relatively small number of attempts have been made to connect image analysis and natural language and in this chapter we describe some of the best known systems.

The VITRA (VIsual TRANslator) [50, 51, 6, 8, 9, 57, 36] is one of the few projects that tries to integrate computer vision and the generation of natural language expressions for the description of image sequences. The domains investigated in the VITRA project have been *traffic scenes* and *soccer sequences.* The approach used in VITRA was based on concurrent image sequence evaluation and natural

language processing which was carried out on an incremental basis. A system named SOCCER [51, 103, 102, 101] was developed, which describes short image sequences of soccer games. The listener is assumed not to be watching the scene, but to have prototypical knowledge about the static background. The system does not require that all data be inputted at once.

The system has an incremental event recognition mechanism (as described in [50, 49]) that analyses incoming geometrical data for the dynamic objects in the scene and provides information in propositional form about the events occurring at the moment. A selection/linearization component then selects the relevant propositions, orders them and passes them on to the encoding component of the system, which processes and transforms non-verbal information into an ordered sequence of words. The output is either written or spoken German. As the scene is described continuously, temporal aspects such as the duration of speech generation were taken into consideration when coordinating event recognition and language generation.

The events that took place in the image sequences were described conceptually using *event models*. The event models were essentially a form of prior knowledge representation about typical occurrences in a scene. The recognition of an event occurring in a particular scene involved the instantiation of the respective generic event model.

SOCCER has the assumption that the input images are relatively noise free and that accurate and detailed data about the players can be easily extracted from the input images (which is hardly the case in real world scenarios which involve multiple objects).

The second system developed as part of the VITRA project that combines image processing with natural language processing is CITYTOUR [7, 6, 9]. CITYTOUR is a German question-answering system that simulates aspects of a fictitious sightseeing tour through an interesting part of a particular city. The questioner is assumed to be sitting in the tour bus, and the system generates answers to the questions based on the current position of the observer — the system uses the position to determine the objects of importance in the scene and what parts of the objects are visible to the observer.

The NAOS (Natural Language description of Object movements in a Street scene) system presented by [86] also attempts to connect computer vision and natural language. The goal in this project has been to derive a natural language description of traffic scenes. The approach in this project is different from VITRA in that recognition of the scene is done after all the visual data is available (in VITRA the descriptions are generated as the data becomes available).

A more recent project that combines natural language understanding with image

processing is Informedia: News-on-Demand [128, 45]. The Informedia [55, 56, 23, 99, 63] digital video library project at Carnegie Mellon University is creating a digital library of text, images, video and audio data available for full content search and retrieval. News-on-Demand [47, 46] is a fully-automatic system that monitors TV, radio and text news and allows selective retrieval of news stories based on spoken queries. The user may choose among the retrieved stories and play back news stories of interest.

The system uses three broad categories of technologies: text processing, image analysis and speech analysis.

Text analysis looks at the textual representation of the words that were spoken, as well as other text annotations. These may be derived from the transcript, from the production notes or from the closed-captioning that might be available. Text analysis can work on an existing transcript to help segment the text into paragraphs. The analysis of keyword prominence allows us to identify important sections in the transcript. Currently there two main techniques used for text analysis.

- If a complete time aligned transcript is available from the closed-captioning or through a human-generated transcription, natural structural text markers such as punctuation can be used to identify the news story boundaries.
- To identify and rank the contents of one news segment, TF/IDF (term frequency/inverse document frequency) is used to identify critical keywords and their relative importance for the video document [98].

Image analysis looks at the images in the video portion of the MPEG stream. This analysis is used for the identification of scene breaks and to select static frame icons that are representative of a scene. Image statistics are computed for primitive image features such as colour histograms, and these are used for indexing, matching and segmenting images.

CNN-AT-WORK [21] is system that is similar to News-On-Demand but is based on a transcript of video instead of speech recognition. The system uses a special digitizer that encodes and sends the video information in the INDEO compressed format. The users can store headlines together with the associated video clips and retrieve them at a later date.

These are some of the systems that combine natural language and image analysis. In the following section we describe a number techniques that are used in object tracking.

4.3 Object Tracking Techniques

4.3.1 Simple Object Tracking Techniques

The problem of object tracking is relevant to the work presented in this chapter as the American Football requires a method of tracking players during the plays. Several methods have been developed over the years to track objects in images. Among the best known techniques are correlation tracking, structural correspondence, space-time based tracking and motion tracking.

Correlation tracking [15] is performed by matching a region of one image to some region in the next image. An initial template is generated from the image in which the object is detected, and then the template is used to determine the position of the object in the subsequent frames.

Structural correspondence [59] is also based on the concept of detecting features that can be used to track the object. However, this method does not use the pixel values directly. One the most frequent methods of structural correspondence is the edge-line technique. The edge features in consecutive images are matched based on a chosen property of the edge such as line length or orientation. Motion is another type of feature that can be used. In this case, the difference is obtained by subtracting the frame at each time step with a known background frame and then subsequently thresholding it.

Space-time based tracking [59] is generally used when objects change shape and position slowly between frames. Several frames are analysed together as a unit of data from which the location and motion characteristics can be determined using edge or curve detectors.

Motion tracking is a technique that generally uses some motion model [75, 113, 54, 106, 16, 100] to supplement the tracking information extracted from images. One of the most common motion models is based on the assumption that the object is moving with a constant velocity or a along a smoothly changing path. One method that uses a smooth motion model is described by Huang and Wu [54]. The trajectory of a moving object in an image sequence is determined using a path and shape coherence constraint. Another algorithm based on the smooth motion model is presented by Sethi and Jain [106]. The method uses an extended frame sequence to establish the correspondence necessary to obtain the trajectories of objects.

Another approach in motion tracking is using optical flow [53, 84] which uses the distribution of apparent velocities of movement of brightness patterns. In this way the optic flow motion methods can be used to predict the position of the object without explicitly matching image features.

The problem with the tracking methods described above is that they do not perform well when applied to complex scenes in sports (such as American Football). This due to the unpredictable nature of the movement and shape of the players. In general football players move quickly, change direction unpredictably and often collide with other players. In addition, players are very difficult to model since they are essentially non-rigid objects and shape can change significantly (player standing vs player in a crouch position).

4.3.2 Object Tracking Techniques using Context Knowledge

Attempts have been made to improve the performance of tracking methods by using domain knowledge such as *geometrical model knowledge* or *context knowledge.*

Methods using geometric knowledge use rigid models of the objects to track them in image sequences. One of the most popular applications of geometrical model knowledge has been vehicle tracking (geometric models for cars are relatively easy due to the rigid shape of the object involved) and several applications that attempt to track cars in traffic scenes have been developed such as the system presented in [64].

Contextual information is a very powerful tool that can aid in the selection of image processing techniques [111, 89, 12]. This is demonstrated by Fu [34] with the system named SHOPPER. The system uses the context to guide its search when attempting to find objects in a scene — the current context determines which one of three different search routines is to be applied to the images. When SHOPPER attempts to find products like pancake mix it uses contextual information such as *cereals are grouped together* and *cereals sit on shelves.*

A method for tracking players in American Football greyscale images that makes use of context knowledge is described by Intille [58, 59]. In his work, Intille proposes the use of "closed-world" [60, 61, 14] analysis to incorporate contextual knowledge into low-level tracking. *A closed world is defined as a space-time region of an image where contextual information such the number and type of objects within the region is known.* The internal state of the closed world is unknown, e.g. the positions of the players, and must be determined from the visual data. The visual routines for computing the internal state of the world are selected using context restricted domain knowledge and any information that has already been learned about the state within the world from previous processing. There are two types of entities in the closed-world: objects and image regions. The objects are the physical things (such as the players) that the system must monitor to develop a useful interpretation. The knowledge of the domain dictates how

objects interact and is independent of the scene. The image regions are simply image data, or the objects projected onto the image plane. Though the tracking method in general produces good results, it is still likely to fail in situations where the player models are imprecise, spatial resolution is low or when the object tracked is very close in appearance to some nearby feature.

Condor [112, 111] is another system that makes use of context knowledge and uses the output of many simple vision processes and local context in the scene for recognition of outdoor imagery. Condor treats objects as component parts of larger contexts from which they cannot be separated, that is objects do not have an independent existence.

Overall, tracking players consistently in complex scenes such as American Football is still a largely unsolved problem. Progress has been made in the last few years by combining contextual knowledge with image processing as demonstrated by Intille. Recently a new approach is being tried to solve the problem of player tracking by completely abandoning image processing. This involves the use of tags on players to determine their positions on the field and has already been tested in soccer games.

This section presented techniques used to track objects in dynamic scenes. The work presented has great relevance to our American Football application that involves tracking players during the plays. In the following section we describe the first part of the American Football application that connects natural language with image analysis.

4.4 Play Model

American Football plays are very complex and involve a well defined set of movements and actions, with each player attempting to complete his task (which can vary from protecting the quarterback to receiving the ball). To recognise the plays we have combined expert knowledge, domain knowledge (game rules), spatial knowledge (player relationships) and temporal knowledge (action sequencing in individual plays) into a model which identifies a particular play.

The play model description used to represent an American Football play has several components which describe the play action in terms of *player significance*, *player relationships*, *player movements*, *player actions*, *player positions* and the *temporal sequencing* of the *player actions and movements*. The significant characteristic of the *play model* representation is that it is mainly made up of *symbolic* data.

The play model has the following features:

- The **play class** — there are three classes of plays in American Football that are defined by the main action that takes place in the play. In a play each one of the players in the team performs some action but generally there is only one action of particular importance for the play. For example, in an offensive pass play the main action is the pass between the quarterback and the receiver. The possible play classes are defined by set A where: $A = \{offensive, defensive, special\}$.

- The **significant players and actions in the play** — each play involves significant players performing predefined tasks. We define as significant the players whose actions are very likely to have a major impact on the main action and outcome of the play. For example, in a pass play, the players we consider to be significant are the quarterback and the receivers since they are the players involved in the main action of the play. The significance of the players can range from low to very-high depending on how much impact they are likely to have on the play. The player significance is determined by the expert and is defined by set B where: $B = \{low, average, high, very\text{-}high\}$.

  An example of the significance of the players involved in a typical Pass Play is shown in Figure 4.1.

Position	Significance	Action
Quarterback	Very-High	pass-ball
Receiver	Very-High	catch-ball
Wide-Receiver	Very-High	catch-ball
Guard	High	block-player
Corner-Back	Average	block-player
Blocker	Low	block-player

Figure 4.1: Player Position and Significance for a typical Pass Play.

  In an analogous manner, the player position, significance and action is defined for all plays. The possible actions are defined by set C where: $C = \{pass, toss, give, hand, catch, get, receive, fake, tackle, block, push, grab, intercept\}$.

- The **player combination** — each play (offensive/defensive/special) requires a specific combination of players. For example an offensive pass play might have the following player combination: 4 blockers, 2 corner-backs, 2 receivers, 1 guard, 1 snap and 1 quarterback.

- The **temporal sequencing** of the actions in the play — in each play certain actions take place at specific times and in a certain order. For

example, in a running play the quarterback will pass the ball immediately to the running back at the same time as the blockers tackle the opposing defence line and once the running back gets the ball, he waits for a breach to appear in the opposing defence line in order to start his run. In a pass play however the quarterback does not pass the ball immediately, instead, he waits for the receivers to break free of their markers and then gets rid of the ball. The temporal representation of the play model is based on a set of simple movements combined with a play action. The *play action* is the main action in the play already defined in the first part of the play model (see significant players and actions definitions). The *movement* describes the type of movement of a given player *at an instance in time.* A player can either *move to a position P* or *make a turn.*

The quantitative value of position P towards which the player is moving is not important — we use it in the model descriptions to indicate players are moving towards separate positions.

The movements *turn-left* and *turn-right* are relative to the direction in which the football player is facing.

To determine when the player is *turning* or *moving* to a new position we segment the video frames using a colour based segmentation method. The reason for selecting a colour based segmentation method is that the teams in American Football have distinct colours which can used to identify/segment the players in the video frames. The coordinates of the centroids of the players of interest to the system, are extracted by hand and complied into a list used to:

- Generate the spatial relationships between the players.
- Generate the temporal sequencing of the player actions.
- Generate a symbolic description of the movement of the player over the duration of the play in terms of *turns* and *moves.*

The coordinates extracted from the image are rectified using the grid lines on the field. For each frame, we manually segmented the grid lines on the pitch and we used a camera model to transform the coordinates so the image plane is parallel to the pitch. This process is shown in Figure 4.2.

The movement of the player is tracked over each frame in the video sequence and a change in the movement that occurs indicates a turn. The change (if any occurred) in the movement of the player is computed by taking the difference in the coordinates extracted from the video frames. The change has to be reinforced for at least 3 consecutive video frames and be above a threshold to indicate a turn. An example is shown is Figure 4.3 where the heading of the player was recorded for each one of the 18 frames — the player made three turns, at frames 3, 9 and 15.

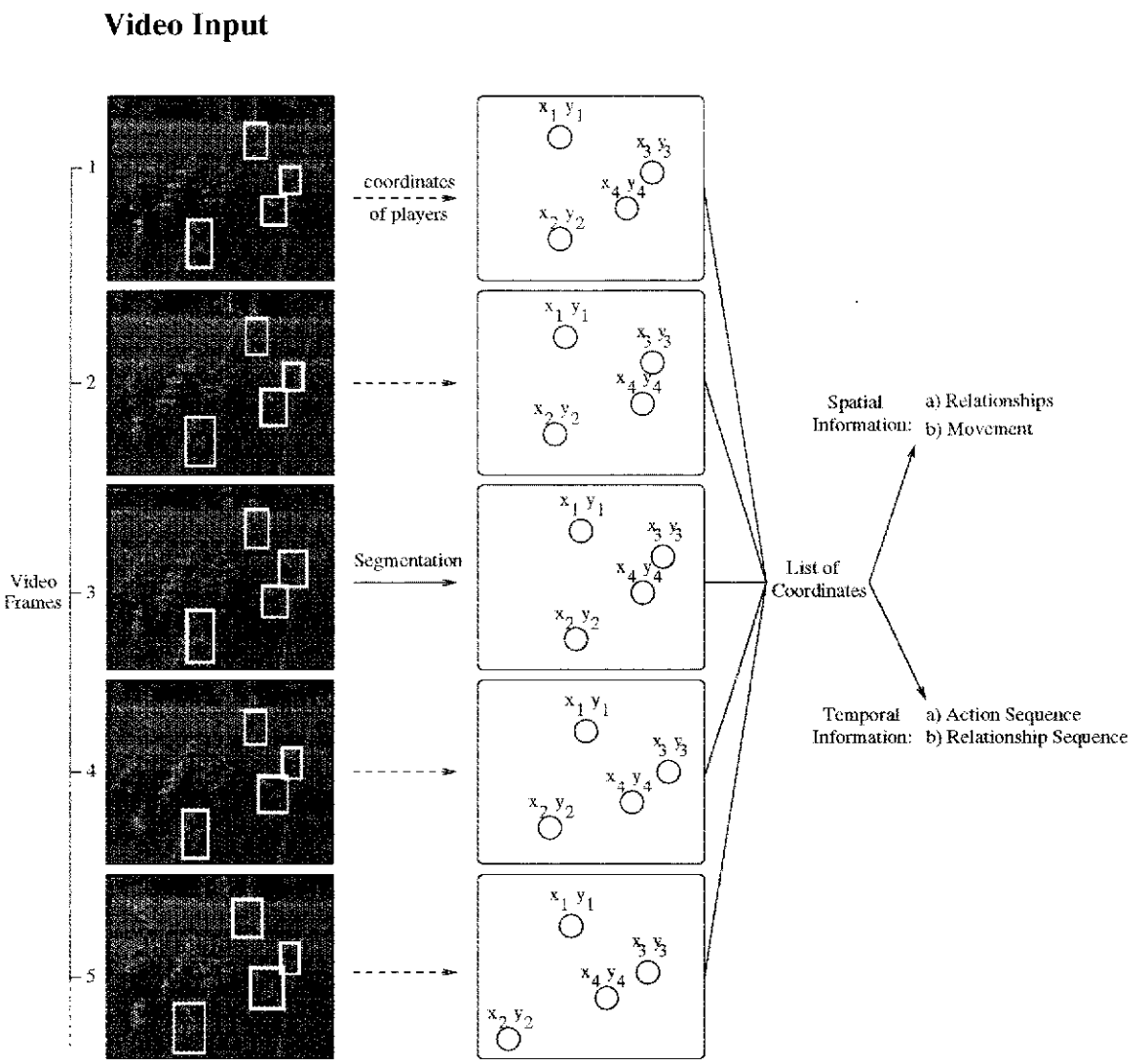


Figure 4.2: The video frames are segmented and the coordinates of the players are compiled into a list. The list is then use to generate spatio-temporal relationships of the players as well as the player movement symbolic description. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

The player movement in Figure 4.3 is converted into *moves* and *turns* as shown in Figure 4.4 and can be described as: Move (from position A to B), Turn-Left, Move (from position B to C), Turn-Right, Move (from position C to D), Turn-Left, Move (from position D to E).

The possible movements of a player are defined by the set D where: $D = \{move, turn\text{-}left, turn\text{-}right\}$.

The order and timing of the combined movement of the players is described with the help of five of Allen's temporal primitives [38, 120, 30, 3, 4, 5, 42, 35, 79]. The set of primitives used in the temporal representation is defined

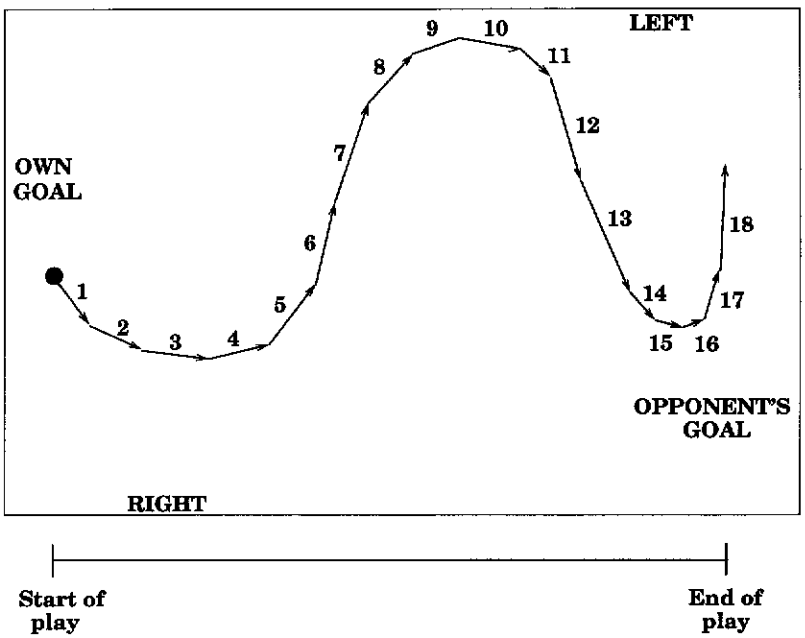


Figure 4.3: The heading of the player at each time instance.

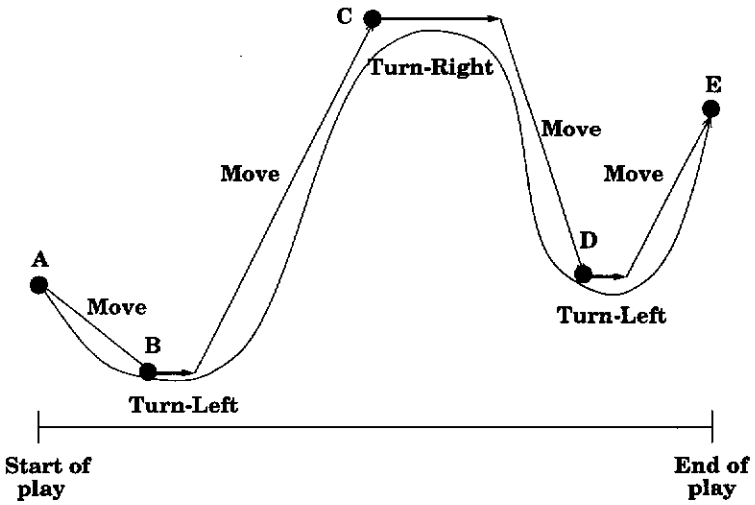


Figure 4.4: The movement of a player over the duration of the play can be described as sequence of Moves and Turns.

by the set E where: $E = \{before, after, during, meets, equals\}$.

We define the temporal models in terms of time instances. We define a time instance to be one in which a change in a player's actions (*turn* or *move*) occurs. The time instance definition is very important as it allows the system to compare two *separate* but *similar actions* without considering the number of frames involved. For example, consider two player movements shown in Figure 4.6. In Play A, player 1 moves to position P1, turns left and then moves to position P2. Player 2 in Play A moves to position P3,

turns left and then moves to position P4. Therefore, in play A, six actions occurred (two move actions and one turn action for each of the players). Therefore the movement can be described in terms of six times instances (one time instance per action). The order in which the actions occur is very important. To maintain the temporal order of the actions we analyse the duration of the actions. Player 1 moved to position P1 over 12 frames, turned left in three frames and then moved to position P2 in seven frames. Player 2 moved to position P3 in 10 frames, turned right in four frames and then moved to position P4 in 12 frames. Both players move out in the same time. Player 2 reaches position P3 while player 1 is still moving to position P1. Therefore the first time instance is set at the end of the movement of player 2 to position 3. Player 1 reaches position P1 while player 2 is turning right so the second time instance is set when player 1 completes his movement to position 1. As player 1 is turning left, player 2 finishes his turn right and this sets time instance three. The rest of player movement is processed in a similar way. Now consider the 2 player movements shown in Figure 4.6 – Play B. The players in Play B perform the same actions as those in Play A but uses more time/video frames. If the system attempts to compare the actions of the players in the two plays, frame by frame, then it will determine that the actions and their temporal ordering do not match, which is incorrect. However, when the system generates a description of the movement based on the time instance definition, it determines that the actions of the players in Play B are the same, use six time instances and occur in the same temporal order. Hence it determines that the two plays match.

The time and general sequencing of the moves by the players used in the temporal models of the plays were recorded from football games. Then using the temporal primitives we derived symbolic text descriptions of the actions of the players such as *player1(move position P2)* ***equals*** *player5(move position P8)*. In English the actions can be described as follows: *player1 was moving to position P2 while player5 was moving to position P8.*

Consider the following example. Assume that we require the temporal description of the actions of *quarterback (Q)* and the left *corner-back (CB)* involved in the pass play shown in in Figures 4.7 and 4.8 (notice that the model has only six time instances — this is because some of the changes in the movement of the players occurs *in the same time* and therefore *share the time instance*).

Both the *quarterback* and the *corner-back* perform very simple movements over the play duration. The *corner-back* moves to the predefined *position P4* while the *quarterback* moves to the predefined *position P9.*

The movements of the *corner-backs* and the *quarterback* are represented by the players which have positions in the temporal model denoted by *CB* and *Q*.

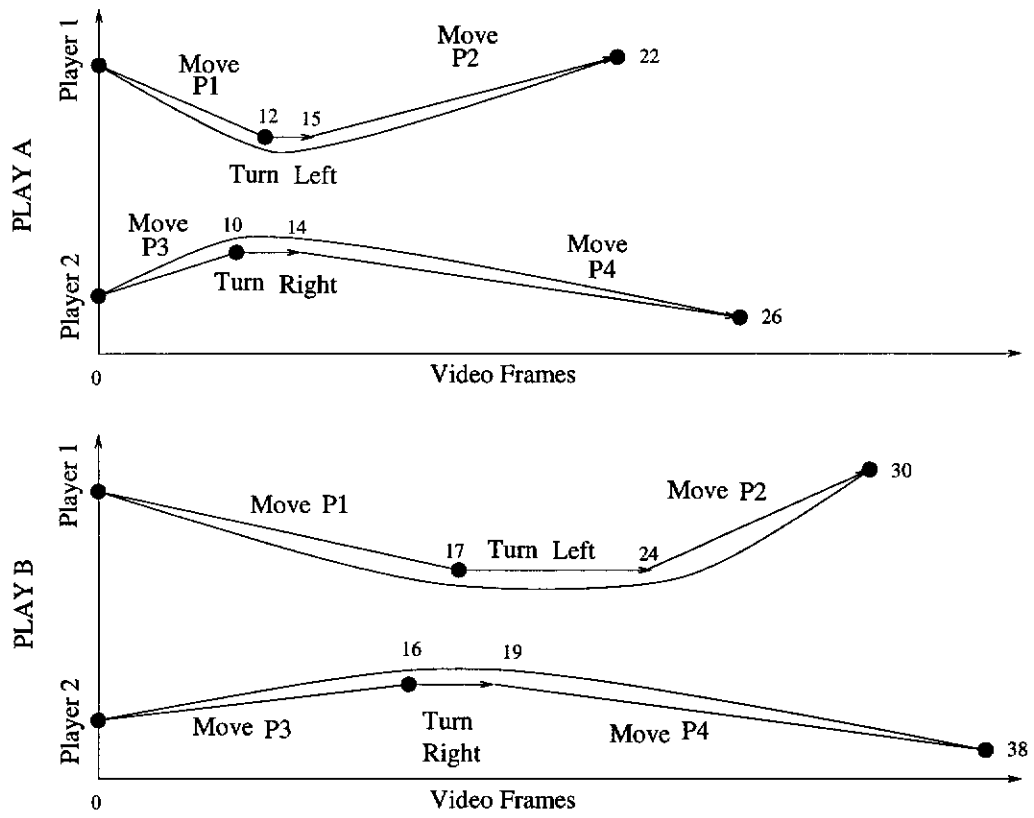


Figure 4.5: Player movements in plays A and B consist of the same actions but the actions in play A occur over a smaller number of frames than in play B.

The movement (as obtained from the timing observed in recorded video) of the *corner-back* finishes before that of the *quarterback*. The temporal actions of the *corner-back* and the *quarterback* can be described as: *corner-back(move position P4)* ***during*** *quarterback(move position P9).*

The English description of the action is: *The player Corner-Back moved to Position P4 while the player Quarterback was moving to Position P9.*

- The **movement of individual players in the play** — players generally have specific movement patterns in specific plays. For example, in a shot-gun pass play, the *receivers* tend to move very fast in a forward direction and across the ground while the *linebackers* move backward protecting the quarterback. In a *Hail-Mary* pass play however, both the *receivers* and *linebackers* move forward straight in an attempt to give the quarterback as many passing options as possible. The movement of the player *over the entire play period* is described in terms of *shape* and *distance*. The shape of the movement indicates whether the movement is **straight** or **contains_turns**. Figure 4.9 shows the movement of a player who makes three turns.

  The distance covered by the player during the play can be either short or long. To determine the type of distance, the system uses the player's

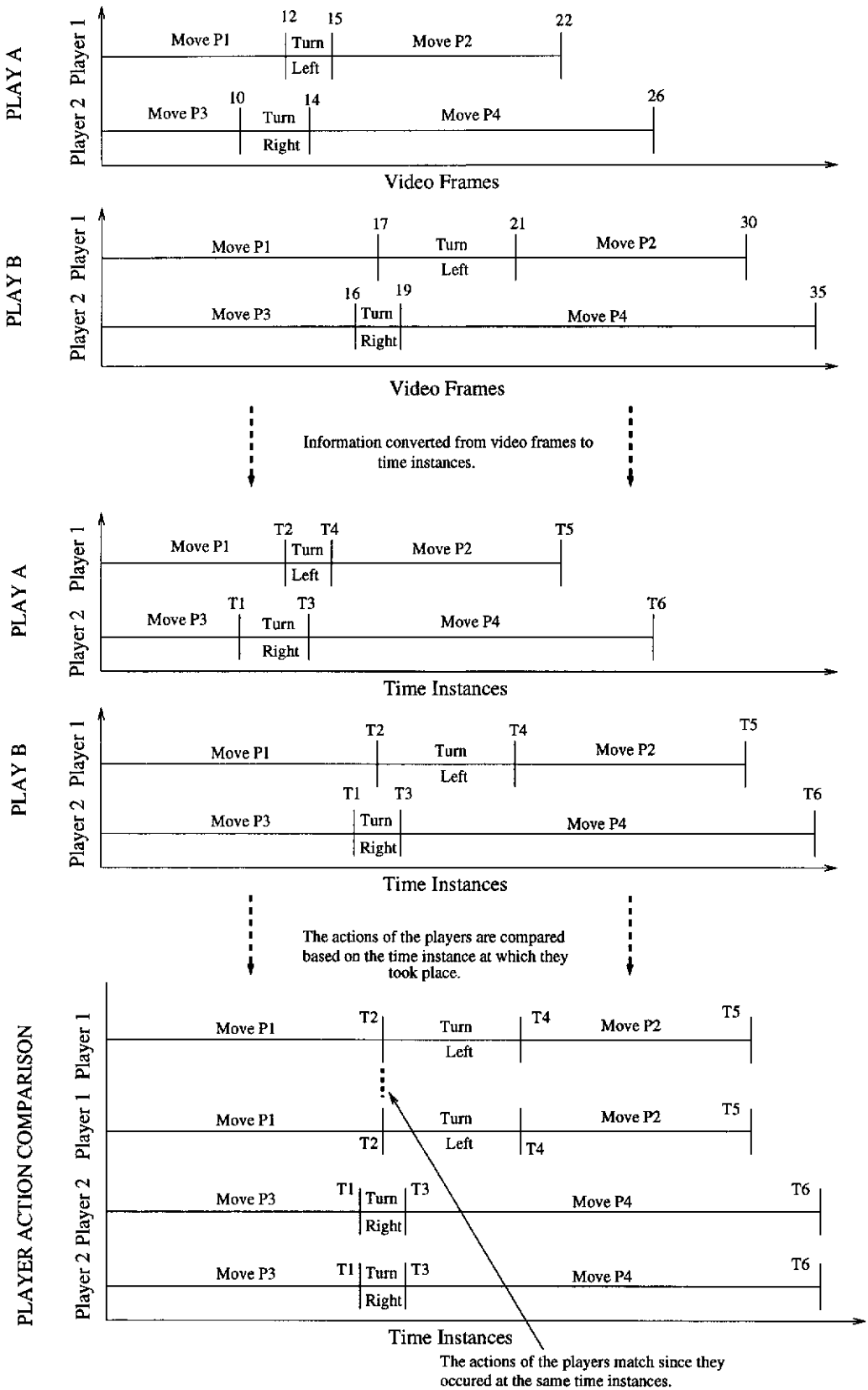


Figure 4.6: The temporal representation of the player actions, in play A and B, using time instances solves the problem of frame based action comparison.

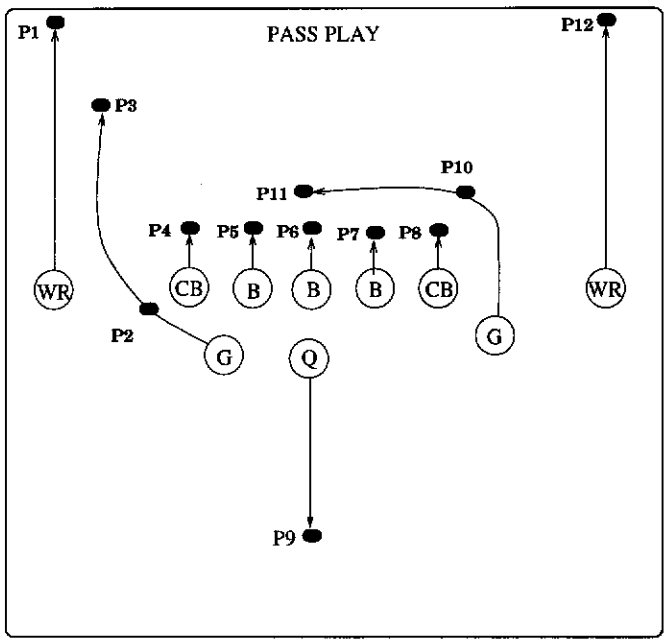


Figure 4.7: Spatial model of a Pass Play.

starting and finishing positions and the position of the defensive line. If the player started *behind* or *on the same line* as the defensive line and finished *in-front* of the defensive line *then*, the distance covered is considered to be *long*, otherwise the distance is *short* (see Figure 4.10). The possible overall movements are shown in Figure 4.11 and are defined by set F where: F ={*short-and-straight, short-and-turn, short-and-many-turns, long-and-straight, long-and-turn, long-and-many-turns*}.

- The **relationships between a player and the other team mates** at different instances throughout the duration of play are described symbolically and derived from four reference points which are assumed to be known at all times. These are the left boundary, the right boundary, the opponent's goal and the team's own goal line. They produce the following six relationships: *in-front, behind, to-the-left, to-the-right, in-line-with-horizontal* and *in-line-with-vertical*. Figure 4.12 shows how the relationships are generated, the positions of two players and the corresponding the relationships generated for these two players. The references are dependent on which way the attacking team is *facing*.

  The possible relationships are defined by the set G where: G = {*in-front, behind, to-the-left, to-the-right, in-line-with-horizontal, in-line-with-vertical*}.

Of the six features described in the play model, the last two (temporal sequencing and the relationships between players) are adding a level of complexity to the play which makes a search based on just the last two features very difficult and time

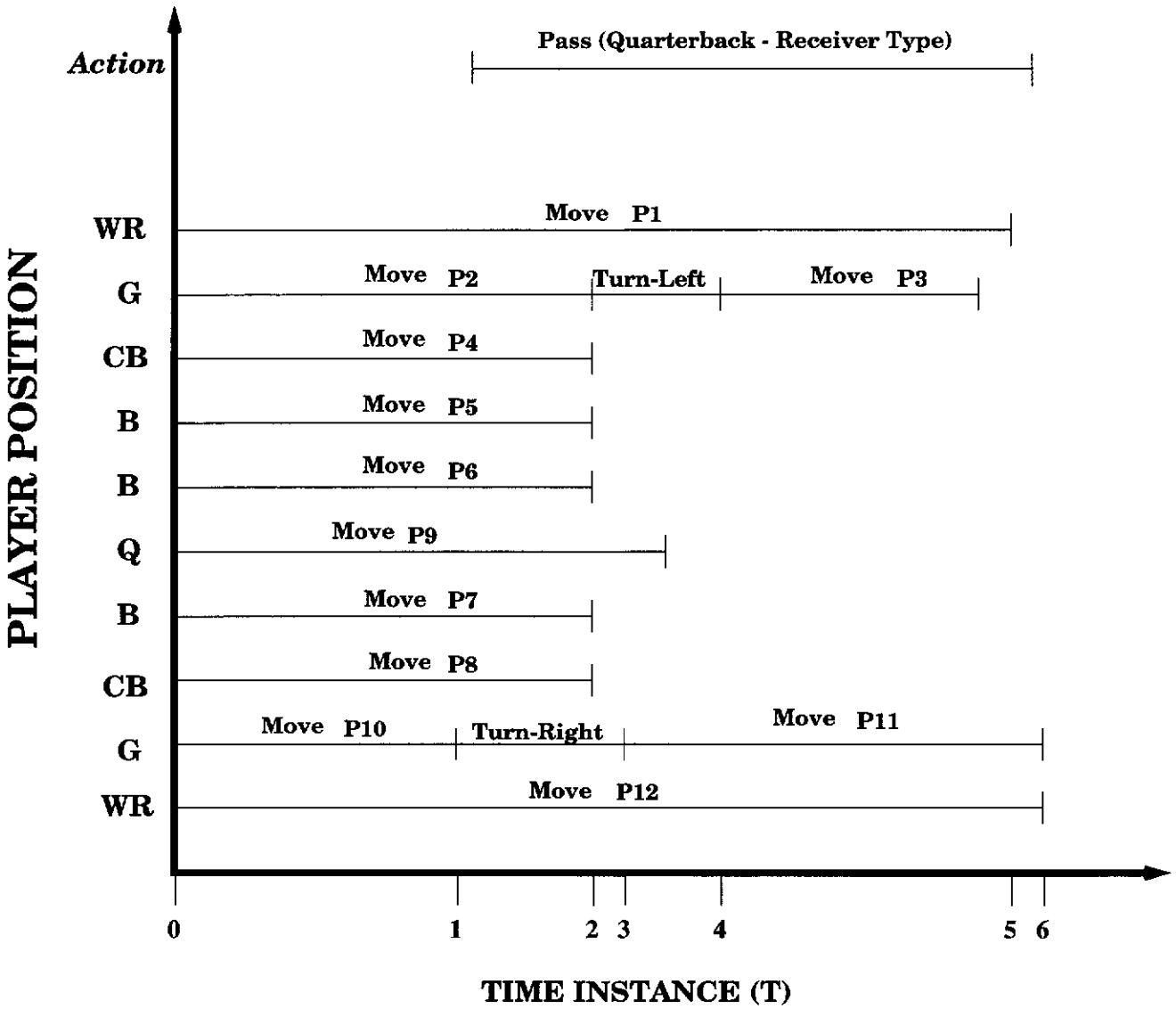


Figure 4.8: Temporal Model of the Pass Play.

consuming. The problem is the representation of actions in temporal terms and the player relationships at different instances in the play. There are so many combinations of action sequences and player relationships that a detailed play description that covers all possibilities is not feasible. We have decided to use only a subset of all the possible actions and player relationships that covers the *actions of the significant players* in the plays. This is because we believe that by capturing only the most important aspects of the play (which always involve the significant players) we can still obtain consistent play model recognition. The schematic of a typical play model is shown in Figure 4.16 while a sample of the symbolic data stored in the model is shown Figures 4.13, 4.14 and 4.15.

To build the play models we have created a database of play information, which

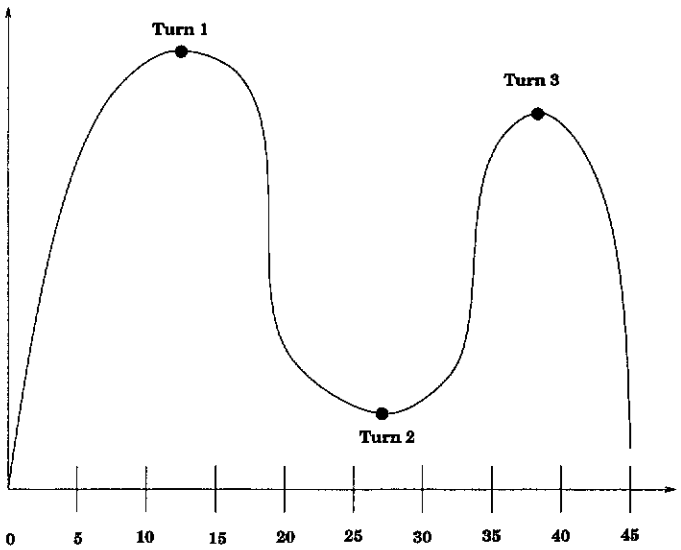


Figure 4.9: Player movement with 3 turns.

includes data about the player combinations, play actions, player significance and player movement over the duration of a "by the book" play. The information about the plays (which has been used to build the text and schematic descriptions of the play models as shown in Figures 4.13, 4.14, 4.15 and 4.16), has been gathered from several sources which include Gridiron play manuals [24, 85, 83], Gridiron tapes [11] and Gridiron computer games [10, 109]. The information about the players (positions and significance) are used to generate the play model description.

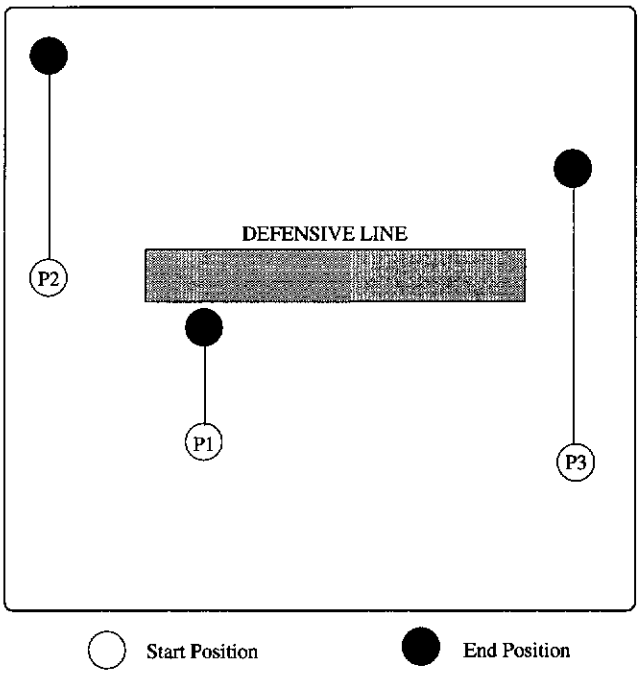


Figure 4.10: Long movement and short movement. The players P2 and P3 have *long movement* since they started behind/in-line with the defensive line and their finishing positions were in front of the defensive line. Player P1 has *short movement* as he started behind the defensive line and finished behind the defensive line.

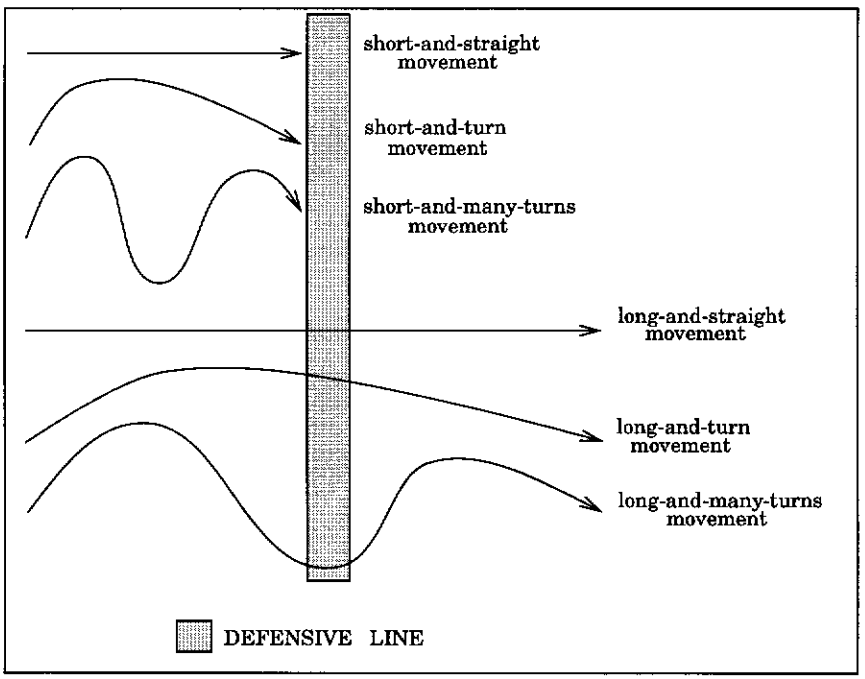
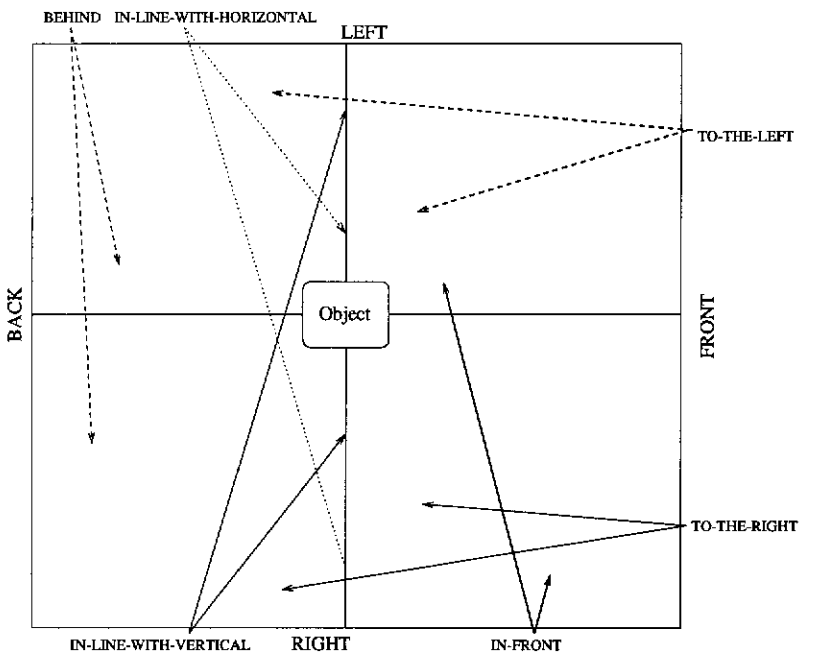


Figure 4.11: The six types of movements of the players.



Relationship	Player	Player
	P1	P2
In-Front-Of	Yes	No
Behind	No	Yes
Left-Of	No	Yes
Right-Of	Yes	No
In-Line-Horizontal	No	No
In-Line-Vertical	No	No

Relationships generated for players P1 and P2

Figure 4.12: The areas used to define the player relationships. In the example shown Player P1 is *in-front* and *to-the-right* of Player P2.

```
Play Name : X-SHOTGUN
=====================
Play Class : OFFENSIVE - PASS
=====================
Players Involved

3 linebackers
2 blockers
2 guards
2 wide-receivers
1 quarterback
1 snap
=====================
Significant Players Position
1) Quarterback
2) Wide-receiver
3) Guard
=====================
Significant Action
1) PASS (quarterback - receiver)
=====================
Player Movement
1) Position : linebacker -> straight-short-no-turns
2) Position : linebacker -> straight-short-no-turns
3) Position : linebacker -> straight-short-no-turns
4) Position : blocker -> straight-short-no-turns
5) Position : blocker -> straight-short-no-turns
6) Position : guard -> long-and-turn
7) Position : guard -> long_and_turn
8) Position : wide-receiver -> straight-long
9) Position : wide-receiver -> straight-long
10) Position : quarterback -> straight-short
11) Position : snap -> straight-short
=====================
```

Figure 4.13: Play Model for a Pass Play.

4.5 The Play Model Hierarchy

The system uses a well defined semi-dynamic hierarchical structure to store the models of the American Football plays. It is semi-dynamic because only two of the three levels in the hierarchy can be updated. The hierarchy shown in Figure 4.17 has three levels: a class level, a condition level and model level (a more detailed figure describing the hierarchy was not included as the actual play hierarchy is too large - 15 megabytes of text).

The class level defines the type of play and no updates are allowed at this level. There are three classes: *offensive*, *defensive* and *special*. The condition level defines the spatial setup of the players at the start of the play and the player combination. Learning at this level is possible as new setups can be encountered or old setups can be updated.

The model level contains the detailed spatio-temporal information about each

```
Action Sequence

OCCUR(shotgun-pass(quarterback receiver-type) T) ->
        [(OCCUR(move(linebacker position1 position2) T1) ->
          OCCUR(move-to(linebacker position3 position4) T2)
          OCCUR(move-to(linebacker position5 position6) T3)
          OCCUR(move-to(blocker position7 position8) T4)
          OCCUR(move-to(blocker position9 position10) T5)
          OCCUR(move-to(guard position11 position12) T6)
          OCCUR(move-to(guard position13 position14) T7)
          OCCUR(move-to(wide-receiver position15 position16) T8)
          OCCUR(move-to(wide-receiver position17 position18) T9)
          OCCUR(move-to(quarterback position19 position20) T10)
          OCCUR(move-to(snap position21 position22) T11)
                    and
                   (T1 during T and
                    T2 during T and
                    T3 during T and
                    T4 during T and
                    T5 during T and
                    T6 during T and
                    T7 during T and
                    T8 during T and
                    T9 during T and
                    T10 during T and
                    T11 during T)
          OCCUR(pass(quarterback receiver-type) T12) ->
               [HOLDS-FOR(moved-from(wide-receiver position15) T13) OR
                HOLDS-FOR(moved-from(wide-receiver position17) T14) OR
                HOLDS-FOR(moved-from(guard position11) T15) OR
                HOLDS-FOR(moved-from(guard position13) T16) AND
                HOLDS-FOR(moved-from(quarterback position19) T17)
                      and
                      ([T8 starts T13] or [T13 during T8])
                      ([T9 starts T14] or [T14 during T9])
                      ([T6 starts T15] or [T15 during T6])
                      ([T7 starts T16] or [T16 during T7])
                      ([T10 meets T12] and [T17 overlaps T10])
          ]
       )]
================
```

Figure 4.14: Play Model for a Pass Play.

play as described in the previous section. The system can perform three operations at this level. It can build a new model if necessary, it can update the spatio-temporal information in existing models, and finally, it can remove models that are no longer of any use.

ILF was used to build the class tree without forgetting. The play model representations were derived using data from coaching books and video. The data did not have any concept drift (hence no forgetting was necessary).

```
Player Relationship

Instance 1  ->

Player Position : wide-receiver(1) IS

TO-THE-LEFT of : wide-receiver2, guard1, guard2, quarterback, linebacker1, linebacker2,
linebacker3, blocker1, blocker2, snap
TO-THE-RIGHT of :
BEHIND :
IN-FRONT-OF : guard1, guard2, quarterback, snap
IN-LINE-WITH : wide-receiver2, linebacker1, linebacker2, linebacker3, blocker1, blocker2
                                .
                                .
                                .
                                .
Player Position : snap IS

TO-THE-LEFT of : wide-receiver1, blocker2, linebacker3, guard1
TO-THE-RIGHT of : wide-receiver2, linebacker1, linebacker2, guard2
BEHIND : wide-receiver1, wide-receiver2, blocker1, blocker2, linebacker1, linebacker2,
linebacker3
IN-FRONT-OF : guard1, guard2, quarterback, snap
IN-LINE-WITH :

Instance 2 ->

Player Position : wide-receiver(1) IS
TO-THE-LEFT of :

 ================
```

Figure 4.15: Play Model for a Pass Play.

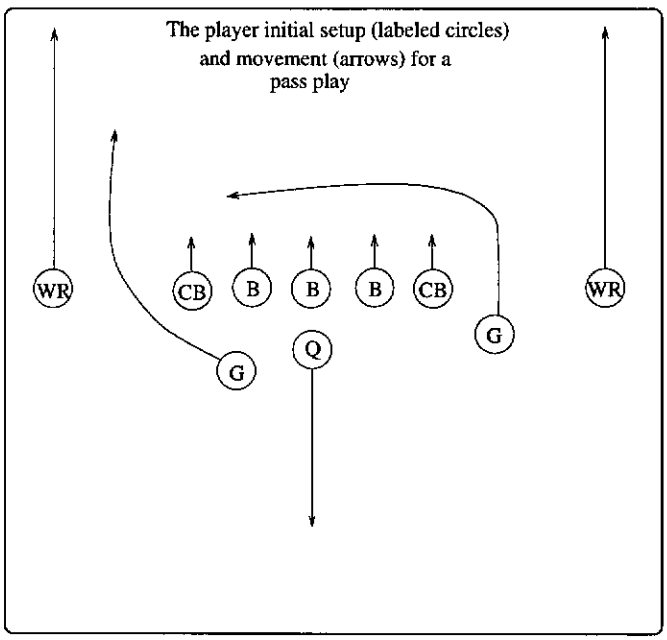


Figure 4.16: Pass Play.

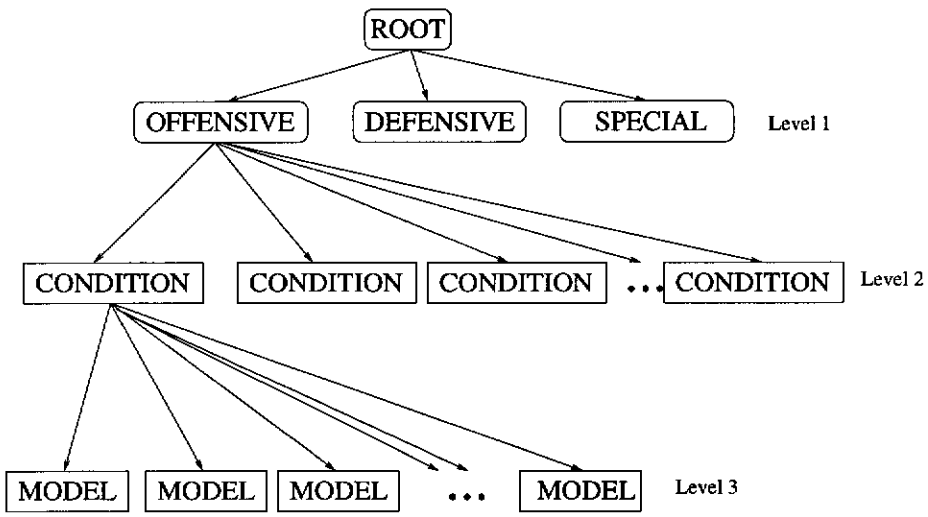


Figure 4.17: The play model hierarchy.

4.6 Summary

In this chapter we covered the knowledge structure used to represent American Football plays. The models contains spatio-temporal information about the play extracted from domain knowledge and expert knowledge. In the following chapter we describe the low level analysis used to determine the labelling of the players necessary to determine the play information described in this chapter.

Chapter 5

N.F.L. Application — Player Labelling

5.1 Introduction

In this chapter we present the method used to label players in the initial setup of American Football plays. In chapter 4 we presented a number tracking of methods used to track objects in a dynamic scenes. However, the object tracking methods presented in chapter 4 cannot track objects in complex scenes such as in American Football games because of the unpredictable nature of the movement and shape of the players. To classify and learn American Football plays requires us to obtain as much information as possible about the movement of the players and their actions. Since we cannot track the players in American Football plays, we combine natural language processing and image analysis to address the problem of player tracking. An important part of our method is *player labelling.* At present there are no systems available to automatically label the players in American Football plays. The problem has been investigated for some time but currently the player labelling is still done by hand. In this chapter we propose a method to automatically label players in American Football plays.

The labelling process entails establishing labels for the blobs segmented from the video. To do this we use background knowledge about the game of American Football combined with the spatial knowledge about the initial setup of the players in the query. The process is very complex as it involves analysing all valid label combinations (sometimes several hundred combinations are possible) and selecting the most appropriate one.

The layout of the chapter is as follows: in Section 2 we describe in detail the processing involved in generating labels for the players in the initial setup. Section

3 shows the results obtained when the method was tested with data from actual American Football plays. A summary of the method is presented in Section 4.

5.2 Labelling Method

The initial setup of the players at the start of the play is very important as it provides valuable information about the type of play likely to occur. Plays in American Football can in some cases be identified based on the starting formation. Each formation has in turn a number of variations and it is to each one of these variations that we refer to as the initial setup of the play.

The initial setup is used to determine:

- The general offensive formation — the offensive formations can be divided into two classes: *one player back class* and *two player back class.*
- The type of play — *short gain play* and *long gain play.*
- The labels of the players.

At the beginning of the labelling stage, the image is processed as described in chapter 4 and the coordinates of the players in the query are converted to a scale of 100×100. Normalising (typically 320×200 pixels) to 100×100 grid enables crude determination of same player positions. For example the receiver takes position near the left or right boundaries of the playing field. Therefore the system considers any player that has a y coordinate value greater than or equal to 90, or smaller than or equal to 10 to be a potential *receiver.* Once the normalization has been done, the system analyses the player coordinates to determine which players are *in line.* This done by computing the Euclidean distance between players and cross-checking it with the number of *vertical* and *horizontal* lines (lines are explained further on).

The system then sorts the player coordinates into ascending order for the following reasons. The first reason is to determine whether the number of players at the *back of the formation* is *one* or *two.* The number of players at the *back of the formation* determines what rules are applied later on in the process of disambiguating the labels generated for the players.

The second reason is to determine which players occupy:

- The leftmost position in the formation or the closest position to the left boundary of the playing field.

- The rightmost position in the formation or the closest position to the right boundary of the playing field.
- The forward-most position in the formation.
- The backward-most position in the formation.

The players in the forward-most position are labelled as *receivers* or *wide receivers*. The players in the leftmost and rightmost positions are given *two labels: flanker* and *receiver*. If the player is positioned in the area considered to be close to the boundary as described above, then the label of *flanker* is removed. The players in the backward-most position are labelled according to how many players are at the *back of the formation*. If there is only *one player at the back of the formation*, then the player is a potential candidate for three labels: *running-back, tail-back, guard*. If there are *two players at the back of the formation*, then both players are given the label of *running-back*.

The third reason is to determine how many *lines* of players are in the initial setup. There are two types of lines: *vertical* and *horizontal*. For example, the initial setup shown in Figure 5.1 is made up of *five vertical lines* and *four horizontal lines*. The system uses the *lines* in four ways. The first way is to determine which is the centermost horizontal line. The players in this line are labelled as potential quarterback candidates. The second way is to determine which side of the initial setup is *"heavier"*. In many plays, the initial setup of the players is arranged in such a way that more players are grouped toward one of the boundaries of the playing field. Figure 5.2 shows the initial setup for a pass play that is *"heavy"* on the left hand side of the playing field. The third way in which the system uses the *lines* is to determine how many players are to the left and right of the centermost line.

The fourth reason is to constrain the number of possible labels for a player. The lines are used as a filter to allocate labels for the players based on the order in which you would expect to see them. As the system processes the vertical lines from right to left, the first players are likely to be *receivers* as they are positioned at the front of the formation. However once the first two vertical lines from the right are processed, the players are labelled as *guards/tight-ends*. As more lines are processed, the preferred label changes from *guard/tight-end* to *running-back/tail-back*. The horizontal lines are used in a similar way. The only difference is that as the system processes the lines from top to bottom, the likely labels vary as follows: *receiver* $--> $ *guard* $--> $ *running-back/quarterback* $--> $ *guard* $--> $ *receiver*.

Once the *lines* are identified, the system attempts to label those remaining players in the query that do not have a label yet. The system labels the players using the information extracted from the *lines* combined with the information on the

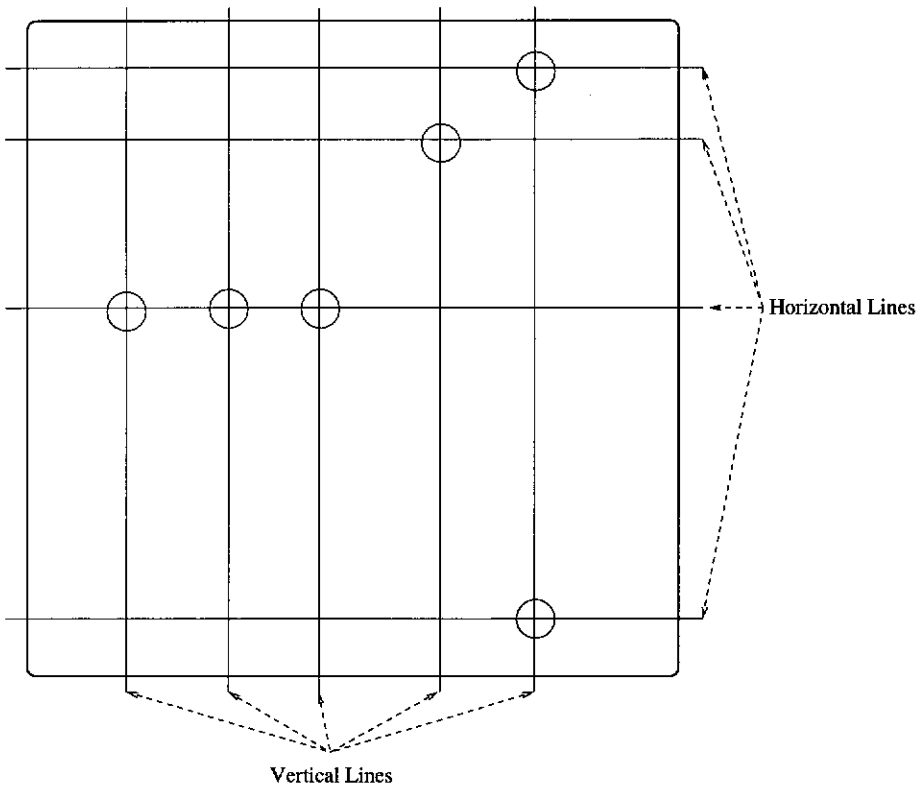


Figure 5.1: Typical query initial setup with 6 players arranged along 5 vertical lines and 4 horizontal lines.

labels currently allocated. For example, consider the initial setup shown in Figure 5.3a. After the lines are identified, the system has labelled players 1, 2 and 3 as *receivers* and player 5 as a *running-back* or *quarterback*. Player 4 does not have a label. The system determines that the play is *heavy* on the left and that there are 3 players to the left of the centermost line in the initial setup — the centermost line in this case is the vertical line with player 5. Player 4 is positioned to the left of the centermost line. He is also *behind the receivers and furthermore is positioned to the right of the receivers on the left hand side of the field.* Therefore player 4 is labelled as a *tight-end*. The last stage in the labelling process involves removing any ambiguity as to the label of a given player as well as validating the label of a player. As mentioned above, some players may have multiple labels. For this reason the system uses spatial information in the form of player symbolic relationships to solve the ambiguity in the player labels. If a player has two labels, the system generates a symbolic description of the relationships with his team mates for each one of the labels. Then the system takes each set of relationships at a time and compares them with the relationships of the corresponding player in the model. A score is computed for each set of relationships generated for each label. The comparison process is simple as it involves a straight match between two symbolic values. If a match is found then the score for the current label is incremented, else the score is left unchanged. When the scores for all labels are

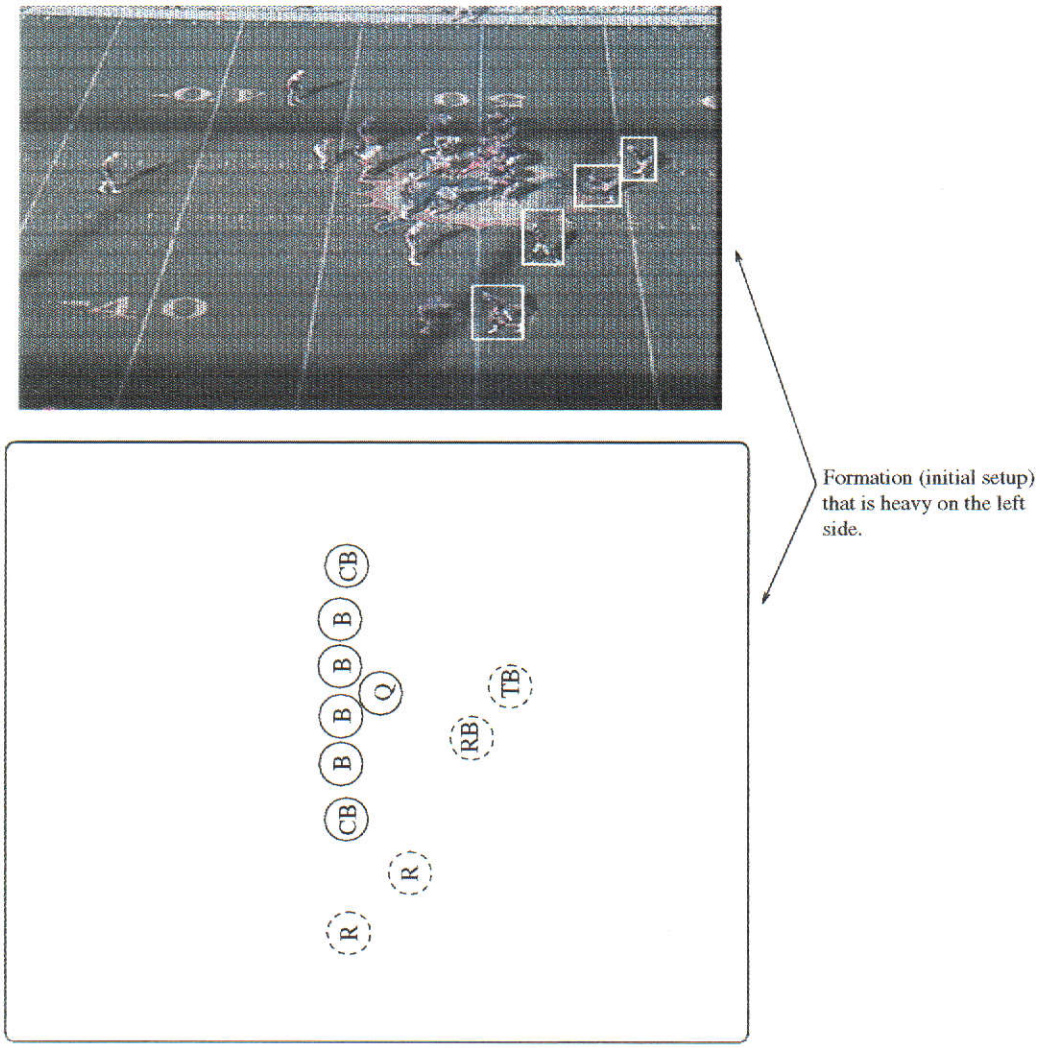


Figure 5.2: Pass initial setup with 2 receivers on the left hand side of the playing field. Image provided with the permission of Wide World of Sports - Channel 9 Australia.

computed, the system selects the highest scoring label as the valid label. After the labelling is completed a more detailed set of labels is obtained by analysing on what side of the field is the position of player. For example, if a player gets the label of wide receiver and is positioned on the left side of the field then the more detailed label becomes left wide receiver.

Overall the labelling module of the system resembles an expert system that consists of sets of rules, which fire based on the information extracted from the initial setup of the players in the video. There are several hundred rules (generated manually) in the labelling module and for this reason we show in Figure 5.4 only a sample of the rules used to classify the players in a initial setup arranged in 5 vertical lines and 3 horizontal lines with one player at the back of the formation.

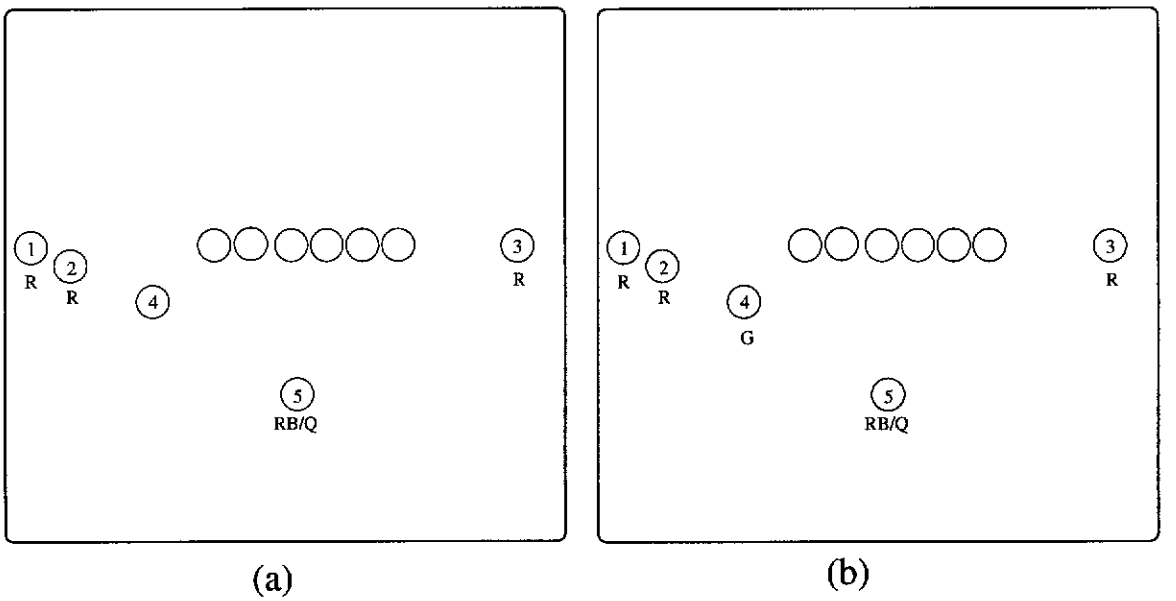


Figure 5.3: (a) Before labelling, (b) After labelling.

The advantage of labelling the players in the query play is that it constrains the search for player matches in the subsequent frames. The play models describe where the players are positioned and what they are doing from the first frame to the last frame — hence we can predict where the player in the query should be positioned and what he should be doing in the rest of the frames.

While labelling the players allows faster matching it has a drawback — if the labelling is incorrect then the solution returned by the system is also likely to be incorrect. Analysing many labellings reveals that misclassification occurs in 10% of initial setups. To further reduce the misclassification is very difficult as more information about the player initial setup is required and more rules need to be generated (all the players need to be segmented out to determine the combination of defensive players in the initial setups — this is not possible since the defensive players are occluded). However even in these 10%, the system still accurately labelled two or more of the players in the setups. Hence we regard this degree of misclassification as having little significance.

5.3 Results

The labelling method has been tested using initial player setups extracted from N.F.L. footage. The tests were conducted as follows. One video frame containing the initial player setup was segmented and the coordinates of the players were extracted as described in the previous chapter. The player coordinates were then

```
Let P1, P2, ..., Pn be the players in the Query.
Let Line_Left be the horizontal lines with the lowest y value,
    Line_Right be the horizontal lines with the highest y value,
    Line_Front be the vertical lines with the highest x value,
    Line_Back be the vertical lines with the lowest x value,
    P1(LL) be the number of horizontal lines left player P1,
    P1(LR) be the number of horizontal lines right player P1,
    P1(LF) be the number of vertical lines in front of player P1,
    P1(LB) be the number of vertical lines behind player P1,
    P1(x), P1(y) be the coordinates of player P1.

 if Vertical_Lines == 5 then
   if Horizontal_Lines == 3 then
     for i = 1 to n
       compute P1(LL), P1(LR), P1(LF), P1(LB);
       if Pi on Line_Left then
          if Pi on Line_Front then
             if Pi(y) < 10 then
               Pi is left wide receiver;
             else
               Pi is left receiver;
           else if Pi(y) < 10 then
                  Pi is left wide receiver;
       if Pi on Line_Right then
          if Pi on Line_Front then
             if Pi(y) > 90 then
               Pi is right wide receiver;
             else
               Pi is right receiver;
           else if Pi(y) < 90 then
                  Pi is right wide receiver;

       if Pi(LL) == 1 and Pi(LR) == 1 then
         if Pi(LF) == 4 then
           Pi is tail back;
       if Pi(LL) == 1 and Pi(LR) == 1 then
         if Pi(LF) == 3 then
           Pi is full back;
       if Pi(LL) == 1 and Pi(LR) == 1 then
         if Pi(LF) == 2 then
           Pi is quarterback;
```

Figure 5.4: Rules used for the case of 5 vertical lines, 3 horizontal lines and one player back formation.

given as input to the labelling module of the system to generate labels for the players in the query. The video frames and the generated labels are shown in Figures 5.5 to 5.14. The labels show the general positions of the players on the field.

The labelling method generated correct labels 27 of the 30 setups. The incorrect labels are highlighted by a bounding box around the label. Notice that though the system failed to generate a completely accurate set of labels for the players in the remaining 3 setups, the system was still able to generate a valid label for at least two of the players.

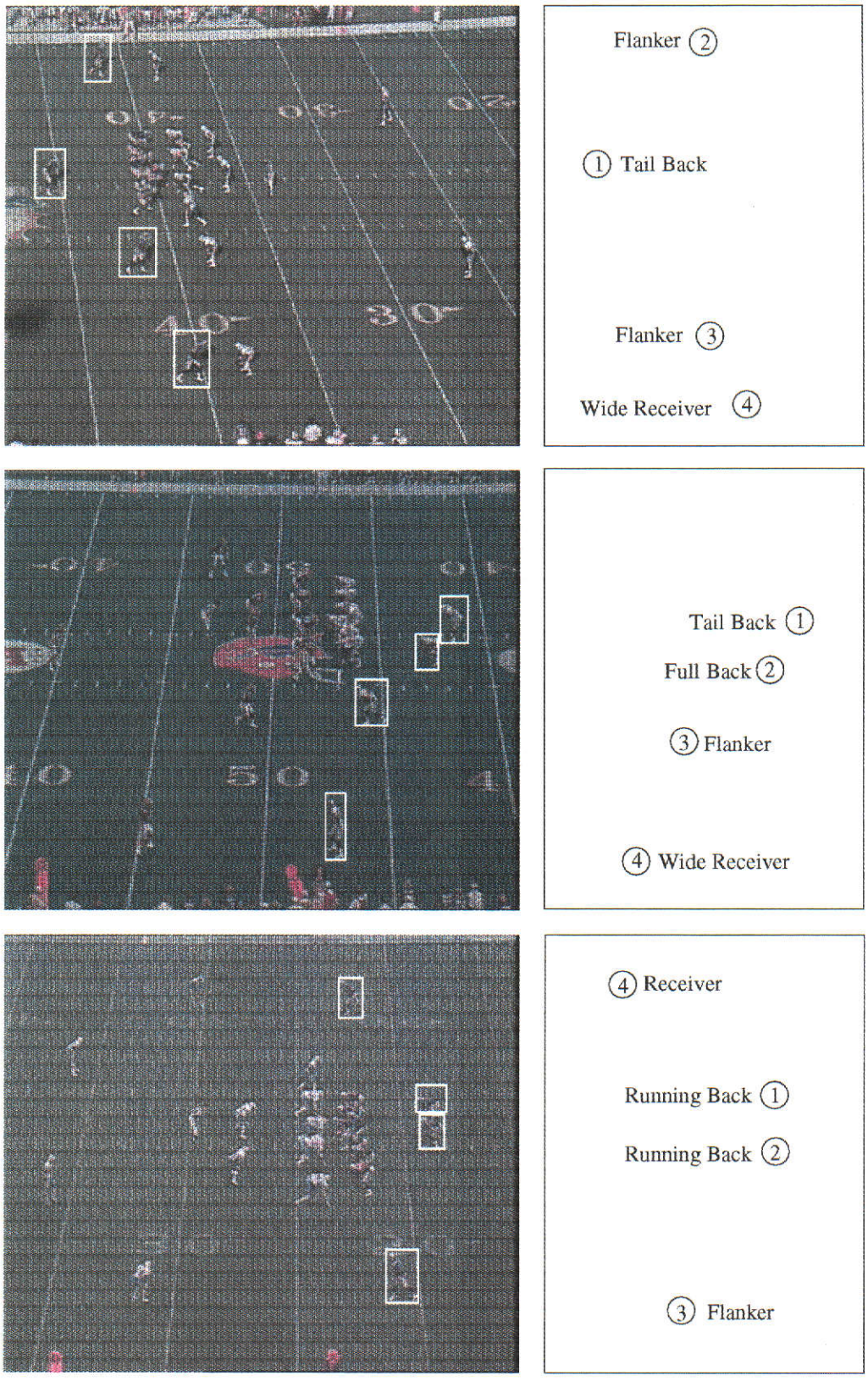
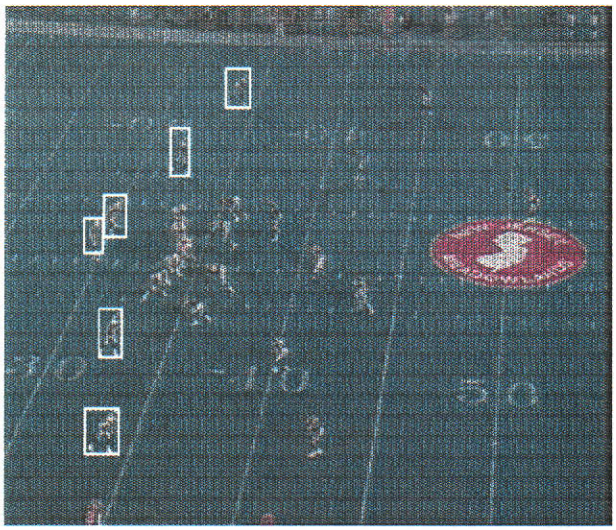
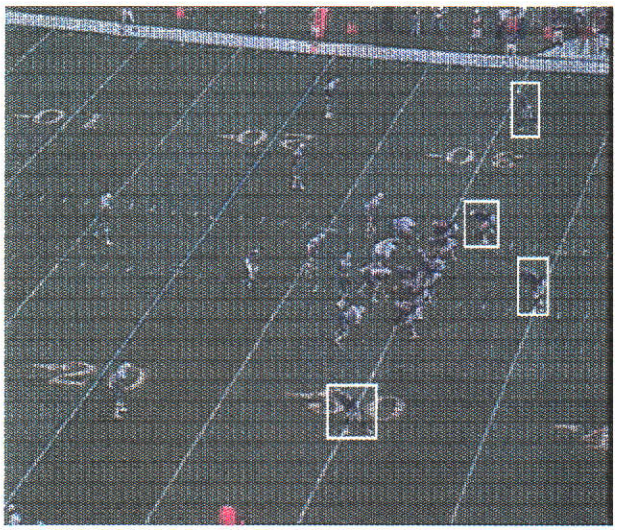


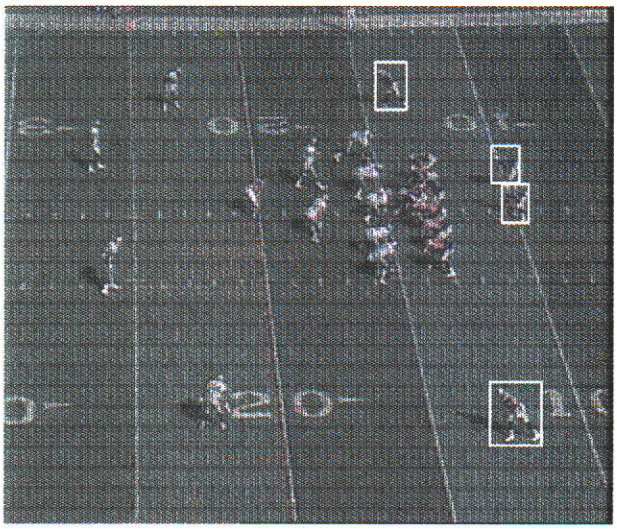
Figure 5.5: Setups 1 - 3. Images provided with the permission of Wide World of Sports - Channel 9 Australia.



(5) Receiver
(3) Flanker
(1) Running Back
(2) Running Back
(4) Flanker
(6) Receiver



(3) Wide Receiver
(Running Back) (2)
Tail Back (1)
(4) Receiver



(4) Receiver
Running Back (1)
Running Back (2)
(3) Flanker

Figure 5.6: Setups 4 - 6. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

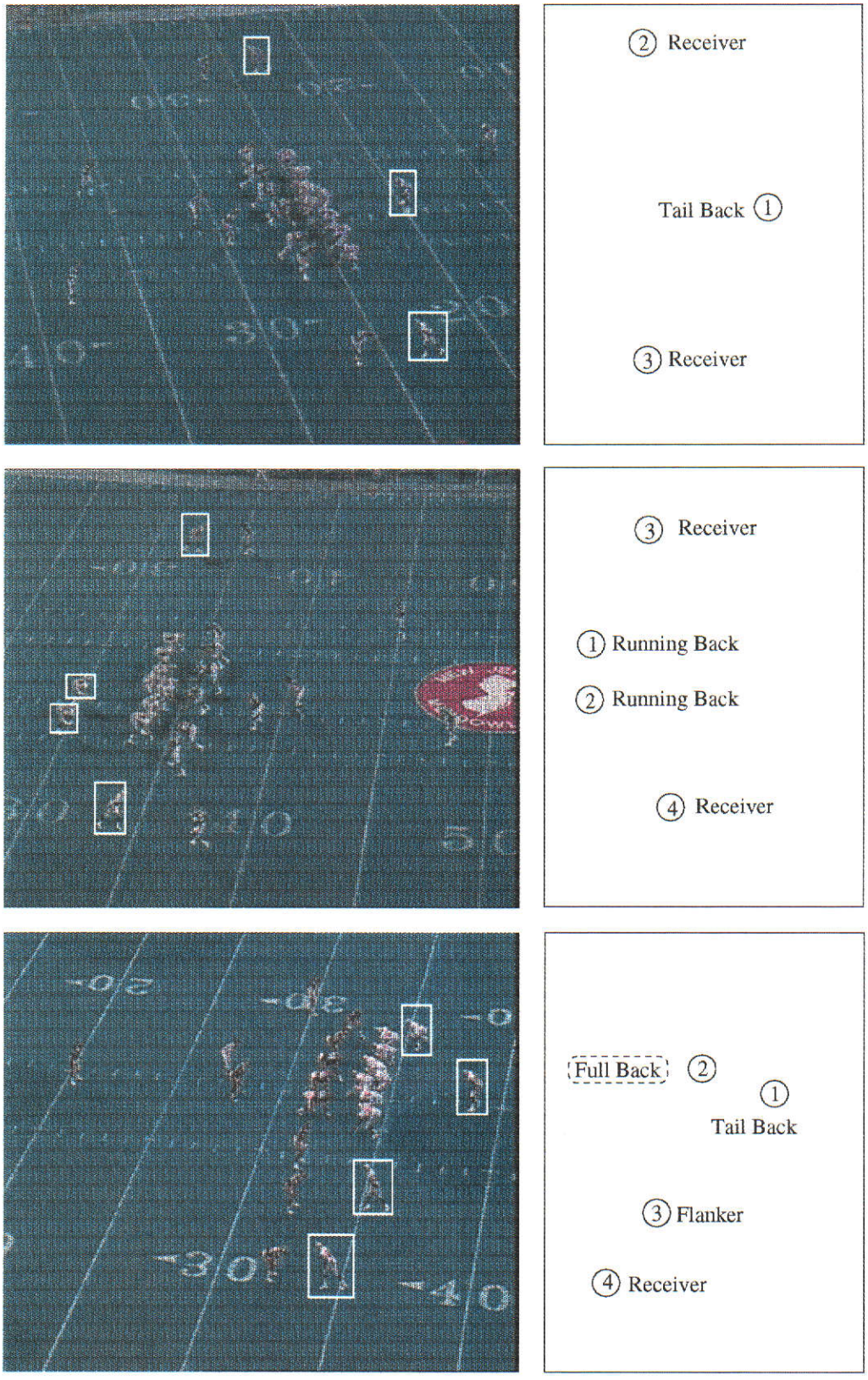
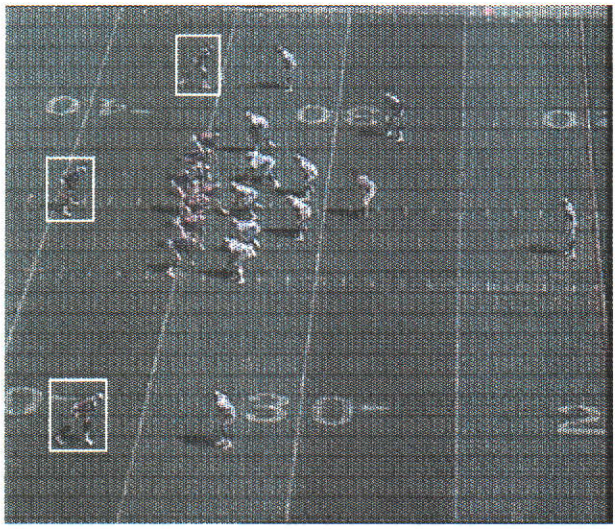


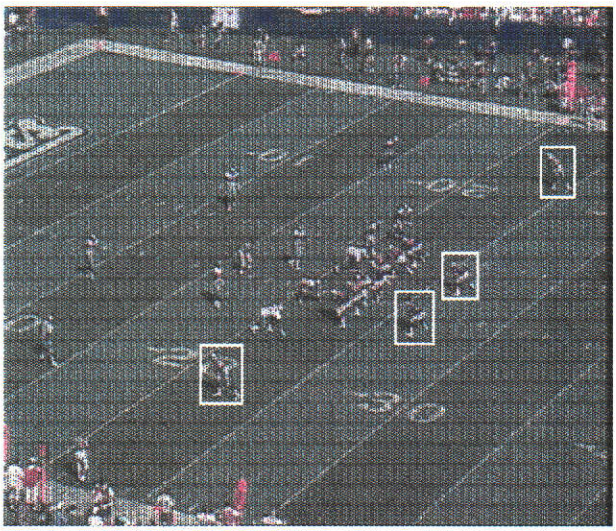
Figure 5.7: Setups 7 - 9. Images provided with the permission of Wide World of Sports - Channel 9 Australia.



② Receiver

① Tail Back

Flanker ③

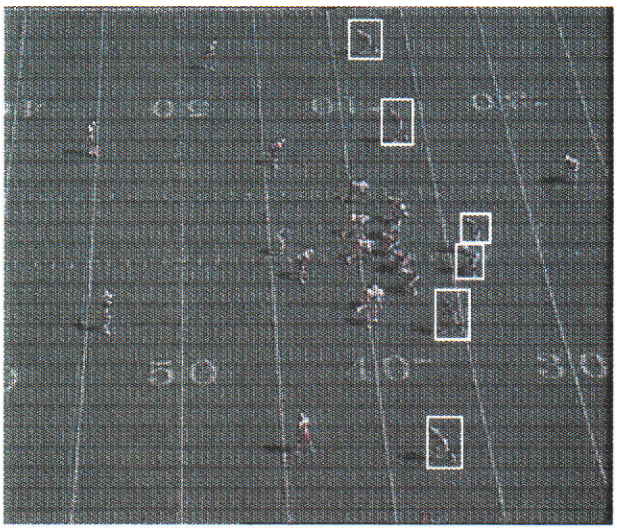


③ Receiver

Running Back ①

Running Back ②

④ Receiver



⑥ Wide Receiver

④ Flanker

Running Back ①

Running Back ②

③ Flanker

⑤ Receiver

Figure 5.8: Setups 10 - 12. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

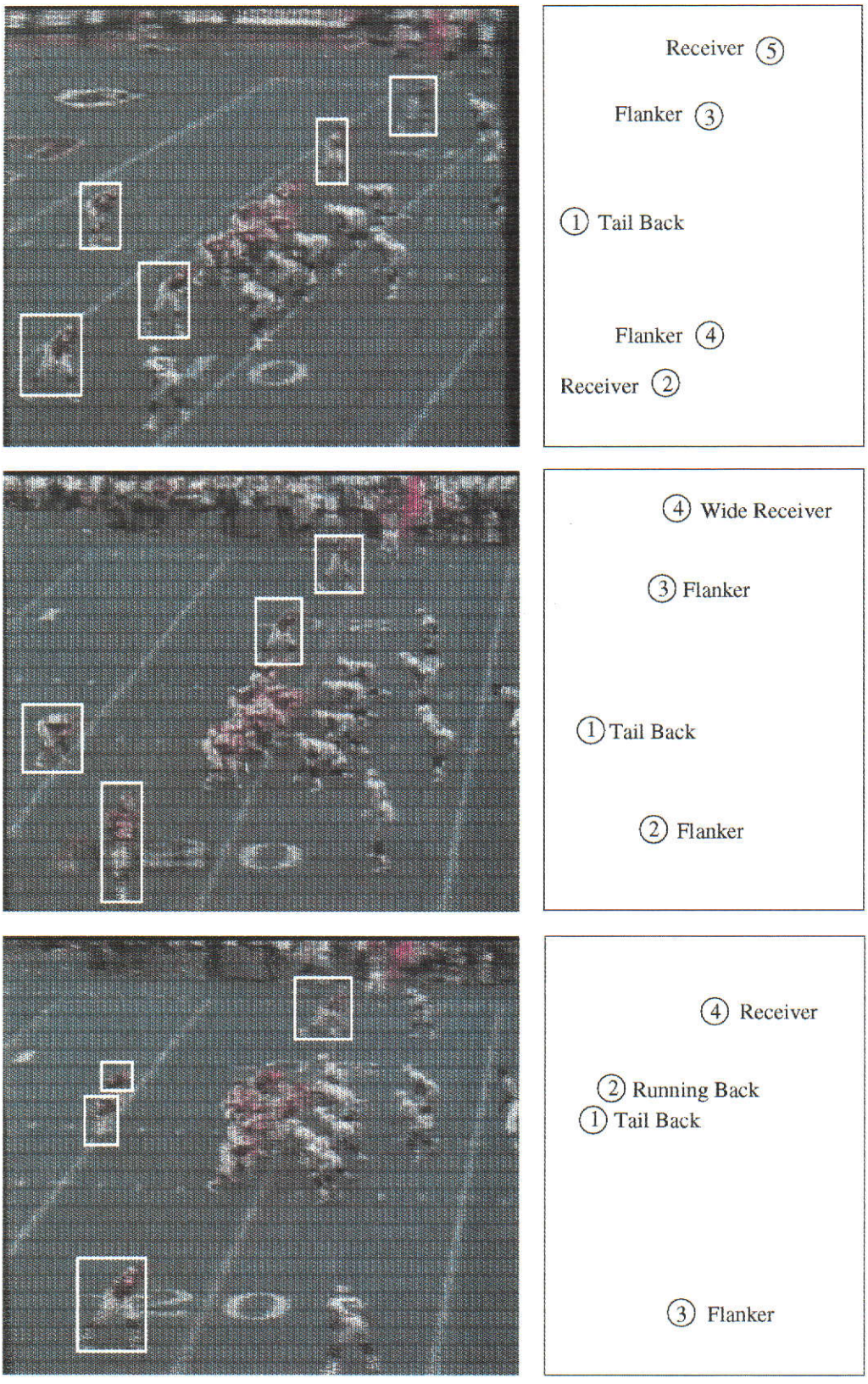


Figure 5.9: Setups 13 - 15. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

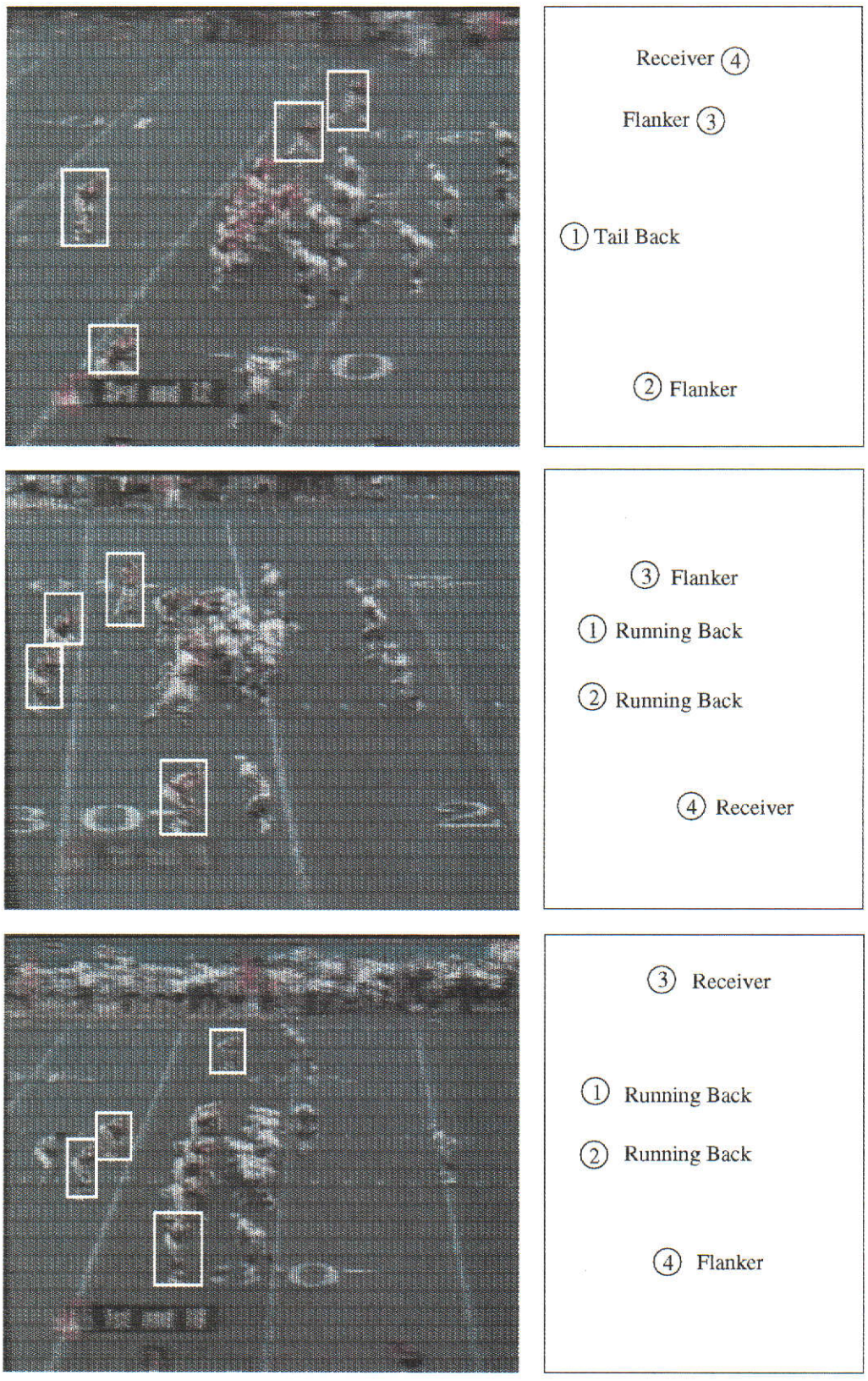
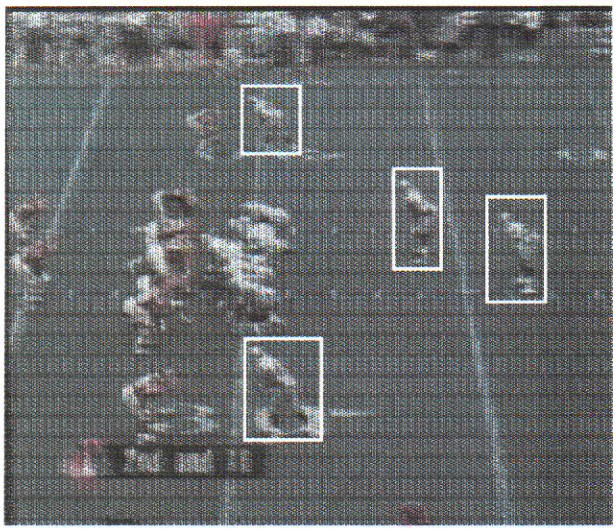


Figure 5.10: Setups 16 - 18. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

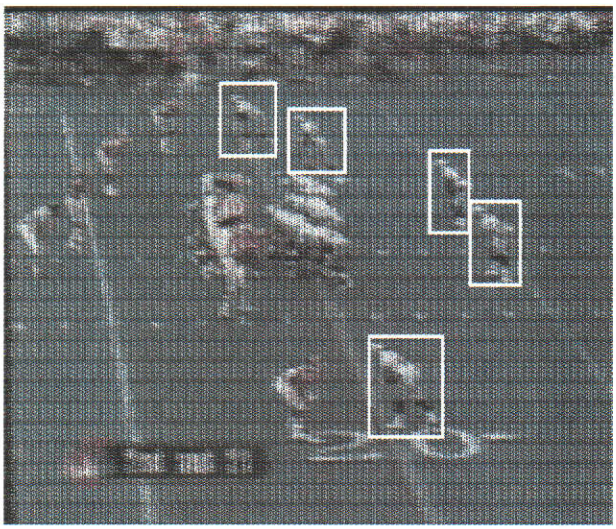


Receiver (4)

(1) Running Back

(2) Running Back

Receiver (3)



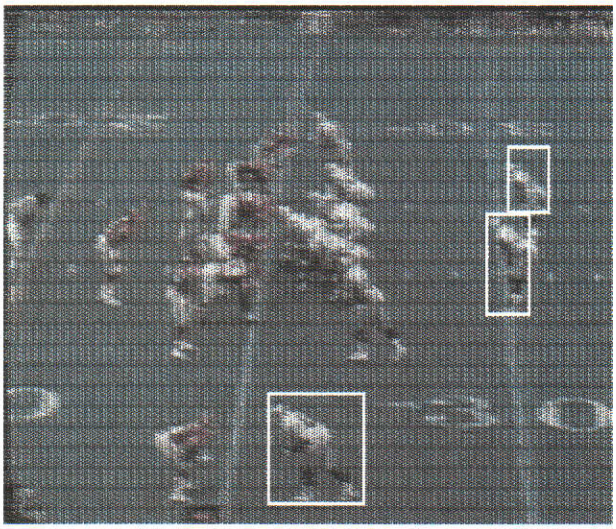
(5) Receiver

(3) Flanker

Running Back (1)

Running Back (2)

(4) Flanker



Running Back (1)

Running Back (2)

(3) Receiver

Figure 5.11: Setups 19 - 21. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

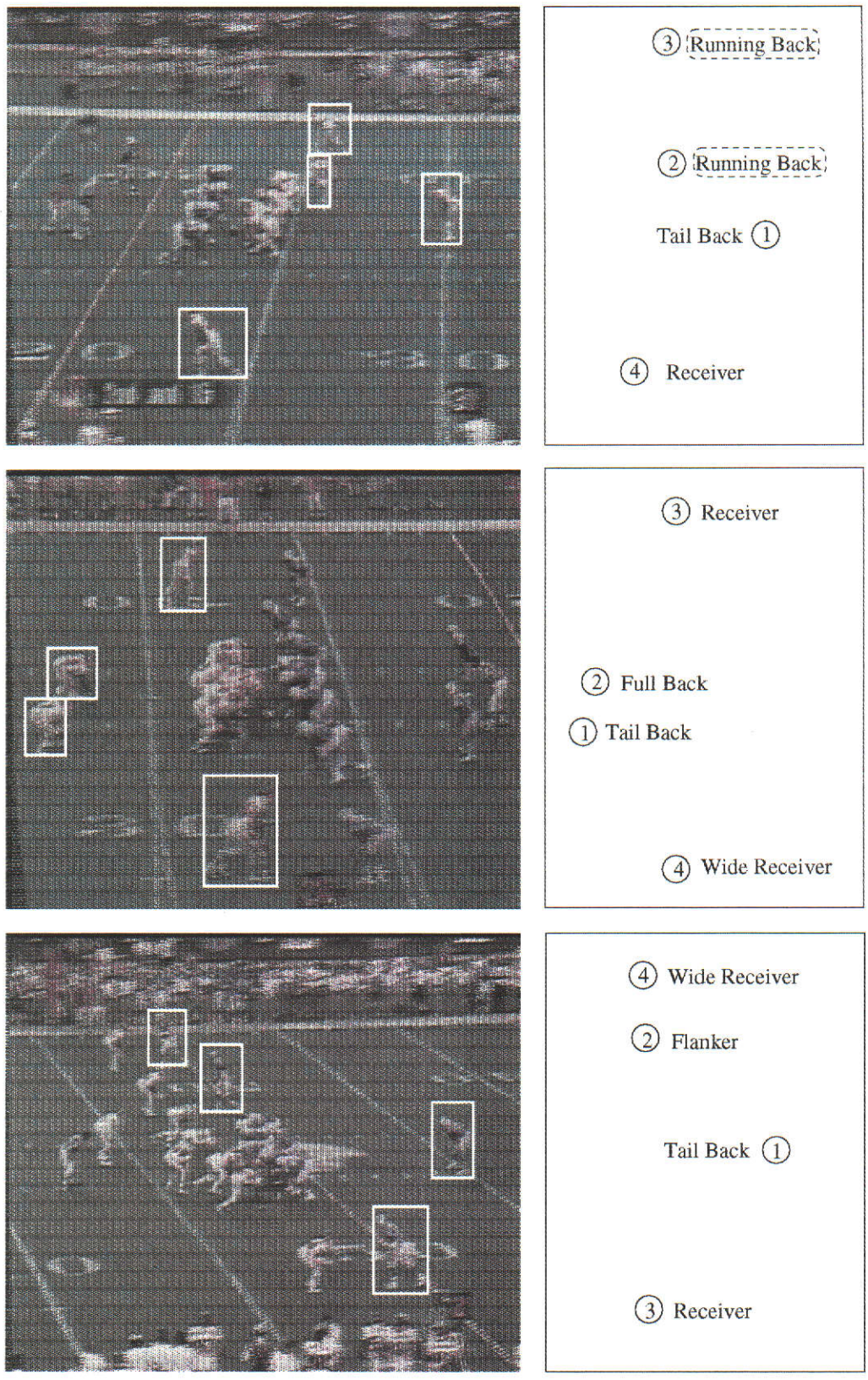
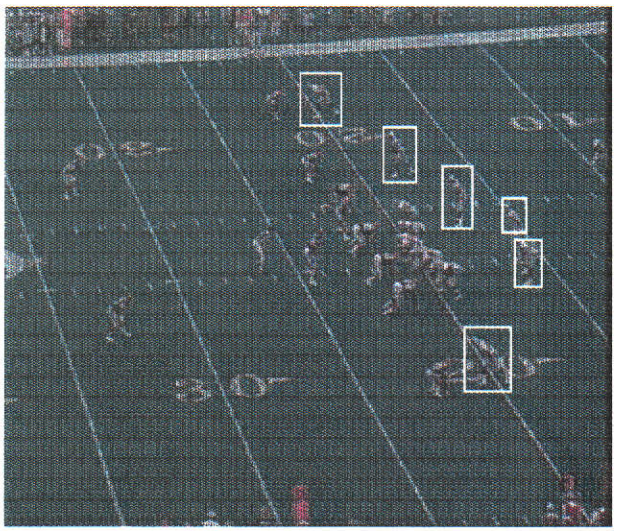
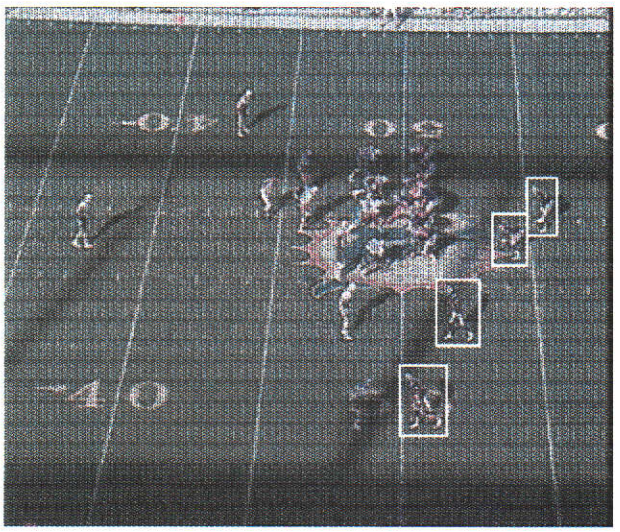


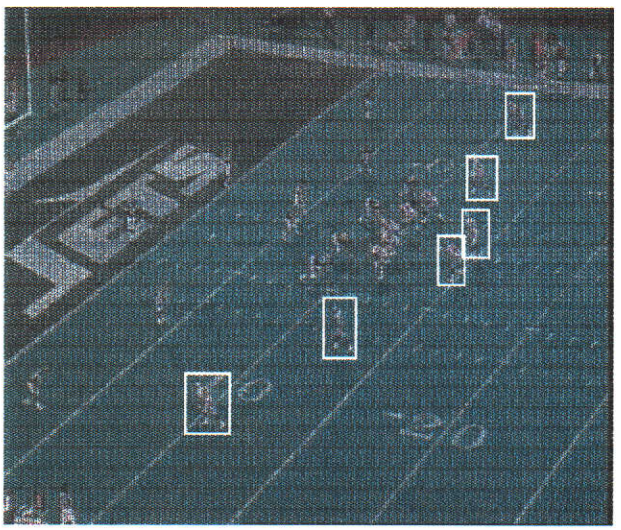
Figure 5.12: Setups 22 - 24. Images provided with the permission of Wide World of Sports - Channel 9 Australia.



⑥ Receiver
④ Receiver
③ Flanker
Running Back ②
Running Back ①
⑤ Receiver



Tail Back ①
Full Back ②
③ Flanker
④ Receiver



⑤ Receiver
④ Flanker
Running Back ①
Running Back ②
③ Flanker
⑥ Receiver

Figure 5.13: Setups 25 - 27. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

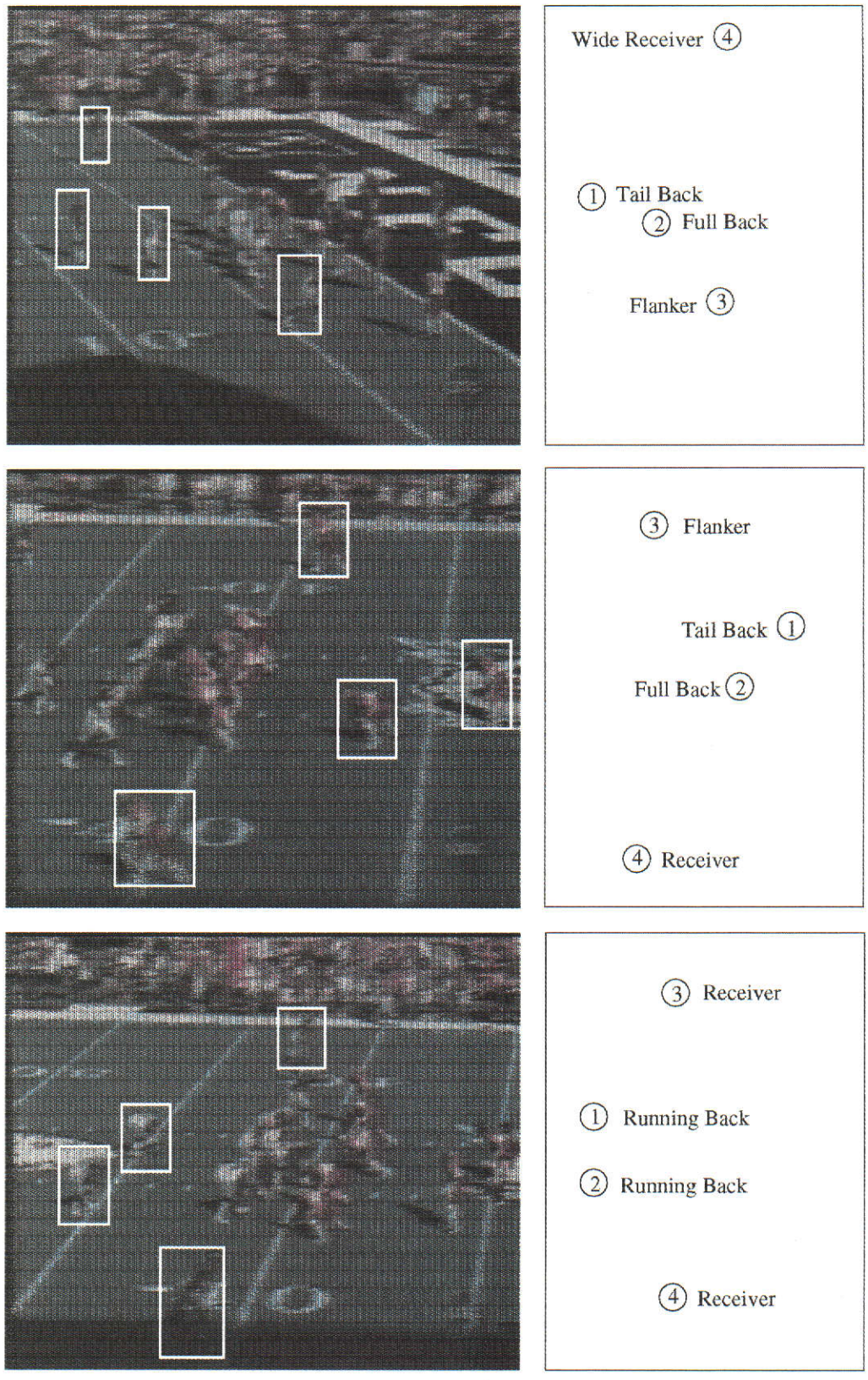


Figure 5.14: Setups 28 - 30. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

5.4 Summary

In this chapter, we described the process of labelling the players in the query. The process involves the use of background knowledge about the game of American Football combined with the spatial knowledge about the initial setup of the players in the query. The final part of the American Football application — the recognition and classification algorithm of American Football plays — is presented in the next chapter.

Chapter 6

N.F.L. Application — Model Recognition

6.1 Introduction

This chapter describes the play model recognition process. The process is built around the play model representation and consists of three stages that check whether the query play is similar to any of the known plays and if it is similar, then it determines the degree of similarity. We refer to a play as a *known* play if the system already has a model representing the play.

The recognition task is essentially to identify which play model, if any exists in the hierarchy (described in chapter 4), best matches the commentary segment and the geometric data extracted from the corresponding video sequence. The chapter is organised as follows: in Section 2 we describe the processing involved in extracting data from the text and video. Section 3 presents the algorithm used in the model recognition process. Some classification results are shown in Section 4.

6.2 Data Inputs

We use three sources of information: the transcript of the natural language commentary of the football game derived from the audio track of the video, geometrical information about the play derived from the video and domain knowledge. The low level video processing was discussed in chapters 4 and 5 of this thesis.

6.2.1 Extraction of Information from Text

The transcript of the natural language commentary plays a crucial role in recognition because it is an expert description of the action in the video and therefore can be used to obtain much information about the play.

The commentary text provides three clues about the play in the video. The first clue, is the type of action that takes place in the game, such as "the ball was passed".

The second clue gives the name and the type of the players that have been involved in the play action. For example, the text "PlayerX throws the ball high to receiver PlayerY" describes a play "pass", in which one of the player's position on the field is that of a receiver by the name of PlayerY.

The third clue that can be obtained from the natural language text is the game statistics. It is possible to extract the score of the game, the time remaining in the game, etc. This information essentially describes the status of the game and by and large dictates the course of action taken by the team in possession of the ball. For example, if the team in possession of the ball is behind and there is little time left in the game, then the team is bound to try to score quickly and hence choose from only a certain subset of the possible types of plays.

Note that it is not the intention here to understand all the text, just the relevant key phrases that are relevant to our goal of understanding the play. This significantly reduces the complexity and enables the use of simple grammar tools (LEX and YACC). The system uses a transcript parser that is essentially a finite state machine (as shown in Figures 6.1 and 6.2). The parser has been built using the UNIX tools LEX and YACC, to search the input commentary for a predefined set of keywords. The keywords found in the commentary are then passed onto a module that attempts to assemble the keywords into predefined patterns that describe various aspects of the game ranging from player positions to play actions. The parser searches the text for two general types of patterns in the text: a play pattern and game statistics pattern (e.g. yard line pattern).

The play pattern (Figure 6.1a) can be of four types: offensive, defensive, special and empty list. The most common offensive play patterns (Figure 6.1b) are throw, pass, lob and hit. Each one of the offensive play pattern types involves an action and one or two player components. The action can consist of one to three keywords. The player component can be lead type or receiver type. The lead type (Figure 6.1c) consists of a position and a name. The position is a single keyword. The name can be one or two keywords. In general, the play patterns consist of an action component and one or two player components. The action component and the player component vary depending on the type of play pattern (offensive, defensive or special).

The game statistics pattern consists of several types patterns of which the most common are yard line, time left, time period and score. The yard line pattern (Figure 6.1e) is made up of seven specific patterns and each one of the patterns is made of between one and four keywords.

The most common defensive plays (Figure 6.2a) are cover, tackle, block and intercept. Each one of the defensive play patterns consists of an action and one or two player components. The player component can be defender type or receiver type. The defender type (Figure 6.2d) consists of a position (Figure 6.2e) and a name (Figure 6.2e).

The special play patterns are similar to the offensive and defensive play patterns and are shown in Figure 6.2b.

The patterns extracted from the commentary text are then used to fill two types of frames: a player frame and a play frame. Appendix A shows the list of keywords.

Two typical information frames are shown in Figures 6.3 and 6.4.

6.2.2 Extraction of Information from Video Data

There are two stages in the processing of the video data. In the first stage the system attempts to determine the size of the temporal window in terms of frames that contains the American Football play, that is the start and end of the shot sequence containing the play. The play shot starts after the players have taken their positions in a predefined pattern on the playing field. Just before the play starts all players stand still and for this reason the beginning of the play shot is characterised by a period of time when there is little movement in the scene i.e. a frame with little movement indicates the start of the play. The play action involves heavy player movement in the scene and it *ends* when the camera focuses on one player, the crowd or when the game/player statistics or advertisments are shown. The time length of a play varies depending on the play. For example, running plays generally take longer than pass plays. In terms of frames, a play can take from 19 to 120 frames. In conclusion a play start is indicated by little movement in the scene and a play end is indicated by the camera focusing on a player or when a cut is detected (game statistics or advertisments are shown).

In the second stage the frames are segmented and the geometrical information extracted from the video frames used to compile a list of unlabelled player co-ordinates from the frames in the video sequence. This information is used to determine direction of movement of the players in the video as well as the initial and final player setups and positions (see Figure 6.5). As shown in the previous two chapters, the centroids of the blobs (which represent the unlabelled players

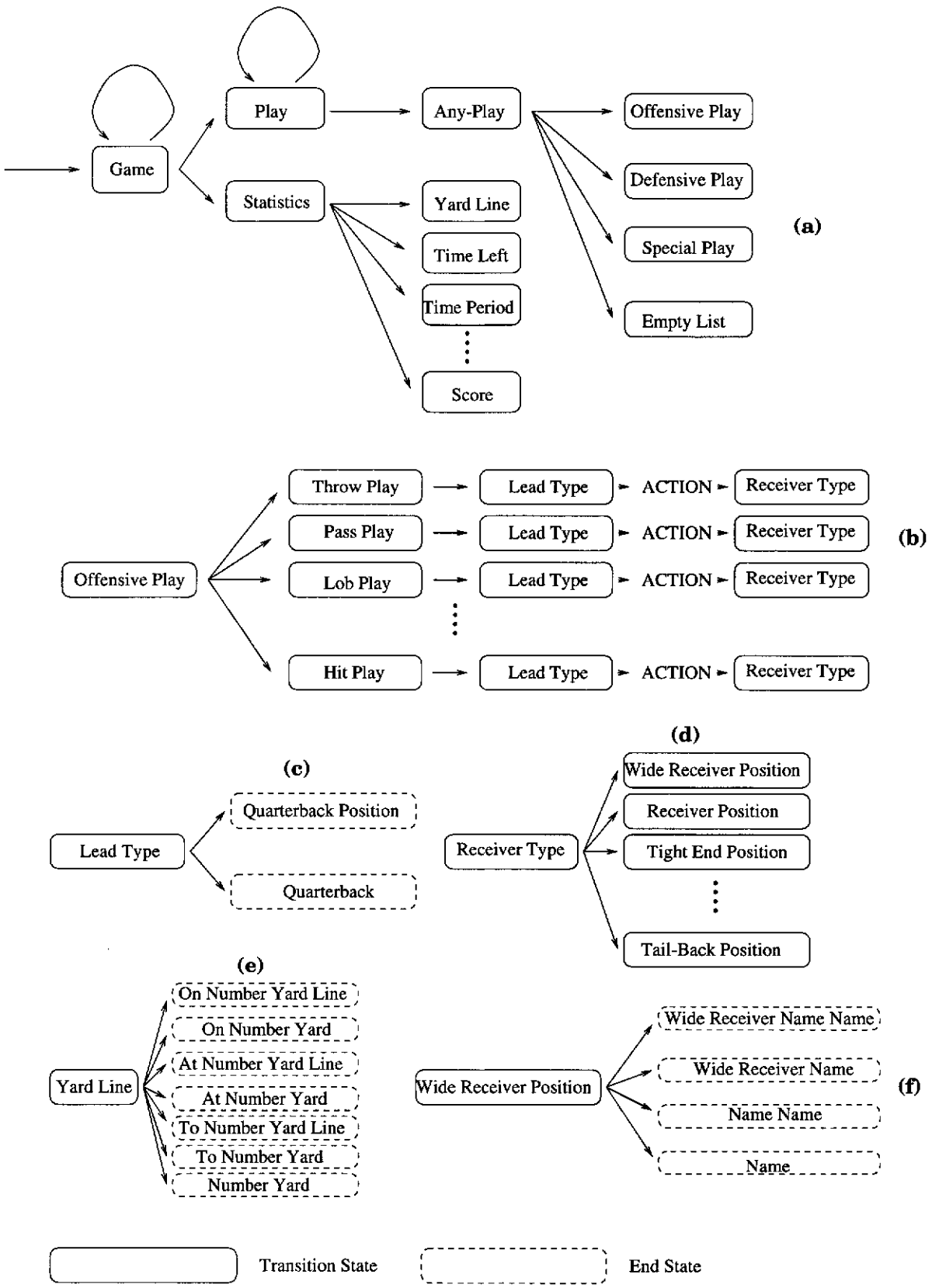


Figure 6.1: Text parser — a continuous line bounding box indicates an intermediate state; a dashed line bounding box indicates an end state.

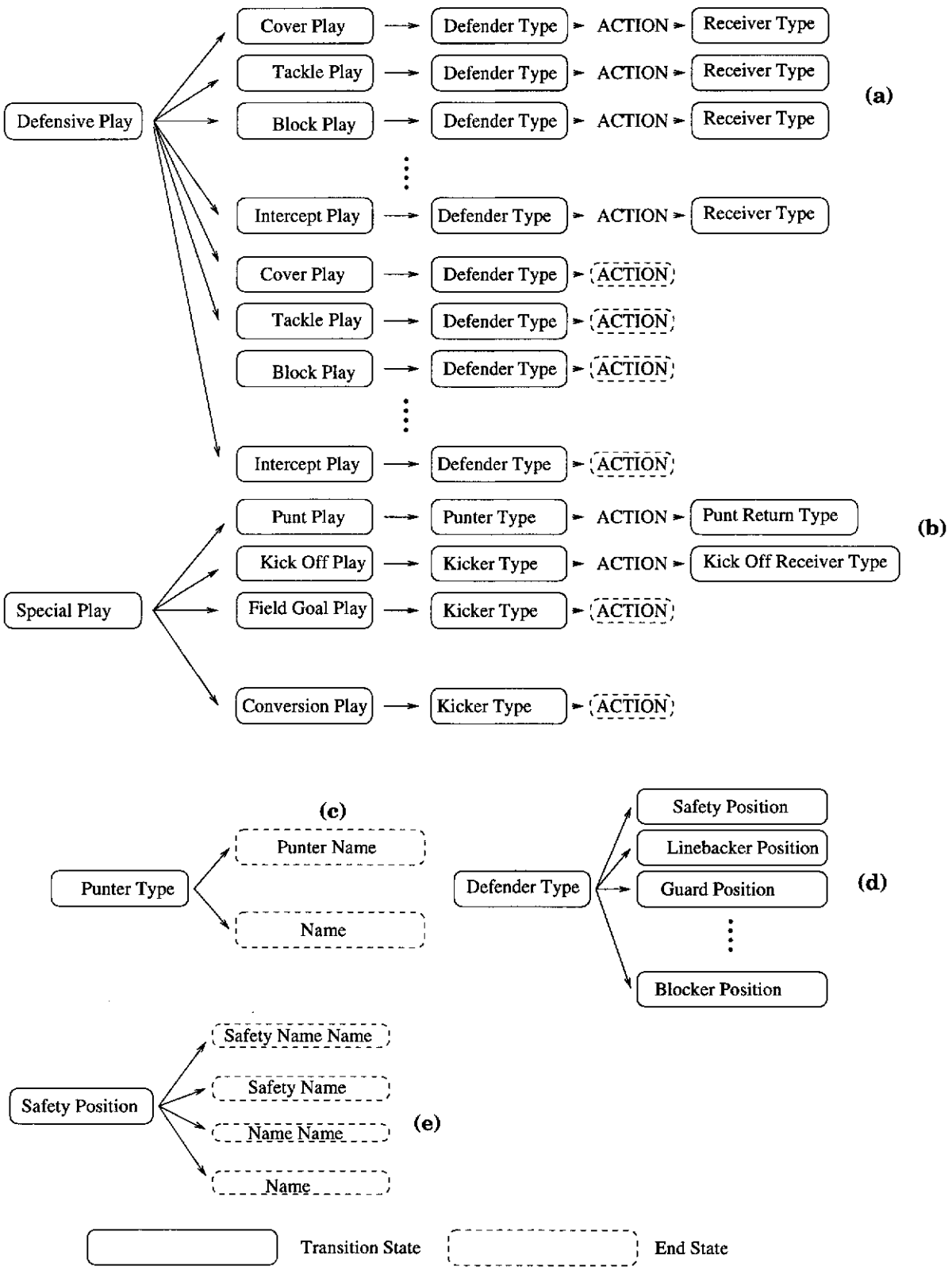


Figure 6.2: Text parser (continued) — a continuous line bounding box indicates an intermediate state; a dashed line bounding box indicates an end state.

Name1	Name2	Position	Action	Team	Play-Type	Id	Instance
PlayerX	PlayerY	Quarterback	Pass-ball	Team1	Offensive	15	19

Figure 6.3: Player Frame with an example.

Play-name	Blitz
Offensive-team	Team1
Defensive-team	Team2
Down-No.	2nd
Yards-to-go	2
Time	4:15
Player1	PlayerX
Player2	PlayerY
Player1-Type	Linebacker
Player2-Type	Quarterback
Instance	68

Figure 6.4: Play Frame with an example.

in the video frames) at time instances T and $T + 1$ are extracted and compiled into lists of coordinates. The pairs of coordinates can then be used to determine a general heading (H) for each player. In the case shown in Figure 6.5a the coordinates of three players at times T and $T+1$ are extracted from the images and compiled into a list (the players are labelled P1, P2, P3 and P4). In Figure 6.5b the coordinates obtained previously are used to compute the heading of the three players.

Using the information from the video, three symbolic descriptions of the player's movement are derived. The first description details the movement of the player for the entire play as a series of *moves* and *turns* (as described in chapter 4). The description is used in the temporal part of the play model. The second description is used to keep track of the player's movement at each instance in time and is used to reconstruct the dynamics of the play. The third description details the player movement over the entire play in terms of shape and distance.

The information from the video is assumed to be incomplete and can possibly contain noisy data about the player coordinates at different frames in the video.

6.3 Model Matching

Once the system has processed the football commentary and the video data to extract information, it proceeds to search the solution lattice (shown in Figure 6.6) for potential matches and outputs a list of plays identified as potential matches in ascending order (best match first). If the system has successfully branched down to the fourth level (where the labelling is done) then a solution exists for the query. The lower levels are then used to refine the solution. The three general levels match those in Figure 4.17.

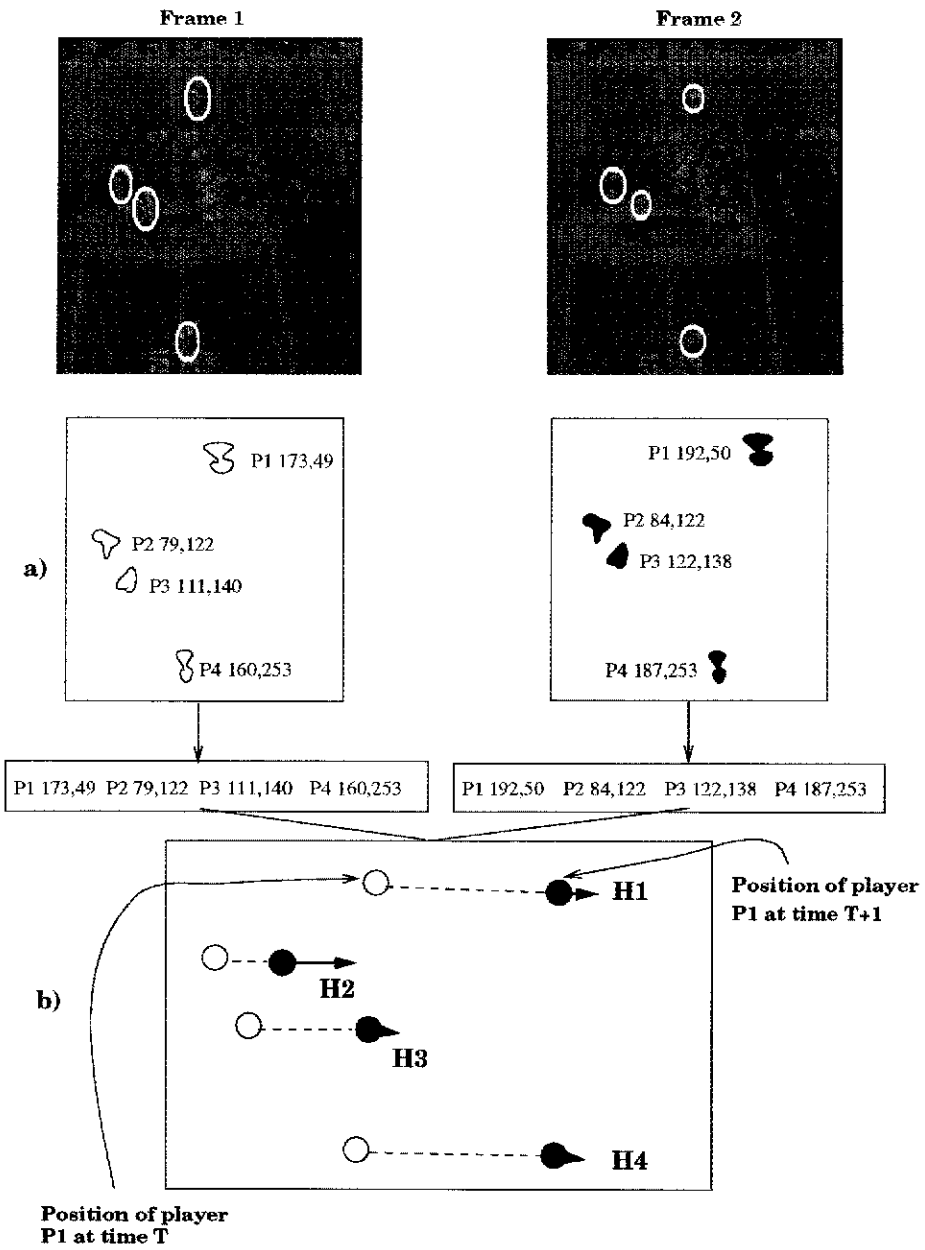


Figure 6.5: Extracting Information from Video Frames.

In the class level information is *extracted from the commentary text.* The other two levels involve the derivation and analysis of symbolic data from the geometric information. Each of these levels is described in detail in the following sections.

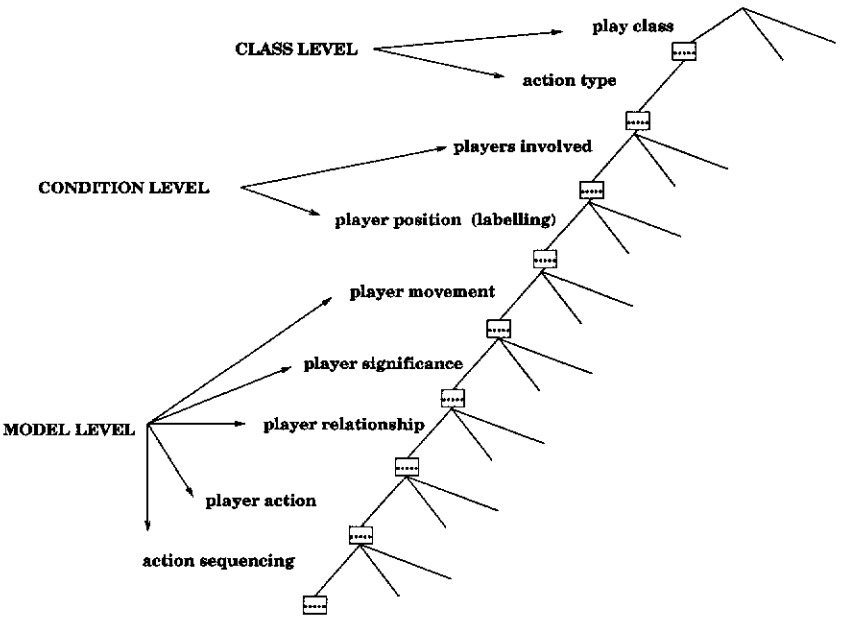


Figure 6.6: The general solution lattice used in the recognition routine.

6.3.1 Level 1: Comparing the Text Information

Initially, all plays are assumed to match the query. As the system descends the first three levels in the lattice, it determines the class of the play, the action in the play and the players involved in the play. Using this information the system *eliminates* (or prunes) potential play candidates that do not agree with the clues obtained from the different data sources. The candidates remaining after the system reaches the fourth level in the lattice, have their similarity scores updated after each successive level of processing until the system reaches the leaf nodes. The first four stages in the recognition process (represented by the first four levels in the decision tree) have to be performed in the given order. The rest of the stages are independent of each other and they can be carried out in any order.

In the majority of cases the text analysis provides the system with important clues on the type of play class occurring in the video sequence. However, there are instances when the commentary will not reveal any information about the play type. In these cases the actual matching is entirely based on the initial setup of the team at the beginning of the play. It is possible to determine from the player positions the general type of play (such as offensive running play) but the accuracy of the recognition process is greatly reduced. Also, teams sometimes use running play setups to disguise pass plays. If the system does not get any clues from the commentary (even the name of an offensive player involved in the action can help the system in determining the type of play) it is difficult to determine the actual play.

6.3.2 Level 2: Player Position Labelling

The initial setup of each model is compared with the initial setup of the query to label the players. The process is described in detail in chapter 5 and involves generating one or more hypotheses about the positions of the players in the query.

6.3.3 Level 2: Player Relationship Comparison

We test the relationships between the players at each instance in the play action. For each instance in the query play we take one player and develop a symbolic representation of his relationship with his teammates. For example, we take the player labelled *receiver* in the query play, develop the symbolic description of the relationships with his teammates and then compare these relationships with those of the player labelled receiver in the play model. If the relationships match then the *similarity score* of the candidate model is incremented, else the score is left unmodified. Consider the following example. Player P1 is *left-of* and *in-front* of player P2 in the query. In the model, Player P1 is *left-of* and *behind* player P2. The relationship *left-of* matches, so the *similarity score* is incremented. The relationship *in-front* does not match and the *similarity score* is left unchanged (the relationships of P1 in the query only partially match those of player P1 in the model).

6.3.4 Level 3: Temporal Data Comparison

The third level deals with the temporal occurrence and sequencing of the actions of the players in the query play. The system aligns the end points of the time intervals in the query and play model on a single axis (as shown in Figure 6.7). Then for each sub-interval, if the action in the query matches the action in the play model, the *similarity score* is incremented, otherwise the score is unchanged. Also, for each sequence of actions for each consecutive pair found in both the query and the model the *similarity score* is incremented. The process is repeated for each player.

Consider the example shown in Figure 6.7. It shows the actions of Player 1 in the model and the query. In the model, Player 1 *moves to position P1* and then *turns left*. In the query, Player 1 also *moves to position P1* and then *turns left* but the duration of the *move* action is longer than in the model although the ordering of the actions is the same in the model and query. Since the actions and the ordering of the actions in the query also match the model, the system increments the score of the model each time the action of Player 1 occurs at the same time instance. Notice that we do not consider duration of importance. In this case the

system increments the score twice as the players *moved at time instance 1* and *turned left at time instance 2.*

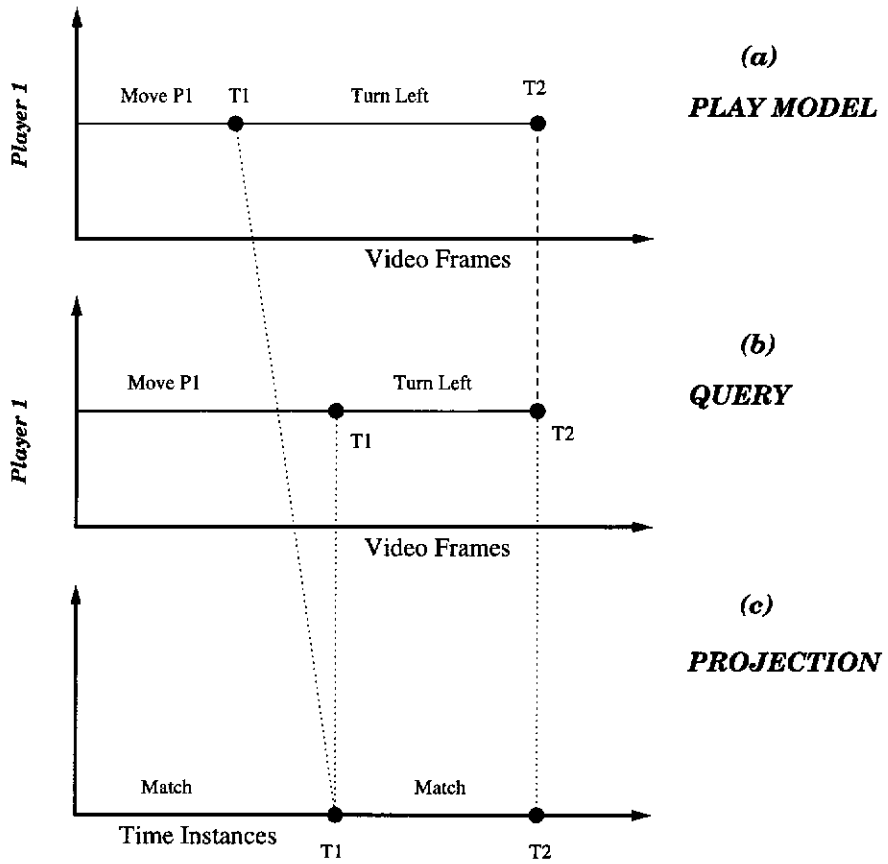


Figure 6.7: The action of Player 1 in (a) the play model and (b) the query are shown. The two actions are aligned as shown in (c).

After the third level in the recognition routine has been completed, the plays are arranged in order of descending *similarity score* and the candidate plays with the best five scores are returned as solutions. There are two reasons for returning the best five solutions.

One reason the best five solutions are returned is that the system in some cases does not find a good match for the query — only partial matches. A partial match is found when the system finds a match for only part of the information in the query — some of the spatio-temporal information in the query is different from that in the candidate model. For example if the relationships of a given query player P1 (Query) match only half of the corresponding model player P1 (Model) in the model, then P1 (Query) is considered to be a partial match for P1 (Model). In case the system finds only a partial match, there are two options: to return a model solution that is just a partial match or to return that no complete match for the query has been found. Rather than returning no solutions, the

system sorts the best five partial matches and returns the possible solutions.

In addition there are American Football plays, specifically *short running plays*, where there are only small differences between different plays. The information from the query is *incomplete* as it contains information about 3 to 6 players and an accurate match in this case is not possible. In such cases it is more appropriate to provide more than one of the matches.

The algorithm to compute the similarity score between the query and the candidate model has 3 stages: a text stage (corresponding to the class level), a spatial relationship stage (corresponding to the condition level and spatial conditions in the model level) and a temporal stage (corresponding to the temporal conditions in the model level). The values used to increment the *Similarity_Score* at the three levels are system parameters. The values used in the application have been: *IncreText 5*, *IncreRel 1* and *IncreTemporal 5*. The algorithm is shown in Figures 6.8 and 6.9.

For a more detailed explanation of how the system process queries, consider the following example.

6.3.5 Test Sequence 1

6.3.5.1 Input

The input to the system consists of a paragraph of American Football commentary (see Figure 6.10) and the video frames shown in Figures 6.11, 6.12 and 6.13. The query is "what is the offensive play in the video sequence?"

6.3.5.2 Extracting Information from the Text

The system uses LEX and YACC to extract the relevant keywords from the text and to build frames with information about the players and the plays described in the commentary paragraph.

The system identifies (using the LEX list of keywords) that there is an action described in the text which involves the players PlayerX and PlayerY (from the words "PlayerX lob PlayerY") and another action which involves the player PlayerZ (from the words "PlayerZ on coverage").

By using the predefined YACC patterns (shown in Appendix A), the system determines that PlayerX's position is that of a quarterback since only the quarterback can pass/lob the ball. Similarly it determines that PlayerY's position is

```
-------------------------------------------------------------------------------
Level 1.    ---   TEXT DATA ---
-------------------------------------------------------------------------------


Let P1, P2, ..., Pn be the labels of the players in the Query and
    POS1, POS2, ..., POSm be the labels of the players in the candidate Model.


Similarity_Score = 0;

 for i = 1 to n
  for j = 1 to m
   if Pi == POSj then
    if POSj is significant then
      Similarity_Score = Similarity_Score + IncreText;



-------------------------------------------------------------------------------
Level 2.    ---   RELATIONSHIP DATA ---
-------------------------------------------------------------------------------


Let P1, P2, ..., Pn be the labels of the players in the Query.
Let P1(Rel1)P2, P1(Rel2)P2, P1(Rel3)P2, P1(Rel4)P2, P1(Rel5)P2 and P1(Rel6)P2 be the
    six valid relationships between players P1 and P2, where Rel1 is Left-Of,
                                                            Rel2 is Right-Of,
                                                            Rel3 is In-Front-Of,
                                                            Rel4 is Behind,
                                                            Rel5 is In-Line-Horizontal,
                                                            Rel6 is In-Line-Vertical.
Let P1(Rel1)P2 == True is P1 is Left-Of P2.

Let POS1, POS2, ..., POSm be the labels of the players in the Model.
Let POS1(Rel1)POS2, POS1(Rel2)POS2, POS1(Rel3)POS2, POS1(Rel4)POS2, POS1(Rel5)POS2 and
    POS1(Rel6)POS2 be the six valid relationships between players POS1 and POS2,
                                                    where Rel1 is Left-Of,
                                                            Rel2 is Right-Of,
                                                            Rel3 is In-Front-Of,
                                                            Rel4 is Behind,
                                                            Rel5 is In-Line-Horizontal,
                                                            Rel6 is In-Line-Vertical.

Let POS1(Rel1)POS2 == True is POS1 is Left-Of POS2.

 for i = 1 to n
  for j = 1 to m
   if Pi == POSj then
    for k = 1 to n
     for p = i to m
      if Pk == POSp and i != k and m != p then
       for s = 1 to 6
        if Pi(Rels)Pk == True and POSm(Rels)POSp == True then
         Similarity_Score++;

-------------------------------------------------------------------------------
```

Figure 6.8: The algorithm used to compute the Similarity Score.

```
-------------------------------------------------------------------------------
Level 3.    ---   TEMPORAL DATA ---
-------------------------------------------------------------------------------

Let P1, P2, ..., Pn be the labels of the players in the Query.
Let POS1, POS2, ..., POSm be the labels of the players in the candidate Model.
Let P1(A1), P1(A2), ..., P1(Ak) be the ordered actions of player labelled P1 in the
    Query.
Let POS1(B1), POS1(B2), ..., POS1(Bk) be the ordered actions of player labelled POS1
    in the Model.

 for i = 1 to n
  for j = 1 to m
   if Pi == POSj then
    for p = 1 to k
     if Pi(Ap) == POSj(Bp) then
       Similarity_Score = Similarity_Score + IncreTemporal;
       if Pi(Ap-1) == POSj(Bp-1) then
         Similarity_Score = Similarity_Score + IncreTemporal;

-------------------------------------------------------------------------------
```

Figure 6.9: The algorithm used to compute the Similarity Score (continued).

==========

On 3rd and 8 or a long 7. 4 wide receivers, PlayerX lobs it high and over the stretching try of PlayerY the former Team1 star, PlayerZ on the coverage.

==========

Figure 6.10: American Football commentary.

that of a receiver. The action is described as a *lob* which is in fact a *high pass.* The action in the play is therefore a *pass* and the action is an *offensive action.*

The system also has the information that the position of PlayerZ is that of a linebacker so the system determines that the second action described is a *defensive action.*

When the system concludes extracting the information from the text, it builds two frames, one for the offensive action and for the defensive action. The frames are shown in Figures 6.14 and 6.15.

The query states that the system should find the best match for an offensive play and as a result the relevant action is the offensive action described in the commentary paragraph. Hence the first play frame will be used in the text analysis.

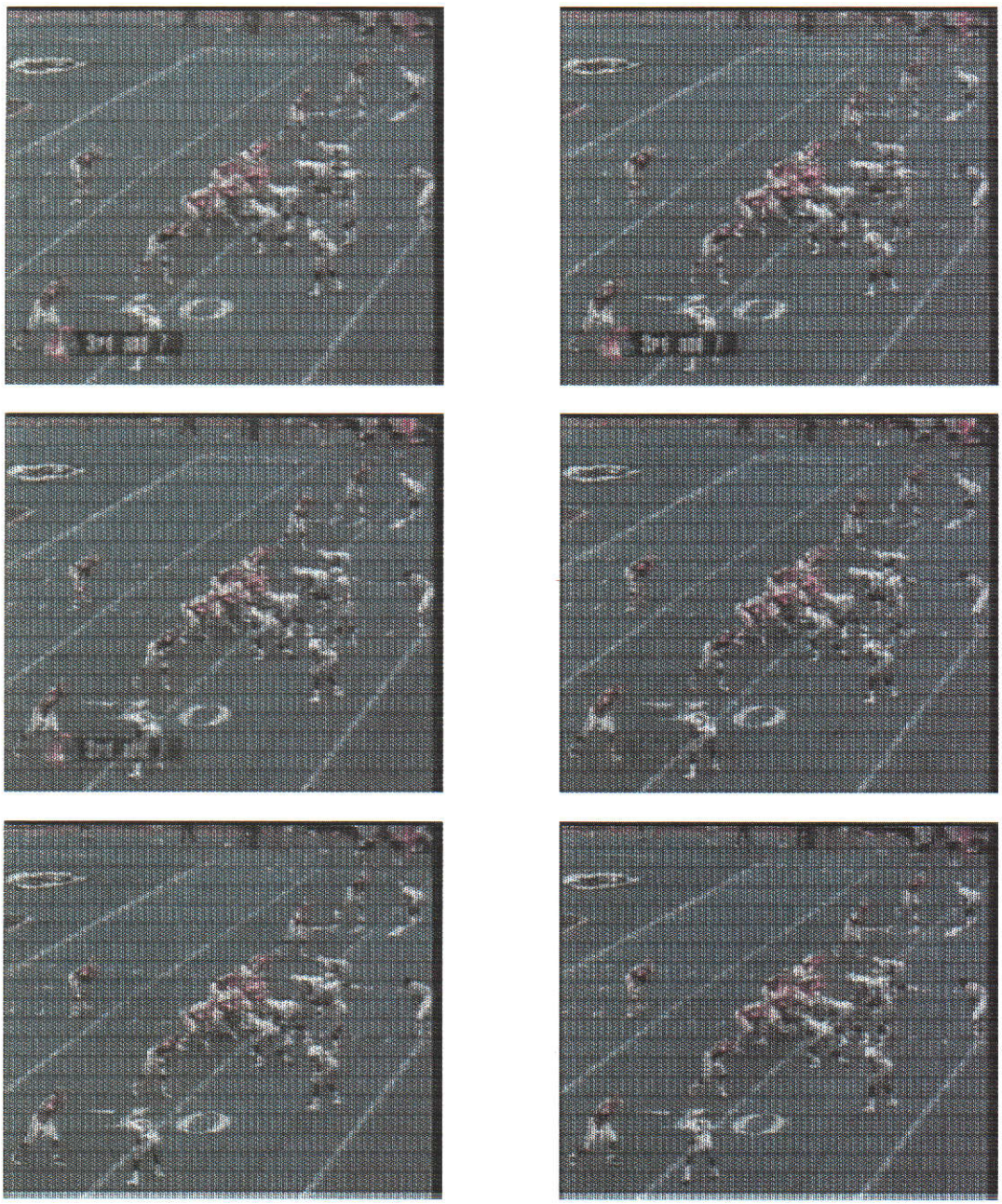


Figure 6.11: Video frames from test sequence 1.

6.3.5.3 Text Analysis

Using the frame describing the offensive action, the system proceeds to search for a play model that best matches the play described in the commentary.

At the start of the search all the play models in the system's memory are considered to be possible solutions to the query, regardless of the play model class

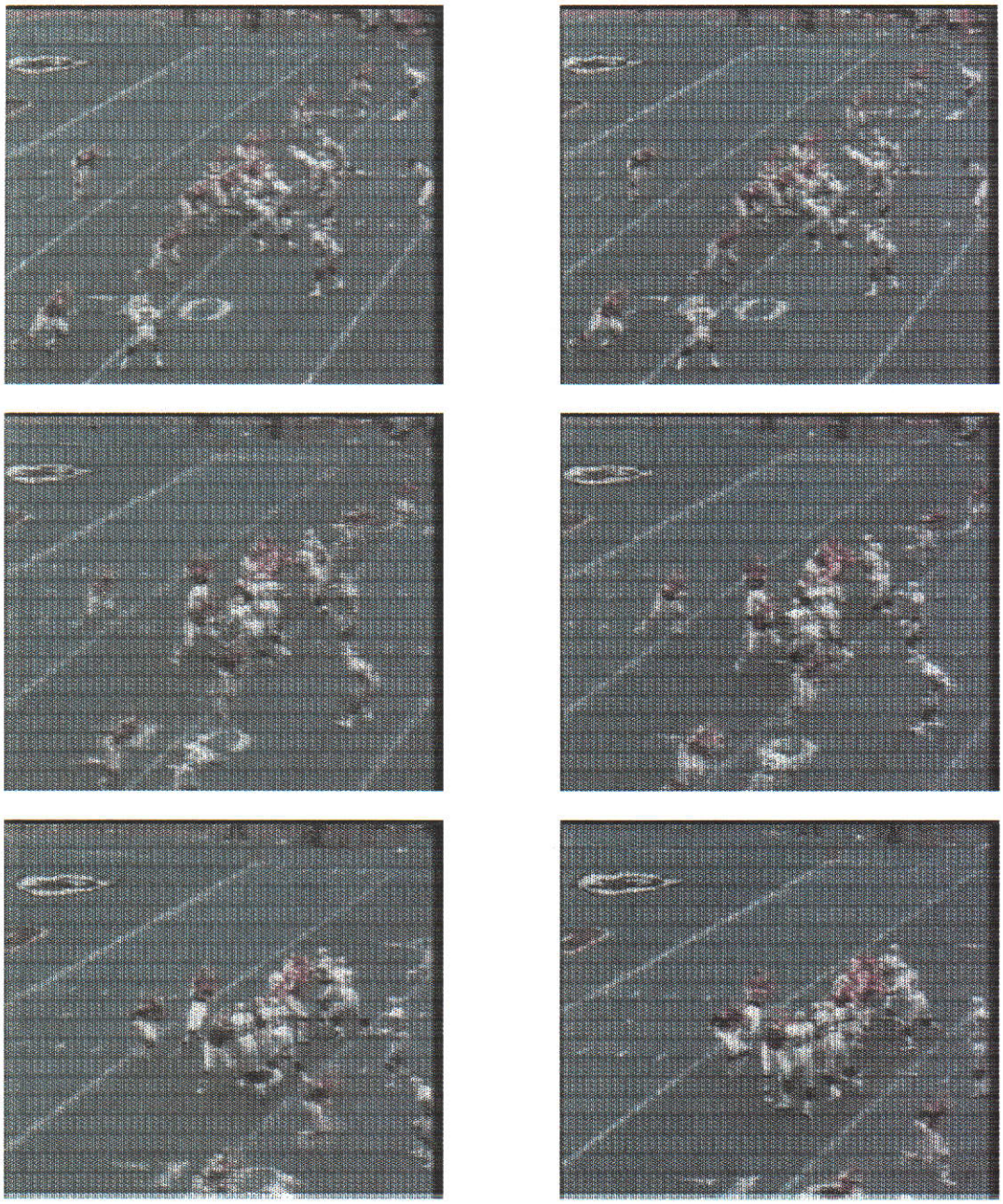


Figure 6.12: Video frames from test sequence 1. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

(offensive, defensive and special).

The search for a solution involves the pruning of a solution tree which contains all the plays in the system's memory. The frame built in the previous stage describes an offensive play and therefore the class of the play is *offensive*. The play model must share the same class with the play in the query so the system prunes the branches in the solution tree which contain defensive and special play models.

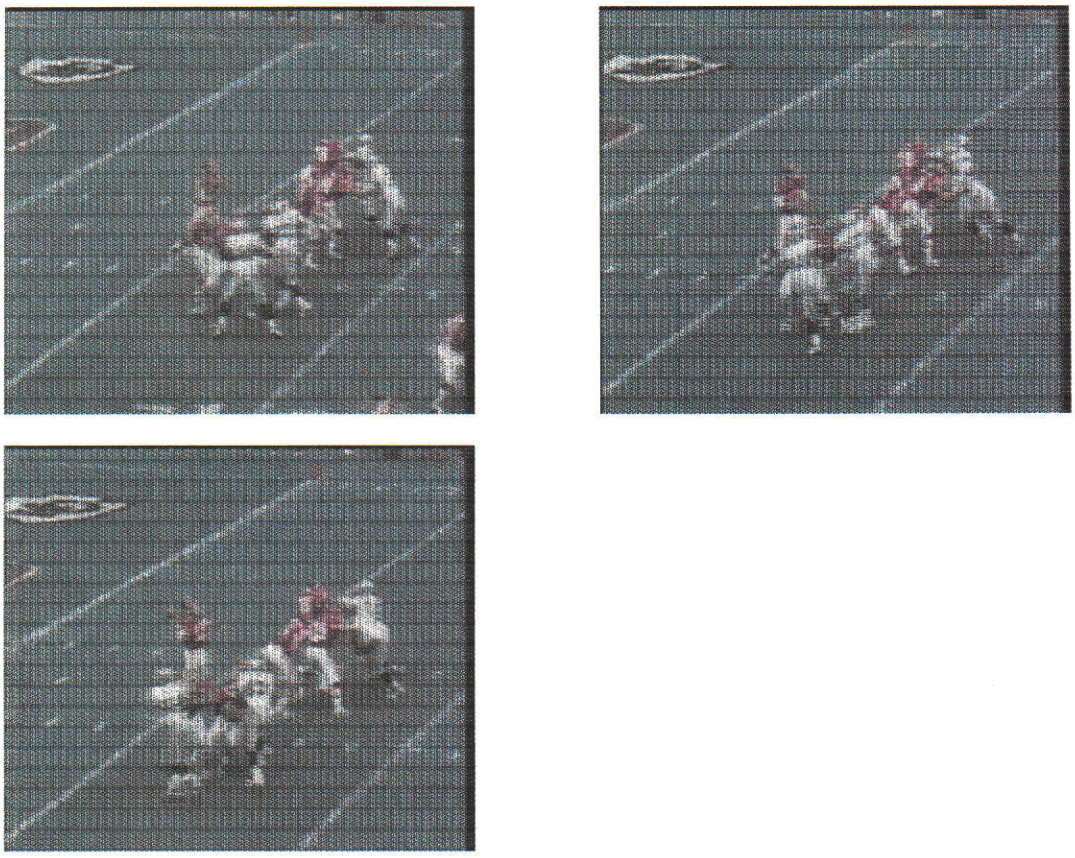


Figure 6.13: Video frames from test sequence 1. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	3rd
Yards-to-go	7
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Figure 6.14: Frame for offensive action.

The *action* described in the query play frame is a *pass* so the system further constrains the search by removing all offensive plays from the solution tree whose action type is not a *pass*.

The frame describing the query is then used by the system to determine the positions of the players involved in the play action. In the text, the players are *quarterback* and *receiver*. The system then uses the player positions to reduce the number of possible solutions by removing the offensive plays whose action did

Play-name	Tackle
Offensive-team	-
Defensive-team	-
Down-No.	3rd
Yards-to-go	7
Time	-
Player1	-
Player2	PlayerZ
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Figure 6.15: Frame for defensive action.

not involve a quarterback and a receiver.

At the end of this stage the solution tree still contains 5 play models and this indicates that the query play is similar to the 5 play models. We will show the processing performed by the system considering one of the candidate play models: the *Single Back Formation: Break-In play model.*

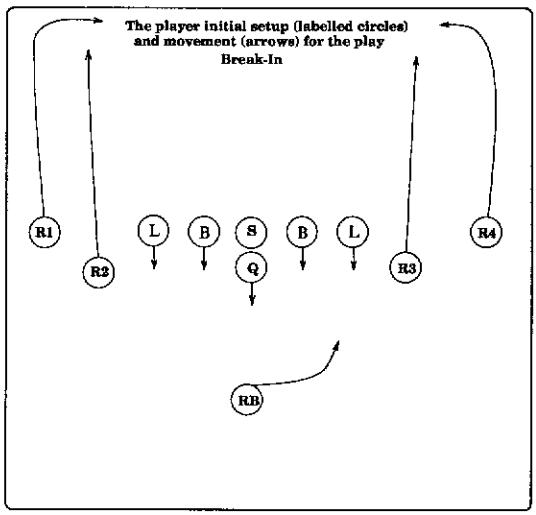


Figure 6.16: Break-In play model.

6.3.5.4 Video Analysis

The video frames are processed and the coordinates of the players are extracted and compiled into a list. The segmented players are highlighted with a bounding box. At the start of the play, 5 players are segmented out. As the play develops, the players progressively move out of the camera view and as a result after 19 frames only one of the original players is possible to segment out. The segmentation of the players stops when the last player is occluded and the play shot

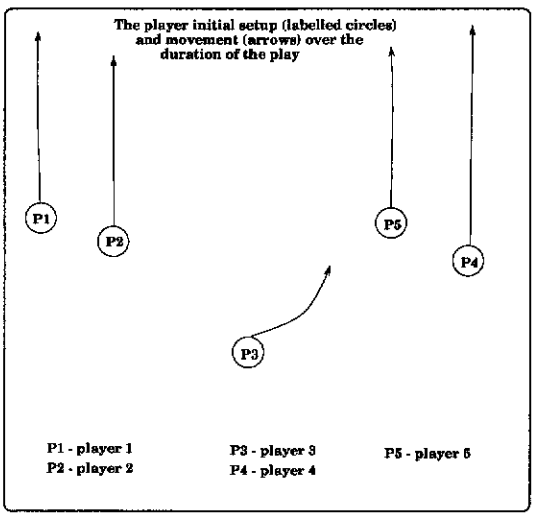


Figure 6.17: Query play.

ends with the camera focusing on the player that catches the ball thrown by the quarterback.

The coordinates are used to build the spatio-temporal description of the query shown in Figure 6.18.

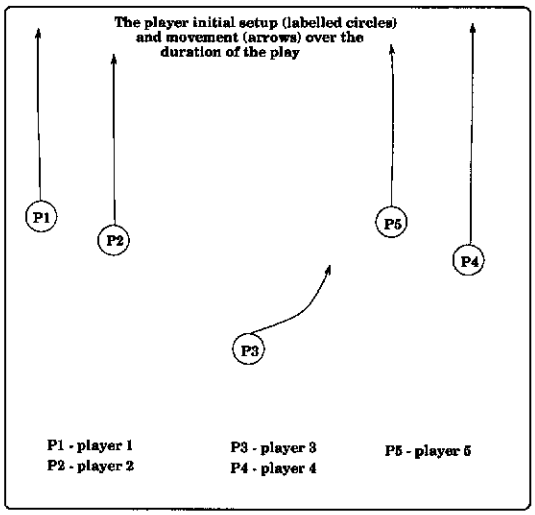


Figure 6.18: The movement and starting positions of the players in the query.

The solution list contains five candidate play models and to determine which is the best match, the system uses the information extracted from the video. It first attempts to label the players in the query.

Player Labelling — The coordinates of the players in the query in the first frame (see Figure 6.19 of the video sequence) are used to label the players (see chapter

5). The process is repeated for each candidate play model. The system first considers *Single Back Formation Break In* which is the (4wr — 4 wide receivers) play model (see Figure 6.20 for player starting positions). Players 1 and 2 are

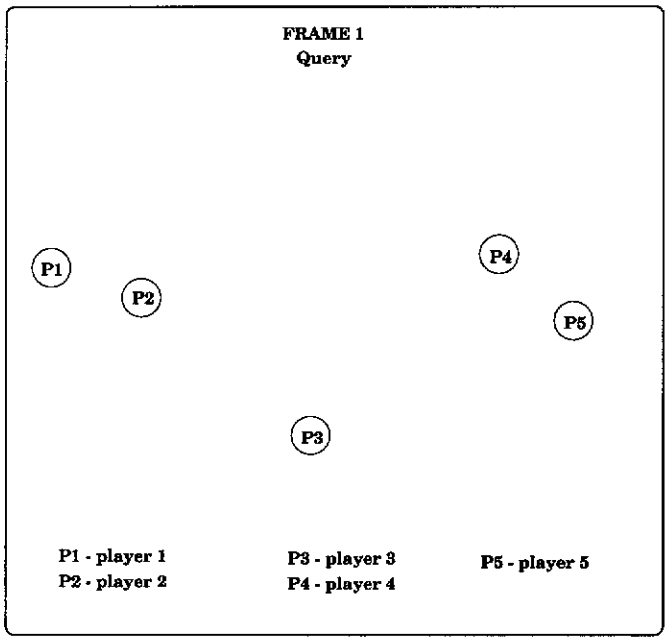


Figure 6.19: The starting position of players in the query at the start of the play.

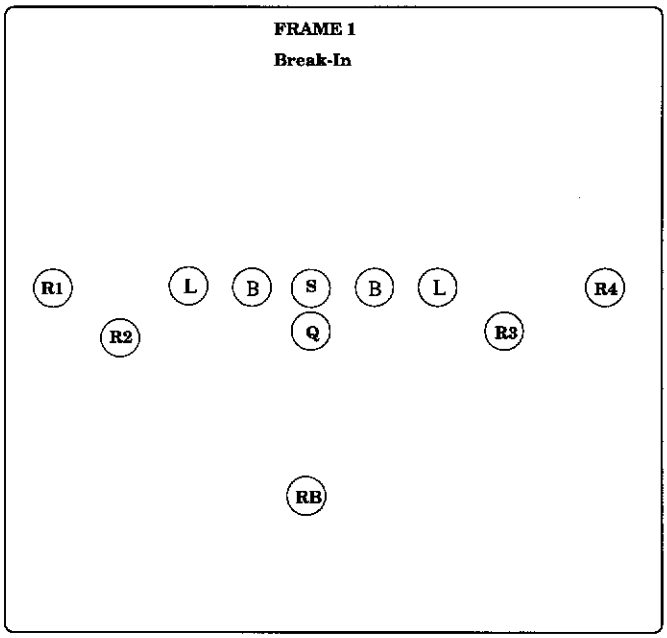


Figure 6.20: The starting positions of the players in the Break-In play model (at frame 1).

close to the left boundary. This restricts the possible matches in the model to just

the players on the left hand side. Next the system analyses the player's spatial relationships to his team mates and labels both of them as *receivers*. Similarly players 4 and 5 (close to the right boundary) are labelled as *receivers*. Player 3 is in the last line of players (there is no player behind him) and gets the label of *tail back*.

When the labelling has been completed, the system considers each play model one at a time and performs a series of tests (described below) which determines the degree of similarity between the query and the play model. Each candidate model starts with a score of 0 and after each test, if appropriate, the *similarity score* is incremented. The play model with the highest similarity score is the best match.

Player Movement — The system uses the information extracted from the video sequence to develop a symbolic description of the player movement in the query as described in chapter 4. Players 1, 2, 4 and 5 start *in_line* or *behind* the defensive line and finish their movement in front of the defensive line hence the system determines that their movement is *long*. Player 3 is starting behind the defensive line and finishes behind the defensive line. His movement is therefore *short*. The movement of Players 1, 2, 4 and 5 is *straight* while the movement of Player 3 has *a turn*. The symbolic description of movement in the query is: Player 1 — long-and-straight, Player 2 — long-and-straight, Player 3 — short-and-turn, Player 4 — long-and-straight, Player 5 — long-and-straight.

Each one of the candidate play models is considered at a time and the movement of the players in the query is compared with the movement of the respective players in the play model. If the movement of a player in the query is similar to that of the respective player in the play model, the *similarity score* is incremented by 5 (the amount by which the score is incremented has been chosen to be 5 because the information evaluated is of great significance).

In the case of the Break-In play model the movement of Receiver 1 is long-and-turn. When compared with the movement of Player 1 the system discovers that the movement is not similar and the *similarity score* remains 0. Similarly the movement of Player 4 is different and the score is not incremented. The movements of Players 2, 3 and 5 are similar to those of the respective players in the Break-In model. Therefore the score is incremented by 15 (5 for each of the 3 players). At the end of the player movement comparison the *similarity score* for the Break-In play model is 15.

Player Significance — The system uses the significance of the players in the query to further differentiate between the candidate play models. For each significant player in the query the *similarity score* is incremented by 5. When the players in the query were labelled using the Break-In play model, the labels were: *receiver1*, *receiver2*, *tail back*, *receiver3* and *receiver4*. All the players are significant and

the system increments the *similarity score* of the Break-In play model by 25.

Player Relationships — The system then analyses the relationships between the players in all the frames in the video sequence. For each time instance in the query play the system takes one player and develops a symbolic representation of his relationship with his teammates. It then compares the relationships with those of his matching player in the known play model and each time a relationship match is found the *similarity score* is incremented by 1 (the amount by which the score is incremented has been chosen to be 1 because the information that evaluates is of smaller significance).

The relationships of the players in the query at time instance 1 are as follows:

Player 1 —

BEHIND: Player 2, 3, 4, 5;

TO-THE-RIGHT: Player 2, 3, 4, 5;

IN-LINE-WITH-HORIZONTAL: Player 4,

Player 2 —

IN-FRONT: Player 1, 4;

BEHIND: Player 3, 5;

TO-THE-RIGHT: Player 3, 4 5;

TO-THE-LEFT: Player 1,

Player 3 —

IN-FRONT: Player 1, 2, 4, 5;

TO-THE-RIGHT: Player 4, 5;

TO-THE-LEFT: Player 1, 2,

Player 4 —

BEHIND: Player 2, 3, 5;

TO-THE-RIGHT: Player 5;

TO-THE-LEFT: Player 1, 2, 3;

IN-LINE-WITH-HORIZONTAL: Player 1,

Player 5 —

BEHIND: Player 3;

IN-FRONT: Player 1, 2, 4;

TO-THE-LEFT: Player 1, 2, 3, 4.

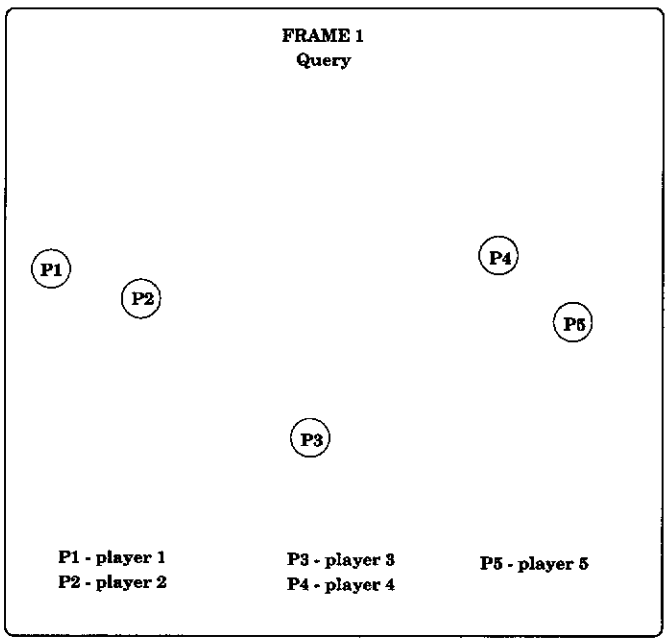


Figure 6.21: The position of the players in the query at time instance 1.

Now consider The Break-In model. The system compares the relationships of *receiver1* in the play model with the relationships of player 1 in the query. The relationships of *receiver1* are:

BEHIND: **receiver** 2, 3, **running-back**;

TO-THE-RIGHT: Player **receiver** 2, 3, 4, **running back**;

IN-LINE-WITH-HORIZONTAL: **receiver** 4.

The difference between the relationships of *receiver1* and player 1 from the query are: *receiver1* is *IN-LINE-WITH-HORIZONTAL* with *receiver4* (which is player 5 in the query), *receiver1* is *not IN-LINE-WITH-HORIZONTAL* with *receiver3* (which is player 4 in the query), and *receiver1* is *not IN-FRONT-OF receiver4*.

The rest of the relationships for *receiver1* are the same as those of player 1 in the query and the score is incremented by 7 (7 relationships match — for each relationship that matches, the score is incremented by 1). The remaining players

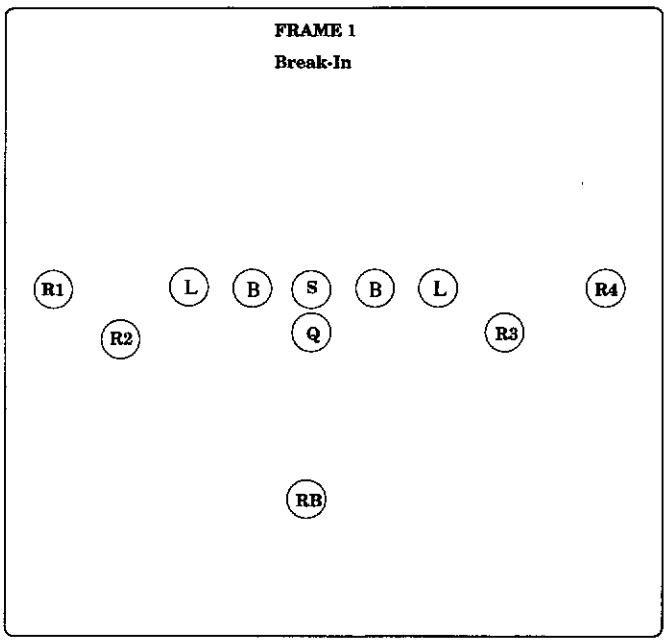


Figure 6.22: The position of the players in the Break-In play model at time instance 1.

are also considered by the system and the score is incremented by 26. The total evidence for the Break-In play model after the first time instance was considered is 73. This is repeated for each time instance in the video sequence (6 instances) and after all the instances have been considered the total score for the play is 181.

The five remaining candidates are processed in a similar fashion and the scores for the play models after the player relationships are examined are shown in Table 6.1.

Play-Model-Name	Play Similarity Score
HB-Draw (4wr)	239
Break-In (4wr)	181
In-Out (4wr)	176
Quick-Slant (3wr)	168
Quick-Slant (normal)	159

Table 6.1: Similarity score for the play models after the player relationships were considered.

When this stage has been completed, the model emerging as the best solution is HB-Draw. Though the differences in score between HB-Draw and the rest of the candidate plays is large, this is not always the case and more processing is necessary to increase the confidence in the classification of the query.

6.3.5.5 Temporal Analysis

To perform the temporal analysis the system takes the movement of each player in the query and describes it as a series of *moves* and *turns*. The descriptions of the player movements are afterwards used to build the temporal description of the query play (see Figure 6.23).

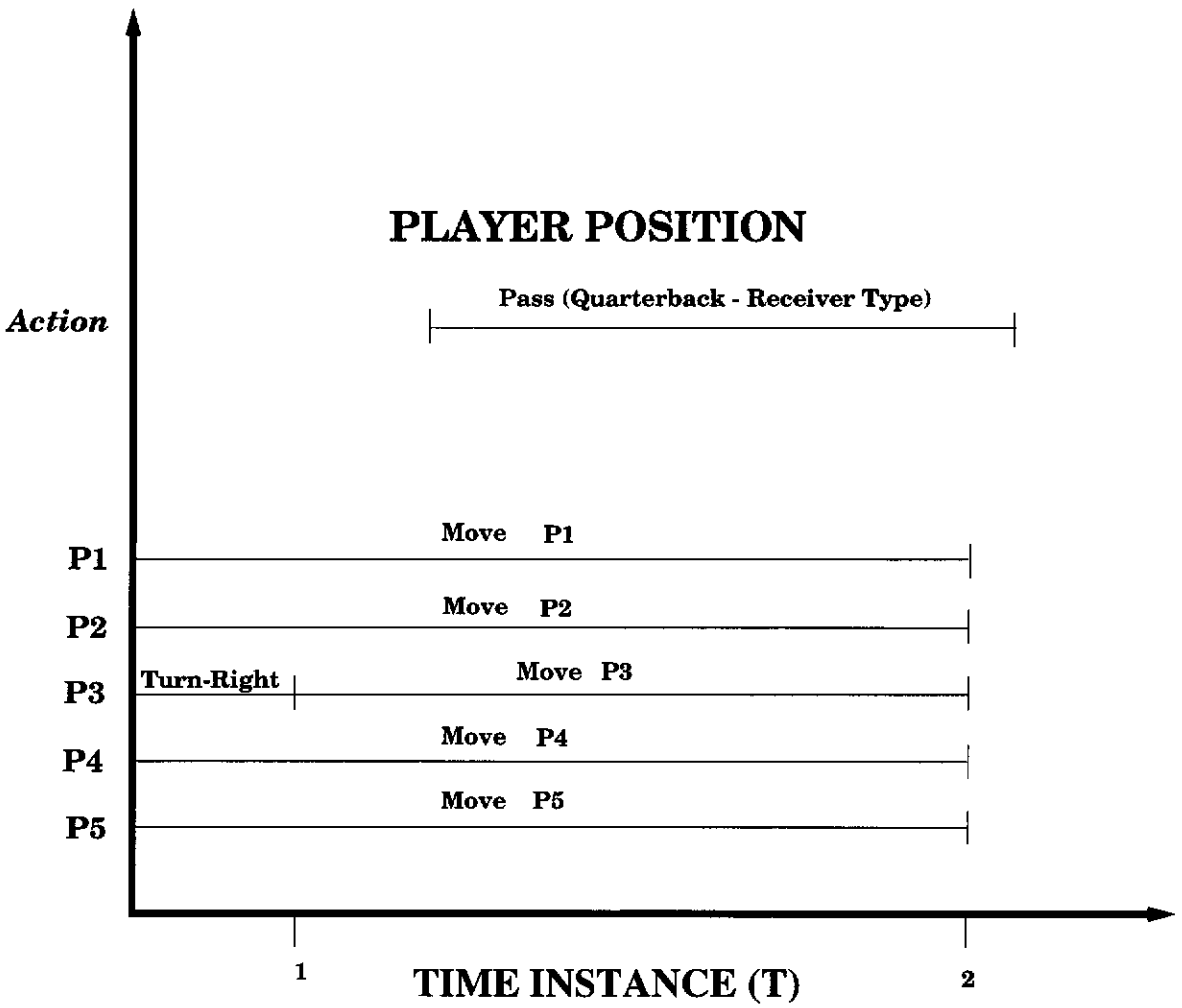


Figure 6.23: Temporal description for the query.

Each player from the query has his movement compared to that of the corresponding player in the candidate model and if the same action occurs in a similar time interval, the *similarity score* for the candidate play model is incremented by 5.

The Break-In play model (see Figure 6.24 for a temporal description of the play model) is considered by the system and Player 1 in the query has his movement compared to the movement of *Receiver* 1 in the model (see Figure 6.25).

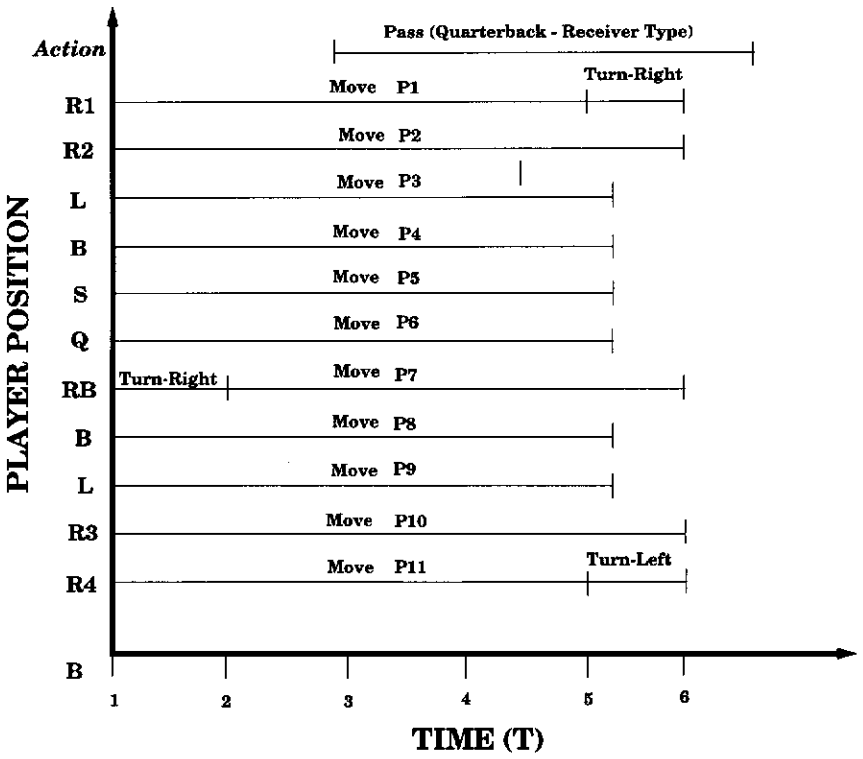


Figure 6.24: The temporal description of the Break-In play model.

Both players move in the same time to predefined positions so the action matches. The score is incremented by 5. In the candidate model temporal description, *Receiver* 1 makes a turn while Player 1 in the query continues his movement so the action does not match. Players 2, 3, 4 and 5 are processed in the same manner. The *similarity score* after the time analysis for the Break-In play is 201. The results for all candidate plays are shown in Table 6.2.

Play-Model-Name	Play Evidence Score
HB-Draw (4wr)	264
Break-In (4wr)	201
In-Out (4wr)	191
Quick-Slant (3wr)	188
Quick-Slant (normal)	174

Table 6.2: Final Result. The best is match the HB-Draw play.

This shows that the temporal analysis improved the difference between the best and the next best interpretation, increasing the the confidence in this result.

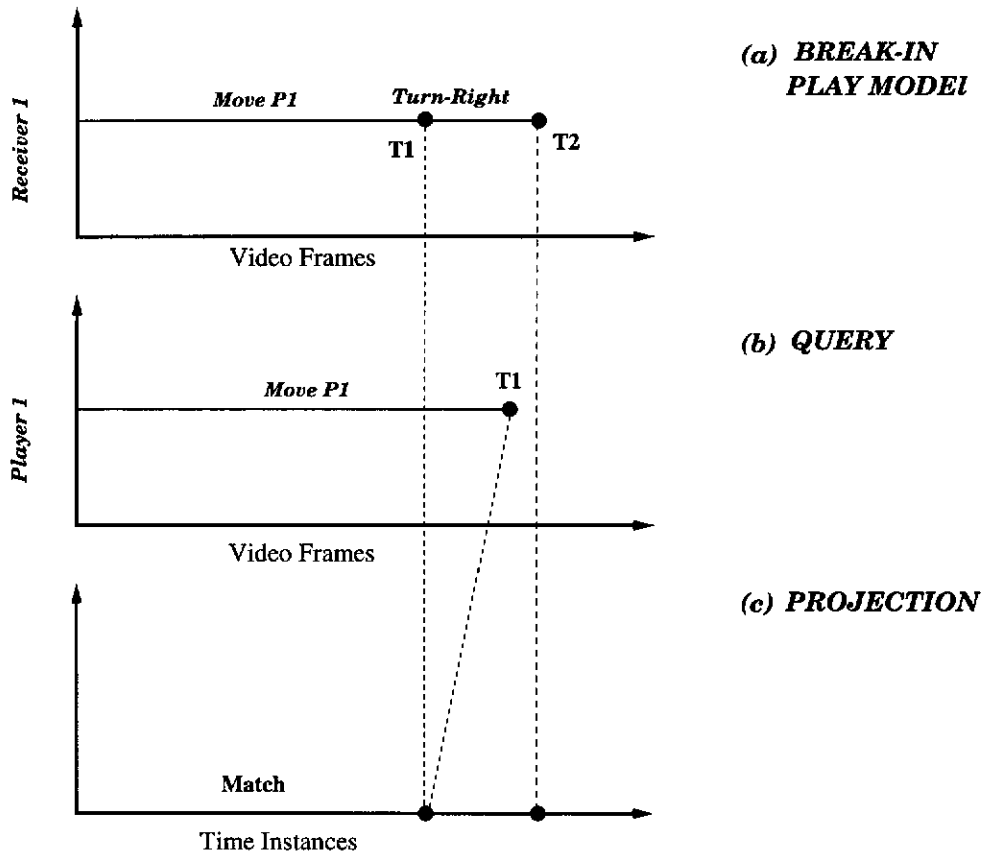


Figure 6.25: The aligment of the actions of Player 1 and Receiver 1.

6.4 Classification Results

Several tests were conducted. We describe two more results in detail and include the others in Appendix C.

6.4.1 Test Sequence 2

The video frames of the play are shown in Figures 6.27 and 6.28 and the transcript of the text for the play is in Figure 6.26. Four players have been segmented and the text processing reveals the play frame in Table 6.3.

PlayerX to the 25 yard line for another Team1 1st down .

Figure 6.26: Transcript of the commentary for the play shown in Figures 6.27 and 6.28.

Play-name	Run
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	Running-Back
Player2-Type	-
Instance	1

Table 6.3: Play frame.

The solutions returned by the system are shown in Table 6.4.

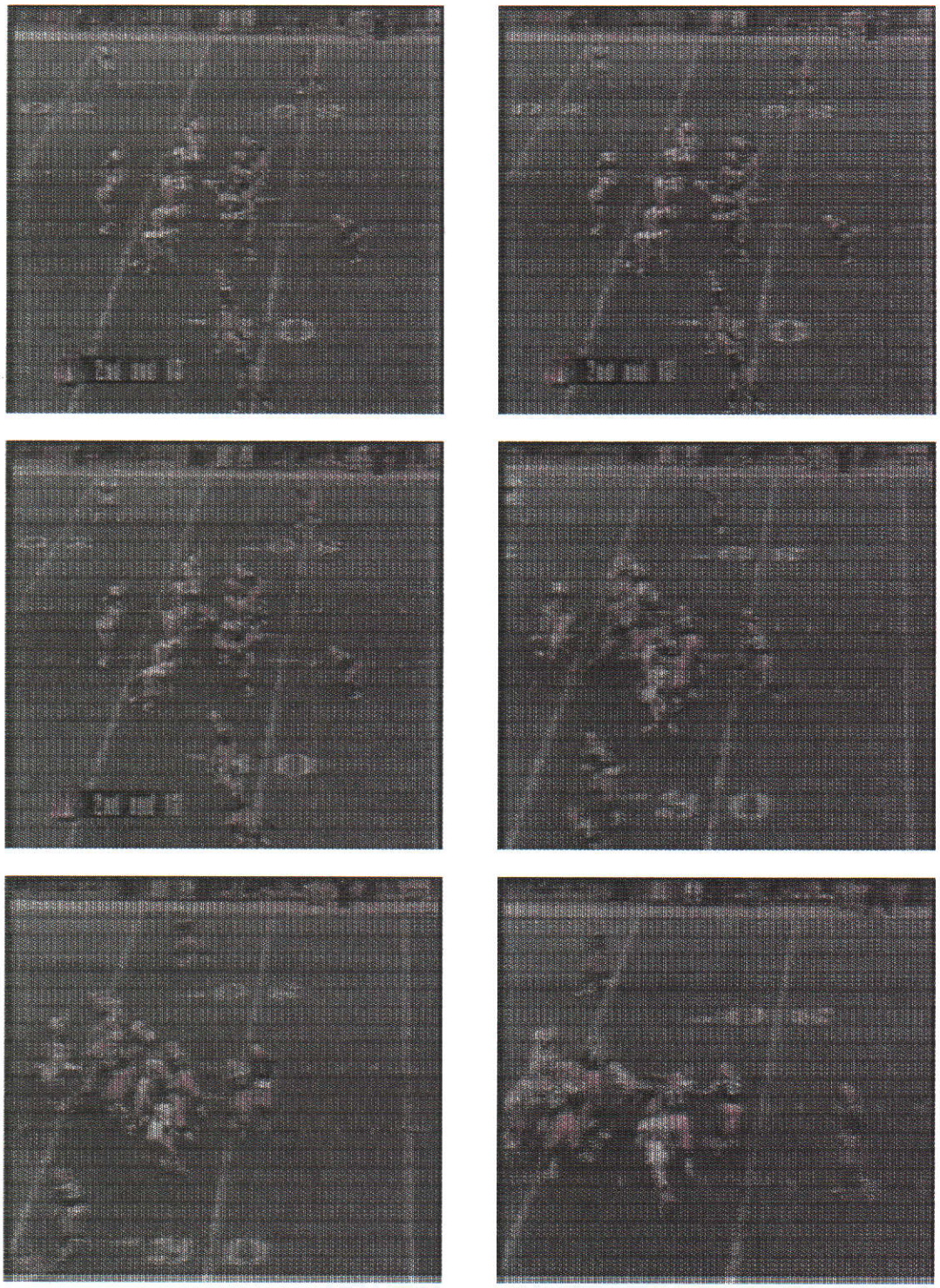


Figure 6.27: Video frames from test sequence 2. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

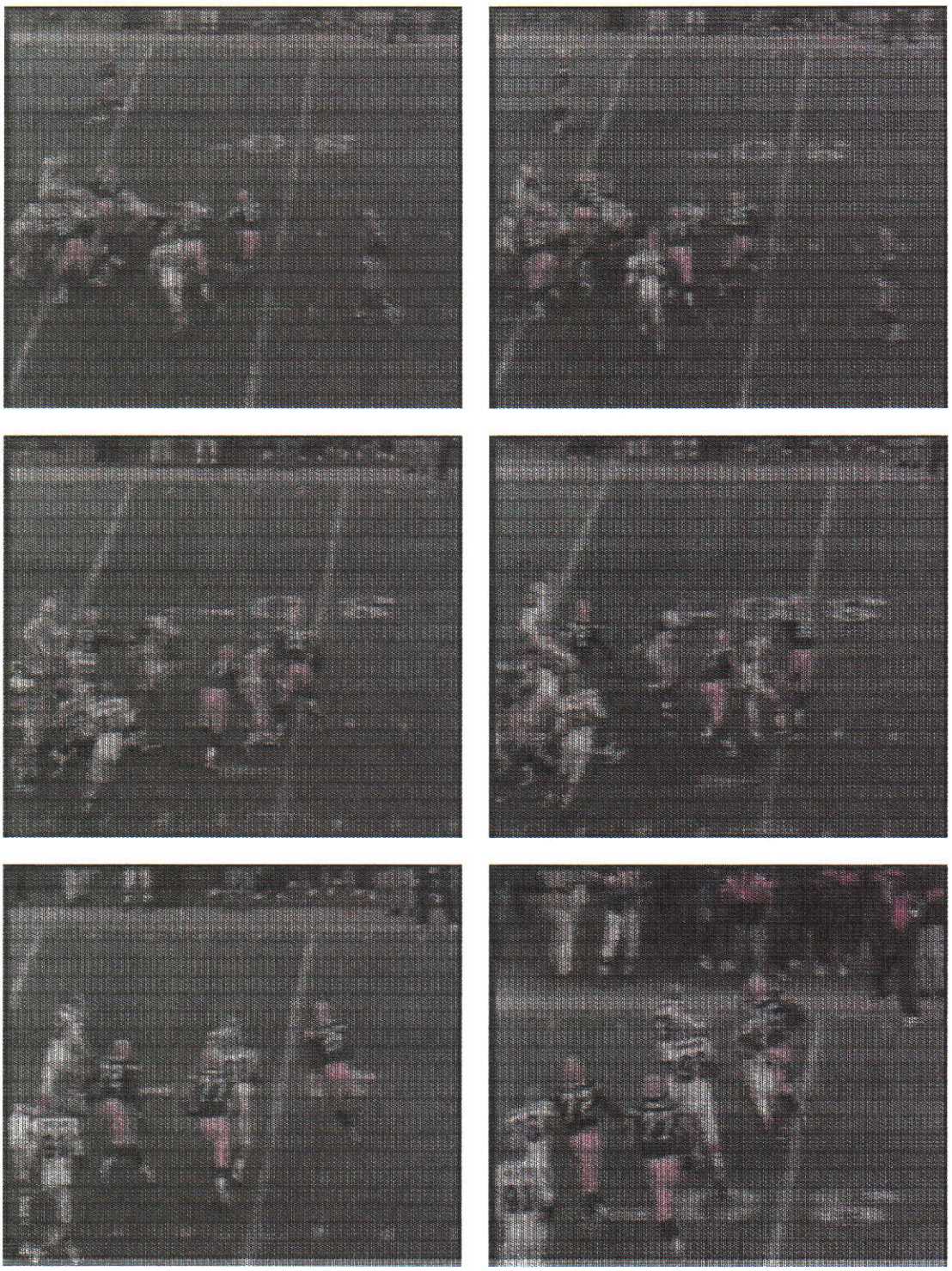


Figure 6.28: Video frames from test sequence 2. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

Play-Model-Name	Play Similarity Score
Single-Back (normal)	245
Single-Back (3wr)	219
HB-Draw	211
Belly-Strong	194
HB-Dive	174

Table 6.4: The solutions and their respective similarity scores.

6.4.2 Test Sequence 3

The video frames of the play are shown in Figures 6.30 and 6.31 and the transcript of the text for the play is in Figure 6.29. Three players have been segmented and the text processing generates two play frames shown in Tables 6.5 and 6.6.

PlayerX had enough speed to get to the 39 yard line, PlayerY made the tackle .

Figure 6.29: Transcript of the commentary for the play shown in Figures 6.30 and 6.31.

Play-name	Run
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	Running-Back
Player2-Type	-
Instance	1

Table 6.5: Offensive Play Frame.

Play-name	Tackle
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerY
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Table 6.6: Defensive Play Frame.

The solutions found by the system are shown in Table 6.7.

We have carried out on the recognition system using real data. Of the 20 tests the system performed correctly 18 times. It failed when the text was irrelevant to the game. If only video analysis is used, it correctly classifies *only* 12 of the 20 plays which demonstrates that by combining natural language processing with video processing the accuracy of the classification is improved.

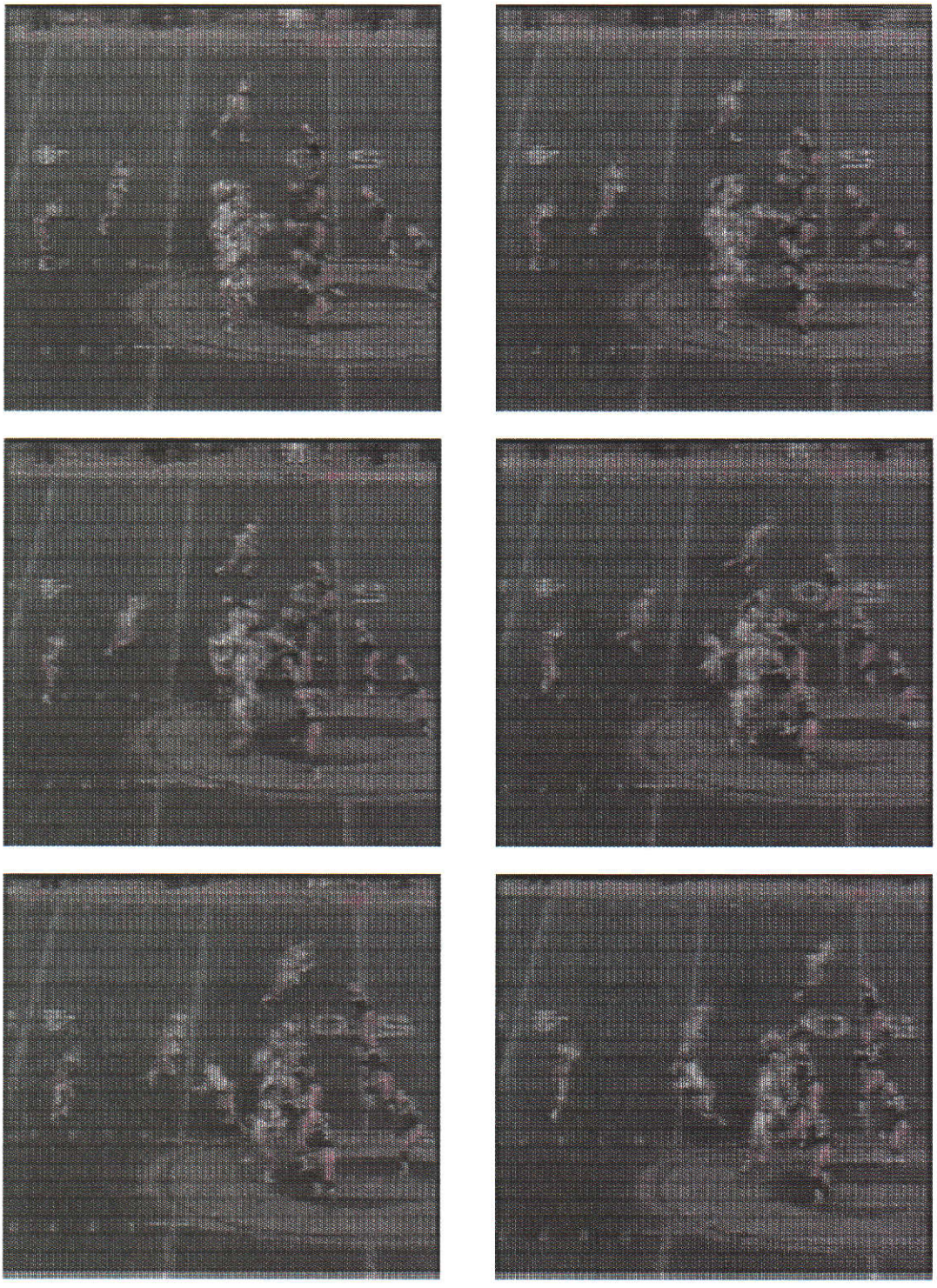


Figure 6.30: Video frames from test sequence 3. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

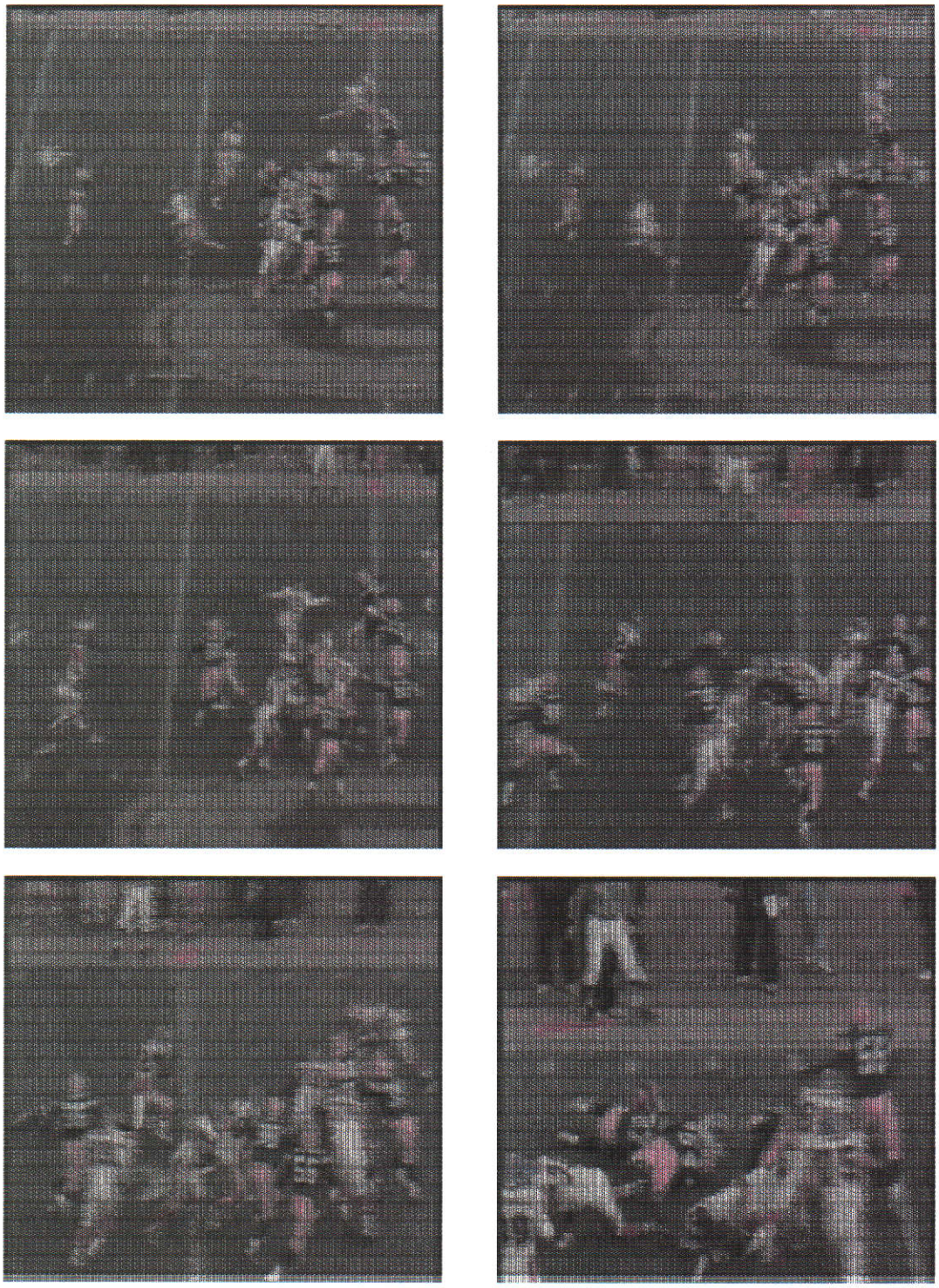


Figure 6.31: Video frames from test sequence 3. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

Play-Model-Name	Play Evidence Score
In-Out	261
Quick-Outs	236
Square-In	224
HB-Draw	210
HB-Dive-Right	198

Table 6.7: Solution table for test sequence 3.

6.5 Summary

This chapter has described processing and results for American Football using ILF with no emphasis on the learning used. ILF was used to generate the concept hierarchy that stores the play models but no forgetting was used. The reason for using only the learning part of ILF is that the models used to build the hierarchy were generated from data extracted from coaching books and video – no forgetting was necessary (all the knowledge used in the models was expert knowledge). Classification of the query plays is only part of the application. The system also has the capacity to learn from the data it processes and update existing American Football play models. This is necessary as play models evolve over time (over a game, over a season) requiring concept drift (in this case play model drift) tracking. The following chapter describes how the system learns spatio-temporal information from the queries it processes and how it continuously updates its models.

Chapter 7

Incrementally Learning Spatio-Temporal Patterns

7.1 Introduction

The previous chapter describes the recognition of N.F.L. plays using a complex decision tree generated by the system using play information only from coaching books. In this chapter we explore issues of machine learning using ILF. Specifically we are concerned with how the system can learn new facts from the input data by either modifying a known model, by building a completely new model, or by filling in the knowledge gaps that may be present in previously built models.

When the system examines a new query, it attempts to find the best match for it in the memory. If the system is successful in finding a model that matches the query, it checks its similarity coefficient. If the coefficient is greater than or equal to a threshold (in the work described in this thesis, the threshold is 0.75) the system searches for any existing differences between the model and query. The differences found are then used to build generalised descriptions, which include the old facts from the model and new facts from the query.

If the coefficient is smaller than a threshold then there is not enough evidence to justify the update of the model. The system therefore builds a new model that is based on the query.

Another case for which the system builds a new concept is when the search for a model that matches the query is unsuccessful.

When a new concept is necessary, the system assigns the values and descriptions generated from the input data to the attributes in the model structure described

previously. The query data is not complete and in the best case the query will contain information about five or six players. All American Football plays involve 11 players and hence gaps in the knowledge *will be present* in any newly created concept. These gaps can only be filled in the future when the system *will learn new facts from similar queries.*

When the model has to be updated to be consistent with the new query, the spatial and temporal attribute descriptions go through a process of generalisation. The layout of the chapter is as follows: in Section 2 we describe the process used to generalise spatial information. Section 3 presents the process involved in generalising temporal information. In Section 4 we show the classification results without learning while in Section 5 we cover the classification results with learning and forgetting.

7.2 Generalising the Spatial Information

In the American Football example there are two spatial attributes in the play model: the player movement and the player relationship.

Generalising the description of the player movement is simple and involves the union of the player movement description in the model with that of the corresponding player in the query. For example if the player movement description in the model was *short-and-straight* and the movement of the corresponding player in the query was defined as *short-and-turn*, then the new generalised description of the player becomes *short-and-straight* **or** *short-and-turn.*

The process of updating the relationship description requires that the order in which the relationships changes occur is preserved. For example consider the relationships of *Player1* and *Player2* in Figure 7.1. Figure 7.1a shows how the relationships between the two players are changing over the duration of the play in the case of the model while Figure 7.1b shows the relationships between the two players in the query.

When the generalisation takes place the system attempts to enlarge the description by adding to it those relationships in the query which are different from those in the model while still keeping the time index at which they took place. The updated description is shown in Figure 7.1c. Notice that now the relationships that occur both in the query and the model can be represented — Player2 now has two patterns.

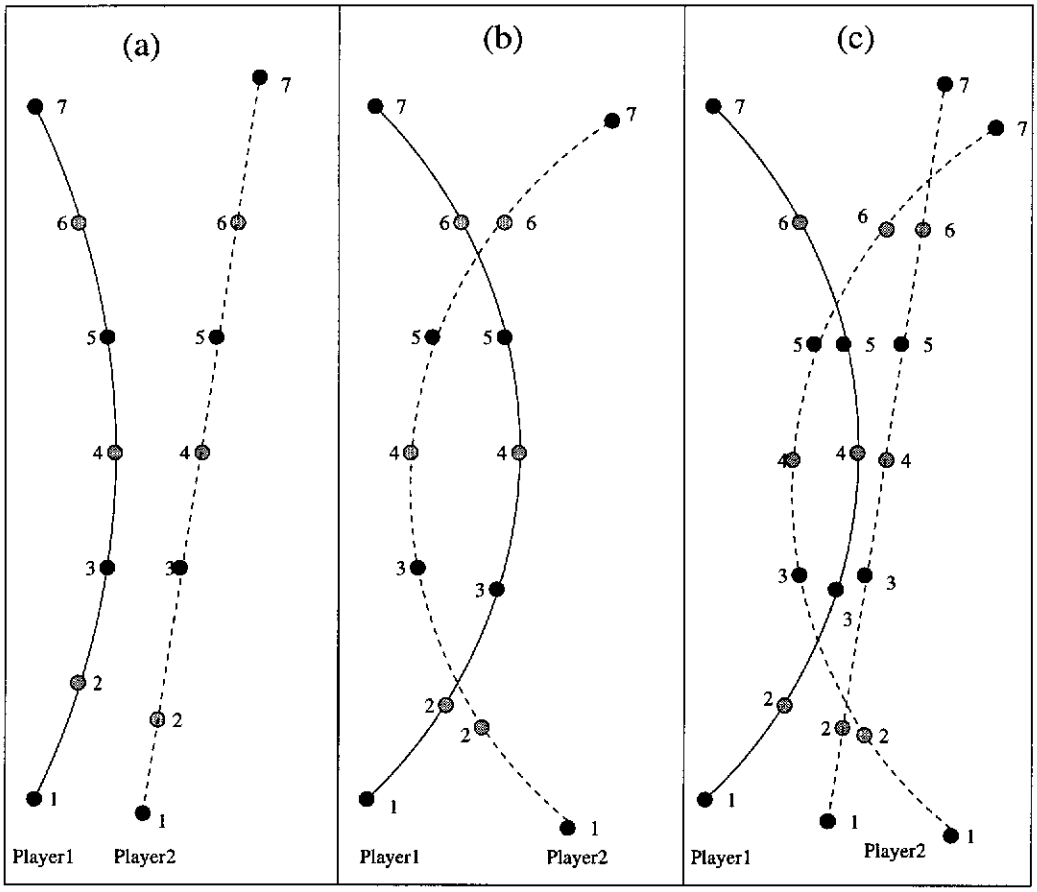


Figure 7.1: The filled circles indicate the position of the players at different instances in time for the entire duration of the play. To update the model (a) the new facts from the query (b) have to be incorporated into the relationship description. After this is done, Player2 has two patterns in the description (c).

7.3 Generalising the Temporal Information

The temporal description is updated in a similar fashion to the spatial relationship. However, in the case of the temporal description the system deals with actions (movement, turns and path crossing) and the occurrence of the actions is defined using Allen's temporal primitives. Consider the movement and actions of the players 1 and 2 in Figures 7.2. The movement of the players in the model is simple when compared to that of the players in the query. For example the movement of player 1 in the query consists of six moves and five turns whereas the movement of player 1 in the model is made up of only four moves and three turns. The systems updates the model to include the new information from the query by adding the new actions as well as maintaining the temporal ordering between the actions. Figure 7.3 shows the model after the update.

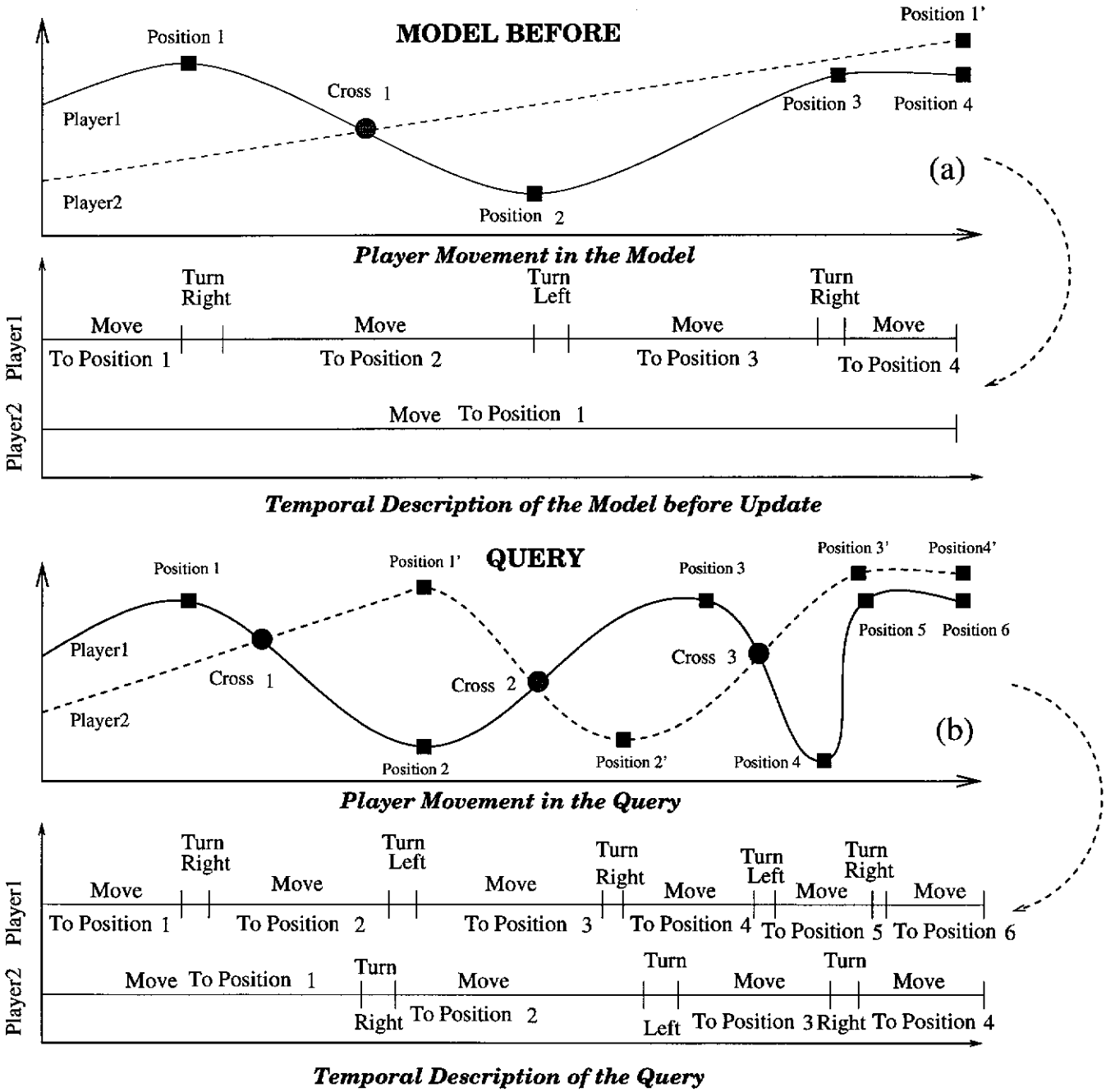


Figure 7.2: The movement and actions of Player1 and Player2 is similar in both the model and the query up to the instance when they cross each other's path. From that instance on the movement and actions are different in the query.

The algorithm to generalise the spatial and temporal information is shown in Figure 7.4 and 7.5. It has five functions. The *Generalise* function controls the processing of new instances. The *Match_Spatial* function determines if two sets of actions are similar while the *Update_Spatial* adds the new information considered to be relevant from the query to the model. Similarly the *Match_Temporal*

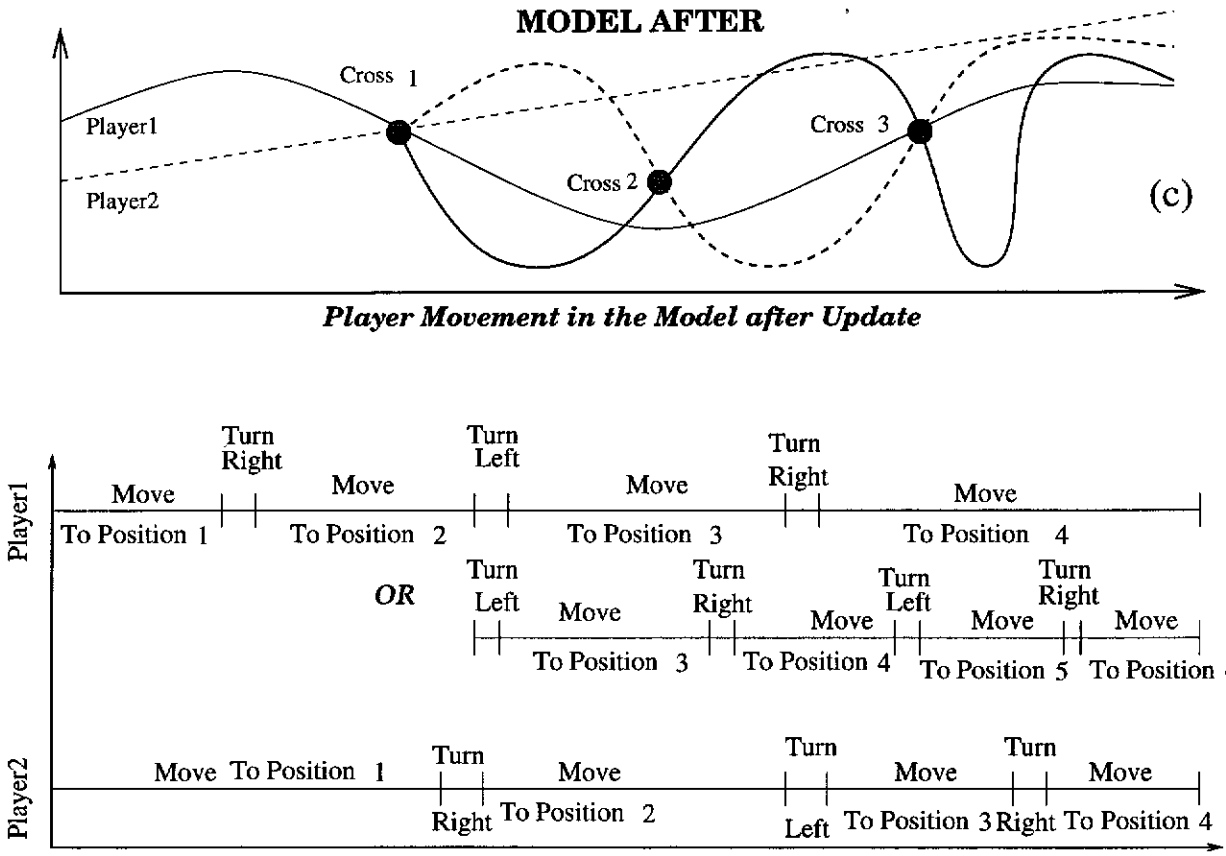


Figure 7.3: To reflect the new data from the query the system updates the temporal description by adding the different movements and actions from the query. The resulting patterns that can be recognised after the changes are made are shown along with the new model.

function determines if two sets of actions are similar while the *Update_Temporal* adds the new temporal information extracted from the query.

7.4 Learning and Classification Results

Over 303 tests were carried out on the system using generated data. Initially the system had a training session in which instances of 50 basic plays were input. The play models developed covered 5 basic formations (Pro-formation, I-formation, Single-Back-formation, Goal-line-formation and Far-formation) performing a combination of 43 running and passing plays. At the end of the training stage the system's memory contained 43 play models.

```
-------------------------------------------------------------------------------
Let Query-Object1, Query-Object2, ..., Query-Objectn be the labels of the objects in the Query.
Let Model-Object1, Model-Objectj2, ..., Model-Objectm be the labels of the objects in the Model.
Let Query-Object1(Rel1)Query-Object2, Query-Object1(Rel2)Query-Object2,
    Query-Object1(Rel3)Query-Object2, Query-Object1(Rel4)Query-Object2,
    Query-Object1(Rel5)Query-Object2 and Query-Object1(Rel6)Query-Object2 be the six
    valid relationships between objects Query-Object1 and Query-Object2 where
    Rel1 is Left-of, Rel2 is Right-of, Rel3 is In-Front, Rel4 is Behind, Rel5 is
    In-Line-Horizontal and Rel6 is In-Line-Vertical.

Let Query-Object1(Rel1)Query-Object2 == True is Query-Object1 is Left-Of Query-Object2.

Let Model-Object1, Model-Object2, ..., Model-Objectm be the labels of the players in the Model.
Let Model-Object1(Rel1)Model-Object2, Model-Object1(Rel2)Model-Object2,
    Model-Object1(Rel3)Model-Object2, Model-Object1(Rel4)Model-Object2,
    Model-Object1(Rel5)Model-Object2 and Model-Object1(Rel6)Model-Object2 be the six
    valid relationships between players Model-Object1 and Model-Object2 where
    Rel1 is Left-of, Rel2 is Right-of, Rel3 is In-Front, Rel4 is Behind, Rel5 is
    In-Line-Horizontal and Rel6 is In-Line-Vertical.

Let Model-Object1(Rel1)Model-Object2 == True is Model-Object1 is Left-Of Model-Object2.

Let Query-Object1(A1), Query-Object1(A2), ..., Query-Object1(Ak) be the actions of object
    Query-Object1 in the Query.

Let Model-Object1(B1), Model-Object1(B2), ..., Model-Object1(Bk) be the actions of object
    Model-Object1 in the Model.

-----------
Generalise.
-----------
Similarity_Score = 0;
Match_Spatial(Query-Object,Model-Object).
Match_Temporal(Query-Object,Model-Object).

 if (Similarity_Score >= Threshold) then
   Update_Spatial; Update_Temporal;

-----------------------
Match_Spatial(Query-Object,Model-Object).
-----------------------
 for i = 1 to n
  for j = 1 to m
   if Query-Objecti == Model-Objectj then
    for k = 1 to n
     for p = 1 to m
      if Query-Objectk == Model-Objectp and i != k and m != p then
       for s = 1 to 6
        if Query-Objecti(Rels)Query-Objectk == True and
           Model-Objectm(Rels)Model-Objectp == True then
            Similarity_Score++;

-----------------------
Update_Spatial(Query-Object,Model-Object).
-----------------------
 for i = 1 to n
  for j = 1 to m
   if Query-Objecti == Model-Objectj then
    for k = 1 to n
     for p = 1 to m
      if Query-Objectk == Model-Objectp and i != k and m != p then
       for s = 1 to 6
        if Query-Objecti(Rels)Query-Objectk != True and
           Model-Objectm(Rels)Model-Objectp == True then
            add Query-Objecti(Rels)Query-Objectk to Model-Objectm(Rels)Model-Objectp;
```

Figure 7.4: The algorithm to generalise the spatio-temporal information.

```
------------------------
Match_Temporal(Query-Object,Model-Object).
------------------------
 for i = 1 to n
  for j = 1 to m
   if Query-Objecti == Model-Objectj then
    for p = 1 to k
     if Query-Objecti(Ap) == Model-Objectj(Bp) then
       Similarity_Score++;
       if Query-Objecti(Ap-1) == Model-Objectj(Bp-1) then
         Similarity_Score++;

------------------------
Update_Temporal(Query-Object,Model-Object).
------------------------
 for i = 1 to n
  for j = 1 to m
   if Query-Objecti == Model-Objectj then
    for p = 1 to k
     if Query-Objecti(Ap) != Model-Objectj(Bp) then
       add Query-Objecti(Ap) to action set of Model-Objectj;
-------------------------------------------------------------------------------
```

Figure 7.5: The algorithm to generalise the spatio-temporal information (continued).

7.5 Test Data

The test input data consisted of query plays which contained information about 3, 4 or 5 players (the information was in the form of lists of coordinates which represented a series of video frames). Also the queries were arranged into two sets. The first set contained "easy" queries with only one of the players in the query having a different movement to that in the play model. The second set contained "hard" queries with one, two or three players having different movements. A detailed run of what the system learns from a simple query is described next.

A test consists of inputting a query play (which could be either easy or hard) which the system would attempt to match against the play models in the system's memory. The best five matches are returned as possible solutions to the query. A total of 303 different queries were used for the tests (103 easy queries and 200 hard queries). When the classification is carried out without any learning the results obtained are shown in Tables 7.1 and 7.2. For the first set the recognition success rate is 65% but for the second set, the success rate is only 49%. In most cases where it failed, the system is still able to identify the correct basic formation.

Basic Formation	No. of Tests Run	Correct Classification	Incorrect Classification
Pro-formation	40	28	12
I-formation	40	26	14
Single-Back-formation	23	13	10

Table 7.1: First set of tests — No Learning (Easy Queries).

Basic Formation	No. of Tests Run	Correct Classification	Incorrect Classification
Pro-formation	80	45	35
I-formation	60	27	33
Single-Back-formation	40	18	22
Goal-line-formation	20	8	12

Table 7.2: Second set of tests — No Learning (Hard Queries).

When the classification is carried out with incremental learning the system produced the results shown in Tables 7.3 and 7.4. For the first set the recognition success rate is 80% while for the second set the success rate is 64%.

Basic Formation	No. of Tests Run	Correct Classification	Incorrect Classification
Pro-formation	40	36	4
I-formation	40	33	7
Single-Back-formation	23	14	9

Table 7.3: First set of tests — with Incremental Learning (Easy Queries).

Basic Formation	No. of Tests Run	Correct Classification	Incorrect Classification
Pro-formation	80	52	28
I-formation	60	42	18
Single-Back-formation	40	21	19
Goal-line-formation	20	13	7

Table 7.4: Second set of tests — with Incremental Learning (Hard Queries).

The accuracy of the classification is significantly affected by the hypothesis generated by the system. In many of the cases where it fails to accurately classify the query, the system generates a hypothesis which makes it difficult to distinguish between the basic formations. This is due to the fact that the formations *Far-formation* and *Pro-formation*, as well as *I-formation* and *Goal-line-formation* are very *similar*.

The main difference between the two cases (with *vs* without learning) is that the system with learning is able to handle the situations where the variations between the query and model are significant more successfully (such as the case where the query contains information about 5 players and where 3 of the players are moving differently when compared with the model).

7.6 Summary

In this chapter we have described how ILF is used to learn spatio-temporal patterns. We also presented how we assessed ILF in the task of learning and classifying American Football plays.

In the next chapter we show how ILF can be used to classify and learn cricket shots. The cricket application uses camera motion parameters to generate sym-

bolic descriptions of the cricket shots. Unlike the case of American Football, the system does not use any natural language in the processing.

Chapter 8

An application: Cricket

8.1 Introduction

We describe another application of ILF— cricket. For American Football natural language understanding was combined with image processing and machine learning. For cricket, we concentrated on image processing and learning. The reason for choosing cricket is that though cricket is a lot simpler than American Football, it still involves well defined discrete actions — *cricket shots* — for which accurate representations can be built. We attempted to extend the American Football learning and classification techniques to other sports and initially it was intended to apply the same techniques as developed for American Football to cricket. However, because of the different nature of the game, a different approach is required.

We define a cricket shot as the way in which the batsman hits the ball in terms of the direction in which the ball is hit and the distance covered by the ball. Then we combine camera motion estimation with incremental learning to classify and learn the cricket shots.

The problem of estimating motion parameters has been researched extensively in the past since it can provide a simple, fast and accurate way to search multimedia databases for specific shots (for example a shot of a landscape is likely to involve a significant amount of pan, whilst a shot of an aerobatic sequence is likely to contain roll).

To learn the description of the cricket shots we use the ILF algorithm. The layout of this chapter is as follows: in Section 2 we present some of the techniques used to extract camera motion parameters and are relevant to the application described in this chapter. Section 3 describes the processing involved in extracting and

converting the camera motion parameters. In Section 4 we describe a method to classify and learn cricket shots using ILF whilst, in Section 5 we show the results obtained from testing the system using real data.

8.2 Motion Parameter Estimation

The problem of estimating motion parameters has been researched extensively in the past since it provides a simple, fast and accurate way to search multimedia databases for specific shots (for example a shot of a landscape is likely to involve a significant amount of pan, whilst a shot of an aerobatic sequence is likely to contain roll). Estimating camera motion parameters is of great relevance to the work described in this chapter because an important part of the cricket application is extracting camera motion parameters. In this section we describe a number of methods that have been used to estimate motion parameters.

The method of Bergen *et al.* [62] is based on two models: a global model that constrains the overall motion estimated and a local model that is used in the estimation process. Affine flow, planar flow, rigid body motion and general optic flow are the four specific models chosen. The same objective function is used in all models and the minimisation is performed with respect to different parameters.

Akutsu *et al.* [1] have proposed a method based on analysing the distribution of motion vectors in Hough space. Seven categories of camera motion are estimated: pan, tilt, zoom, pan and tilt, pan and zoom, tilt and zoom, and pan, tilt and zoom. Estimation of the motion parameters is based on Hough-transformed optic-flow vectors measured from the image sequence and determining which of the signatures best matches the data in a least squares sense.

Park *et al.* [88] describe a method of estimating camera parameters that establishes feature based correspondence between frames. The camera parameters representing zoom, focal length and 3D rotation are estimated by fitting the correspondence data to a transformation model based on perspective projection.

Tse and Baker [117] present an algorithm to compensate for camera zoom and pan. The global motion in each frame is modelled by just two parameters: a zoom factor and a pan and tilt factor based on local displacement vectors found by conventional means.

Wu and Kittler [129] present a technique to extract rotation, change of scale, and translation from an image sequence without establishing correspondence. A multi-resolution iterative algorithm that uses a Taylor series approximation of the spatial and temporal gradient of the image is used.

Hoetler [52] describes a differential motion estimation technique in which the global motion in the image due to zoom and pan are estimated. A three-parameter motion model is used and a reduction in the systematic estimation errors due to displacement measurement is reported.

To extract the camera motion parameters from a sequence of images we use a method developed by Srinivasan *et al* [76]. The method uses residual optic flow to detect and estimate camera pan, tilt, zoom, roll, and horizontal and vertical tracking. Unlike most other comparable techniques, this method can distinguish pan from horizontal tracking, and tilt from vertical tracking. The procedure used to extract and process camera motion parameters for the cricket application is described in the next section.

8.3 Extracting and Converting the Camera Motion Parameters

There are two ways in which a cricket shot can be determined. One method involves segmenting out either the batsman or the ball. Unlike American Football where it is possible to track *some* of the players in the play, in cricket it is much more difficult to track the batsman because of two reasons: the high speed of the shot (few frames and too much blur as shown in Figure 8.1) and the batsman is often confused with the wicket keeper. Furthermore, even if the batsman could be consistently segmented out from the image, it is still difficult to distinguish the action of the batsman (the bat cannot be identified consistently so its pattern of movement cannot be accurately classified). It is similarly difficult to segment out the cricket ball in video especially since the cricket ball is *small* and difficult to distinguish from the background (generally there is no *significant* difference between the ball and background). An alternative method is to use the camera motion parameters to determine the path of the ball because throughout the cricket game the camera generally focuses on the ball trajectory and hence it is possible to generate a hypothesis on the type of cricket shot based on the camera motion parameters, which in turn define the direction of the ball.

The algorithm to process the camera motion parameters has three stages and is shown in Figures 8.2 and 8.3.

In the first stage the system attempts to determine the size of the temporal window that contains the cricket shot, that is the start and end frames of the shot sequence containing the shot (the camera position is assumed to be behind one of the two bowling ends of the cricket ground to capture the bowling action). The cricket action consists of two parts: the bowler action and the batsman action. The bowler's action is defined by a sequence of frames in which there is

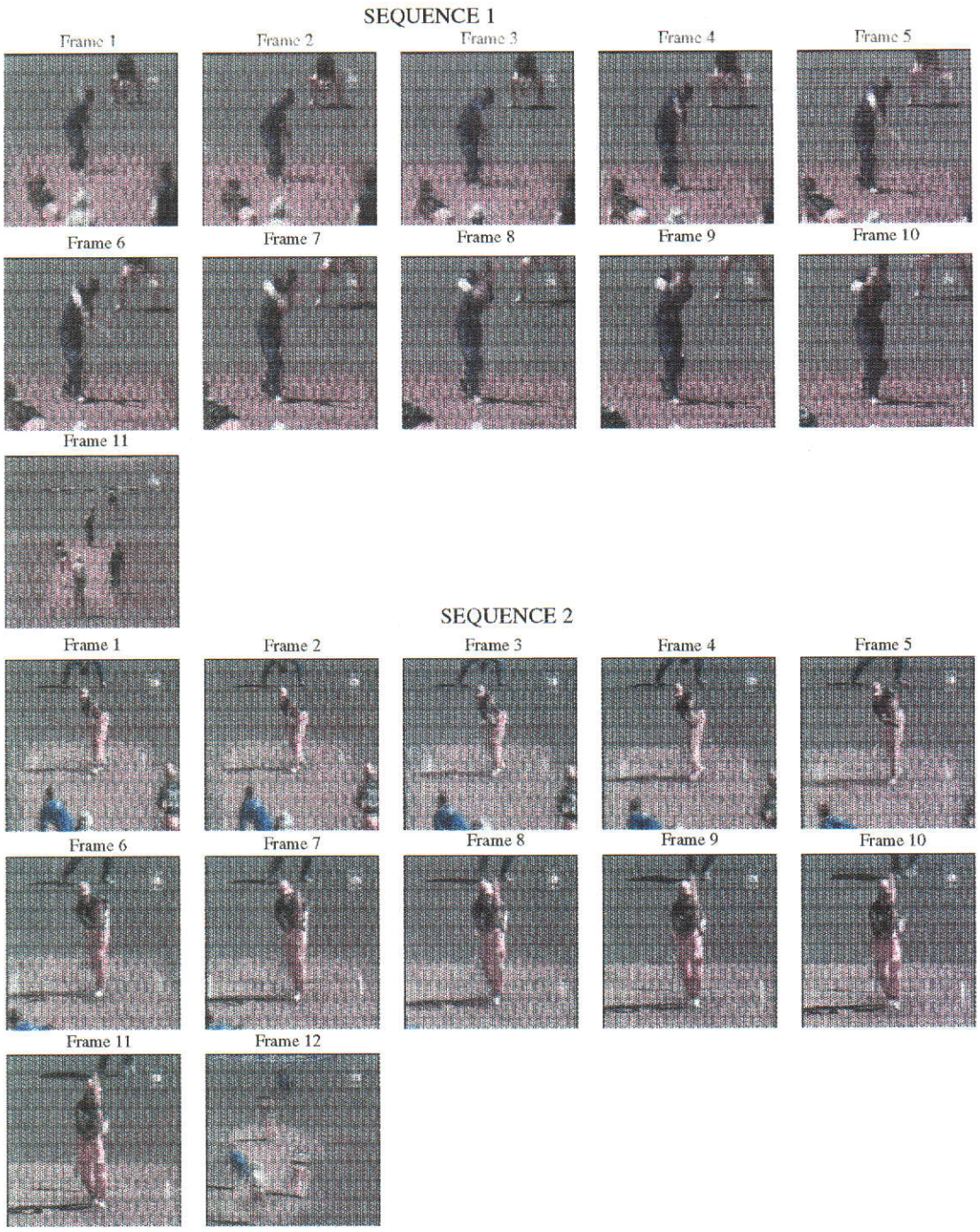


Figure 8.1: Two typical sequences of a batsman attempting a shot. Images provided with the permission of Wide World of Sports - Channel 9 Australia.

a substantial amount of zoom and tilt but little pan. This is because the camera is tracking and zooming on the bowler and the cricket ball. Both the bowler and the ball move fairly straight and hence there is no substantial panning. The batsman action is defined by a sudden change in the direction of the ball which

```
-------------------------------------------------------------------------------------------
STAGE 1. Cricket Shot Window Detection
-------------------------------------------------------------------------------------------
Let Pan(V1), Pan(V2), ..., Pan(Vn) be the values of the pan camera parameter over a video
    sequence of n frames and,
    Tilt(V1), Tilt(V2), ..., Tilt(Vn) be the values of the tilt camera parameter over a video
    sequence of n frames and,
    Roll(V1), Roll(V2), ..., Roll(Vn) be the values of the roll camera parameter over a video
    sequence of n frames and,
    Zoom(V1), Zoom(V2), ..., Zoom(Vn) be the values of the zoom camera parameter over a video
    sequence of n frames.
Let Pan1, Pan2, Pan3, Tilt1, Tilt2, Tilt3, Roll1, Roll2, Roll3, Zoom1, Zoom2 and Zoom3 be
    temporary values of the Pan, Tilt, Roll and Zoom camera parameters.
Let DirectionX be the direction in which the camera is moving along the x - axis (left/right).
    DirectionY be the direction in which the camera is moving along the y - axis (zoom in/out).
    DirectionZ be the direction in which the camera is moving along the z - axis (up/down).
    Rotation be the direction in which the camera is rotating (clockwise/anticlockwise).
Let T_DirectionX, T_DirectionY, T_DirectionZ and T_Rotation be temporary values of the
    movement and rotation of the camera.

 Pan1 = Pan(V1), Pan2 = Pan(V2), Pan3 = Pan(V3).
 Tilt1 = Tilt(V1), Tilt2 = Tilt(V2), Tilt3 = Tilt(V3).
 Roll1 = Roll(V1), Roll2 = Roll(V2), Roll3 = Roll(V3).
 Zoom1 = Zoom(V1), Zoom2 = Zoom(V2), Zoom3 = Zoom(V3).
 DirectionX = (Pan1 + Pan2 + Pan3)/3; DirectionY = (Tilt1 + Tilt2 + Tilt3)/3;
 DirectionZ = (Zoom1 + Zoom2 + Zoom3)/3; Rotation = (Roll1 + Roll2 + Roll3)/3;
 for i = 4 to n
   T_DirectionX = DirectionX;   T_DirectionY = DirectionY;   T_DirectionZ = DirectionZ;
   T_Rotation = Rotation;
   Update(DirectionX,i); Update(DirectionY,i); Update(DirectionZ,i); Update(Rotation,i);
   if (Sign of T_DirectionX != Sign of DirectionX) || (T_DirectionX/DirectionX > Threshold) then
        Change = True;
   if (Sign of T_DirectionY != Sign of DirectionY) || (T_DirectionY/DirectionY > Threshold) then
        Change = True;
   if (Sign of T_DirectionZ != Sign of DirectionZ) || (T_DirectionZ/DirectionZ > Threshold) then
        Change = True;
   if (Sign of T_Rotation != Sign of Rotation) || (T_Rotation/Rotation > Threshold) then
        Change = True;
-------------------------------------------------------------------------------------------
STAGE 2. Dominant Motion Detection
-------------------------------------------------------------------------------------------
Let Positive_Pan, Negative_Pan, Positive_Tilt, Negative_Tilt, Positive_Zoom, Negative_Zoom,
    Positive_Roll and Negative_Roll be the counters of positive and negative values of the
    camera motion parameter values.
 for i = frame_start to frame_end
  if (Pan(Vi) >= 0) Positive_Pan++;   if (Pan(Vi) < 0) Negative_Pan++;
  if (Tilt(Vi) >= 0) Positive_Tilt++;  if (Tilt(Vi) < 0) Negative_Tilt++;
  if (Zoom(Vi) >= 0) Positive_Zoom++;  if (Zoom(Vi) < 0) Negative_Zoom++;
  if (Roll(Vi) >= 0) Positive_Roll++;  if (Roll(Vi) < 0) Negative_Roll++;

 if Positive_Pan > Negative_Pan then
    Pan_Motion is Right;
 else
    Pan_Motion is Left
 if Positive_Tilt > Negative_Tilt then
    Tilt_Motion is Up
 else
    Tilt_Motion is Down
 if Positive_Zoom > Negative_Zoom then
    Zoom_Motion is Zoom In
 else
    Zoom_Motion is Zoom Out
 if Positive_Roll > Negative_Roll then
    Roll_Motion is Clockwise
 else
    Roll_Motion is Anticlockwise
```

Figure 8.2: Stages 1 and 2 of the camera processing algorithm.

```
-------------------------------------------------------------------------------------------
STAGE 3. Average Value Computation.
-------------------------------------------------------------------------------------------

Let Pan(V1), Pan(V2), ..., Pan(Vn) be the values of the pan camera parameter over a video
    sequence of n frames and,
    Tilt(V1), Tilt(V2), ..., Tilt(Vn) be the values of the tilt camera parameter over a video
    sequence of n frames and,
    Roll(V1), Roll(V2), ..., Roll(Vn) be the values of the roll camera parameter over a video
    sequence of n frames and,
    Zoom(V1), Zoom(V2), ..., Zoom(Vn) be the values of the zoom camera parameter over a video
    sequence of n frames.

 for i = frame_start to frame_end
  if (Pan(Vi) Sign == Pan_Motion Sign) then
   Sum_Pan = Sum_Pan + Pan(Vi); Pan_Count++;
  if (Tilt(Vi) Sign== Tilt_Motion Sign) then
   Sum_Tilt = Sum_Tilt + Tilt(Vi); Tilt_Count++;
  if (Roll(Vi) Sign == Roll_Motion Sign) then
   Sum_Roll = Sum_Roll + Roll(Vi); Roll_Count++;
  if (Zoom(Vi) Sign == Zoom_Motion Sign) then
   Sum_Zoom = Sum_Zoom + Zoom(Vi); Zoom_Count++;

  Average_Pan = Sum_Pan/Pan_Count; Average_Tilt = Sum_Tilt/Tilt_Count;
  Average_Zoom = Sum_Zoom/Zoom_Count; Average_Roll = Sum_Roll/Roll_Count;
```

Figure 8.3: Stage 3 of the camera processing algorithm.

involves a significant amount of pan. The cricket shot ends with a cut when the camera focuses on the crowd, the ground or a player who has fielded the cricket ball. The *cricket shot window* therefore starts at the frame where the system has detected a sudden change in the camera parameters and ends at the frame where the system detects a cut. An example is shown in Figure 8.4. The second stage involves determining the *dominant motion* of the camera in the cricket shot. The reason for checking for a *dominant motion* is that the movement of the camera during a shot is not always smooth and contains varying amounts of zoom and tilt as it all depends on how good the camera man is at tracking the ball: the more experienced, the smoother the action. Hence it is quite likely that a drive on the left side will contain some small movement to the right which occurs while the camera man attempted to track the ball. Such movement is essentially noise and must be eliminated to be able to determine the real camera movement. The system analyses the entire sequence and determines the *dominant movement* of the camera (for example whether the overall movement was to the left or to the right) by computing the frequency of the *negative* and *positive* values for the camera parameters. The *most frequent sign determines the dominant motion* for that category of camera parameters. For example if the values for the pan parameter are mostly positive then the dominant motion is to the right otherwise the movement is to the left. The symbolic values classifying the dominant motion for all camera parameters are shown in Table 8.1 — the actual pan/tilt/roll value is not considered at this stage when converting it to symbolic form — just the sign. The third stage involves a more refined classification of the camera motion.

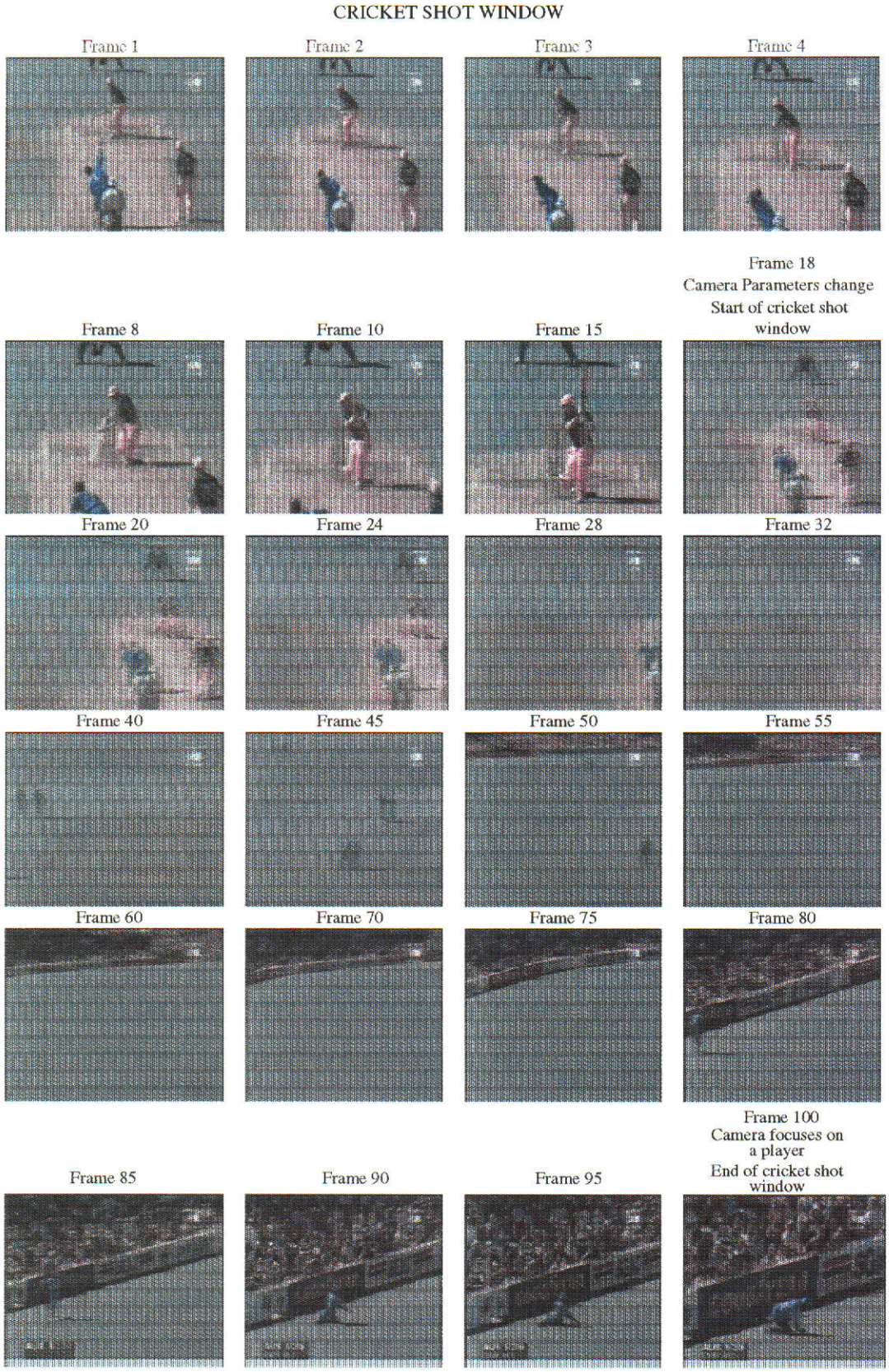


Figure 8.4: The cricket shot window.

Camera Parameter	Symbolic Value	Value Sign
Pan	Right	Positive
Pan	Left	Negative
Tilt	Up	Positive
Tilt	Down	Negative
Roll	Clockwise	Positive
Roll	Anticlockwise	Negative
Zoom	Zoom-In	Positive
Zoom	Zoom-Out	Negative

Table 8.1: Symbolic values classifying the dominant camera motion.

Once the *dominant motion* has been identified much of the noise is removed. This is done by eliminating all the camera parameter values that have a different sign to the one indicated by the dominant motion. For example, if the dominant pan movement is determined to be to the right, then all negative pan values are removed from the data. The percentage of values removed varies from 2% to 10% — it generally depends on the length of the cricket shot and on the ability of the camera man (how smooth he/she is able to track the ball). Then for each camera parameter in turn, the system collects all the values from the sequence and computes an average. This average value indicates how far and how fast the camera moved during the cricket shot. The average value is computed as follows: a cumulative histogram is computed and we determine the bins that contribute 70% of the data. The average value is then computed using these bins. The histogram generated from a processed sequence with a majority of values close to zero is shown in Figure 8.5. Figure 8.6 is shows the histogram generated from a sequence consisting in majority of high negative values (to get the threshold we use the bins that contain 70% of the data — the number of bins considered varies depending on the spread of the data, for example in Figure 8.5 only 4 bins are considered, while in Figure 8.6 5 bins are considered). Table 8.2 shows the symbolic values used to describe the average camera parameter values. The average value is converted into a symbolic value by thresholding. The threshold values are fixed at plus or minus 0.8 with values smaller than -0.8 or greater than 0.8 indicating significant movement.

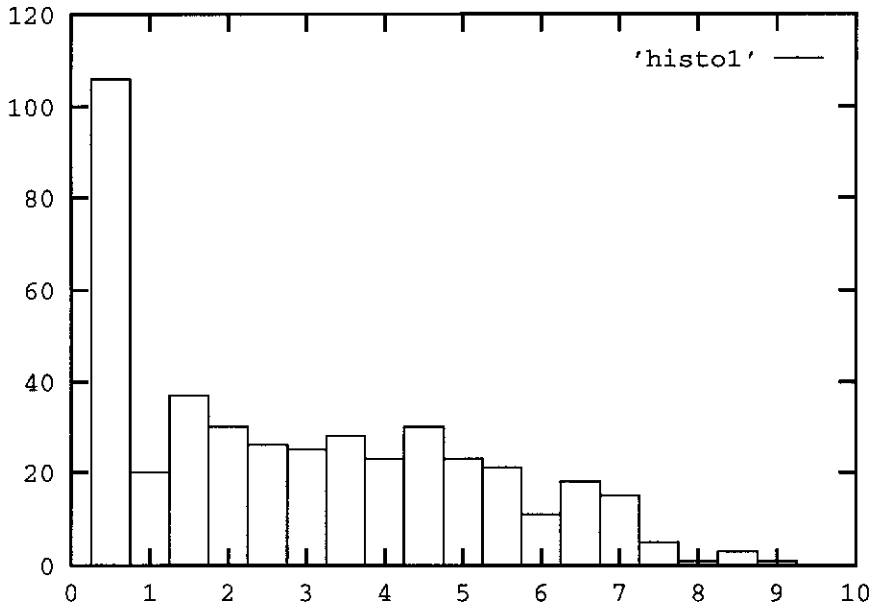


Figure 8.5: Histogram generated for the pan values of sequence where there was little pan movement – the majority of values are close to zero.

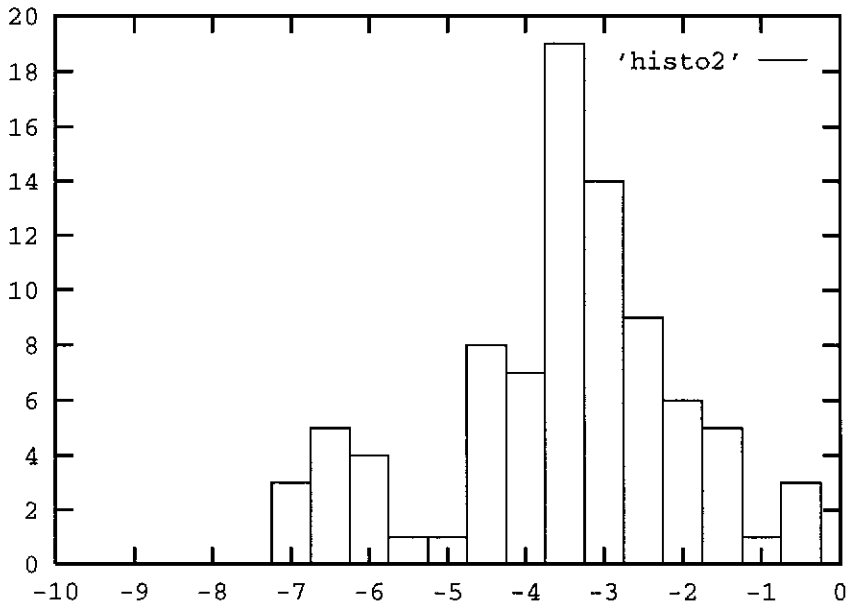


Figure 8.6: Histogram generated for the pan values of sequence where there was significant pan movement to the left – the sequence is made up of mostly high negative values.

Camera Parameter	Symbolic Value	Average Value
Pan	Right	High Positive Values
Pan	Left	High Negative Values
Pan	Middle	Close To Zero
Tilt	Up	High Positive Values
Tilt	Down	High Negative Values
Tilt	Centered	Close To Zero
Roll	Clockwise	High Positive Values
Roll	Anticlockwise	High Negative Values
Roll	Static	Close To Zero
Zoom	Zoom-In	High Positive Values
Zoom	Zoom-Out	High Negative Values
Zoom	Steady	Close To Zero

Table 8.2: Symbolic values for the average camera motion parameter value.

Each shot can, therefore, be expressed as a sequence of symbolic values of dominant and average motion. In addition to the 8 attributes (dominant and average motion) for each of pan, tilt, roll and zoom, the cricket shot length is needed. The length of the shot is simply derived from the duration of the video shot sequence (the number of frames in a shot).

Figure 8.7 shows types of cricket shot which the system is attempting to identify using these motions and shot length. For example consider the shots *long straight drive* (on the left side) and *right pull*. One possible set of symbolic values for the camera parameters is shown in Tables 8.3 and 8.4. The values shown in the tables were obtained after the system was trained with 63 cricket shots. However the

Camera Parameter	Dominant Motion	Average Value
Pan	Left	Close To Zero
Tilt	Down	High Negative Value
Roll	Clockwise	High Positive Value
Zoom	Zoom-Out	High Negative Value

Table 8.3: Symbolic values classifying the dominant camera motion and average motion for a long straight drive (on the left side).

Camera Parameter	Dominant Motion	Average Value
Pan	Right	High Positive Values
Tilt	Up	High Positive Value
Roll	Clockwise	High Positive Value
Zoom	Zoom-Out	High Negative Value

Table 8.4: Symbolic values classifying the dominant camera motion and average motion for a right pull shot.

values of two of the parameters with which a shot is classified, namely dominant motion and average camera motion, vary from game to game. This is due to the following factors:

- Camera position varies on different cricket grounds mainly in the angle at which the shot is captured (right behind the wicket or at a slight angle) and the perspective from which the shot is taken (low down or high up in the stands).
- Ground shape and size also varies on different cricket grounds (for example the Brisbane Gabba Cricket Ground is smaller than the Melbourne Cricket Ground and a shot such as a drive involves slightly different camera motion — different pan and zoom values).

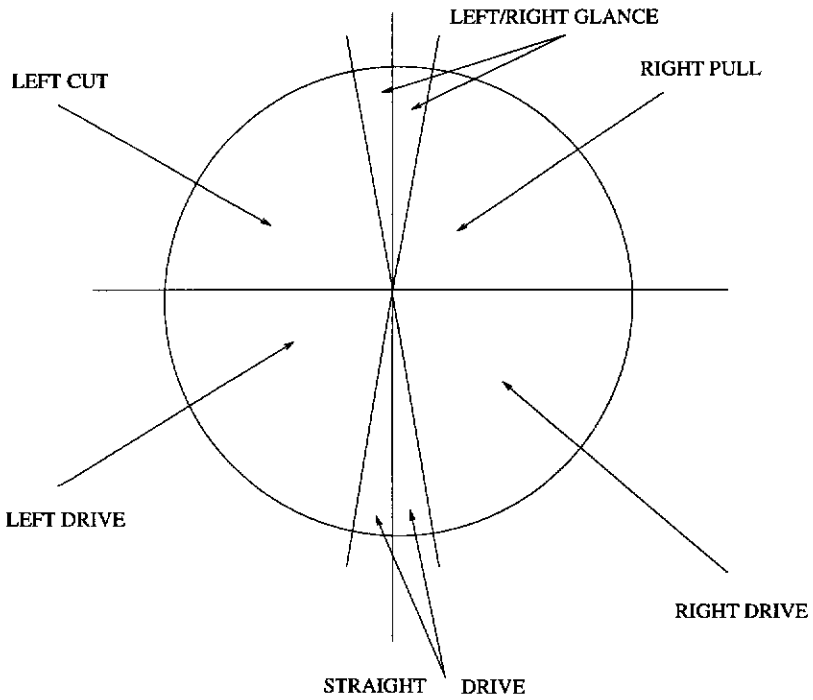


Figure 8.7: The six cricket shots the system attempts to identify: the drive shot (left, right or straight), the pull shot (left or right) and the hook shot.

The goal of our work is to build general cricket shot descriptions to enable the system to classify shots from different cricket grounds and therefore any ground specific information needs to be removed. To deal with variations in the parameters we use incremental learning. We build representations of the shots using symbolic data and update these representations when necessary.

8.4 Classifying and Learning Cricket Shots

Training data is extracted from a video sequence, such that several shots are extracted and their descriptions are derived in terms of the symbolic attributes described in the section 8.3. Classifying and learning uses the incremental learning algorithm ILF described in chapter 3.

The way in which the system learns from the incoming cricket shot descriptions is as follows. The symbolic descriptions generated by the data analysis module are passed on to the incremental learning module which first attempts to find a match for the shot in the existing hierarchy of shots. Each new shot description is compared with the current description in the hierarchy to determine if there is enough evidence to justify the update of the current description. Each description in the hierarchy produces an evidence score which determines whether the shot

does or does not match the current description. The general format of the cricket shot type description is shown in Figure 8.8. This shows that there are n shots, with shot 3, for example, having m generalised examples in its description. A description of a cricket shot has 9 attributes. Each attribute in the description can have a set of values. Each one of the values has an age value associated with it (see ILF concept structure description in chapter 3). The set of values possible for the dominant motion attributes is defined by set $A = \{\text{Positive, Negative}\}$. The set of values for the average camera motion is defined by set $B = \{\text{High Positive, High Negative, Close to Zero}\}$. In this way multiple descriptions (i.e. more than one type of shot) can be updated (provided enough evidence was found) by the same shot. While this procedure results in the system updating descriptions which should not be updated when one considers the overall set of shots, results show that over time the unnecessary modifications are "aged out" of the descriptions. That is unless a particular shot description gets reinforced by other similar ones, it is forgotten. The main reason for choosing to update multiple descriptions is that it is a simple way of representing a multiple match between the existing description and the input cricket shot which is more appropriate than an absolute match (as we mentioned above, the camera parameters vary from game to game). In essence, we keep as much information as possible to catch infrequent shots. The update process is based on data ageing (see chapter 3 for a detailed description of how ILF uses ageing in the update process). The algorithm uses ageing at two levels: the *data level* and the *description level.*

When a new cricket shot is processed there are three possible outcomes. The first is that the system finds a match for it in the existing hierarchy of shot descriptions. The description that matches the input cricket shot is updated to be consistent with the new data. The second outcome is that the system does not find a match for the new cricket shot so a new description is added to the hierarchy while updating the existing shot descriptions. The third outcome from processing a new cricket shot is that one (or possibly more) descriptions in the hierarchy get removed since the data is "aged out".

8.5 Results

We have trained the system on 76 cricket shots and tested it on 82 shots. The video segments used in our work had a length varying from 24 to 190 frames. The shots have been collected from five cricket games and cover six types of shots. Two of the shots occur rarely (drive and glance) and as a result very few instances were present in either the training or the test sets. The make-up of the training data is shown in Table 8.5 and the classification tree generated for the data consisted of four cricket shot descriptions (all training data was correctly classified). The test data used is shown in Table 8.6. The results of

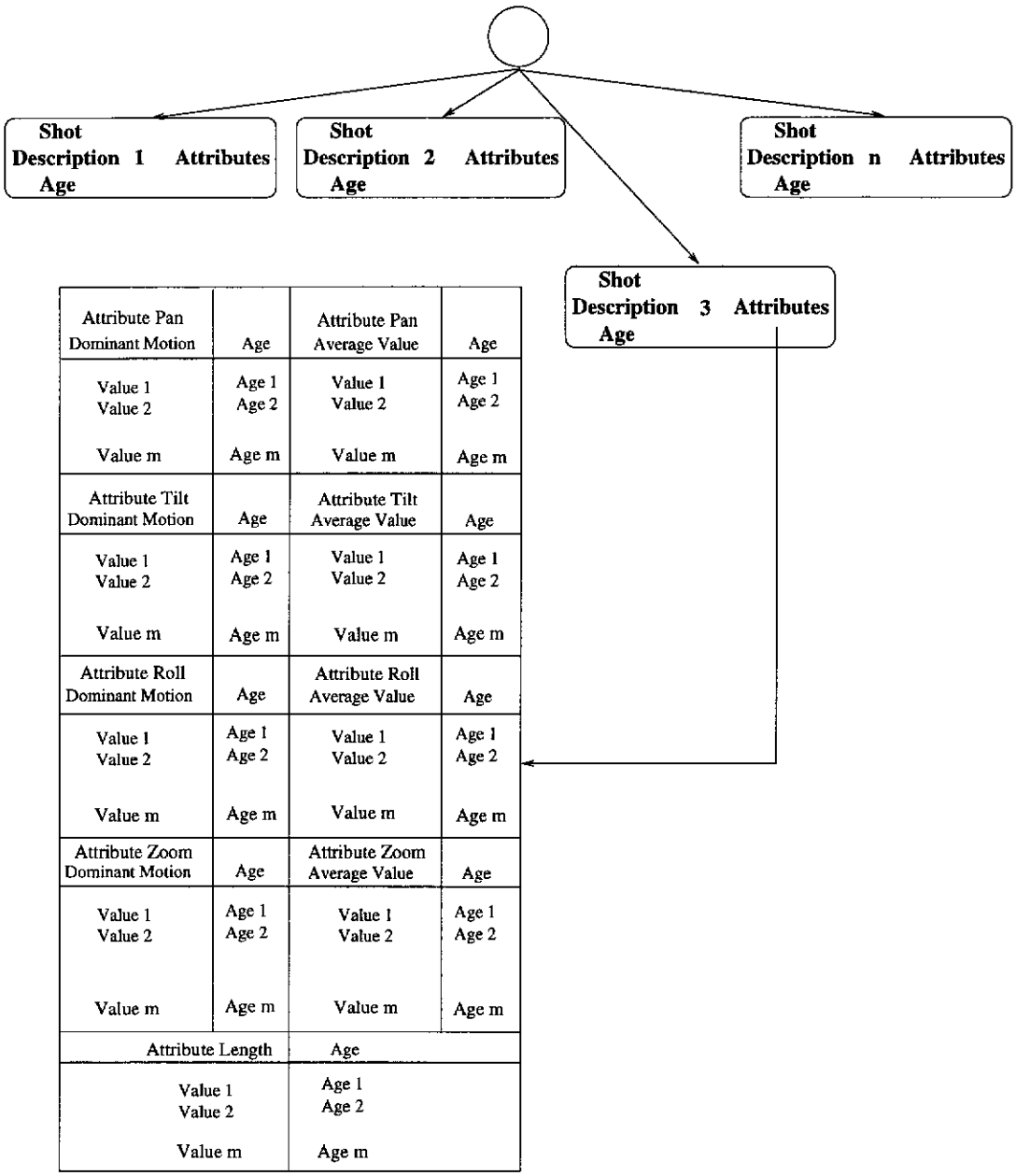


Figure 8.8: A typical description hierarchy used by the incremental learning algorithm. Each shot description has an age and 9 attributes and each one of the attributes has a set of data values+age values associated with it.

Number of Shots	Side	Shot Type
25	Right	Drive
15	Left	Drive
18	Right	Pull
14	Left	Pull

Table 8.5: The training data.

Number of Shots	Side	Shot Type
24	Right	Drive
20	Left	Drive
20	Right	Pull
18	Left	Pull

Table 8.6: The test data.

the classifications are shown in Table 8.7. The system performed very well when

Side	Shot Type	Correct	Incorrect
Right	Drive	19	5
Left	Drive	18	2
Right	Pull	15	5
Left	Pull	12	6

Table 8.7: The results on the test data.

classifying four types of shots: the pull shot (left and right) and drive shot (left and right). The general description generated by the system for the right pull shot is: (Left, Left) - (Down, Up) - (Clockwise, Clockwise) - (Zoom-Out, Zoom-Out) - (Short). An example of a right pull shot is shown in Figure 8.9. For the right drive shot, the general description generated is: (Left, Right) - (Up, Up) - (Clockwise, Clockwise) - (Zoom-Out, Zoom-Out) - (Short). A right drive shot is shown in Figure 8.10. In the case of the left pull shot, the description generated is: (Right, Right) - (Down, Up) - (Clockwise, Clockwise) - (Zoom-In, Zoom-In) - (Long). A typical right pull shot is shown in Figure 8.11. Finally, the description generated for a left drive shot is:(Left, Right) - (Up, Up) - (Clockwise, Clockwise) - (Zoom-Out, Zoom-In) - (Short). An example of a left drive shot is shown in Figure 8.12.

The overall success rate averaged at 77%. The results also show that the system was not able to build an accurate description for the straight drive shot and hence it was unable to accurately classify it in the test data. The system has simply been unable to identify any significant differences between the camera parameters for a straight drive and a drive on each side of the ground (the pan values do not vary as much as expected).

8.6 Summary

This chapter has explored the application of the ILF algorithm to the problem of recognising cricket shots from video data. The methods developed for American

Figure 8.9: Example of a right pull shot. The symbolic description is: (Left,Left) - (Down,Up) - (Clockwise,Clockwise) - (Zoom-Out,Zoom-Out) - (Short).

Football were not suitable for this application so other techniques have been developed although they still use symbolic attributes. As for the American Football, the start and finish of each shot or play is well defined.

In the following chapter we investigate how incremental learning can be combined with incremental recognition to solve classification tasks where the two main assumptions used in learning for American Football and cricket are no longer applicable. The first is that the system always *knows* that the starting point in the training data is always the same as the one in the classes/concepts stored in the knowledge base and therefore the system is able to attempt a comparison straight away. The second is that all the information on the example/query is known before any classification is attempted.

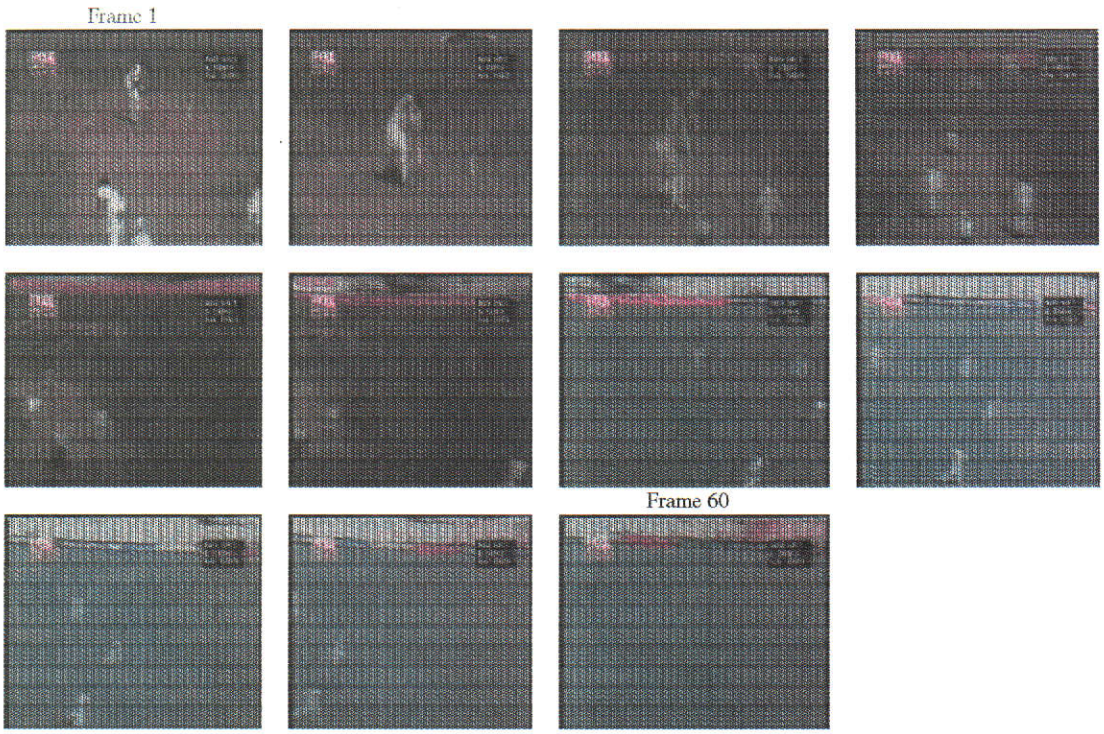


Figure 8.10: Example of a right drive shot. The symbolic description is: (Left,Right) - (Up,Up) - (Clockwise,Clockwise) - (Zoom-Out,Zoom-Out) - (Short).

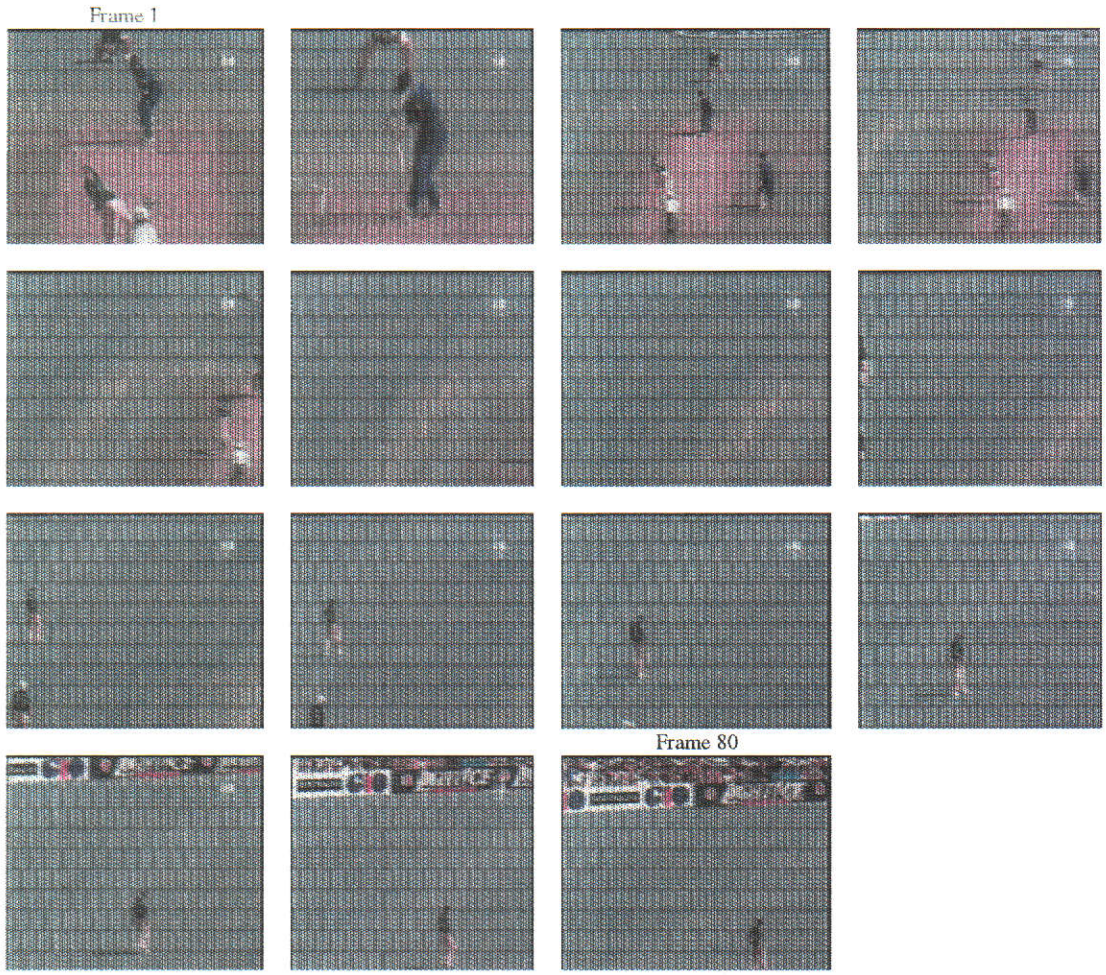


Figure 8.11: Example of left pull shot. The symbolic description is: (Right,Right) - (Down,Up) - (Clockwise,Clockwise) - (Zoom-In,Zoom-In) - (Long).

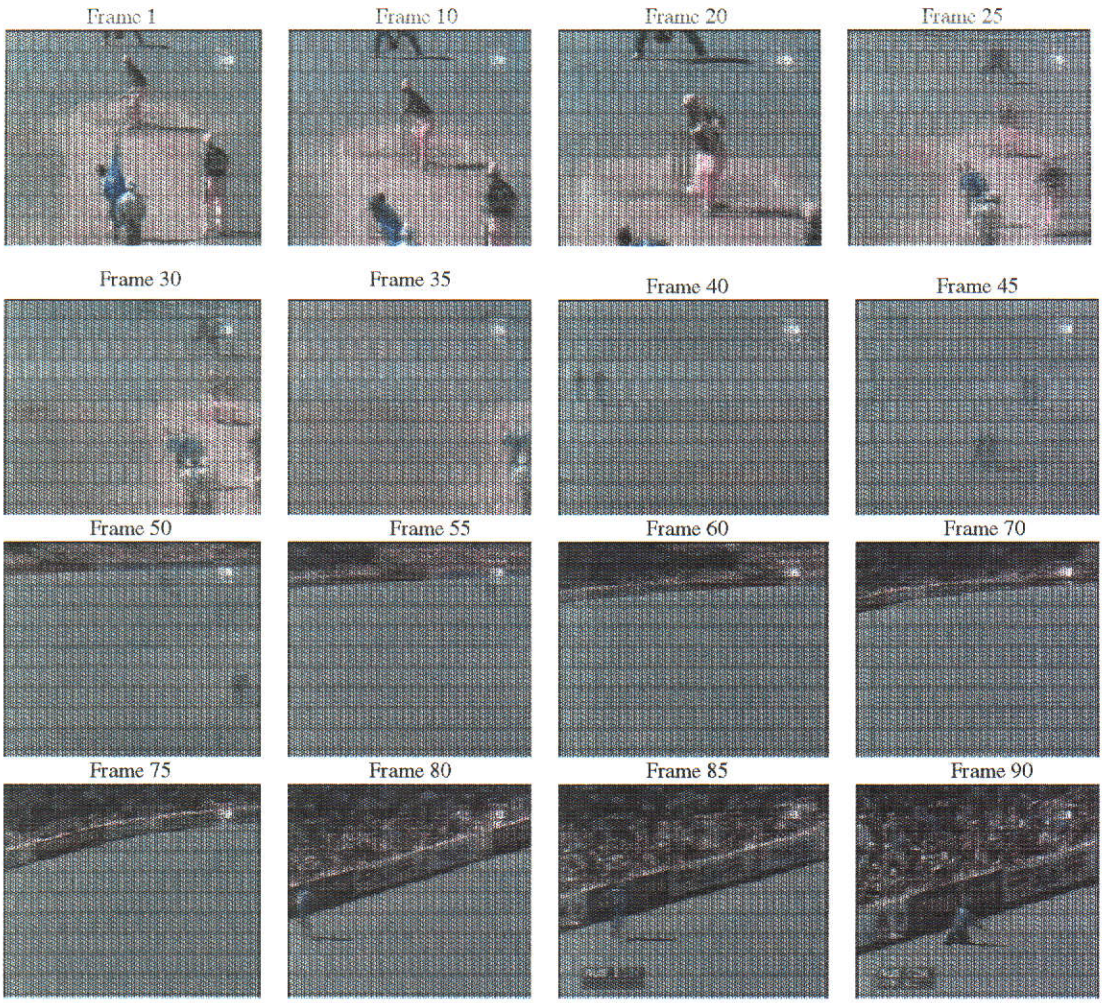


Figure 8.12: Example of the left drive shot. The symbolic description is: (Right,Right) - (Up,Up) - (Clockwise,Clockwise) - (Zoom-Out,Zoom-In) - (Short).

Chapter 9

Incremental Recognition and Learning

9.1 Introduction

The last issue we explore is the combination of incremental recognition and incremental learning. In this chapter we describe an extended version of the ILF algorithm which has been modified to do incremental recognition of queries.

Incremental recognition is a progressive classification (possibly involving backtracking) which considers the data at each time instance in the query and thus provides a probable answer before all the query information becomes available. ILF has been also augmented to enable learning in either a stepwise fashion (the system learns after each time instance) or an overall best match fashion (the system only learns when all time instances have been acquired).

Two main assumptions are generally made for both incremental learning and non-incremental learning.

The first assumption is that the system always *knows* that the starting point in the training data is always the same as the one in the classes/concepts stored in the knowledge base and therefore the system is able to attempt a correspondence and comparison straight away. This was the case for American Football and cricket. For example in American Football, the play has a defined start (the players move) and finish (the camera zooms on a player or advertisments are shown).

The second assumption is that all the information about the example/query is known before any classification is attempted. That is, all feature values have

been acquired. This is usual for situations which have definite identifiable begin and end points so the features are determined once the end point is reached.

Learning is generally concerned with the recognition of independent patterns acquired at a particular time. *There are however situations where sequences of patterns need to be recognised i.e. for each time step we have a multidimensional pattern. Furthermore it is possible that the start of the sequence is unknown.*

Consider the following example. A radar station monitoring the movement of hostile aircraft detects that some of the planes seem to be moving towards an important strategic installation. To formulate the appropriate response action, the radar controller needs to know what is the actual intention of the enemy (it is possible that the enemy might simply attempt to draw valuable resources from another area). In this situation, analysing previous enemy attacks is useful but the traditional classification method no longer applies — the response has to be formulated to counter the enemy attack as soon as possible (before the enemy has finished carrying out its attack).

Incremental recognition and learning of situations such as the one described above is desirable as it would allow a better and more efficient response to be formulated. But there are two problems that have to be addressed. The first problem is that one cannot precisely define the starting point of the scenario (the starting point of the sequence) while the second problem is that the scenario is dynamic and one has to reach a solution before all data is known (before the enemy bombs the target). It is these problems that are now addressed using ILF.

The layout of this chapter is as follows: Section 2 covers the concept of incremental recognition of multidimensional patterns whilst, in Section 3 we describe two incremental learning methods that use incremental recognition. In Section 4 we compare the two learning methods and show their advantages and disadvantages. Section 5 covers results of classification of fighter combat manoeuvers using incremental learning and incremental recognition.

9.2 ILF and Incremental Recognition

ILF was augmented to enable it to carry out the incremental recognition and classification of queries. The main differences between the original classification and the augmented classification and recognition process used by ILF are that the augmented recognition processing allows it to search for a starting point for a sequence of patterns in the models in the hierarchy, and that it can backtrack to search for a better solution as shown in the following example.

9.2.1 Example of Incremental Recognition

The general format of an example E_i for i = *1,2,3,..,m* models used in our research is an ordered collection of data over n time instances T_i where i = *1,2,3,...,n* where each time instance T_i is a conjunction of feature-value pairs $(F_j V_k)$ where j = *1,2,3,...,p* and k = *1,2,3,...,s* as shown below:

$E_i = T_1(F_1, V_1),(F_2, V_2),...,(F_p, V_s),$

$T_2(F_1, V_1),(F_2, V_2),...,(F_p, V_s),...,$

$T_n(F_1, V_1),(F_2, V_2),...,(F_p, V_s).$

The models and the query contain the spatial relationships between the objects in the scene as well as their temporal ordering. The spatio-temporal relationships are derived as in the case of the American Football application.

The processing of a query involves a labelling stage as the system attempts to address the correspondence problem of the objects in the model and query. It is possible to have a large number of hypotheses for a single model (see Figure 9.1). *A hypothesis is one of the valid mappings of the objects in the query onto the objects in the model.* The example shown in Figure 9.1 shows a model that contains 7 objects organised in two separate "four-finger" formations. The query contains only 3 *unlabelled* objects but their arrangement is also in a four-finger formation. When considering the spatial relationships between the objects, several hypotheses or mappings can be derived to label the objects in the query with the best two hypotheses being [(a,1), (b,2) (c,3)] and [(a,5), (b,6), (c,7)].

Now consider the case in which the system's conceptual hierarchy is made up of three models with each model consisting of four objects and five time instances as shown in Figure 9.2. The query consists of three objects and four time instances (Figure 9.3). The processing is done as follows. The system first attempts to determine what is the equivalent instance in the model for the first instance in the query starting with the first instance in Model 1.

Less than 75% of the relationships between the objects in the model match the relationships in the query so Model 1 is discarded. The system proceeds then to analyse the first instance of Model 2. The two instances do not match and Model 2 is also discarded. Similarly the first instance of Model 3 fails to match the first instance in the query, so the system starts analysing the second instance in the models.

The second instance of Model 1 matches the first instance in the query (100% similarity) so the system has determined that the starting point is instance 2 in the models. Model 1 is *marked* as a match for the query and the score generated

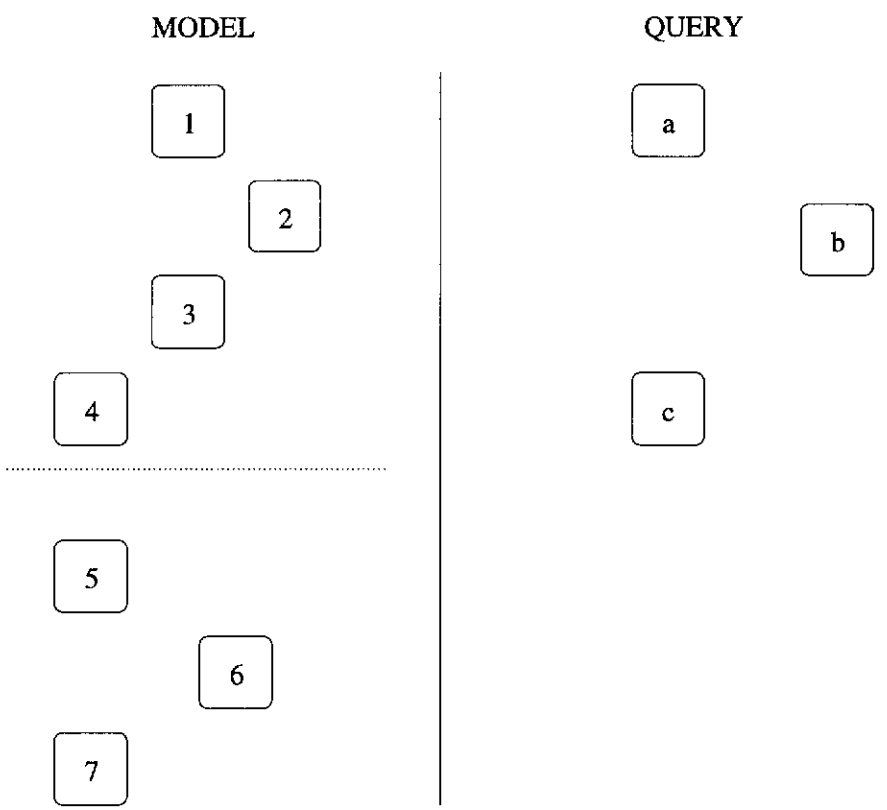


Figure 9.1: Several hypotheses can be generated for the 3 planes in the query when their spatial relationships are compared to the planes in the model.

for the second instance is recorded.

The first instance in the query also matches the second instance in Model 2 but does not match Model 3. Model 2 is also *marked* as a match and the associated score is recorded.

The second instance in the query becomes available and is shown in Figure 9.3. The system attempts to match the second instance in the query by comparing it against the third instance in the *marked* models (Model 1 and 2). Both models match the query so their scores are updated. When the second instance in the query has been considered, the probable solutions are still models 1 and 2. When information about the third instance in the query is available, the system attempts to match it against the fourth instance in the marked models. Model 1 matches while Model 2 does not. Therefore the system *changes* the *mark* on Model 2 to *not matched* and only the score of Model 1 is updated. After the third instance in the query has been processed, there is only one possible solution — Model 1. The fourth instance in the query is matched by the system against the fifth instance in Model 1 but no match is found. The system *changes* the match mark on Model 1 to *not matched* and backtracks to the fourth instance in the models (third instance in the query). The system attempts to build a new list of candidate

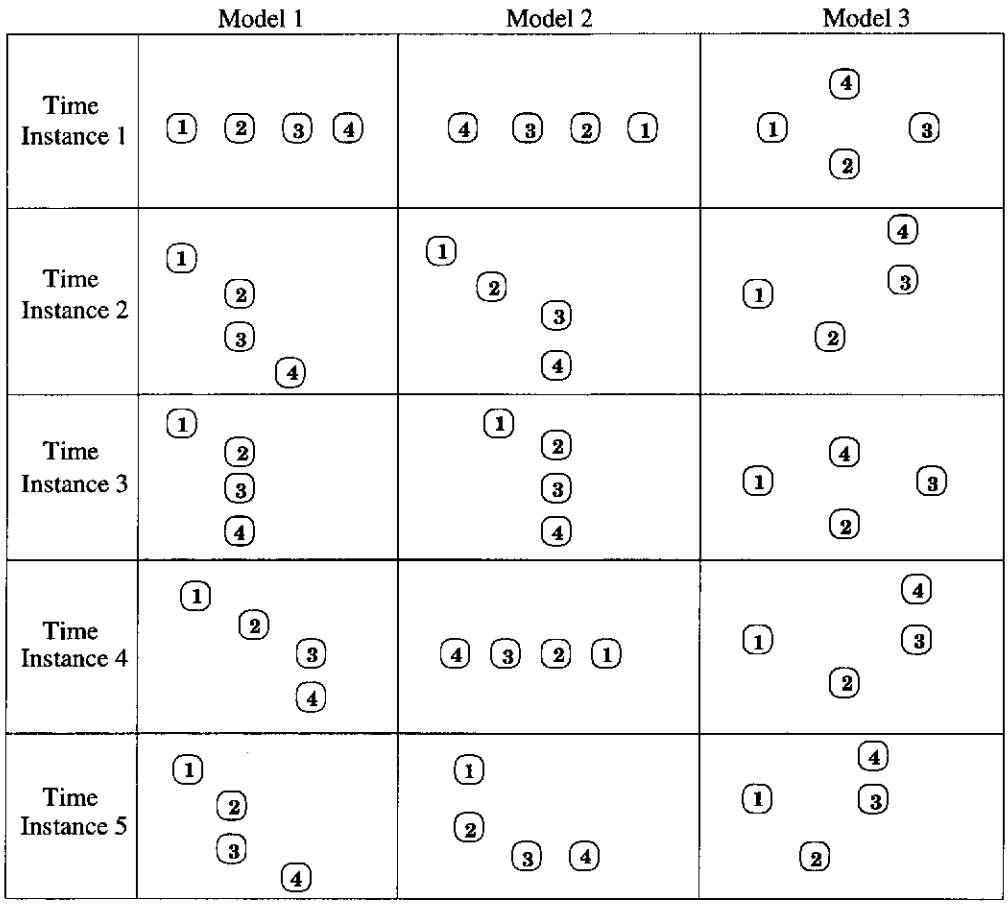


Figure 9.2: The models stored in the concept hierarchy.

models by analysing all the models that are not marked. Both Model 1 and 2 are marked, so Model 3 is the only potential candidate. The fourth instance in Model 3 matches the third instance in the query, so the model is marked as a match and the associated similarity score is recorded. The last instance in the query also matches the fifth instance of Model 3. Therefore at instance 4 in the query the most likely solution is Model 3.

As demonstrated with this example, by using the information available at each step, the system generates a list of solutions which changes according to the new data. This offers the advantages that a solution list is available at all the steps in the query and the system can backtrack each time the current solutions are determined to be inaccurate. The disadvantage of this method is that many solutions which appear to be accurate for several steps can lead to dead ends.

This configuration gives the most consistent match for each of the time instances in the query and not the overall match for the whole query. This allows queries to change halfway through, i.e. Models 1 and 3 are the best partial matches. If the best overall match is important then in this example, Model 1 is the best

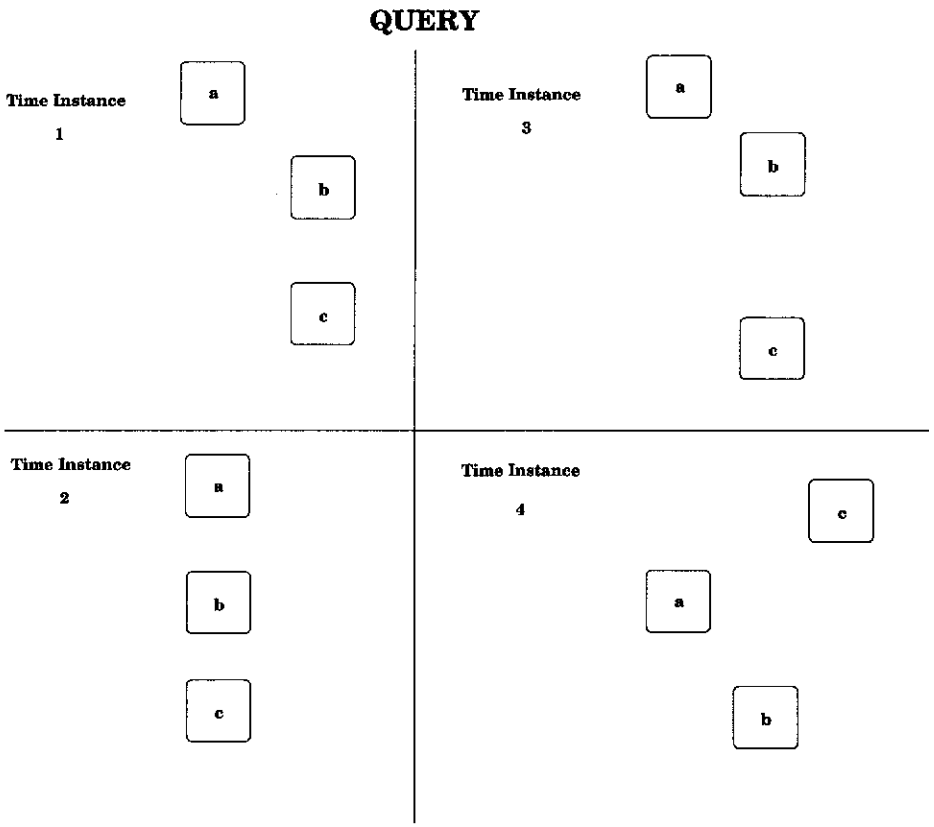


Figure 9.3: The time instances in the query.

match.

9.3 Incremental Learning using Incremental Recognition

Incremental learning using ILF can be used to update the conceptual hierarchy using two methods. The first modification enables it to carry out the incremental recognition/classification of queries while the second modification enables it to learn using two methods:

- Best Overall Model — the system only learns is when all time instances are analysed — the algorithm is shown in Figures 9.4, 9.5).
- Time Instance Based Method — the system learns after each time instance — the algorithm is shown in Figure 9.6.

```
Let example E and be an ordered collection of time instances T1,T2,T3,..,.Tn where each time
instance is a conjunction of features F1,F2,F3,...,Fp.
Let Model Mi and be an ordered collection of time instances T1,T2,T3,..,.Tn where each time
instance is a conjunction of features F1,F2,F3,...,Fp.

Best-Model = -1; Best-Score = -1;
Start-Instance = -1;

STAGE A.
-------------------------------------------------------------------------------------------

for Model Mi, i = 0 to n do
 for Time Instance Ij, j = 1 to n do
   Similarity-Score = 0;
   compare Model Mi, Instance Ij with Example, Instance I1 and compute Similarity-Score
   if Similarity-Score >= Similarity-Threshold then
     Start-Instance = j;
     Exit;
   end-if
 end-for
end-for

-------------------------------------------------------------------------------------------
STAGE B.
-------------------------------------------------------------------------------------------

if Start-Instance != -1 then
  Backtrack-Flag = 0;
  while Time Instance, Ij < n do
   Instance-Similarity-Score = 0;
   for Model Mi, i = 1 to n do
     if i > 1 then
       if Model Mi is marked and Backtrack-Flag == 0 then
         compare Model Mi, Instance I(j+Start-Point) with Example, Instance Ij and
         compute Similarity-Score
         if Similarity-Score >= Similarity-Threshold then
            Model Mi(Score) = Similarity-Score + Model Mi(Score);
            Model Mi is marked;
           if (Instance-Similarity-Score == 0) then
            Instance-Similarity-Score = 1;
           end-if
          end-if
       end-if
     else
       compare Model Mi, Instance I(j+Start-Point) with Example, Instance Ij and
       compute Similarity-Score
        if Similarity-Score >= Similarity-Threshold then
           Model Mi(Score) = Similarity-Score + Model Mi(Score);
           Model Mi is marked;
          if (Instance-Similarity-Score == 0) then
            Instance-Similarity-Score = 1;
          end-if
        end-if
      end-if
   end-for
   if Best-Score < Model Mi(Score) then
    Best-Score = Model Mi(Score);
    Best-Model = i;
   end-if
   if (Instance-Similarity-Score != 0)
     j = j + 1; Backtrack-Flag = 0;
   else
     j = j; Backtrack-Flag = 1;
   end-if
  end-while
end-if
-------------------------------------------------------------------------------------------
```

Figure 9.4: Best Overall Model — Algorithm.

```
STAGE C.

-------------------------------------------------------------------------------------

if Best-Model != -1 then
  for Time Instance Ij, j = 1 to n do
    if Similarity-Score > Similarity-Threshold then
      update Model M(Best-Model), Time Instance I(j+Start-Point)
    else
      add new condition to Model M(Best-Model), Time Instance I(j+Start-Point)
    end-if
  end-for
end-if

-------------------------------------------------------------------------------------
```

Figure 9.5: Best Overall Model — Algorithm (continued).

The first method is *similar* to traditional incremental learning where all the modifications (if any are necessary) are done after the entire query has been processed. This method involves a search for the models which best match the query over its entire set of instances. The algorithm (shown in Figures 9.4 and 9.5) has three stages.

In stage A, the system attempts to determine which time instance in the query to use in the matching process. It starts by comparing the first instance in the query with the first instance in the models. If this fails, it tries the next time instance in the query. The process is repeated until there are no more instances left in the query or a match is found.

In stage B, the system attempts to identify which candidate model best matches the query. The system builds a candidate list that is dynamically updated based on the evidence from the matching process. If all solutions lead to dead ends, the system backtracks one time instance and attempts to generate a new list of candidates. The process continues until all the query instances are processed or until the candidate list is empty.

In stage C, the system updates the model which best matches the query. The update process is done if the Similarity-Threshold condition is satisfied.

Consider the following example. Let us say the system is processing a query which consists of 30 time instances and that it generates 6 solutions for the first instance in the query. After 10 instances, the system determines that all the original solutions are leading to dead ends, so backtracks one instance and generates 4 new solutions. The last 20 instances of the query are processed and the system determines that 2 of the 4 solutions match the query. Therefore the best solutions are not the ones generated at instance 1 (6 solutions) but the ones generated at instance 9 (2 out of 4 solutions) — the ones which match the latter

```
Let example E and be an ordered collection of time instances T1,T2,T3,..,.Tn where each time
instance is a conjunction of features F1,F2,F3,...,Fp.
Let Model Mi and be an ordered collection of time instances T1,T2,T3,..,.Tn where each time
instance is a conjunction of features F1,F2,F3,...,Fp.

Start-Instance = -1;

STAGE A.
-------------------------------------------------------------------------------------------

for Model Mi, i = 0 to n do
 for Time Instance Ij, j = 1 to n do
   Similarity-Score = 0;
   compare Model Mi, Instance Ij with Example, Instance I1 and compute Similarity-Score
   if Similarity-Score >= Similarity-Threshold then
     Start-Instance = j;
     Exit;
   end-if
 end-for
end-for

-------------------------------------------------------------------------------------------
STAGE B.
-------------------------------------------------------------------------------------------

if Start-Instance != -1 then
  Backtrack-Flag = 0;
  while Time Instance, Ij < n do
   Instance-Similarity-Score = 0;
   for Model Mi, i = 1 to n do
     if i > 1 then
       if Model Mi is marked and Backtrack-Flag == 0 then
         compare Model Mi, Instance I(j+Start-Point) with Example, Instance Ij and
         compute Similarity-Score
         if Similarity-Score >= Similarity-Threshold then
            Model Mi(Score) = Similarity-Score + Model Mi(Score);
            Model Mi is marked;
            update Model Mi, Time Instance I(j+Start-Point);
           if (Instance-Similarity-Score == 0) then
            Instance-Similarity-Score = 1;
           end-if
          end-if
       end-if
     else
       compare Model Mi, Instance I(j+Start-Point) with Example, Instance Ij and
       compute Similarity-Score
        if Similarity-Score >= Similarity-Threshold then
           Model Mi(Score) = Similarity-Score + Model Mi(Score);
           Model Mi is marked;
           update Model Mi, Time Instance I(j+Start-Point);
          if (Instance-Similarity-Score == 0) then
            Instance-Similarity-Score = 1;
          end-if
        end-if
      end-if
   end-for
   if (Instance-Similarity-Score != 0)
     j = j + 1; Backtrack-Flag = 0;
   else
     j = j; Backtrack-Flag = 1;
   end-if
  end-while
end-if

-------------------------------------------------------------------------------------------
```

Figure 9.6: Best Time Instance — Algorithm.

part of the query. Therefore the main difference between the traditional and the new learning method is that the best solutions found match to only the latter part of the query and backtracking is necessary.

The second method involves updating the conceptual hierarchy after each instance in the query has been processed. The assumption in this case is that once the *current set of solutions is reinforced at the current instance* the system has found enough evidence to justify the update of the model's information for the previous time instance.

The algorithm is shown in Figure 9.6. In stage A, the system searches for the time instance in the query that can be used in the matching process. It starts by comparing the first instance in the query with the first instance in the models. If this fails, it tries the next time instance in the query. The process is repeated until there are no more time instances left in the query or a match is found.

In stage B, the system processes the query and model time instances by time instance. At each step it checks to see which models match the query at the current time instance. Any model matching the query is marked. All marked models that match the query at the current time instance as well as the previous time instance, and satisfy the Similarity-Threshold condition, are updated. In case the system does not find any matches in the candidate list at a given time, it backtracks to the previous time and generates a new list of candidates. The matching process continues until all time instances in the query are processed or no candidates exist.

Consider the following example. The system is analysing the first instance in the query and builds a list of solutions which contains 3 models, say Model 1, 2 and 3. The system is then processing the second instance in the query and checks the models in the solution list. After matching the query against Model 1, 2 and 3, the system determines that Model 1 and 3 match the query, but that Model 2 does not. In this case the assumption is that since the system has confirmed that Model 1 and 3 are *still* possible solutions, then there is enough evidence to backtrack to instance 1 in the two models and update them (if necessary).

9.4 Discussion

These learning methods have their own sets of advantages and disadvantages. The first method has the advantage that it involves the update of the conceptual hierarchy only after all the information on the query becomes available and when the best solution has been found. Another advantage is the number of models updated is kept to a minimum so the conceptual hierarchy remains compact. The dilemma in this method is how to update the concepts. The system always selects

the best candidates but, as it is shown above, it is possible that the best solution only matches part of the query so should the entire model be updated or only the parts that match the query? If only parts of the model are updated then potentially useful information could be lost. If the entire model is updated (all instances including the ones which do not match) then it is likely that the model will also contain irrelevant or out of date information. ILF implements the latter type of updating and the reasoning behind this choice is twofold. It is the safest method (no potentially useful information is lost) and with an effective forgetting mechanism the incorrect/noisy data can be "aged out" from the models.

The second method offers the advantage that the update operations are done only when there is evidence to do so and therefore the amount of incorrect or useless data added is kept to a minimum. The update operation is also more dynamic and reflects better the incoming stream of information from the query. The disadvantage of this method of learning is that the number of models modified is potentially very large and therefore the amount of memory required to store the conceptual hierarchy is increased significantly.

To demonstrate the differences between the methods consider the following example. Let us assume that the concept hierarchy consists of the 3 models shown in Figure 9.2 and that the query consists of the time instances as shown in Figure 9.3. The threshold value for a match between the query and the model is set at 67% (two out of three features must match). The start age for all *new* time instance representations is set to -190 while the age for the original representations is set to 0. The lowest age a representation can reach before it is removed is -200.

To show the effects on the models and the time instance representations, we will assume that the same query is processed by the system 5 times. If the processing is done using the first method when the system analyses the query, it selects Model 1 as the best *overall match*. The system then modifies the representations for instances 2, 3 and 4 in Model 1. Apart from the original condition, the system adds the new hypotheses generated for each time instance. Notice that time instance 5 in Model 1 has a new representation. The system could not generate a hypothesis for time instance 5 in the Model and since it determined that the Model and the query are similar, it added time instance 4 in the query to time instance 5 in Model 1. The new updated model is shown in Figure 9.7. The modifications made contain both correct data — such as Hypothesis 1 for instance 3 — and incorrect data — such as Hypothesis 4 for time instance number 2. The age associated with each representation is shown in Table 9.1.

The second time the query is processed, the system again selects Model 1 as the best *overall match* and attempts to make the necessary modifications. The model now contains all the possible hypotheses for the query, so the process is now to determine which representation best matches the query. For example the query matches 100% of the original representation with Hypothesis 2 but does not match

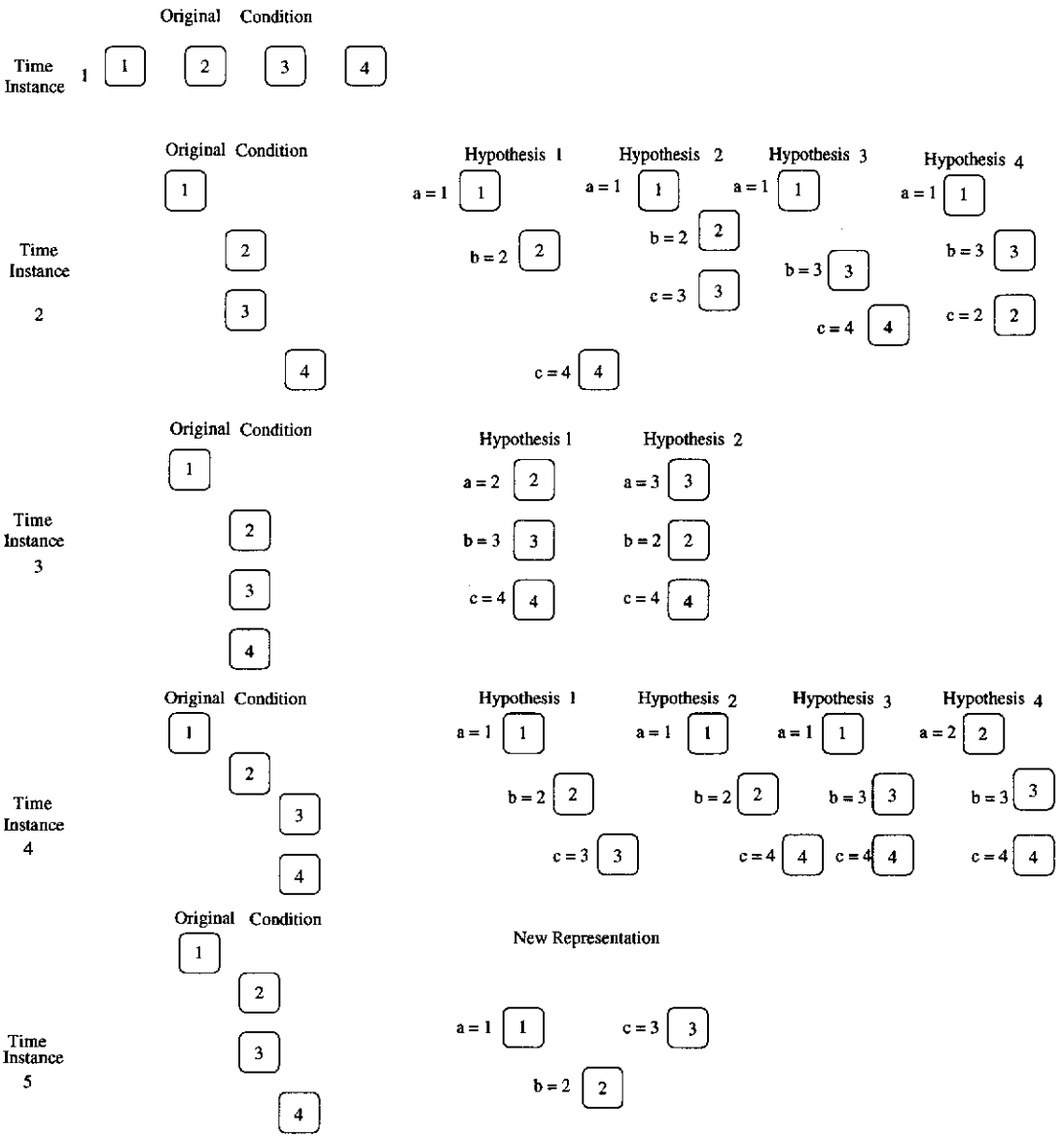


Figure 9.7: Model 1 after the query was processed for the first time.

Hypothesis 1, 3 and 4 at time instance 2. The system checks all representations and updates the age values as required. The age values of the representations after the query was processed the second time are shown in Table 9.2. The representations which are a good match for the query get reinforced while the ones which are not a good match get their associated age values decremented. As the same query is processed another three times, hypotheses which were not a good match get "aged out" and the representations left in the model have the age values shown in Table 9.3.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 1	2	-190
Hypothesis 2	2	-190
Hypothesis 3	2	-190
Hypothesis 4	2	-190
Original	3	0
Hypothesis 1	3	-190
Hypothesis 2	3	-190
Original	4	0
Hypothesis 1	4	-190
Hypothesis 2	4	-190
Hypothesis 3	4	-190
Hypothesis 4	4	-190
Original	5	0
New Rep	5	-190

Table 9.1: Age of Representations after the query was processed once.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 1	2	-192.5
Hypothesis 2	2	-187.5
Hypothesis 3	2	-192.5
Hypothesis 4	2	-192.5
Original	3	0
Hypothesis 1	3	-187.5
Hypothesis 2	3	-192.5
Original	4	0
Hypothesis 1	4	-192.5
Hypothesis 2	4	-192.5
Hypothesis 3	4	-187.5
Hypothesis 4	4	-187.5
Original	5	0
New Rep	5	-187.5

Table 9.2: Age of Representations after the query was processed twice.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 2	2	-177.5
Original	3	0
Hypothesis 1	3	-177.5
Original	4	0
Hypothesis 3	4	-177.5
Hypothesis 4	4	-177.5
Original	5	0
New Rep	5	-177.5

Table 9.3: Age of Representations after the query was processed 5 times.

The resulting Model 1 is shown in Figure 9.8. The original conditions have been augmented by the hypotheses shown on the right. Significantly, a new model pattern has been added.

If the second method of learning is used all three models in the concept hierarchy would be modified. When the system analyses the query for the first time, it determines that the query matches time instances 2, 3 and 4 in Model 1, time instances 2 and 3 in Model 2 and time instance 5 in Model 3. The updated Models are shown in Figures 9.9 to 9.11. The age of the representations is shown in Tables 9.4 to 9.6. The second time the query is processed, the system attempts to determine which representations in the models best match the query. For example the query matches 100% of the original representation in Hypothesis 1 but does not match Hypothesis 2 at time instance 5 for Model 3. The system processes all representations of one model — one instance at a time and modifies the associated age values as required. The new age values are shown in Tables 9.7 to 9.9. The age values of the models after the query was processed 5 times are shown in Tables 9.10 to 9.12.

From the example it can be seen that the two types of learning offer different advantages — memory requirement and less processing *vs* more accurate representation of the flow of information.

9.5 Results

The *extended version of ILF which uses the best model update method* was used in a fighter combat application. Unlike the sports domain, the data used for the fighter combat simulation uses only 3D spatio-temporal features.

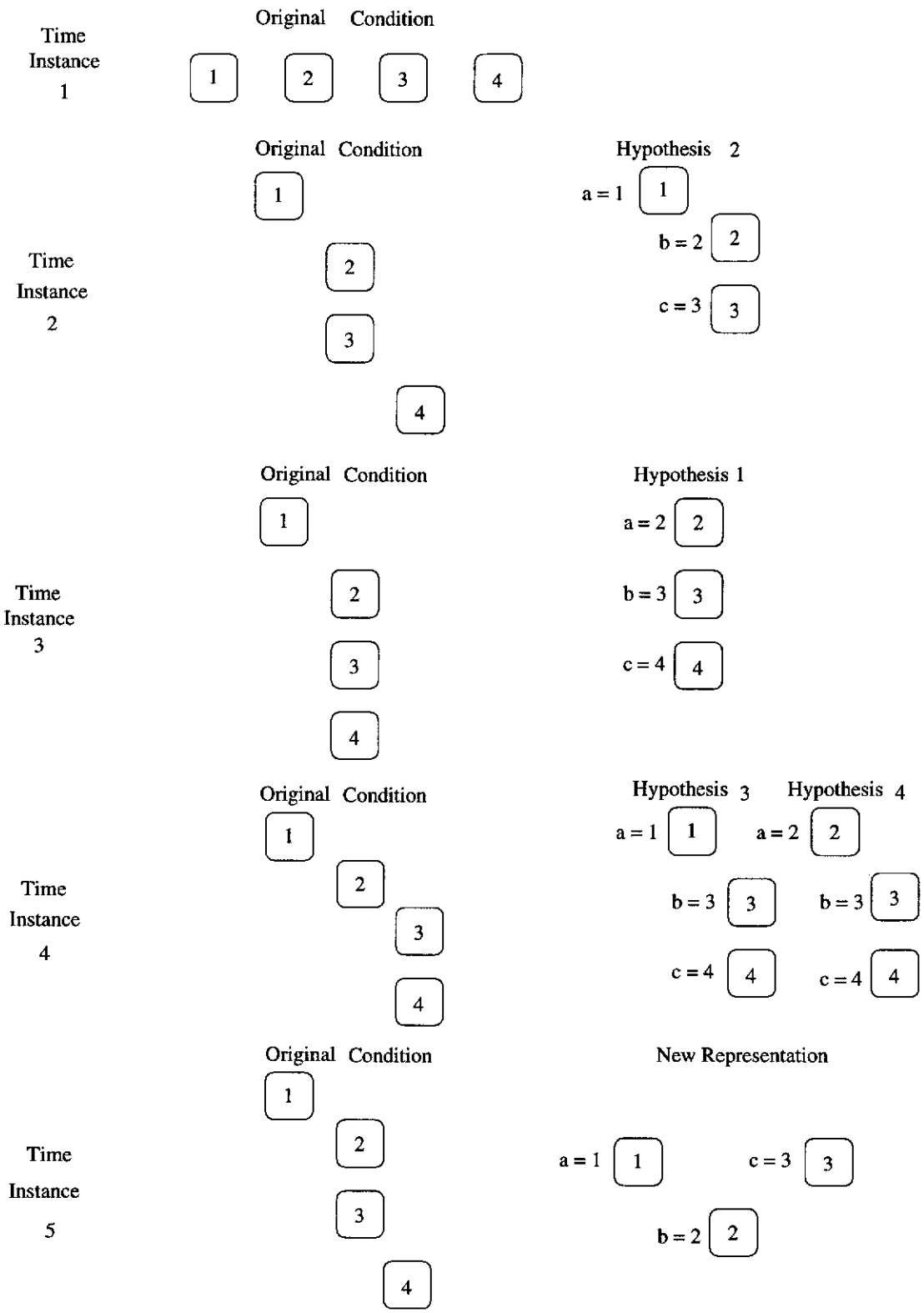


Figure 9.8: Model 1 after the query was processed 5 times.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 1	2	-190
Hypothesis 2	2	-190
Hypothesis 3	2	-190
Hypothesis 4	2	-190
Original	3	0
Hypothesis 1	3	-190
Hypothesis 2	3	-190
Original	4	0
Hypothesis 1	4	-190
Hypothesis 2	4	-190
Hypothesis 3	4	-190
Hypothesis 4	4	-190
Original	5	0

Table 9.4: Age of Representations after the query was processed once for Model 1.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 1	2	-190
Hypothesis 2	2	-190
Hypothesis 3	2	-190
Hypothesis 4	2	-190
Hypothesis 5	2	-190
Original	3	0
Hypothesis 1	3	-190
Hypothesis 2	3	-190
Original	4	0
Original	5	0

Table 9.5: Age of Representations after the query was processed once for Model 2.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Original	3	0
Original	4	0
Original	5	0
Hypothesis 1	5	-190
Hypothesis 2	5	-190

Table 9.6: Age of Representations after the query was processed once for Model 3.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 1	2	-192.5
Hypothesis 2	2	-187.5
Hypothesis 3	2	-192.5
Hypothesis 4	2	-192.5
Original	3	0
Hypothesis 1	3	-187.5
Hypothesis 2	3	-192.5
Original	4	0
Hypothesis 1	4	-192.5
Hypothesis 2	4	-192.5
Hypothesis 3	4	-187.5
Hypothesis 4	4	-187.5
Original	5	0

Table 9.7: Age of Representations after the query was processed twice for Model 1.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 1	2	-192.5
Hypothesis 2	2	-192.5
Hypothesis 3	2	-192.5
Hypothesis 4	2	-187.5
Hypothesis 5	2	-187.5
Original	3	0
Hypothesis 1	3	-187.5
Hypothesis 2	3	-192.5
Original	4	0
Original	5	0

Table 9.8: Age of Representations after the query was processed twice for Model 2.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Original	3	0
Original	4	0
Original	5	0
Hypothesis 1	5	-187.5
Hypothesis 2	5	-192.5

Table 9.9: Age of Representations after the query was processed twice for Model 3.

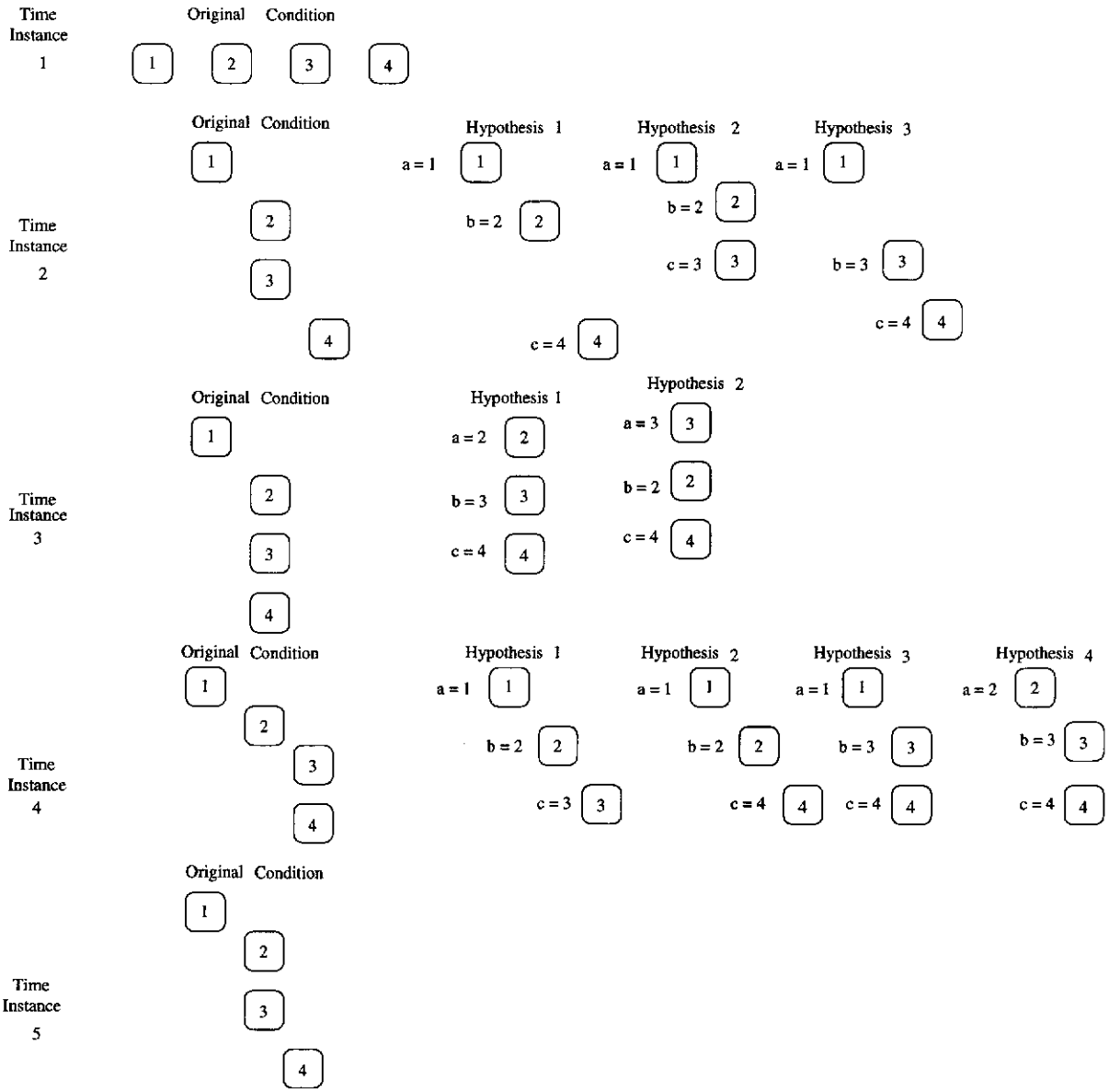


Figure 9.9: Model 1 after the query was processed the first time.

The fighter combat manoeuvers were simulated using information gathered from two sources: fighter combat textbooks [107] and computer flight simulator manuals [44, 25]. The scenarios simulated were divided into an "easy" set and a "hard" set.

The easy set involved between 2 and 6 planes per scenario and covered four basic fighter manoeuvres: one circle turn fight, two circles turn fight, lead pursuit and lag pursuit. The manoeuvres are shown in Figures 9.12, 9.16, 9.13, 9.17, 9.14, 9.18 and 9.15, 9.19. The figures show a plane view of the manoeuvres as well as how the planes change their heights.

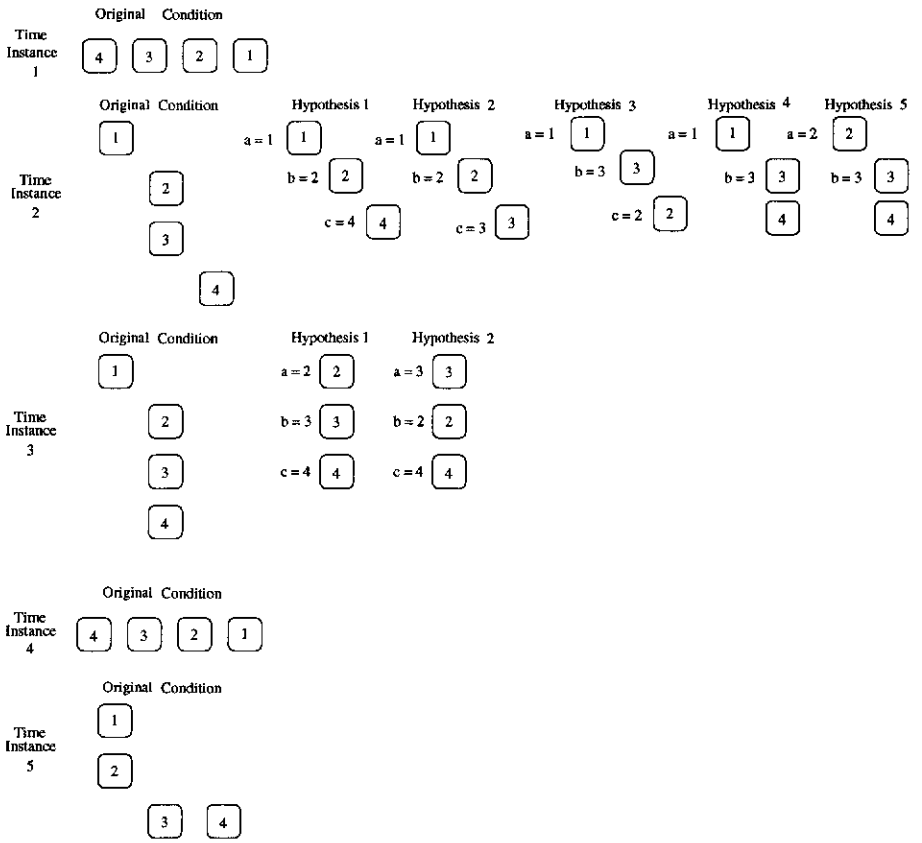


Figure 9.10: Model 2 after the query was processed the first time.

The *difference* between a query and a model was that the *query contained information where 1 (or 2) of the planes were moving differently when compared to the model.* To recognise the four manoeuvres the system was trained using 134 examples. The data which made up an example consisted of the list of (x,y,z) coordinates of the positions of the planes during the scenario. The processing of an example has three steps. In the first step the system tentatively labels the planes in the query. In the second step the system searches for the starting point in the query and models in the database. When the system identifies the starting points it processes all the time instances in the query and determines which is the best matching model (if there are more than one candidate). If the similarity between the best model and the query is greater than or equal to 75% then the model is updated.

Once the training stage was completed, the system attempted to classify 120 new examples of the manoeuvres. The results are shown in Table 9.13.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 2	2	-177.5
Original	3	0
Hypothesis 1	3	-177.5
Original	4	0
Hypothesis 3	4	-177.5
Hypothesis 4	4	-177.5
Original	5	0

Table 9.10: Age of Representations after the query was processed 5 times for Model 1.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Hypothesis 4	2	-177.5
Hypothesis 5	2	-177.5
Original	3	0
Hypothesis 1	3	-177.5
Original	4	0
Original	5	0

Table 9.11: Age of Representations after the query was processed 5 times for Model 2.

Representation	Time Instance	Age
Original	1	0
Original	2	0
Original	3	0
Original	4	0
Original	5	0
Hypothesis 1	5	-177.5

Table 9.12: Age of Representations after the query was processed 5 times for Model 3.

Manoeuvre	Number of Tests	Correct	Incorrect
One Circle Turn Fight	30	26	4
Two Circles Turn Fight	30	24	6
Lead Pursuit Fight	30	25	5
Lag Pursuit Fight	30	24	6

Table 9.13: Easy Scenarios — Results.

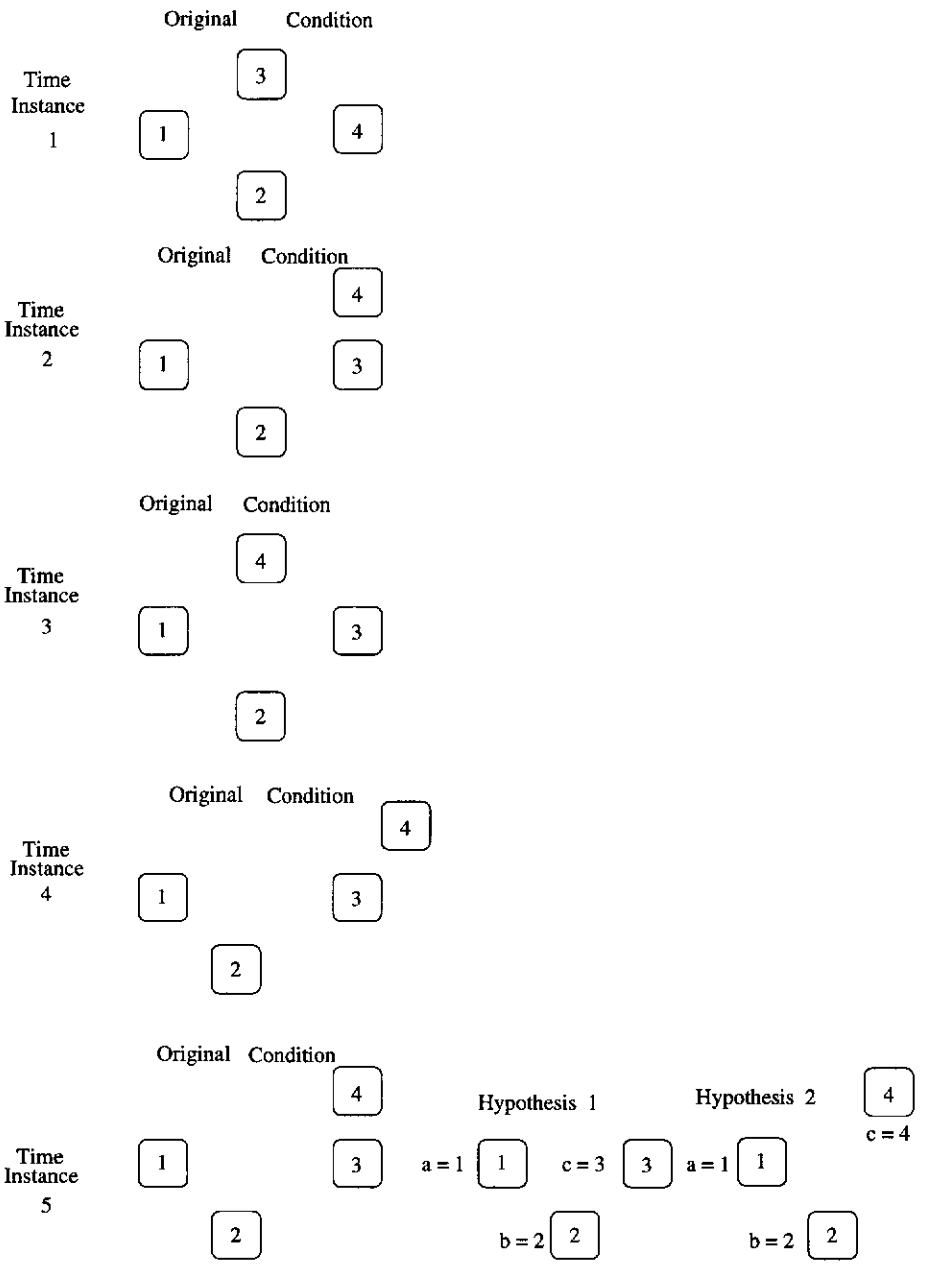


Figure 9.11: Model 3 after the query was processed the first time.

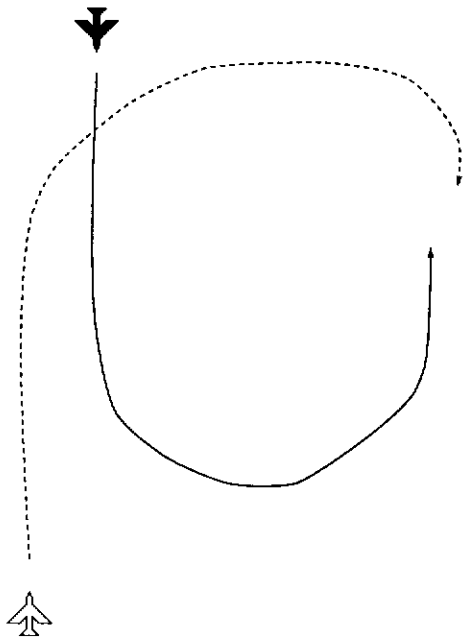


Figure 9.12: One circle manoeuver.

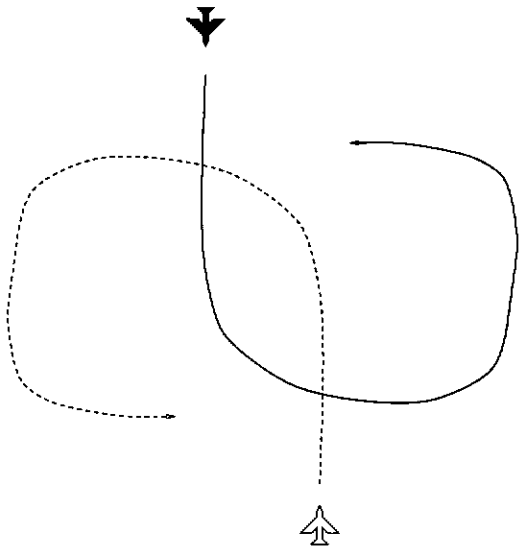


Figure 9.13: Two circles manoeuver.

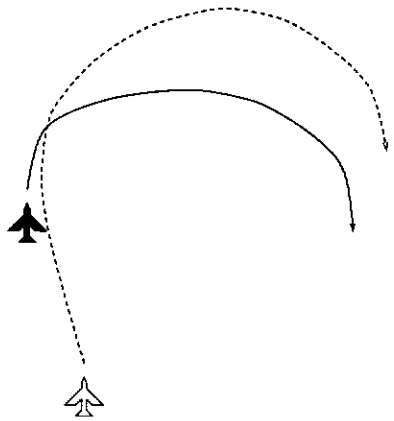


Figure 9.14: Lag pursuit manoeuver.

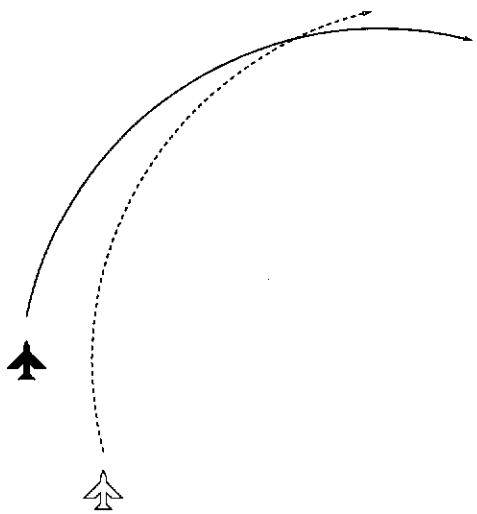


Figure 9.15: Lead pursuit manoeuver.

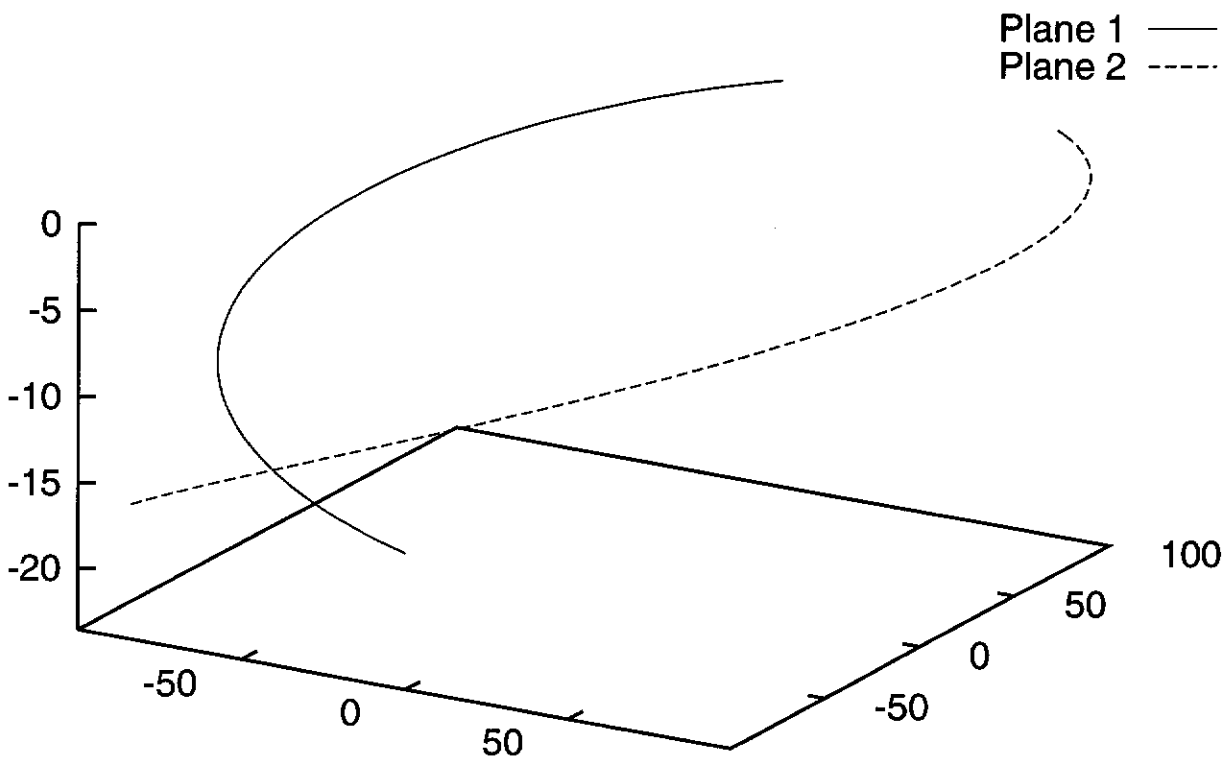


Figure 9.16: One circle manoeuver.

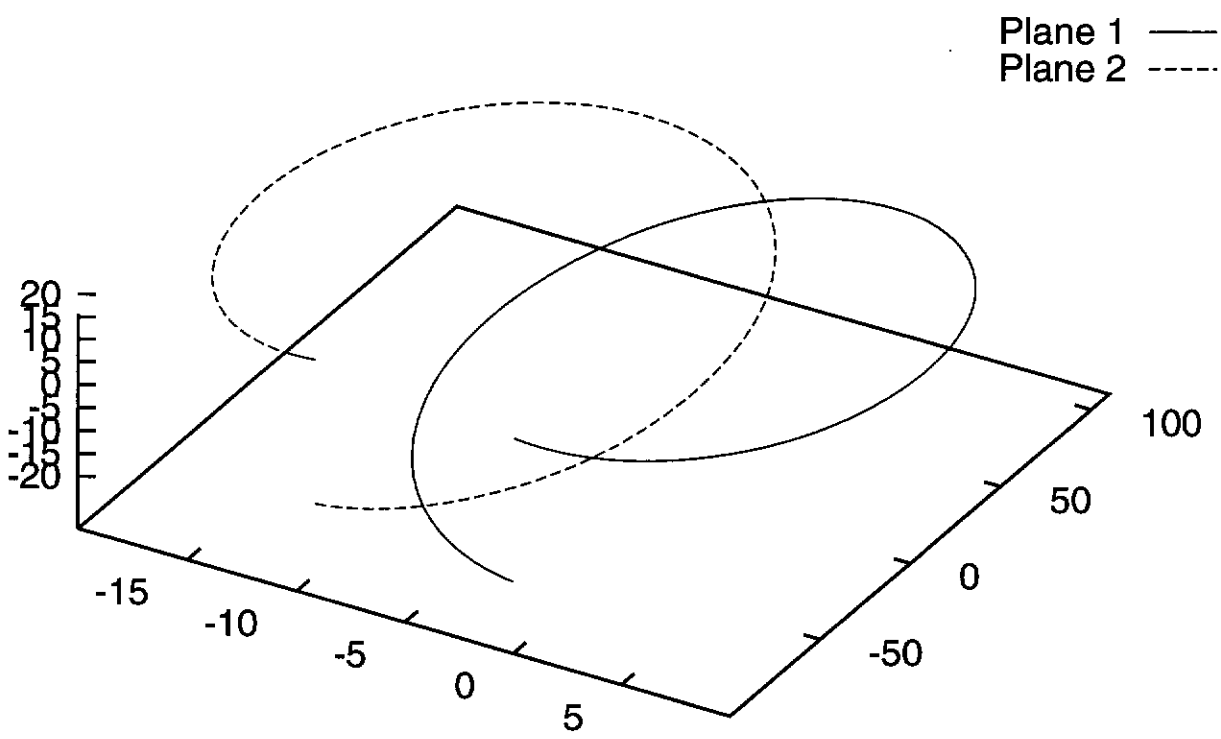


Figure 9.17: Two circles manoeuver.

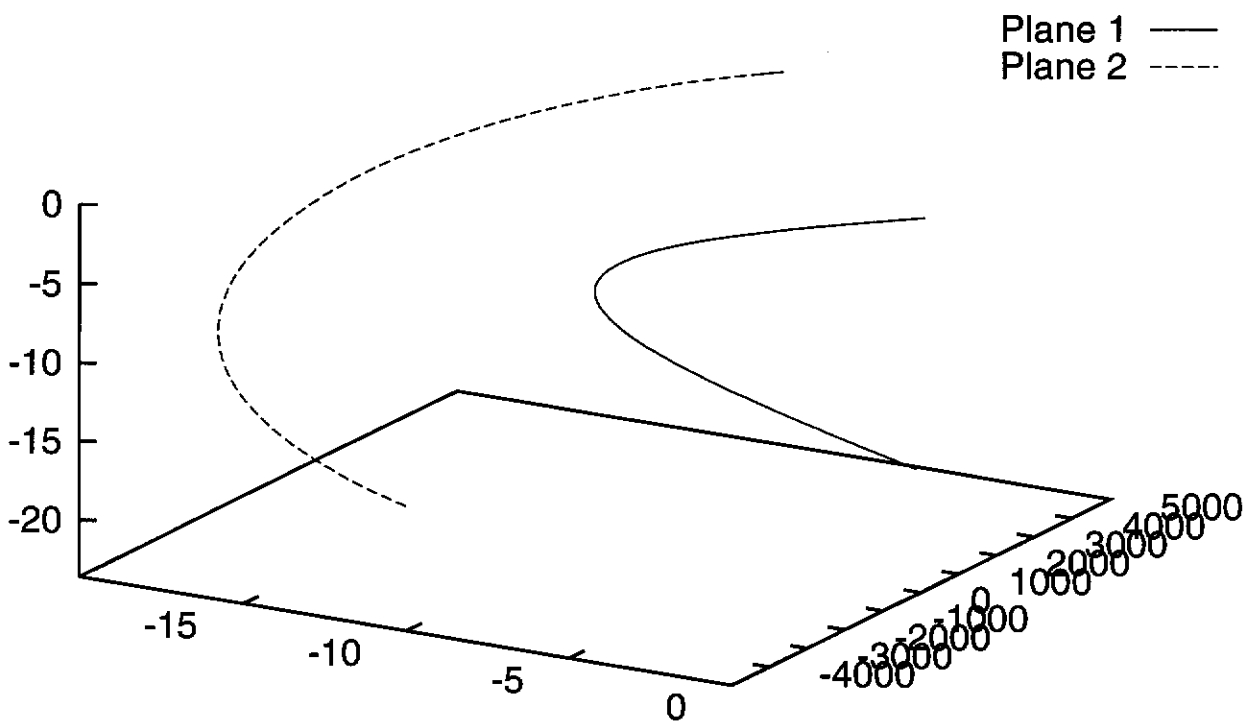


Figure 9.18: Lag pursuit manoeuver.

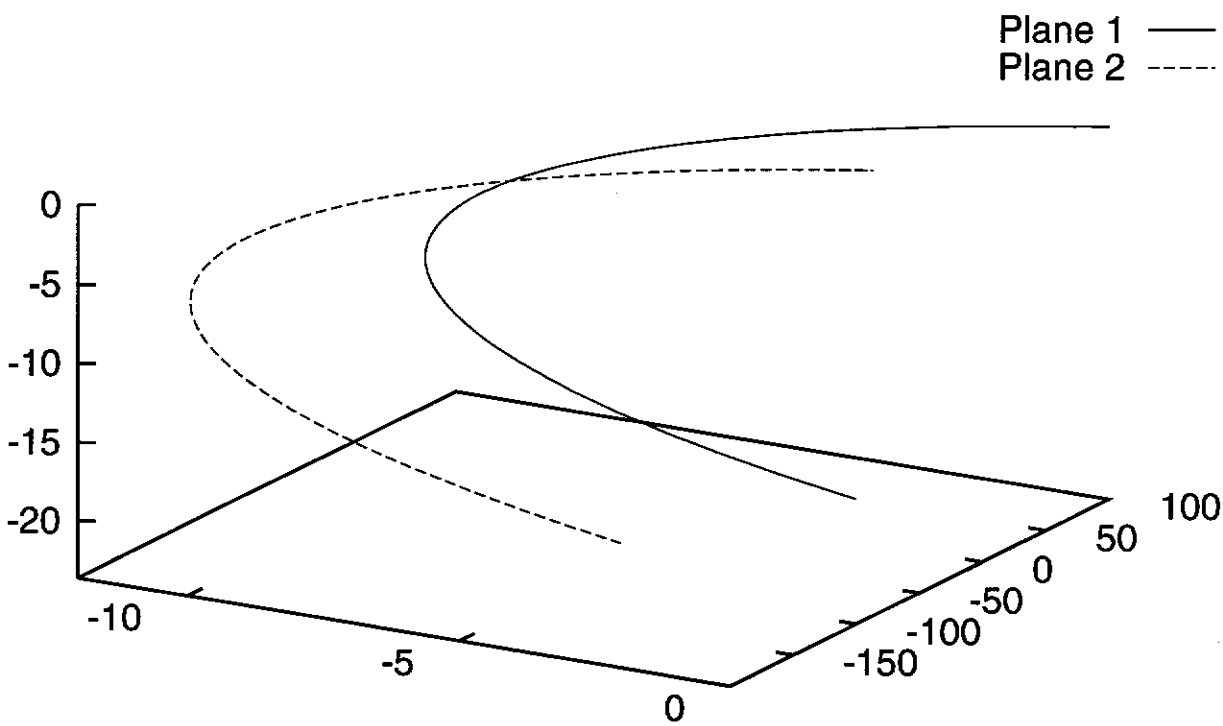


Figure 9.19: Lead pursuit manoeuver.

The hard set involved between 8 and 14 planes per scenario and covered the following fighter manoeuvers: pince attack, single offset attack, champagne attack and chainsaw attack. The manoeuvers are shown in Figures 9.20 9.24, 9.21, 9.25, 9.22, 9.26 and 9.23.

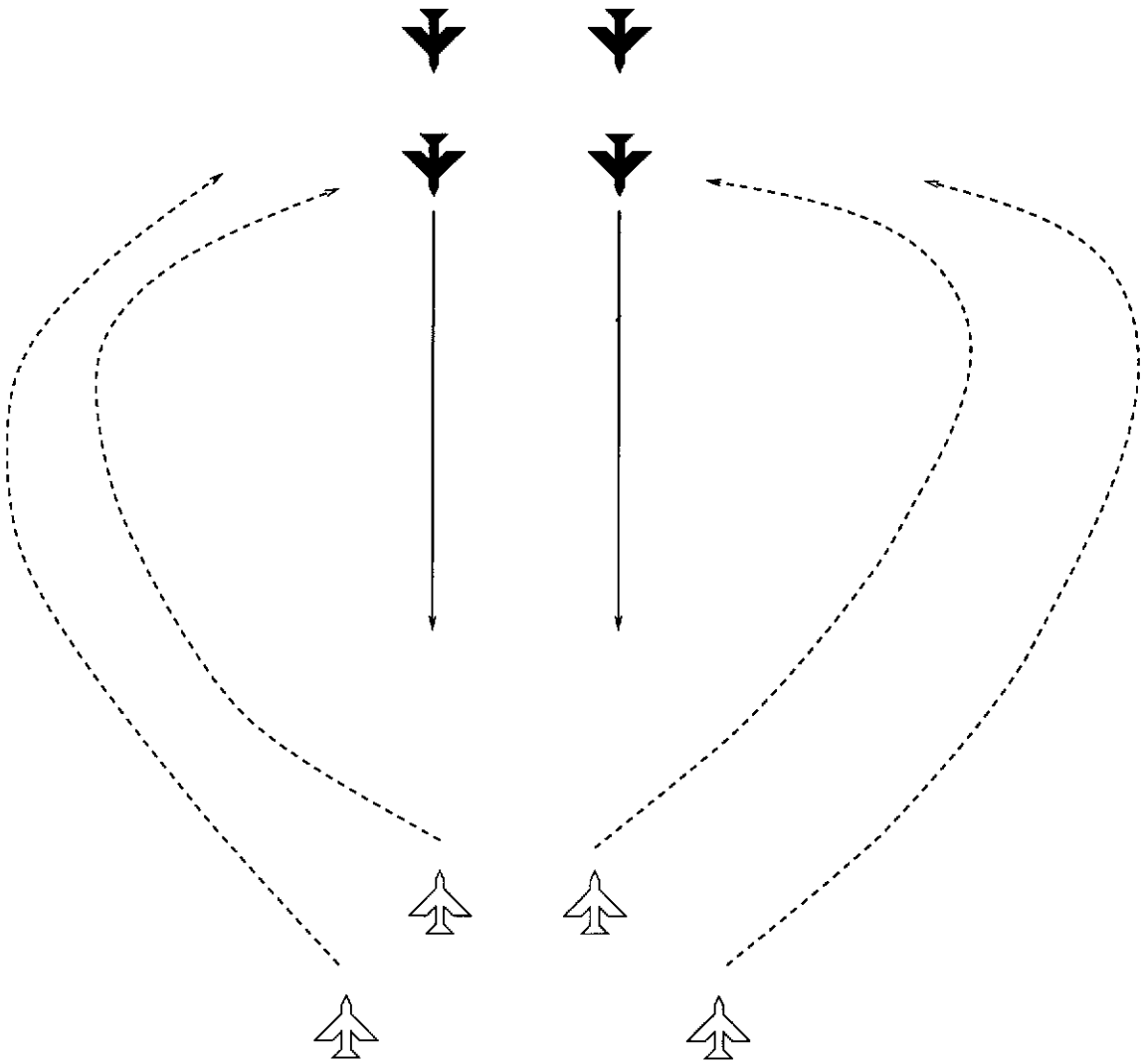


Figure 9.20: Pince manoeuvre.

The *difference* between query and model was that the *query contained information where 3 (or more) of the planes were moving differently when compared with the model.* To recognise the four manoeuvres the system was trained using 172 examples and then tested on 120 new examples of the manoeuvres. The results are shown in Table 9.14.

In both the easy and hard scenarios the system failed to correctly classify some of the manoeuvres. In the case of the easy scenarios, the system misclassified some of the manoeuvers due to the similarity between the data involved in the

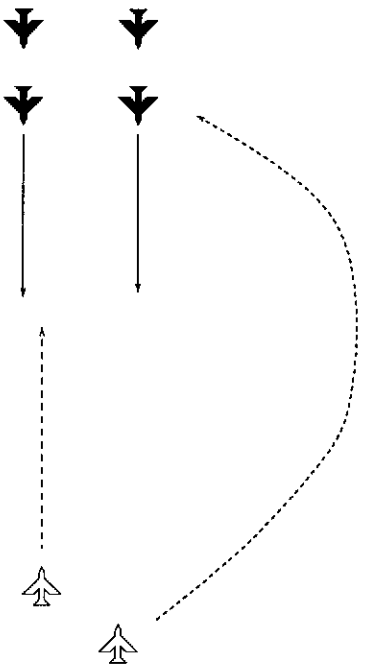


Figure 9.21: Single OffSet manoeuvre.

Manoeuver	Number of Tests	Correct	Incorrect
Pince Attack	25	18	7
Single Offset Attack	25	20	5
Champagne Attack	35	26	9
Chainsaw Attack	35	23	12

Table 9.14: Hard Scenarios — Results.

scenarios (lag and lead pursuit are very similar manoeuvres — the main difference is the approach angle). In the case of the hard scenarios, the system generated new models to describe the manoeuvers. This was due to the complexity of the scenarios (especially the case of chainsaw scenarios — a complex manoeuvre that involves several steps).

9.6 Summary

ILF has been extended to be able to do incremental recognition of queries. This uses a progressive classification (possibly involving backtracking) which considers the data at each time instance in the query and thus provides a probable answer before all the query information becomes available. The extended version of ILF has been also augmented to enable learning in either a stepwise fashion or an overall best match fashion.

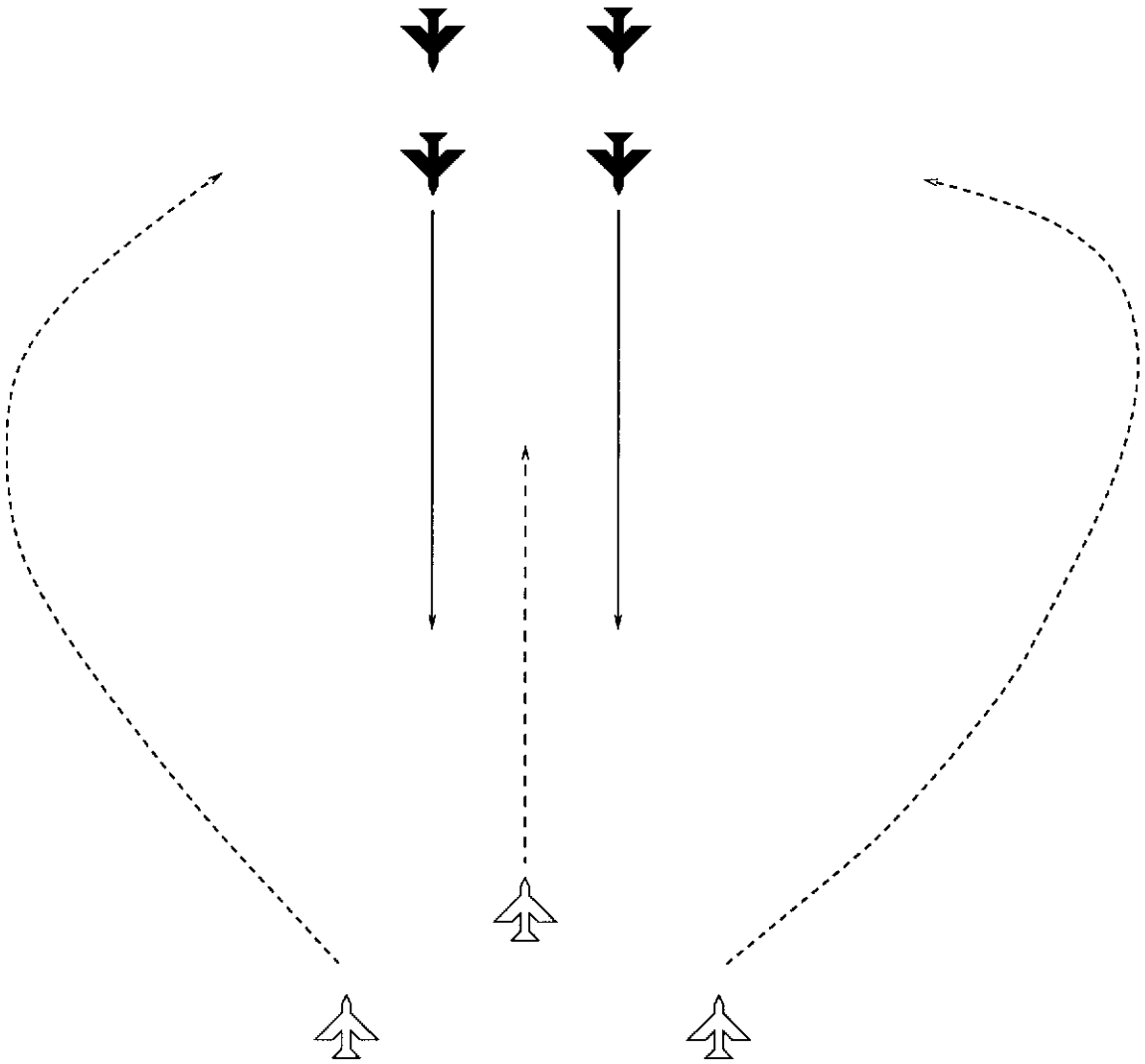


Figure 9.22: Champagne manoeuvre.

Two major issues arise when attempting to combine incremental learning and incremental recognition. The first issue is that of the overall goal. Both methods can produce partial matches for a query but if the goal is to find the best overall match for a query, the first method is more appropriate. However if the goal is to find the best interpretation (or partial match) of data at a given instance in time then the second method should be used.

The second issue is that of the concept update. In the example of incremental recognition shown in this section, we have presented the concepts as an ordered collection of time instances, where each time instance had its own set of objects having a well defined spatial arrangement.

This type of representation works well in cases where *recognition* or *classification* and *learning* is performed after the entire query is processed. However once we

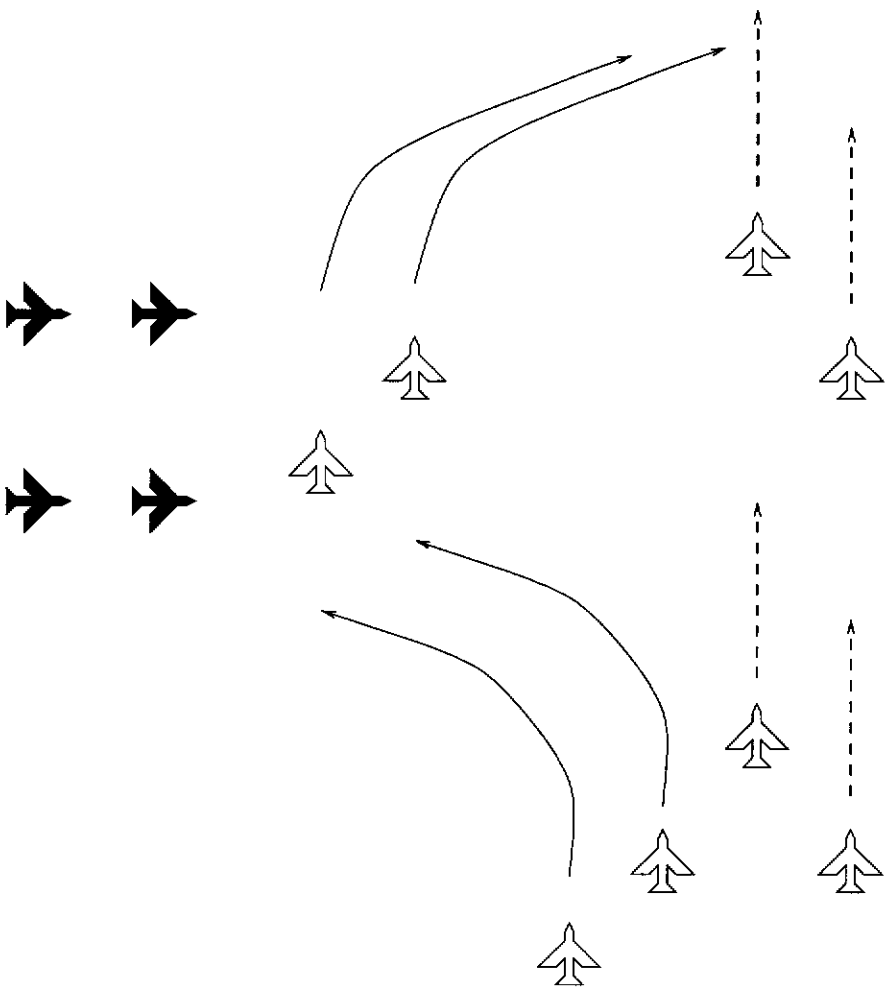


Figure 9.23: Chainsaw manoeuvre.

try to combine incremental recognition with incremental learning, the question is how should the model be updated? Should queries that partially match several models become models themselves? Should the parts of the models which are not matched be updated or not? One potential solution to the problem of model updating is to change the representation of the spatial information and that involves dividing the original model into its time instances — each time instance becomes a model in its own right.

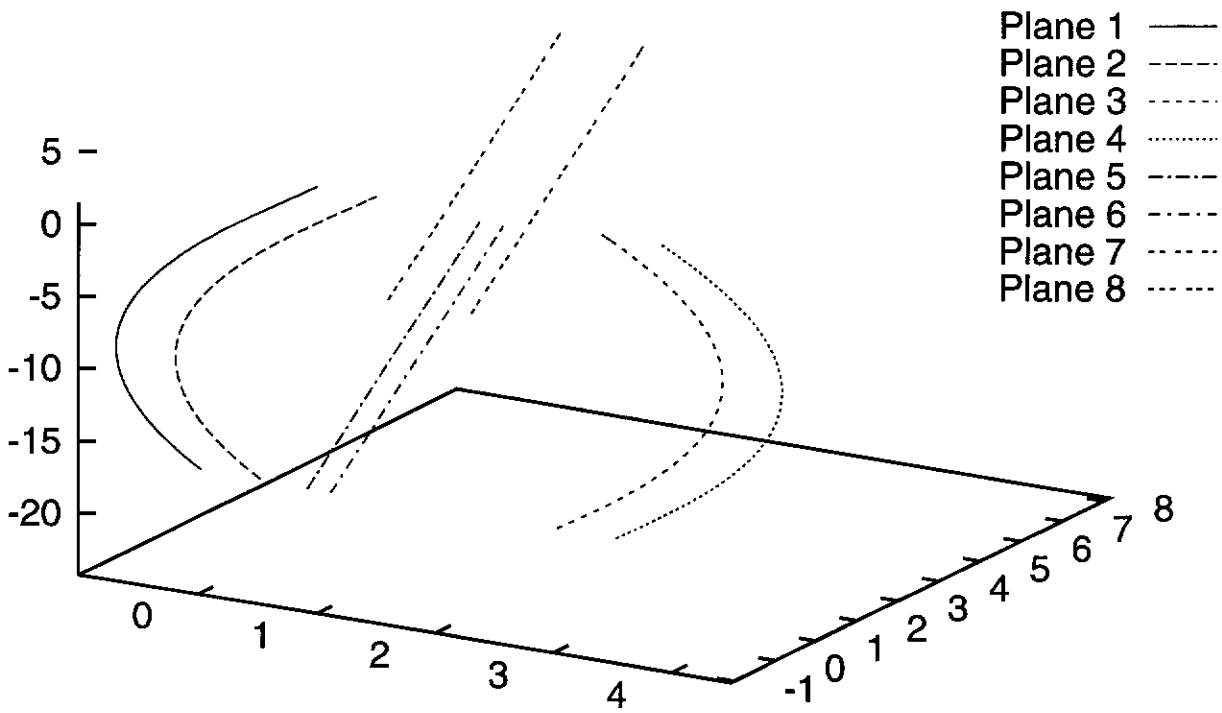


Figure 9.24: Pince manoeuvre.

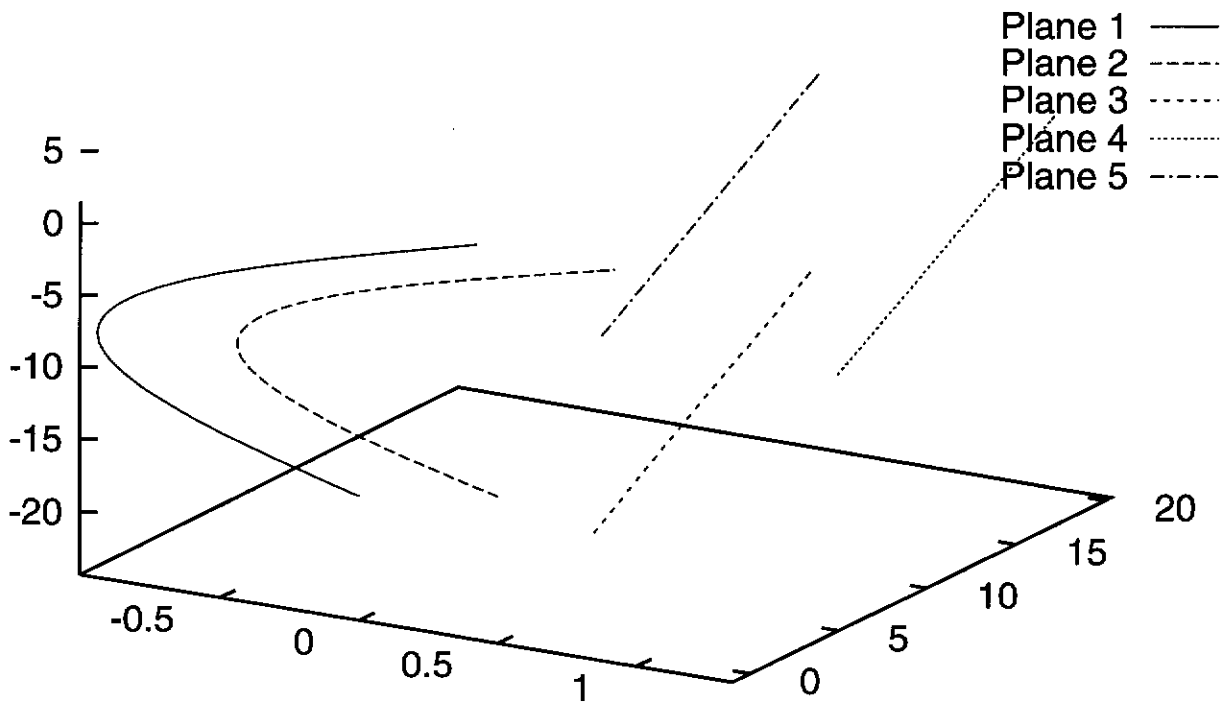


Figure 9.25: Single OffSet manoeuvre.

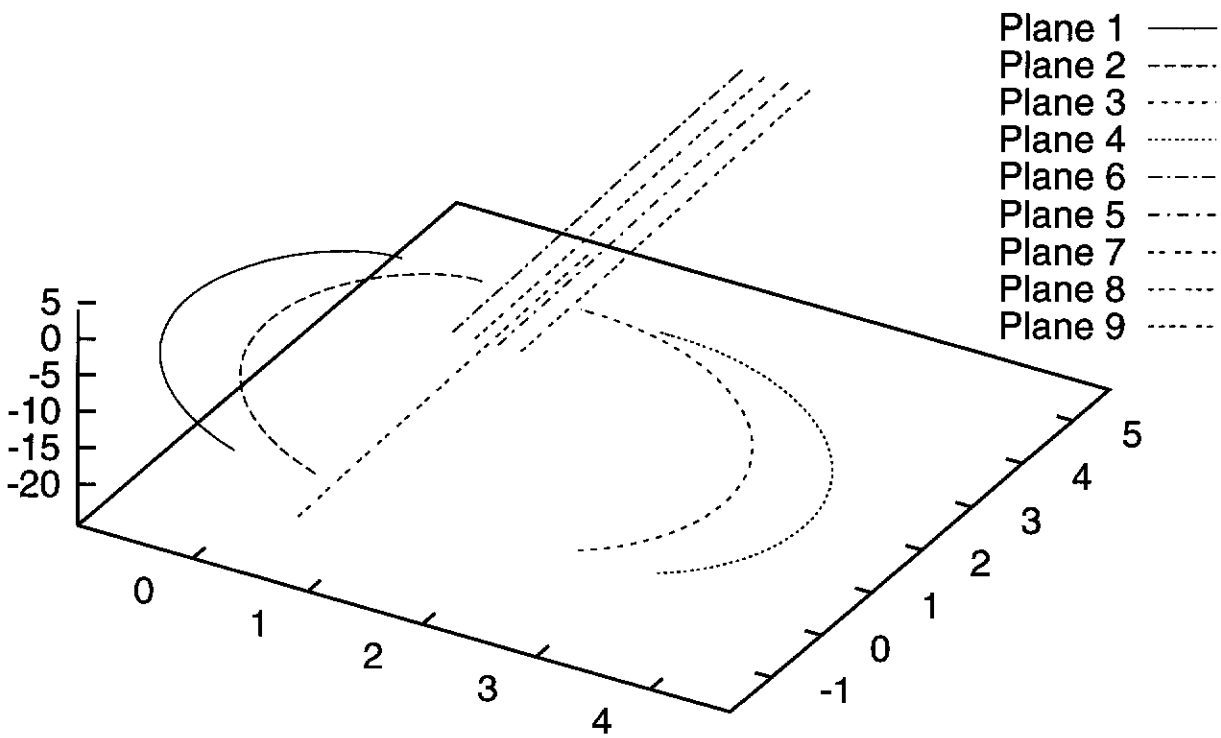


Figure 9.26: Champagne manoeuvre.

Chapter 10

Conclusions and Future Work

10.1 Summary

This thesis has described research performed into the classification and indexing of complex video using machine learning. The first aim was to develop an algorithm that can classify and learn complex spatio-temporal patterns. The second aim was to show that a complex problem such as classifying American Football plays can be solved by combining image processing, text processing and learning. The third aim was to demonstrate that camera parameter estimation and learning can be used to recognise and learn cricket shots whilst, the fourth aim of this thesis was to investigate combining incremental learning and incremental recognition.

Chapter 3 presented the incremental learning algorithm: ILF. The algorithm generates compact conceptual hierarchies, uses an evidence based forgetting method to prune old or irrelevant data and tracks concept drift. When comparing ILF with two similar algorithms — UNIMEM and COBWEB — ILF performs better.

In chapter 4, the knowledge structure to represent American Football plays was presented. American Football plays are very complex and involve a well defined set of movements and actions with each player attempting to complete his task (which can vary from protecting the quarterback to receiving the ball). To capture the knowledge about the plays, a model structure has been developed that combines expert knowledge, domain knowledge (game rules), spatial knowledge (player relationships) and temporal knowledge (action sequencing in individual plays). The play model has several components which describe the play action in terms of *player significance*, *player relationships*, *player movements*, *player actions*, and the temporal sequencing of the *player actions* and *movements*.

In chapter 5, it was shown that by combining simple low-level image processing

with context knowledge it is possible to solve the complex problem of player labelling. The labelling of the players was done with the help of an expert system that uses over 1200 rules to generate the labels for the players based on the information extracted from images.

Chapter 6 presented the algorithm used to recognise plays of American Football. The algorithm has three stages: a text comparison stage (where the clues from the transcript of the commentary are analysed), a spatial relationship stage (where the relationships between the players are compared) and a temporal analysis stage (where the actions of the players and their temporal sequencing are compared). The algorithm can return both complete and partial matches between two plays compared and demonstrates that combining image analysis, text processing and background knowledge can help solve the problem of classifying American Football plays.

In chapter 7, we described how ILF learns spatio-temporal patterns. The algorithm generalises the spatial information by adding any new relationships encountered to a known set of relationships while maintaining the temporal order in which they occurred. The temporal information is generalised in a similar way. The algorithm is simple and demonstrates that symbolic data can be used to describe complex spatio-temporal patterns.

Chapter 8 presented the application of the ILF algorithm to the problem of recognising cricket shots from video data. The methods developed for American Football were not suitable for this application so other techniques have been developed although they still use symbolic attributes. The cricket application shows that combining camera motion estimation and learning can be used to classify and learn cricket shots.

In chapter 9, we explored how incremental learning could be combined with incremental recognition to solve classification tasks where the two main assumptions used in learning were no longer applicable. The first assumption was that the system always knew that the starting point in the training data is always the same as the one in classes/concepts stored in the knowledge base and therefore the system would be able to attempt a comparison straight away. The second was that all the information about the example/query was known before any classification was attempted. ILF was extended to enable incremental recognition of queries. This used a progressive classification (which possibly involved backtracking) that considered the data at each time instance in the query and thus providing a probable answer before all the query information became available. The extended version of ILF was also augmented to enable learning in either a stepwise fashion or an overall best match fashion. Two issues were identified: overall goal and concept update.

10.2 Future Work

The algorithm we have developed in this thesis addresses three issues in incremental learning: concept drift, forgetting and memory size. However more work could be done in the methods of data ageing. ILF uses an evidence based forgetting mechanism. Other forgetting techniques that can be used are exponential decay ageing or time based ageing.

Another aspect of the work that can be further investigated is that of combining incremental learning and incremental recognition. Our investigations were in a domain where the concepts were stable and one possible extension of the work could be to apply the augmented ILF algorithm to domains where true concept drift occurs (see chapter 3 for explanation). Another possible extension of the work would be to investigate different ageing mechanism while attempting to combine incremental learning and incremental recognition.

Appendix A

A

A.1 Introduction

The parser used to process the transcript of the natural language commentary from American Football games was developed using UNIX tools LEX and YACC. LEX is a tool that recognizes lexical patterns in text. LEX reads the given input files for regular expressions and whenever it finds one, it executes the corresponding C code. YACC is used to generate rules that in combination the information extracted by LEX from the text can be used to search the text for complex text patterns. In this Appendix we present the type of keywords used by the parser.

A.2 Text Keywords

There are three classes of keywords that the system is parsing the commentary for:

- **Class 1** - *keywords* that may indicate the name of a *player, play, team or a player position.*
- **Class 2** - *keywords* that may indicate a *play action.*
- **Class 3** - *keywords* that may indicate a game statistic such as *score or down number.*

The keywords take the form of character strings which can be defined by the user in three ways:

- By specifying the entire character string such as *ball*
- By specifying a character string followed by character intervals. For example *(safet)([y]—[ies]* where an acceptable match is either the string *safety* or the string *safeties.*
- By specifying character intervals such as [a-b][a-b]. Valid matches are *aa*, *ab*, *ba* and *bb.*
- By specifying any of the above followed the + or * wildcards. The + stand for match *anything of length 1 or more.* The * wildcard character stands for match *anything of length 0 or more.*

The four classes are described in more detail in the following sections.

A.2.1 Class 1 Keywords

A.2.1.1 Player Names

The assumption made before the parsing is started, is that a database of the player's names in the American Football League is available. The commentary is searched for any strings that start with a capital letter. If the string is found in the database then the string is identified as *player name keyword.*

A.2.1.2 Play Name

A database of American Football plays is also available before the parsing of the commentary is done. The names of the plays were extracted from three sources:

- *Coaching books*
- *Video tapes*

There are general play names for each the three types of possible plays: offensive, defensive or special. There are 42 plays names for the offensive plays and 38 for the defensive plays. The names of the special plays are listed below:

- Punt
- Fake Punt
- Field Goal

- Conversion
- Kick Off
- Kick Return
- Punt Return
- Punt Block
- Field Goal Block
- Stop Clock

A.2.1.3 Team Name

The names of the 30 teams are stored as part of the background knowledge. When searching for the name of a team, the system looks for:

- The full team name such Denver Broncos
- The name of the city where the team is based such as Denver
- The team name such as Broncos

The names of the teams are shown below:

- New England Patriots
- New York Jets
- Buffalo Bills
- Miami Dolphins
- Indianapolis Colts
- Dallas Cowboys
- Washington Redskins
- New York Giants
- Philadelphia Eagles
- Arizona Cardinals
- Jacksonville Jaguars
- Pittsburgh Steelers

- Baltimore Ravens
- Cincinnati Bengals
- Tennesse Oilers
- Tampa Bay Bucanners
- Green Bay Packers
- Minnessota Vikings
- Detroit Lions
- Chicago Bears
- Denver Broncos
- Kansas City Chiefs
- San Diego Chargers
- Oakland Raiders
- San Francisco 49
- Carolina Panthers
- Saint Louis Rams
- New Orleans Saints
- Atlanta Hawks

A.2.1.4 Player Position Name

The positions of the players who take part in offensive plays are:

- Quarterback
- Guard
- Receiver
- Wide-Receiver
- Half-Back
- Running-Back

- Blocker
- Corner-Back
- Linebacker
- Tight-End
- Kicker
- Punter
- Snap
- Full Back
- Tail Back
- Slot Back
- Flanker
- Right Tackle
- Left Tackle
- Right Guard
- Center
- Left Guard

Similarly the system is searching the text for 14 defensive players and 8 special teams players.

A.2.2 Class 2 Keywords

Any verb that may describe an action in the play is a Class 2 keyword. There three types of action: offensive, defensive and special. In the case of the offensive action, the system searches for verb that indicate:

- That the ball was passed.
  - Pass
  - Throw
  - Toss

  - Hit
  - Lob
  - Unload
  - Telegraph
  - Find
  - Connect
  - Target
- That the ball was handed
  - Hand
  - Give
  - Leave
- That the ball was kicked
  - Kick
  - Punt

In the case of defensive action the system is looking for:

- Tackle
- Grab
- Push
- Hit
- Grill
- Hammer
- Flatten
- Shove
- Hold
- Block
- Cover
- Intercept
- Fumble

- Level
- Blind
- Intervene
- Sack
- Stop
- Deflect
- Blitz
- Bump
- Overpower

For the special plays, the releveant character strings are:

- Return
- Kick
- Fake

A.2.3 Class 3 Keywords

The system looks for both character strings and numerical strings. The keywords of interest are shown below:

- 1st down
- 2nd down
- 3rd down
- 4th down
- yard number
- quarter number
- yard line
- time left

Appendix B

B

B.1 Introduction

The keywords defined in Appendix A are used by YACC to recognise the patterns two types of patterns: complex patterns and simple patterns. The complex patterns are themselves constructed using simpler patterns and keywords while the simple patterns are constructed using *only* keywords.

B.2 Text Patterns

The patterns are shown below (the text analysis starts with most complex pattern and goes down the simplest pattern).

```
game : play game
     | statistics game
     | play
     | statistics;

play : any_play play
     | any_play;


any_play : offensive_play
         | defensive_play
         | special_play
         | empty_list;

offensive_play : throw_play
                 .
                 . for each verb describing an offensive action
                 . (see Appendix A) there is a pattern such as the throw_play
                 .
               | hit_play;
```

```
special_play : kick_play
             | receive_play
             | punt_play;

throw_play :  lead_type THROW TO receiver_type
            | lead_type THROW receiver_type
            | lead_type THROW;

receiver_type : guard_position
              | wide_receiver_position
              | receiver_position
              | tight_end_position
              | running_back_position
              | half_back_position;

guard_position : GUARD NAME NAME
               | GUARD NAME
               | NAME;
```

Similar patterns are defined for the rest of the receiver positions.

```
defensive_play : tackle_play
                 .
                 . for each verb describing a defensive action (see Appendix
                 . A) there is a pattern such as the cover_play
               | cover_play;


tackle_play : receiver_type TACKLE BY defender_type
            | lead_type TACKLE BY defender_type
            | defender_type WITH TACKLE
            | TACKLE ON receiver_type BY defender_type
            | TACKLE ON lead_type BY defender_type
            | TACKLE BY defender_type;

statistics : yard_line
           | yards_gained
           | yards_lost
           | down_number
           | attack zone line
           | defensive zone line
           | score
           | time period
           | time left;


yard_line : NUMBER YARD
          | ON NUMBER YARD LINE
          | ON NUMBER YARD
          | AT NUMBER YARD LINE
          | AT NUMBER YARD
          | TO NUMBER YARD LINE
          | TO NUMBER YARD;
```

Similar patterns are defined for the rest of the statistics options.

```
empty_list : TO empty_list
           | ON empty_list
           | ART empty_list
           | TO
           | ON
           | ART;_position;
```

Appendix C

C

The results of the remaining 17 video sequences processed by the system are presented in this Appendix. The format of the results is: commentary, information frames and result. The results marked by a * indicate that the text processing failed, while the results marked by ** indicate that the video analysis failed to choose the correct play.

C.1 Test Sequence 1

C.1.1 Commentary 1

3rd down and 1 . PlayerX to PlayerY , finally a 1st down and PlayerZ covering PlayerY .

Figure C.1: Transcript of the commentary for the play 1.

C.1.2 Information Frames

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	3rd
Yards-to-go	1
Time	9
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Table C.1: Play Frame.

Play-name	Cover
Offensive-team	-
Defensive-team	-
Down-No.	3rd
Yards-to-go	9
Time	9
Player1	PlayerZ
Player2	PlayerY
Player1-Type	Linebacker
Player2-Type	Receiver
Instance	2

Table C.2: Play Frame.

Play-Model-Name	Play Evidence Score
Square-In (broken)	248
Pa-Pass (normal)	223
Weak-Flood (hb-back)	205
Quick-Outs	196
Quick-Slant (normal)	190

Table C.3: Result for test sequence 1.

C.2 Test Sequence 2

C.2.1 Commentary 2

Took roughly 31 minutes for Team1 to get a 1st down , PlayerX , and PlayerY gets him into the turf at the 45 yard line .

Figure C.2: Transcript of the commentary for the play 2.

C.2.2 Information Frames

Play-name	Run
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	-
Player2-Type	Running-Back
Instance	1

Table C.4: Play Frame.

Play-name	Tackle
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	9
Player1	PlayerY
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Table C.5: Play Frame.

Play-Model-Name	Play Evidence Score**
HB-Off-Tackle (normal)	232
FB-Off-Tackle (normal)	210
HB-Off-Tackle (2te)	193
FB-Opt-Drive	186
HB-Hook	174

Table C.6: Result for test sequence 2.

C.3 Test Sequence 3

C.3.1 Commentary 3

PlayerX to PlayerY this time , unsuccessful , PlayerZ made the tackle .

Figure C.3: Transcript of the commentary for the play 3.

C.3.2 Information Frames

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Table C.7: Play Frame.

Play-name	Tackle
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerZ
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Table C.8: Play Frame.

Play-Model-Name	Play Evidence Score**
HB-Off-Tackle	203
HB-Dive	184
Cross-Pass	178
FB-Inside-Run	169
HB-Toss	157

Table C.9: Result for test sequence 4.

C.4 Test Sequence 4

C.4.1 Commentary 4*

==========

3rd and 10 for PlayerX , PlayerY going forward . The throw to PlayerZ is complete and we have a touchdown.

==========

Figure C.4: Transcript of the commentary for the play 4.

C.4.2 Information Frames 4

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	3rd
Yards-to-go	10
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Running-Back
Instance	1

Table C.10: Play Frame.

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	3rd
Yards-to-go	10
Time	-
Player1	PlayerZ
Player2	-
Player1-Type	Receiver
Player2-Type	-
Instance	2

Table C.11: Play Frame.

Play-Model-Name	Play Evidence Score
In-Out	264
Square-In	242
Break-In	210
Pa-Pass	194
HB-Sprint-Draw	183

Table C.12: Result for test sequence 4.

C.5 Test Sequence 5

C.5.1 Commentary 5*

PlayerX's 4 touchdowns get him a seat on the bench and PlayerY from the Citadel or is it the Citadel .

Figure C.5: Transcript of the commentary for the play 5.

C.5.2 Information Frame

Play-name	-
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Quarterback
Instance	1

Table C.13: Play Frame.

Play-Model-Name	Play Evidence Score**
Belly-Weak	210
Square-In	180
Out-n-Ups	173
Quick-Slant	165
Power-Weak	160

Table C.14: Result for test sequence 5.

C.6 Test Sequence 6

C.6.1 Commentary 6

PlayerX again and his jersey is on the ground , PlayerY made the tackle .

Figure C.6: Transcript of the commentary for the play 6.

C.6.2 Information Frames

Play-name	Run
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	9
Player1	PlayerX
Player2	-
Player1-Type	Running-Back
Player2-Type	-
Instance	1

Table C.15: Play Frame.

Play-name	Tackle
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerY
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Table C.16: Play Frame.

Play-Model-Name	Play Evidence Score
Square-In	257
HB-DIve-Left	245
HB-Draw	231
FB-Inside-Run	210
Power-Weak	187

Table C.17: Result for test sequence 6.

C.7 Test Sequence 7

C.7.1 Commentary 7

Number 19 , PlayerX maybe gained about 2 yards .

Figure C.7: Transcript of the commentary for the play 7.

C.7.2 Information Frames 7

Play-name	Run
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	Running-Back
Player2-Type	-
Instance	1

Table C.18: Play Frame.

Play-Model-Name	Play Evidence Score
Square-In	256
FB-Inside-Run	210
HB-Toss	198
HB-Trap	194
HB-Draw	175

Table C.19: Result for test sequence 7.

C.8 Test Sequence 8

C.8.1 Commentary 8

PlayerX to PlayerY . I think this game is going to make them become a better team . PlayerY has 1st down at the 27 yard line .

Figure C.8: Transcript of the commentary for the play 8.

C.8.2 Information Frames 8

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Table C.20: Play Frame.

Play-Model-Name	Play Evidence Score
In-Out	256
HB-Hook	210
HB-Toss	198
HB-Toss-Pass	194
Quick-Outs	175

Table C.21: Result for test sequence 8.

C.9 Test Sequence 9

C.9.1 Commentary 9

Team1 has the home field advantage and Team2 is working on it . PlayerX and you might you want to bring up the question Conference1 versus Conference2 .

Figure C.9: Transcript of the commentary for the play 9.

C.9.2 Information Frames 9

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	Receiver
Player2-Type	-
Instance	1

Table C.22: Play Frame.

Play-Model-Name	Play Evidence Score**
FB-Off-Tackle	224
FB-Inside-Run	210
HB-Wham	187
Belly-Strong	179
Power-Weak	162

Table C.23: Result for test sequence 9.

C.10 Test Sequence 10

C.10.1 Commentary 10

PlayerX, they call the grand cannon in State near the Grand Canyon and PlayerY has gone down to the 49 yard line .

Figure C.10: Transcript of the commentary for the play 10.

C.10.2 Information Frames 10

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Table C.24: Play Frame.

Play-Model-Name	Play Evidence Score**
HB-Hook	230
Belly-Weak	221
HB-Sprint-Draw	215
Power-Weak	
FB-Inside-Run	184

Table C.25: Result for test sequence 10.

C.11 Test Sequence 11

C.11.1 Commentary 11

Our congratulations to Team1 , Conference1 champions , as PlayerX has a 1st down at the Team1 47 yard line .

Figure C.11: Transcript of the commentary for the play 11.

C.11.2 Information Frames 11

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	Receiver
Player2-Type	-
Instance	1

Table C.26: Play Frame.

Play-Model-Name	Play Evidence Score**
Belly-Weak	241
HB-Wham	227
HB-Sprint-Draw	220
FB-Inside-Run	215
Power-Weak	184

Table C.27: Result for test sequence 11.

C.12 Test Sequence 12

C.12.1 Commentary 12

Team1 has lost 7 games in a row to Team2 in Texas Stadium . PlayerX underneath , incomplete with 11 seconds left , PlayerY the intended receiver .

Figure C.12: Transcript of the commentary for the play 12.

C.12.2 Information Frames

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Table C.28: Play Frame.

Play-Model-Name	Play Evidence Score**
Quick-Slant	236
Square-In	219
Cross-Pass	194
Quick-Outs	189
Pa-Curls	175

Table C.29: Result for test sequence 12.

C.13 Test Sequence 13

C.13.1 Commentary 13

After the penalty the Team1 start inside their 9 . PlayerX on the goaline , the tail back , and PlayerY to throw down the middle , incomplete as PlayerZ was unable to hang on at the 18 , PlayerW and PlayerU on the coverage .

Figure C.13: Transcript of the commentary for the play 13.

C.13.2 Information Frames

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerY
Player2	PlayerZ
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Table C.30: Play Frame.

Play-name	Cover
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerW
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Table C.31: Play Frame.

Play-name	Cover
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerU
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	3

Table C.32: Play Frame.

Play-Model-Name	Play Evidence Score
In-Out	249
Break-In	232
Square-In	225
Quick-Outs	210
HB-Draw	167

Table C.33: Result for test sequence 13.

C.14 Test Sequence 14

C.14.1 Commentary 14

2nd and 10 . PlayerX gets it and picks his way to the 10 maybe the 11 for a 3 yard gain .

Figure C.14: Transcript of the commentary for the play 14.

C.14.2 Information Frames 14

Play-name	Run
Offensive-team	-
Defensive-team	-
Down-No.	2nd
Yards-to-go	10
Time	10
Player1	PlayerX
Player2	-
Player1-Type	Running-Back
Player2-Type	-
Instance	1

Table C.34: Play Frame.

Play-Model-Name	Play Evidence Score**
HB-Dive-Left	253
HB-Draw	240
Square-In	229
Break-In	208
FB-Inside-Run	197

Table C.35: Result for test sequence 14.

C.15 Test Sequence 15

C.15.1 Commentary 15

Team1 against the Kansas crowd , and almost intercepted as PlayerX was hit as he throwed . PlayerY , number 99 was on PlayerX .

Figure C.15: Transcript of the commentary for the play 15.

C.15.2 Information Frames

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	Quarterback
Player2-Type	-
Instance	1

Table C.36: Play Frame.

Play-name	Tackle
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerY
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Table C.37: Play Frame.

Play-Model-Name	Play Evidence Score**
Pa-Pass	209
In-Out	197
Pa-Curls	182
Quick-Outs	177
Break-In	166

Table C.38: Result for test sequence 15.

C.16 Test Sequence 16

C.16.1 Commentary 16

PlayerX to throw again to the sidelines , incomplete , PlayerY unable to hang on , PlayerZ was covering .

Figure C.16: Transcript of the commentary for the play 16.

C.16.2 Information Frames

Play-name	Pass
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	PlayerY
Player1-Type	Quarterback
Player2-Type	Receiver
Instance	1

Table C.39: Play Frame.

Play-name	Cover
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerZ
Player2	-
Player1-Type	Linebacker
Player2-Type	-
Instance	2

Table C.40: Play Frame.

Play-Model-Name	Play Evidence Score
Quick-Outs	262
Cross-Pass	247
In-Out	225
Out-n-Ups	219
Quick-Slant	186

Table C.41: Result for test sequence 16.

C.17 Test Sequence 17

C.17.1 Commentary 17

The 75th game between these two teams , dead even with 36 wins each and 2 ties , but Team1 dominating lately , winning 12 of the 13 as PlayerX bowling his way forward and earning a couple of extra yards down to the 10 , 5 yards gained on the play.

Figure C.17: Transcript of the commentary for the play 17.

C.17.2 Information Frames

Play-name	Run
Offensive-team	-
Defensive-team	-
Down-No.	-
Yards-to-go	-
Time	-
Player1	PlayerX
Player2	-
Player1-Type	Running-Back
Player2-Type	-
Instance	1

Table C.42: Play Frame.

Play-Model-Name	Play Evidence Score
HB-Wham	231
Belly-Strong	214
HB-Sprint-Draw	193
Power-Weak	185
HB-Off-Tackle	178

Table C.43: Result for test sequence 17.

Bibliography

[1] H. Hashimoto A. Akutsu, Y. Tonomura and Y. Ohba. Video indexing using motion vectors. In *Conference Proceedings of SPIE Visual Communication and Image Processing 1992*, pages 1522–1530, 1992.

[2] Personal communication, 1997.

[3] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, (23):123–154, 1984.

[4] James F. Allen. Maintaining knowledge about temporal intervals. In Ronald J. Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*, pages 510–521. Morgan-Kaufmann, 1985.

[5] James F. Allen and Patrick Hayes. A common-sense theory of time.

[6] Elisabeth Andre, Guido Bosch, Gerd Herzog, and Thomas Rist. Characterizing trajectories of moving objects using natural language path descriptions. In *Proceedings of the 7th European Conference on AI*, volume 2, pages 1–8, 1986.

[7] Elisabeth Andre, Guido Bosch, Gerd Herzog, and Thomas Rist. Coping with the intrinsic and deictic uses of spatial prepositions. In K. Jorrand and L. Sgurev, editors, *Artificial Intelligence II: Methodology, Systems, Applications*, pages 375–382. North-Holland, 1987.

[8] Elisabeth Andre, Gerd Herzog, and Thomas Rist. On the simultaneous interpretation of real world image sequences and their natural language description: The system soccer. In *Proceedings of the 8th European Conference on AI*, pages 449–454, 1988.

[9] Elisabeth Andre, Gerd Herzog, and Thomas Rist. Natural language access to visual data: Dealing with space and movement. 1st Workshop on Logical Semantics of Time, Space and Movement in Natural Language, 1989.

[10] Electronic Arts. Madden NFL 98, 1998.

[11] Channel 2 Australia. American football - game of week (season 93, 94, 95).

[12] Mosche Bar and Shimon Ullman. Spatial context in recogition. Technical report, The Weizmann Institute of Science, June 1996.

[13] F. Bergadano, S. Matwin, R.S. Michalski, and J. Zhang. Learning two-tiered descriptions of flexible concepts: The poseidon system. *Machine Learning*, 8:5–43, 1992.

[14] Aaron F. Bobick. Video annontation: Computers watching video. In *Second Asian Conference on Computer Vision 1995*, pages 19–23, December 1995.

[15] R.D. Boyle and R.C. Thomas. *Computer Vision - A first course.* Blackwell Scientific Publications, 1990.

[16] A.J. Bray. Tracking objects using image disparities. *Image and Vision Computing*, 8:4–9, 1990.

[17] Gary Briscoe and Terry Caelli. *A Compendium of Machine Learning.* Ablex, 1996.

[18] Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell. Learning from observation: conceptual clustering. In . Michalski R. S, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 331–363. Morgan Kauffman, 1983.

[19] Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell. An overview of machine learning. In . Michalski R. S, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 3–23. Morgan Kauffman, 1983.

[20] Peter Clark. Machine learning: Techniques and recent developments. In A. R. Mirzai, editor, *Artificial Intelligence: Concepts and Applications in Engineering*, pages 65–93. Chapman and Hall, 1990.

[21] Cable news network/intel cnn at work - live news on your networked pc. http://www.intel.com/comm-net/cnn-work/index.html.

[22] Michael T. Cox and Ashwin Ram. An explicit representation of forgetting. In *Proceedings of the Sixth International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden*, pages 115–120, August 1992.

[23] M. Cristel and K. Pendyala. Informedia goes to school: Early findings from the digital video library project. http://www.dlib.org/dlib/september96/informedia/09christel.html, September 1996.

[24] Croydon rangers play book. Coaching Book, 1991.

[25] Digital Image Design. E.F. 2000 manual, 1996.

[26] Mark Devaney and Ashwin Ram. Dynamically adjusting concepts to accomodate changing contexts. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, pages 1441–1450, 1994.

[27] Mark Devaney and Ashwin Ram. Efficient feature selection in conceptual clustering. In *Machine Learning: Proceedings of the Fourteenth International Conference, Nashville*, July 1997.

[28] Pedro Domingos. Exploiting context in feature selection. *Artificial Intelligence Review*, 1997.

[29] M. W. Eysenck. Memory and arousal. In Anthony Gale and John A. Edwards, editors, *Physiological Correlates of Human Behaviour - Attention and Perf ormance*, volume 2, chapter 10. Academic Press Ltd., 1983.

[30] Sadri Fariba. *Temporal Logics and their Applications*, chapter 4, pages 121–168. Academic Press Limited, 1987.

[31] Douglas Fisher. Iterative optimization and simplification of hierarchical clusterings. Technical Report CS-95-01, Vanderbilt University - Nashville, January 1995.

[32] Douglas Fisher and Jeffrey C. Schlimmer. Models of incremental concept learning: a coupled research proposal, February 1997.

[33] Douglas H. Fisher. Knowledge aquisition via incremental conceptual clustering. In Jude W. Shavlik and Thomas G. Diettrich, editors, *Readings in Machine Learning*, pages 267–284. Morgan Kauffman, 1990.

[34] Daniel D. Fu, Kristian J. Hammond, and Michael J. Swain. Vision and navigation in man-made environments: Looking for syrup in all the right places. *IEEE*, 3:20–26, 1994.

[35] Antony Galton. *Temporal Logic and Computer Science: An overview*, chapter 1, pages 2–52. Academic Press Limited, 1987.

[36] Klaus-Peter Gapp. Basic meanings of spatial relations: Computation and evaluation in 3d space. Technical Report 101, Universitat des Saarlandes, April 1994.

[37] John H. Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[38] Hans Werner Geusgen. Spatial reasoning based on allen temporal logic. Technical report, International Computer Science Institute, Berkley, August 1989.

[39] Diana F. Gordon. Queries for bias testing. In *Proceedings of the Workshop on Change of Representation and Problem Reformulation*, 1992.

[40] Diana F. Gordon and Marie des Jardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20:1–17, 1995.

[41] Philip M. Groves and George V. Rebec. *Introduction to Biological Psychology.* Wmc. C. Brown Publishers, College Division, 3rd edition, 1988.

[42] Roger Hale. *Temporal Logic Programming*, chapter 3, pages 92–119. Academic Press Limited, 1987.

[43] Michael Harries and Kim Horn. Learning stable concepts in domains with hidden changes in context. Technical report, University of New South Wales, 1994.

[44] Hasbro-Microprose. Falcon4.0 manual, 1998.

[45] A. Hauptmann and M. Witbrock. Informedia on demand: Using speech recognition to create a digital library, 1997.

[46] Alexander Hauptmann and Michael A. Smith. Text, speech and vision for video segmentation. In *AAAI-95 Fall Symposium on Computational Models for Integrating Language and Vision*, November 1995.

[47] Alexander G. Hauptmann, Michael J. Witbrock, Alexander I. Rudnicky, and Stephen Reed. Speech for multimedia information retrieval. In *UIST-95 Eighth Annual Symposium for User Interface Software and Techonology*, November 1995.

[48] D. P. Helmbold and P. M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14:27–45, 1994.

[49] Gerd Herzog. Utilizing interval-based event representations for incremental high-level scene analysis. Technical Report 91, Universitat des Saarlandes, November 1992.

[50] Gerd Herzog and Karl Rohr. Integrating vision and language: Towards automatic description of human movements. In *Proceedings of the 19th Annual German Conference on Artificial Intelligence (KI-95)*, June 1995.

[51] Gerd Herzog and Peter Wazinski. Visual translator: Linking perceptions and natural language descriptions. *Artificial Intelligence Review*, 8:175–187, 1994.

[52] M. Hoetler. Differential estimation of gloabl motion paramters zoom and pan. *Signal Processing*, 16:249–265, 1989.

[53] B.K.P. Horn and B.G. Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[54] Chung-Lin Huang and Chi-Hou Wu. Dynamic scene analysis using path and shape coherence. *Pattern Recognition*, 25:445–461, 1992.

[55] The CMU Informedia Project. http://www.informedia.cs.cmu.edu.

[56] The CMU Speech Project. http://www.cs.cmu.edu/afs/user /air/WWW/SpeechGroup/Home.html.

[57] Vitra. http://www.dfki.uni-sb.de/vitra/.

[58] Stephen S. Intille. Tracking using a local closed-world assumption: Tracking in the football domain. Master's thesis, Architecture and Planning, August 1994.

[59] Stephen S. Intille. Real-time closed-world tracking. In *International Conference on Computer Vision and Pattern Recognition*, pages 697–703. IEEE Computer Society Press, June 1997.

[60] Stephen S. Intille and Aaron Bobick. Visual tracking using closed-worlds. Technical Report 296, MIT Media Laboratory, Perceptual Computing Section, Massachusetts Institute of Technology, 20 Ames St. Cambridge, November 1994.

[61] Stephen S. Intille and Aaron F. Bobick. Visual tracking using closed worlds. In *International Conference on Computer Vision 1995*, page ??, June 1995.

[62] K. Hanna J. R. Bergen, P. Anandan and J. Hingorani. Hierarchical model based estimation. In *Proceedings of ECCV92*, pages 232–252, 1992.

[63] T. Kanade. Immersion into visual media: New applications of image understanding. *IEEE Expert Intelligent Systems and Their Applications*, 1:73–80, 1996.

[64] K. Daniilidis Koller and H.H. Nagel. Model based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10:257–281, 1993.

[65] Miroslav Kubat. Conceptual inductive learning : the case of unreliable teachers. *Artificial Intelligence*, 52:169–182, 1991.

[66] Miroslav Kubat and Ivana Krizakova. Forgetting and aging of knowledge in concept formation. *Applied Artificial Intelligence*, 6:195–206, 1992.

[67] M. Lazarescu, S. Venkatesh, G. West, and T. Caelli. Classifying and incrementally learning american football plays using natural language and video processing. Technical Report 2, Curtin University of Technology, May 1998.

[68] Mihai Lazarescu, Svetha Venkatesh, and Geoff West. Classifying and learning cricket shots using camera motion. In *Proceedings of AI'99*, Dec 1999.

[69] Mihai Lazarescu, Svetha Venkatesh, and Geoff West. Incremental learning with forgetting (i.l.f.). In *Proceedings of ICML-99 Workshop on Machine Learning in Computer Vision*, June 1999.

[70] Mihai Lazarescu, Svetha Venkatesh, and Geoff West. Learning about dynamic scenes. In *Proceedings of Bad Homeburg Workshop on Computer Vision*, May 1999.

[71] Mihai Lazarescu, Svetha Venkatesh, Geoff West, and Terry Caelli. Combining nl processing and video data to query american football. In *Proceedings of IAPR'98*, pages 1274–1277, August 1998.

[72] Mihai Lazarescu, Svetha Venkatesh, Geoff West, and Terry Caelli. Using natural language and video data to query and learn american football plays. In *Proceedings of ICAPR'98*, pages 63–73, nov 1998.

[73] Mihai Lazarescu, Svetha Venkatesh, Geoff West, and Terry Caelli. Applications of machine learning to dynamic scene analysis. In *Proceedings of DICTA'99*, Dec 1999.

[74] Mihai Lazarescu, Svetha Venkatesh, Geoff West, and Terry Caelli. On the automated interpretation and indexing of american football. In *Proceedings of ICMCS'99*, pages 824–829, June 1999.

[75] David G. Lowe. Robust model-based motion tracking through the integration of search and estimation. Technical Report 92-11, University of British Columbia, May 1992.

[76] Srinivasan M., S. Venkatesh, and R. Hosie. Qualitative estimation of camera motion parameters from video sequences. *Pattern Recognition*, 30:593–606, 1997.

[77] Stan Matwin and Miroslav Kubat. The role of context in concept learning. Technical report, Department of Computer Science, University of Ottawa, 1994.

[78] D.P. McKenzie, R.D. McGorry, C.S. Wallace, L.H. Low, D.L. Copolov, and B. S. Singh. Constructing a minimal diagnostic decision tree. *Methods of Information in Medicine*, 32:161–166, 1993.

[79] Itay Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 85:343–385, 1996.

[80] R. S. Michalski and R. Stepp. Revealing conceptual structure in data by inductive inference. In J. E. Hayes, D. Mitchie, and Y-H Pao, editors, *Machine Intelligence*, volume 10, pages 173–196. 1982.

[81] R.S. Michalski. Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4), 1980.

[82] C. Mitchell. The need for biases in learning generalizations. Technical Report TR-117, Rutgers University, 1980.

[83] Tony Morgan. *The Australian Guide to American Football.* ABC Entreprises, 1991.

[84] H.H. Nagel. On the estimation of optical flow: relations between different approaches and some new results. *Artificial Intelligence*, 33:299–324, 1987.

[85] Joe Namath. *American Football - A Complete Guide to Playing the Game.* Pan Books, London, 1986.

[86] Bernd Neumann. Natural language description of time-varying scenes. In D.L. Waltz, editor, *Semantic Structures: Advances in Natural Language Processing*, pages 167–207. Lawrence Erlbaum, 1989.

[87] Jonathan J. Oliver. Decision graphs - an extension of decision trees, 1992.

[88] J. Park, N. Yagi, K. Enami, and E. Kiyoharu. Estimation of camera parameters from image sequence for model video based coding. *IEEE Transactions on Circutis Systems and Video Technology*, 3(4):288–296, 1994.

[89] L.F. Pau. Context related issues in image understanding. In L.F. Pau C.H. Chen and P.S.P. Wang, editors, *Handbook of Pattern Recognition and Computer Vision*, pages 741–768. World Scientific Publishing Company, 1994.

[90] J. R. Quilan. Semi-autonomous aquisition of pattern-based knowledge. In J. E. Hayes, Donald Michie, and Y-H Pao, editors, *Machine Intelligence*, volume 10, pages 159–173. Ellis Horwood Limited, 1982.

[91] J. R. Quilan. Decision trees and multi-valued attributes. In J. E. Hayes, Donald Michie, and J. Richards, editors, *Machine Intelligence*, pages 305–318. Claredon Press, 1988.

[92] J. R. Quinlan. Decision trees and decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:339–345, April 1990.

[93] J. R. Quinlan. *C4.5 : programs for machine learning.* Morgan Kaufmann Publishers, 1993.

[94] Ashwin Ram. Incremental learning of explanation patterns and their indices. In Porter B. W. and Mooney R., editors, *Machine Learning*, pages 313–321. Morgan Kauffman, 1990.

[95] Ashwin Ram. Indexing, elaboration and refinement : Incremental learning of explanatory cases. *Machine Learning*, 10(3):201–248, 1993.

[96] Yoram Reich. Constructive induction by incremental concept formation. In Y. A. Feldman and A Bruckstein, editors, *Artificial Intelligence and Computer Vision*, pages 191–204. Elsevier Science Publishers, 1991.

[97] R. E. Reinke and R. S. Michalski. Incremental learning of concept descriptions : A method and experimental results. In J. E. Hayes, D. Mitchie, and J. Richards, editors, *Machine Intelligence*, volume 11, pages 263–289. 1988.

[98] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[99] S. Sato and T. Kanade. Name-it: Association of face and name in video. Technical report, Carnegie Mellon University School of Computer Science, December 1996.

[100] Robert J. Schalkoff. *Digital Image Processing and Computer Vision.* John Wiley and Sons, 1989.

[101] Jorg Schirra. Connecting visual and verbal space preliminary considerations concerning the concept 'mental image'. In M. Arnague, A. Borillo, M. Borrillo, and A. Bras, editors, *4th International Workshop on Time, Space, Movement and Spatio-Temporal Reasoning*, pages 105 – 121, Chateau des Bonas, September 1992. Groupe Langue, Raisonnement , Calcul, Toulouse.

[102] Jorg Schirra. *A Contribution to Reference Semantics of Spatial Prepositions: The Visualisation problem and its solution in VITRA*, volume 3 of *Natural Language Processing.* Mounton de Gruyter, 1993.

[103] Jorg Schirra and Eva Stopp. Antlima - a listener model with mental images. In *Proceedings of the 13th International Joint Conference on AI (IJCAI-93)*, pages 175–180, Chambery, August 1993.

[104] Jeffrey C. Schlimmer. Incremental learning from noisy data. *Machine Learning*, pages 317–330, 1986.

[105] Jeffrey C. Schlimmer and Richard H. Granger. Beyond incremental processing : Tracking concept drift. In *Proceedings of AAAI-86 : Fifth National Conference on Artificial Intelligence*, pages 502–507. Springer, 1986.

[106] Ishwar K. Sethi and Ramesh Jain. Finding trajectories of feature points in a monocular image sequence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:56–73, January 1987.

[107] Robert L. Shaw. *Fighter Combat - Tactics and Maneuvering.* Naval Institute Press, Annapolis, Maryland, 1985.

[108] Leon Shklar and Haym Hirsh. Imposing bounds on the number of categories for incremental concept formation. In Russel Greiner, Thomas Petsche, and Stephen Jose Hanson, editors, *Computational Learning Theory and Natural Learning Systems*, volume 4, chapter 3. MIT Press, Combridge, Massachusetts, 1997.

[109] Sierra. Gridiron Legends - NFL 98, 1998.

[110] William M. Spears and Diana F. Gordon. Is consistency harmful ? In *Proceedings of the Workshop on Biases in Inductive Learning*, 1992.

[111] Thomas M. Strat and Martin A. Fischler. The role of context in computer vision. In *Proceedings of the Workshop on Context Based Vision*, pages 2–11, June 1995.

[112] T.M. Strat and M.A. Fischler. Context-based vision: recognizing objects using information from 2d and 3d imagery. *PAMI*, 13:1050–1065, 1991.

[113] Tomas Svoboda and Tomas Pajdla. Efficient motion analysis. Technical Report K335-1995-95, Czech Technical University, October 1995.

[114] G. Tiberghien. Natural and artificial memory systems. In J. Demongeot, T. Herve, V. Rialle, and C. Roche, editors, *Artificial Intelligence and Cognitive Sciences*, chapter 16, pages 221–237. Manchester University Press, 1988.

[115] Luis Torgo. Controlled redundancy in incremental learning. In *Machine Learning : ECML-93*, pages 185–195. Springer, 1993.

[116] Luis Torgo and Miroslav Kubat. Knowledge integration and forgetting. Technical Report, 1991.

[117] Y.T. Tse and R.L. Baker. Global zoompan estimation and compensation for video compression. In *ICASSP'91*, pages 2725–2728, 1991.

[118] P.D. Turney. Exploiting context when learning to classify. In *European Conference on Machine Learning*, Springer-Verlag, pages 402–407, 1993.

[119] P. Utgoff. Shift of bias for inductive concept learning. In . Michalski R. S, J. Carbonell, and T. Mitchell, editors, *Machine Learning II*, pages 107–148. Morgan Kauffman, 1986.

[120] Peter van Beek. Approximation algorithms for temporal reasoning. In *Knowledge Representation*, pages 1291–1296. 1990.

[121] P. M. Warburton. Towards a neurochemical theory of learning and memory. In Anthony Gale and John A. Edwards, editors, *Physiological Correlates of Human Behaviour - Basic Issues*, volume 1, chapter 7. Academic Press Ltd., 1983.

[122] Gerhard Widmer. Recognition and exploitation of contextual clues via incremental learning. Technical report, Department of Medical Cybernetics and Artificial Intelligence, University of Vienna, Schottengasse 3, A-1010 Vienna, Austria, 1994.

[123] Gerhard Widmer. Learning in dynamically changing domains: Recent contributions of machine learning, 1998.

[124] Gerhard Widmer and Miroslav Kubat. Adaptive strategy selection for concept learning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-92), Vienna*, 1992.

[125] Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *European Conference on Machine Learning*, pages 227–243. Springer-Verlag, 1993.

[126] Gerhard Widmer and Miroslav Kubat. Combining robustness and flexibility in learning drifting concepts. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94), Amsterdam*, 1994.

[127] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.

[128] Michael Witbrock, A. Hauptmann, and M. Cristel. News-on-demand - an application of informedia technology. *D-LIB magazine*, 1995.

[129] S.F. Wu and J. Kittler. A differential method for simultaneous estimation of motion change of scale and translation. *Signal Process. Image Comm.*, 2:69–80, 1990.

[130] J. Z. Young. *Programs of the Brain.* Oxford University Press, 1978.