Supporting the Tennis Coach: Automatically Analyzing and Evaluating Tennis Footage

Master's thesis, October 13 2010

Marc Kuijpers

Supporting the Tennis Coach: Automatically Analyzing and Evaluating Tennis Footage

THESIS

submitted in partial fulfilment of
the
requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

Marc Kuijpers
born in Delft, The Netherlands

Web Information Systems Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, The Netherlands
http://eemcs.tudelft.nl

Supporting the Tennis Coach: Automatically Analyzing and Evaluating Tennis Footage

Author:	Marc Kuijpers
Student id:	1178555
Email:	kuijpersmj@gmail.com

Abstract

Video support is becoming an indispensable tool in tennis practice sessions, especially at the professional level. Cameras are used to record a player and the current available software is used optimize the player's tennis technique, i.e. the biomechanics. Unfortunately, the tactical side of tennis is underexposed in terms of available software. Tennis can be seen as a spatial-temporal game. The the dimensions of the court are fixed and the ball goes from player a to player b in a finite amount of time. The work presented in this thesis shows a method to flexibly evaluate a tennis game based on the footage of a single mounted camera. Software is used to extract spatial-temporal data from the tennis footage and a spatial-temporal language based on first order logic is designed to query the spatial-temporal data. The implemented prototype of this thesis' work provides a graphical user interface in which the user is able to execute queries and to see the movie fragments that meet the requirements of the spatial-temporal query.

Keywords: Video analysis, tennis software, spatial-temporal language, SQL query optimization

Acknowedgements

I would like to thank my daily supervisor Jan Hidders for the stimulating discussions and guidance he has offered me throughout the course of this work. I also owe many thanks to Bill Christmas from the University of Surrey who was always available to help me with installing and using the Vampire software. I would also like to thank my fellow roomates and my late friend Jeroen for their interest in my work and their support.

Special thanks are also due to the people of the Royal Dutch Lawn Tennis Federation, Frank van Fraayenhoven, Ruurd Creemers, Peter Starke and Jan-Willem Baars, who were always there to provide me with useful advice.

Finally, and most importantly, I would like to express my gratitude to my family for the support they have shown to me during this long journey.

Marc Kuijpers,
13 October 2010, Delft

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The support of visual aids in sports is still an emerging area of research. Using videos in sports can have numerous advantages. Firstly, an important general feature of video is that it can be watched over and over again. Secondly, motions in sports can occur too quick for the human eye to track. The human eye is capable of accurately tracking an object that moves with a speed of 70 (rad/s). Higher radial velocities cause blur [19]. Using high-speed cameras or simply replay the video in slow-motion can take away this human shortcoming. When using a multiple camera setting one is able to view a sport scene from different angles and a sports coach is also able to keep track of multiple matches simultaneously with the use of cameras.

The work presented in this thesis tries to further exploit the use of video cameras in tennis practice sessions. The objective is to support and extend the tactical insights of a tennis game for tennis players by means of a single video camera and additional video analysis software.

The introduction is divided in two sections. In section 1.1 tennis background information is provided in order to expose the possibilities to use video analysis. Also the current methods of video analysis will be briefly discussed as well as the comparison to other sports in which video analysis is already used. Section 1.2 and 1.3 introduce respectively the problem description and the objective of this work.

1.1 Background

This work will purely be concerned with the use of visual aids in tennis. Tennis can be seen as a spatial-temporal game. Two players battle against each other in which the ball travels from player one to player two in a limited amount of time and space. Professional tennis is based on four pillars: technical, tactical, physical and mental. The technical pillar is concerned with the biomechanical characteristics of tennis players. Optimizing the movement of body joints, muscles and other tissues to hit as effectively as possible is the core subject of biomechanics. The tactical pillar looks at the spatial-temporal characteristic of the game. In general this comes down to the question: where should player one hit the ball in order to make it player two as hard as possible. This question is not as trivial as it seems since it depends on multiple apsects, e.g.: player position, incoming ball, position relative to the court, etc. The physical pillar tries to get the body in shape to avoid injuries and to enable the body to perform the biomechanically most optimal movements throughout a tennis match. The last pillar, mental, looks at the positive and negative emotions that every tennis player faces. Analyzing physical and mental aspects by means of video analysis is not probable and are therefore beyond the scope of this work.

The video analysis software currently available for tennis coaches focuses mainly on the technical domain. The software enables the coach to record a player and to analyze the player's movement. An example is shown in Figure 1.1 where the trajectory of the player's racket is highlighted after a serve. Another package frequently used by tennis coaches is called TimeWarp[1]. TimeWarp records a player and replays it with an adjustable delay. The goal of this software is to provide instant feedback to a player after a number of hits. The software is intended for improving the player's tennis technique.

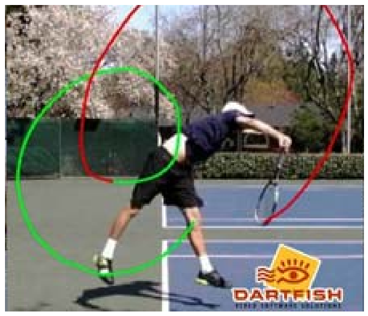


Figure 1.1: Analysis of the serve using biomechanical analysis software

In tennis, compared to other ball-oriented sports, there is a lack of tactical analysis software. In professional soccer the use of tactical software is more and more embedded in practice sessions. The goal of the software is to analyze the performance of the team with respect to tactical situations and to find weaknesses in the tactics of the opponent. Hockey is another example of a sport in which tactical analysis software has become an indispensable tool for the coach.

[1]TimeWarp is a product of SiliconCoach. For more information `http://www.siliconcoach.com`

1.2 Problem Description

The lack of dedicated tactical software is a shortcoming for the tennis coach. For the application to be successful, the following requirements should be met:

- **Portability**
  The system should be able to be used at different locations easily. The problem encountered here is the number of cameras. To increase portability this number should be kept as low as possible. Furthermore, the software needs to be flexible in terms of positioning the camera(s) behind the tennis court. Footage originating from one or multiple camera(s) behind the court, approximately in the center, should be sufficient for the software to work with.

- **Flexibility**
  Given a tennis video, the user of the system needs to be able to specify exactly what he wants to see. A set of predefined functions should be available in the system to extract the most common tennis events from the video. On top of this set, the user should be able to easily indicate all the tactical situations that he wants to extract from the video.

- **User-friendly** In order for the system to be used by tennis coaches, the learning curve of the system should be as steep as possible and the number of actions to be taken to produce the desired output should be kept as low as possible. The system should therefore contain an user interface that is intuitive, easy to understand and fast.

1.3 Objective of This Thesis

The objective will be to design and develop a prototype that is able to, given a tennis video, automatically analyze and evaluate tactical tennis situation in a portable, flexible and user-friendly way. The research questions that should be answered are:

1. How to gather and save spatial-temporal data from tennis footage and what is the optimal number of cameras taking the ratio portability/accuracy into account?

2. How to give the user the flexibility to query the collected data in a quick and easy way?

3. What requirements should the user interface meet in order to make the input of the coach as easy as possible and at the same time produce useful output?

Chapter 2 discusses research that is related this work. In this chapter the latest research on tennis video analysis software is discussed. Linear Temporal Logic will also be elaborated on since it is an acknowledged method to query temporal data. Chapter 3 thoroughly elaborates on the prototype. This chapter is divided in three sections: the ball and player tracking software, the spatial-temporal language and the user interface. In the next chapter the work is evaluated and the thesis is ended with a chapter Discussion and Future Work.

Chapter 2

Related Work

In this section research literature related to automated tennis video analysis and temporal logic will be briefly presented. Objective of this chapter, next to positioning this thesis in the research field, is to show the alternative options related to tennis video analysis software and query languages like temporal logic. The first section will discuss current tennis video analysis software. As mentioned in the introduction the emphasis of this work is not to come up with new video analysis algorithms to optimize the tracking of balls and players during a tennis match. Merely, the emphasis of this work is to build a layer on top of this types of software to create software that can be used by professional tennis coaches to analyze tactical situations. The tennis video analysis (TVA) software used for building the prototype is called "Vampire" and is able to track the ball and both players during a tennis match. However, TVA literature will be discussed to see the benefits of Vampire compared to alternative TVA software.
The next section discusses temporal logic since it is the most common method to query spatial-temporal data. In this section temporal logic is briefly explained and Linear Temporal Logic (LTL) is compared to first-order logic in order to see the differences and the similarities.

2.1 Tennis Video Analysis

For the TVA software to be usable it should output what, when and at which location an event has occured. Where an event can either be an hit or a bounce event. Other requirements that must be met by the TVA software are:

- Track the location of players based on single-camera footage
- Track the location of the tennis ball based on single-camera footage
- Generate 2-dimensional spatial-temporal data from the above data sources

The TVA literature available can roughly be divided in three categories. Firstly, the software that examines broadcast videos. The emphasis is mainly on distinguishing between camera shots (e.g. close-up, replay, game play, etc.) and trying to extract the most fundamental events. Secondly, the software that examines tennis videos and tries to track both players. This software goes deeper and tries to detect more specific events. For example: detecting when a serve is hit or when a rally is played. The third category is concerned with player and ball tracking. This is computationally the most heavy type of software and is concerned with tracking both the players and the ball as it goes from location a to location b. For this thesis only the last category is interesting, but for comparison also the second category, player tracking, will be discussed.

2.1.1 Player Tracking Methods

Huang et al. [5] examined broadcast videos. Broadcast tennis videos always use multiple cameras, but the camera position that records the game play is always fixed. The difficulty of this type of footage is to distinguish the various types of camera shots: game play, close-ups, replays, audiance shots, etc. By using a combination of video and audio they are able to detect when a serve is hit and when a rally is played with accuracies between the 84.9% and 94.1%.

Another player tracking algorithm based on broadcast video is developed by Dang et al. [4]. They are able to track players and to detect the lines of the court with an accuracy of respectively 97% and 87%. Their prototype is able to detect serves, baseline rallies and net approaches. The extraction of this higher semantic information is based on:

- Instant speed of the player to detect whether a player is still or running.
- Speed change of the player. Changes in action behavior occur simultaneously with acceleration and deceleration of a player.
- Relative position of the player on the court model. This is the main source for the recognition of position-based events (i.e. serve, baseline rally, approach). See Figure 2.1.
- Temporal relations among each event. In tennis there are strong temporal correlations between events, e.g. a serve always happen before a rally starts and net approaches can occur during a baseline rally.

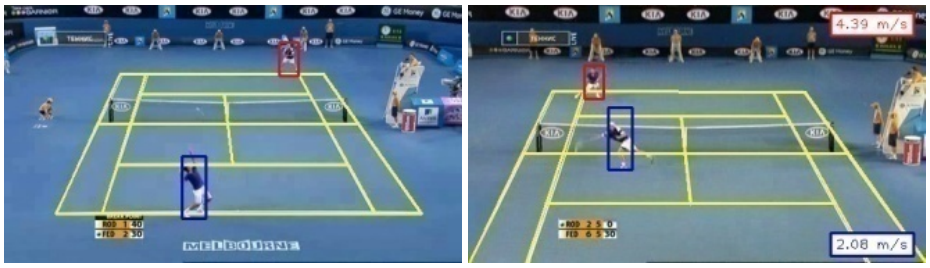


Figure 2.1: The detection of respectively a serve and a net approach based on the relative positions of both players

This prototype recognizes global events, but is not capable of tracking ball trajectories and detecting hit/bounce events.

The software of Tien et al. [21] characterizes events based on audiovisual features and hence are able to detect more specific events. The five events it recognizes are: fault, double fault, ace/unreturned serve, baseline rally and net approach. Next to the relative position of the player on the court, this software uses more audiovisual feature extractions to:

- Detect player movement. This is similar to the feature proposed in [5], except that this software doesn't detect speed changes. Looking at Figure 2.2, the bottom player moves from one region into another and hence can be categorized as a net approach.
- Applause/cheer sound effects. The use of audio features can help to detect events, since audience tends to give applauses after an ace or a baseline rally, but not when a double fault has been hit.
- Length of the play. Different events have different lengths. They take the length of an event in account to, for example, distinguish an ace from a baseline rally.

The use of audio to distinguish among events is for the application of tactical software not practical since the software should also perform accurately without an audience.

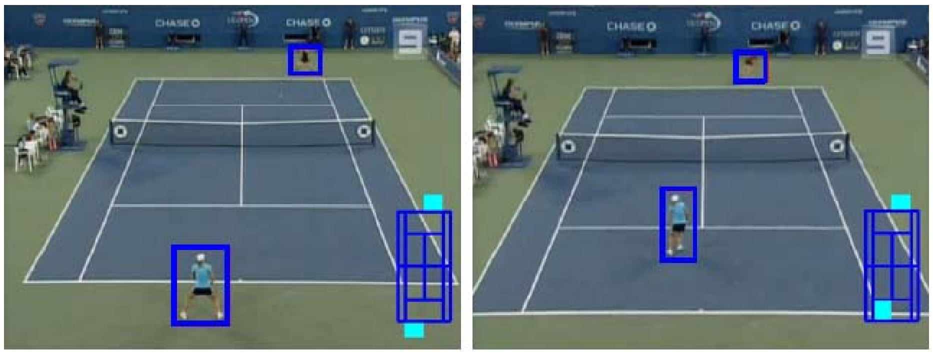


Figure 2.2: Player movement detection. The mapping to the homography is depicted in the right bottom corner.

Shaeib et al. [20, 22] propose a methodology called TennisSense for automated processing of wireless sensor data[1]. The system requires eight video cameras and six sensors around the court, one camera above the court and one sensor attached to the body of the player. The obtained data, which is structured in XML files, can be requested using a standard query language. The benefit of this approach is that queryable statistics can be provided to coaches immediately after matches so that they can evaluate the sensed data and modify the behaviour of their athletes. The location of the players on the court is recorded through sensors and the tennis court is mapped into 24 zones, each a range of (x,y) coordinates corresponding to a section of the court. The court location can be exploited to categorize the construction of generic queries and the detection of similar patterns of play. The court mapping is a necessary step to allow basic queries in XPath and XQuery following enrichment.
According to their test results the Ubisense network is computationally less expensive compared to approaches using video. The downside of the system is the level of portability since a total of ten cameras and seven sensors is required. Also, TennisSense is not able to track the ball. Attaching a sensor at the outside of the ball is not possible and implementing a sensor within a ball goes at the cost of portability. Benefit of the system is the possible extension to the third dimension (z-coordinate, height) in which it can track players.

Miyamori et al. [16] propose a method to detect different actions. They distinguish between three different actions: a stroke (forehand or backhand), the volley and the serve. They have developed an appearance-based annotation system based on silhouette transitions. Extracting silhouettes from a tennis video and comparing it to a set of existing silhouettes (see Figure 2.3) can help to categorize tennis actions.

[1]The sensor technique is provided by UbiSense (http://www.cdvp.dcu.ie/tennisireland/ (July -2010))

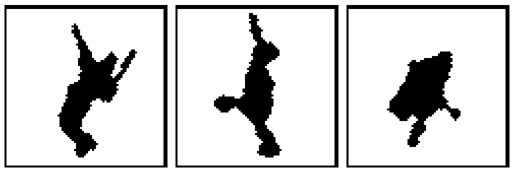


Figure 2.3: Typical silhouettes of an over-the-shoulder swing

The results of this method vary strongly among the different tennis actions. The retrieval result of stroke-detection, e.g. recognizing forehand or backhand, has an error rate of 5% when the player is at the bottom-side of the screen and an error rate of 26.4% when the player is at the top-side of the screen. This discrepancy is due to occlusion that comes into play when single-camera video is analyzed. Occlusion is the effect that objects tend to shrink when they are further away in the screen due to perspective. In far-view frames, a player-object can shrink to 30 pixels tall [26]. Separating objects from their environment gets more difficult when the occlusion increases.
By adding domain-based knowledge to the system, the succes rate increases. For example, given the "over-the-shoulder-swing" both the smash and the serve qualify. The distinction can be made by looking at the court position of the player. When positioned close to the net the player hits a smash. When positioned around the baseline, the player is more likely to hit a serve. Miyamori et al. improved their original method by integrating audio features and combining player and ball positions [15]. Another detection and tracking algorithm that focuses on complete player extraction is proposed by Jiang et al. [7] and shows similar results.

2.1.2 Player and Ball Tracking Methods

Attempts to succesfully track a tennis ball have been performed by only a few research groups [17, 25, 16, 11]. The most well-known method nowadays is called "Hawk-Eye". It uses at least four high-speed cameras mounted around the court and has a deviation of only 4 milimeters [14]. The attempt of Yu et al. [25] tries to track the trajectory of the ball. It makes use of the location relation between the players and the ball to improve the ball candidate quality as the hitting player must be near to the ball in tennis. Secondly, to detect start and end points of the ball trajectory it tries to determine hit events. To further improve their results, they added domain knowledge to their application. The success rate of the software sticks around the 96.6%. Disadvantage of the method is that the software relies on multiple cameras.

Most existing tennis ball tracking algorithms for single-camera sequences focus on the estimation problem rather than the data association problem [23]. Yan et al. [18] propose a three-layered data association method to track a tennis ball with a single-camera setup. The first layer is responsible for searching ball candidates in the window. The second layer is concerned with constructing a directed and weighted graph in which a shortest path is created that correspond to the ball trajectory. The third layer deals with filtering non-relevant ball trajectories based on the assumption that there is only one ball to track and that the first and last tracklets of this ball are known. The eventual result is shown in Figure 2.4.

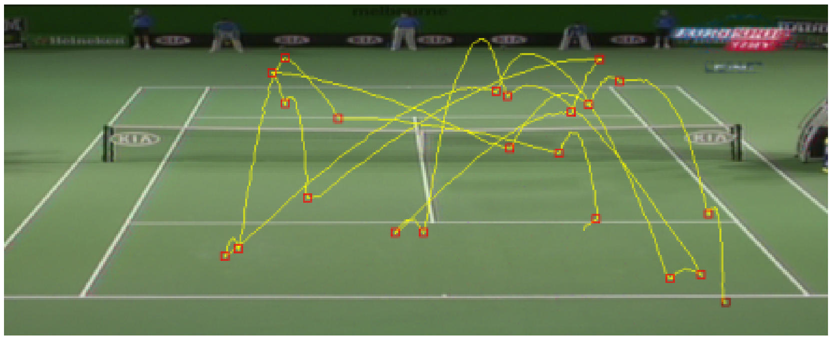


Figure 2.4: Eventual ball tracking result based on a three-layered data association method

The results of the algorithm depend on the surface of the court. In Figure 2.5 two different court surfaces are depicted. The success rate of the algorithm executed on the left surface differs between the 90.6% and 91.3%. The succes rate drops to 83.2% when the algorithm is exectued on the surface depicted in the right image. This ball tracking method is the foundation for the Vampire system.

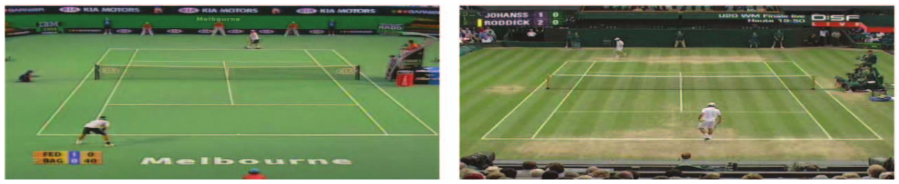


Figure 2.5: Different court surfaces turn out to have different results in the layered data association method. Left image is the Australian Open court (artificial, hard-court), the right image is Wimbledon (grass).

2.1.3 Conclusion

In this section several methods to analyze tennis videos were discussed. The most lightweighted methods are concerned with player tracking only. The success rates are high and most of the methods can be executed in real-time. Most of the methods examine broadcast videos in which the first task is to filter the various camera shots prior to examine the court and the players. A problem

that pops up is occlusion. Player objects in the top half of the screen can become too small to extract them from their environment. This gives problems with tracking these player objects and results in a drop of the success rate.
The other methods discussed in this section are concerned with tracking both the players and the ball. These methods are, with respect to computation power, more heavy. For the methods discussed applies: the more cameras that are used, the more accurate the ball trajectory can be determined. The most accurate method known today is "Hawk-Eye" and uses at least four high-speed cameras to track the ball trajectory.

Both methods use domain specific knowledge to increase their success rates. Examples are: a serve is always hit by a player standing at the baseline, a rally can't start before a serve has been hit and a net approach can be derived by looking at the position of both players. For the application of a tactical tennis tool there is no need for heavy real-time processing software. Single-camera footage has troubles with accuracy both in tracking players and the ball. However, smart filtering techniques can, to a certain extent, overcome this problem and hence will not lead to severe performance issues. The Vampire software turns out to be the most suitable since it generates spatial-temporal data, it can cope with single-camera footage and has an acceptable success rate with a sufficient accuracy. In the next chapter Vampire is elaborated on more extensively.

2.2 Temporal Logic

The term temporal logic is used in the area of formal logic, to describe systems for representing and reasoning about propositions and predicates whose truth depends on time [3]. A temporal logic basically takes a classical propositional or predicate logic and extends it with temporal quantifiers. Classical logic deals with timeless propositions, where temporal logic typically contain some reference to time [13]. The strength of temporal logic can be explained with an example[2]. Take the statement "I am hungry". This statement can be expressed in first-order logic and can either be true or false, but can't take both truth values simultaneously. The strength of temporal logic comes into play when one wants to add temporal constraints. Temporal logic makes it possible to make statements like: "I am *always* hungry", "*Eventually* I will be hungry" and "I am hungy, *until* I eat something". An important application of temporal logic is formal verification, where requirements of hardware of software systems can be stated. A well-known formal verification method is called linear temporal logic and will be discussed next.

2.2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) is a modal logic over a linear frame and was first proposed by Amir Pnueli in 1977. The initial goal was to verify computer

[2]The example originates from `http://en.wikipedia.org/wiki/Temporal_logic`

programs [12]. A model of LTL is an infinite sequence of states where each point in time has a unique successor. The alphabet of LTL is composed of:

- atomic proposition symbols p, q, r, ...
- boolean connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\iff$
- temporal connectives $\circ$, $\Box$, $\diamond$, $\mathcal{U}$, $\mathcal{R}$

LTL knows two main types of properties: safety properties (*something bad nevers happens*) and liveness (*something good eventually happens*). Safety properties can be used to avoid deadlock or unreachable states in a system. With liveness properties one is able to check whether there is still progress in the system [3].
The atomic proposition symbols and the boolean connectives represent the same symbols and meaning of the equivalent first-order logic symbols. The semantics of the temporal connectives are commented in Table 2.1

$\circ$	$\circ\ \phi$ holds, if ϕ holds at the next state on the path
$\Box$	$\Box\ \phi$ holds, if ϕ eventually occurs, i.e., ϕ holds at some state on the path
$\diamond$	$\diamond\ \phi$ holds, if ϕ holds globally, i.e., at every state along the path
$\mathcal{U}$	The formula $\phi\, \mathcal{U}\ \psi$ holds, if ψ holds until ϕ occurs, i.e., there is a state on the path at which ϕ holds, and at every state before ψ holds
$\mathcal{R}$	The formula $\phi\ \mathcal{R}\ \psi$ holds, if, whenever $\neg\ \psi$ occurs at a state on the path, A occurs before. Or equivalently, either ψ holds globally on the path, or ϕ occurs before the first state at which ψ is violated.

Table 2.1: Temporal connectives

The formulas of LTL are true or false on computation paths, that is, sequences of states s_0, s_1, ... as shown in Table 2.2.

$\circ\ \phi$	$\bigcirc \longrightarrow \textcircled{\phi} \longrightarrow \ldots$
$\Box\ \phi$	$\textcircled{\phi} \longrightarrow \textcircled{\phi} \longrightarrow \ldots \longrightarrow \textcircled{\phi} \longrightarrow \textcircled{\phi} \longrightarrow \textcircled{\phi} \longrightarrow \ldots$
$\diamond\ \phi$	$\bigcirc \longrightarrow \bigcirc \longrightarrow \ldots \longrightarrow \bigcirc \longrightarrow \textcircled{\phi} \longrightarrow \bigcirc \longrightarrow \ldots$
$\phi\ \mathcal{U}\ \psi$	$\textcircled{\phi} \longrightarrow \textcircled{\phi} \longrightarrow \ldots \longrightarrow \textcircled{\phi} \longrightarrow \textcircled{\psi} \longrightarrow \bigcirc \longrightarrow \ldots$
$\phi\ \mathcal{R}\ \psi$	$\textcircled{\psi} \longrightarrow \textcircled{\psi} \longrightarrow \ldots \longrightarrow \textcircled{\psi} \longrightarrow \textcircled{\psi} \longrightarrow \textcircled{\psi} \longrightarrow \ldots$ or
	$\textcircled{\psi} \longrightarrow \textcircled{\psi} \longrightarrow \ldots \longrightarrow \textcircled{\psi} \longrightarrow \textcircled{\phi\psi} \longrightarrow \bigcirc \longrightarrow \ldots$

Table 2.2: Formulas with temporal connectives depicted in computation paths

2.2.2 Querying Temporal Data: First-Order Logic vs. LTL

There are two different approaches to query a temporal database using a first-order language [1]:

1. Using temporal logic. As discussed standard temporal logic is an extension of classical logic with the temporal operators *since*, *until*, *next* and *previous*. A form of temporal logic is LTL discussed in section 2.2.1.

2. Using first-order logic or predicate logic, augmenting the database schema with a "timestamp" column. This way one is able to query the records using relational calculus with variables ranging either over data elements or over timestamps.

First-order logic deals with temporal data by using its two quantifiers: the universal ($\forall$) and the existential ($\exists$). LTL and first-order logic can shown to be equivalent over the relation FO[$<$] [10].

2.2.3 Conclusion

This section has briefly outlined the concept of temporal logic. Advantage of LTL is the flexibility and expressivity it offers. However, the syntax of LTL is more complex compared to the syntax of predicate logic. Using predicate logic to reason about temporal data results makes the mapping to a SQL query easier and the user won't be limited in expressing his queries since the equivalence relation FO[$<$] that exists between predicate logic and LTL.

Chapter 3

Technical Architecture

The development and implementation of the prototype has been subject to several design decisions during the process. This chapter elaborates on those design decisions and tries to justify them against design alternatives.
Firstly, a global overview of the technical architecture is described. The system has several steps that have to be walked through consecutively in order to analyze and evaluate tennis footage. For that reason the system can also be seen as a pipeline with a fixed number of stages. This pipeline is disucssed in section 3.1. Each next section discusses one stage of the pipeline. In the second section the TVA software, Vampire, is comprehensively discussed. The operation and the functionalities of Vampire, in terms of video analysis, data gathering and data processing, is explained. In section 3.3 the developed spatial-temporal language to query the data gathered by Vampire is proposed. This includes a thorough description of the grammar, the way the grammar is parsed and how the language fetches the spatial-temporal data. Section 3.4 is concerned with elaborating on the graphical user interface. The decisions made to come up with a user-friendly and usable interface are proposed here.

3.1 The Pipeline

The graphical overview of the several stages of the technical architecture is depicted in Figure 3.1. Every step in the pipeline is included within a Graphical User Interface (GUI). There are two paths in the pipeline. Looking at the figure, the top path is responsible for the tennis video analysis. A tennis video is inserted in the system. It is analyzed by the Vampire software resulting in a database filled with spatial-temporal data. The bottom path in the figure makes it possible for the user to define, with a definition expressed in space and time, what fragments he wants to extract from the video. The first step in this bottom path is to define a definition. This definition is then analysed and parsed in order to convert the definition to a SQL query. The eventual query is exectued on the spatial-temporal database, where the top and bottom path come together, and the returned resultset contains the points in time that meet the requirements of the definition. The moments in time that are returned by the database are extracted from the tennis video and are offered to the user.

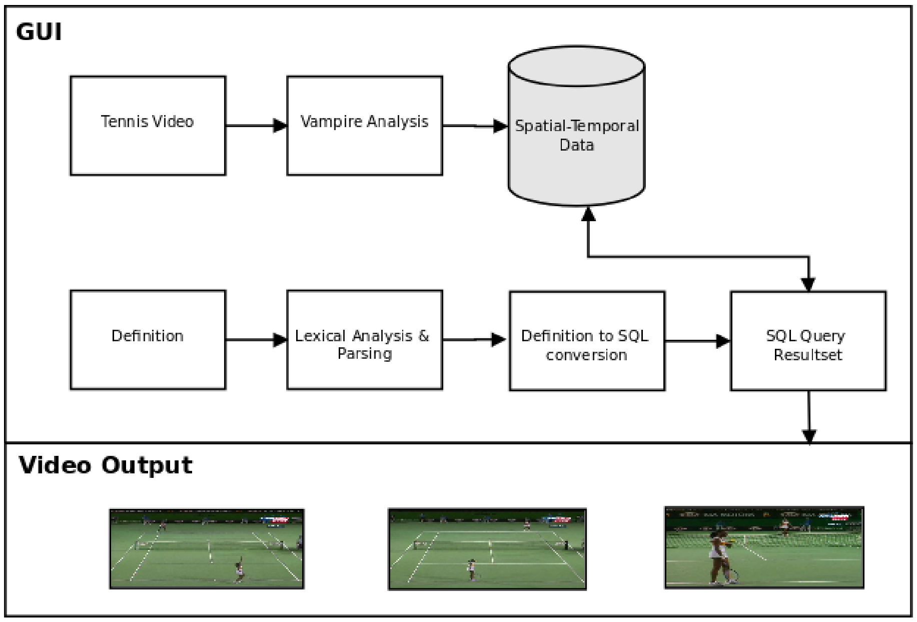


Figure 3.1: Graphical overview of the pipeline

3.2 Vampire: Visual Active Memory Processes Interactive Retrieval

Visual Active Memory Processes and Interactive REtrieval, or 'Vampire', is the software used by the protoype for the first analysis of a tennis video. The Vampire system is intended for use with low-quality, off-air video from a single camera. It is able to, based on the ability to track both the players and the ball, detect and annotate events in a tennis match. The events that can be detected are, among others: serve, hit, bounce and the end of a point [9].
In order to succesfully detect events Vampire divides its system up into three parts. The first part, consisting of the low level modules, is responsible for deinterlacing the video and filtering the coarse scenes of the video to eventually filter video into "non-play" and the useable "play" scenes. The second part is responsible for both the ball tracking and the player tracking simultaneously and the third part, which is comprised of the high-level modules, is concerned with combining the results of the ball and player tracking to generate an analysis of each play shot.
This section explains the working of the higher-level modules of Vampire; how

it is able to track the objects in a video, how the events of the video are distinguished and the way data is outputted by Vampire and how it is used in this prototype.

3.2.1 Player Tracking

For player tracking, the players and the ball are the objects *closest to the court* during play. Extracting the background, i.e. the court, from the foreground, players and ball, the candidate player locations can quite exactly be determined since players appear as objects of reasonable size (see Figure 3.2). Vampire combines this candidate information with temporal tracking methods, applying standard techniques such as particle filters, to more efficiently track player objects.

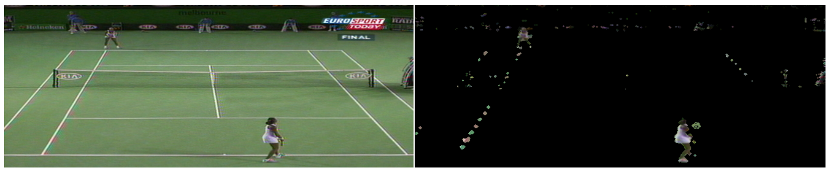


Figure 3.2: Left image: original game-play image. Right image: background extracted from the game-play leaving the player objects behind.

3.2.2 Ball Tracking

The ball tracking process in particular is a challenging task under standard single-camera conditions: the movement of the tennis ball is so fast that sometimes it is blurred into the background, and is also subject to temporary occlusion, sudden change of motion direction and distortion by other objects [8]. Three example frames are shown in Figure 3.3. In the left figure, the space between the ball and the player's head is about five pixels and the color of the ball is strongly affected by the color of the background. The center figure shows an example of the effects of high ball velocities. The ball is almost completely blurred into the background. In the right figure is shown how a wristband can look very similar to a ball, and can form also a plausible trajectory as the player hits the ball.

For ball tracking, the ball appears as a single, very small region of a certain color. From the frame differences, one can extract blobs that have the appearance expected of a tennis ball in such a context - that is, a very small, elliptical blob of relatively bright color. These are considered as ball candidates. However, it is clear that not all of these candidates indeed correspond to the ball; some blobs generated by image noise or actual patches within the field difference image that are visually similar to the ball be misleading when trying to detect where the ball is in each field [9]. Assembling a feature vector containing the properties of all objects that can be seen as 'ball' at a given frame support the tracking process. Since the ball has an expected flight path, the real ball can be extracted from all the ball candidates. The extensive details of the ball tracking process are beyond the scope of this work, but are comprehensively discussed in [24].

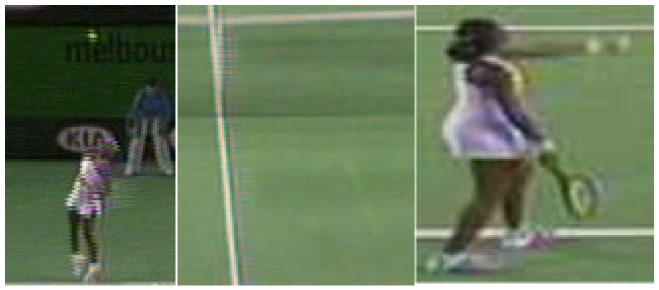


Figure 3.3: Left image: far player serving. Center image: fast moving tennis ball. Right image: Wristband that could easily be confused with a tennis ball.

3.2.3 Event Detection

Based on the tracking results of both the players and the ball-trajectory, Vampire tries to distinguish between events. An important factor in detecting events is an abrupt change in the observed scene. An example is a player who suddenly changes his position. Recognizing interactions between two objects enables one to define which of the scene objects is responsible for a specific event and track them accordingly. For example, when a players hits, the ball changes its direction.

Vampire uses its high-level modules to analyze the data orginating from the ball and player tracking process. The first step is to detect ball events. The ball events that can be distinguished are: tennis ball being hit, bouncing on the court or a collision with the net. Whenever a point on the ball trajectory exceeds specific thresholds in both orientation and motion magnitude, the point is marked as an event. Higher level modules take care of the events in the ball trajectory that are not recognized.

Detecting the player's serves is left to the 'serveDetection' module. To detect a serve, three operations on each player are performed. Firstly, check if one of the players is located in a possible serving position. If a potential serve, based on player position, is detected create a contour of each player. Secondly, check if the body pose of one of the player matches the body pose one would normally associate with a serve hit (see Figure 2.3). The last step in the serve detection is to verify that the tennis ball is directly above the serving player. The serve detection process terminates if, for any of the players all of the above hold. The two parameters that need to be recorded are which player served and from where.

The collected data is reinterpreted in 3D space to further increase the quality of the event detection. The module '3DCues' performs several action:

- All the events that are detected by the ball tracking module are interpreted again. The position of the player and the location of the ball are compared. If a player is close to the ball and the ball is located where a hit can reasonably be made, the event is labelled as a hit. Otherwise it is labelled as a bounce.
- Projecting the court coordinates of the players and the ball on the court surface model. A ball is hit when the ball is approximately near the player's feet.
- No event labels are permitted within 3 frames from each other, to avoid multiple instances of the same event in close succession.

To fully annotate a tennis match one extra module is required. The module looks at tennis specific rules in order to track the evolution of a tennis match. This module, called 'highLevel', uses a model to award a point in a tennis match. In the model are possible paths defined that specify the sequence of a point. Every path starts with a serve and ends with granting the point to one of both players. This module also has to take into account that the system deals with ambiguous and noisy data. Encountering and coping with event detection errors, next to false input events, is also the responsibility of this module. The solution that is used is based on a Hidden Markov Model (HMM) with a look-ahead decision mechanism. This algorithm enables the module to take events into account that occur after the current one. The length of the look-ahead window is set to one event resulting in the correction of only isolated errors. Making the window too large can result in the wrong interpretation of short shots. Another reason to keep the window bound to one event is the fact that ball bounces are the most susceptible to errors. Since a ball bounce can only occur between two successive hits (if the ball in still in play), detecting a player hit automatically means the point can not be awarded yet even if the bounce point is wrongly detected or not detected at all.

3.2.4 Data Handling

Vampire uses XML log files to collect and analyze the video data. In order to gather coordinates of the events the video images need to be transformed to two-dimensional coordinates by means of an homography. In a homograpy the real projective plane is transformed to the projective plane and straight lines are mapped to straight lines. Since the geometry of a tennis court is standardized a model court can be used to detect a court in a given scene. Figure 3.4 and 3.5 show the results of the transformation process.

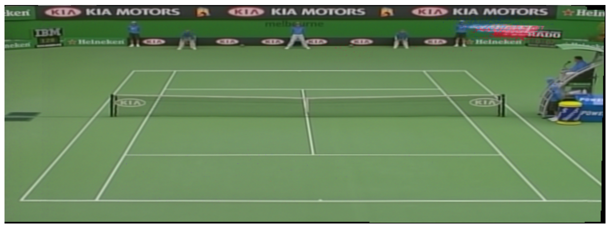


Figure 3.4: Example video image

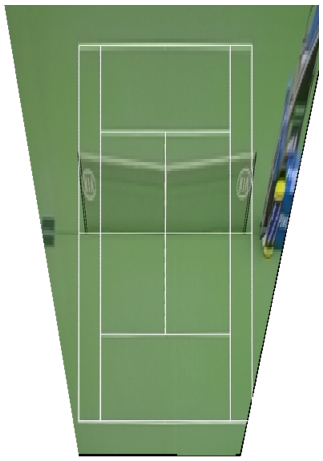


Figure 3.5: Resulting projection after matching the observed court lines to the model court ones

XML Output

Given the possibility to collect coordinates in the projective plane, Vampire is able to write the spatial-temporal data and the associated events to log files. For this thesis' work, the data collected in the 3DCues module is interesting. The log file uses abbreviations for labeling the different events. In the abbreviations every letter contains a chunk of information. The letters that are used are: S (serve), H (hit), B (bounce), I (in), O (out), N (near) , F (far), L (left), R (right). By means of these letters one can annotate the events of a tennis match. A possible seqeunce of letters describing a rally is:

- SNR: *Serve Near Right* - A serve is hit at the bottom half of the screen at the right side of the center.
- BIFSL: *Bounce In Far Serve Left* - The ball bounces in the serve box in the top half of the screen.

- HF: *Hit Far* - Top player hits the ball, BN *Bounce Near*, HN *Hit Near*, etc.
- NET (optional) - The ball hits the net.
- TimeOut

One may notice in the items above the TimeOut at the end of the sequence. Vampire denotes the end of a rally with the TimeOut label. Also the NET label corresponds to the end of a rally. In the XML log file the location of the event is also recorded. The location is recorded by means of coordinates in the two dimensional plane. The top left corner of the model court is (50,100) and the right bottom corner is located at (316,518). One pixel on the model court corresponds to two inch distance on the real tennis court. In Figure 3.6 a part of the 3DCues XML log file is printed in which a possible sequence of events is displayed.

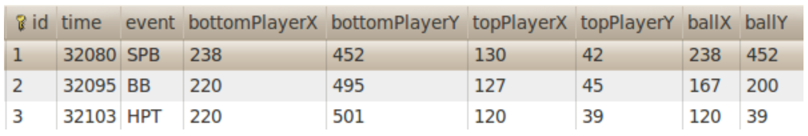
```
<item>
<time beg="169" end="169" framerate="50" start="00:00:03:19">
<data topPlayer="21 143" event="SNR" bottomPlayer="500 247">
</item>
<item>
<time beg="190" end="190" framerate="50" start="00:00:03:40">
<data topPlayer="6 142" ball="191 171" event="BIFSL"
bottomPlayer="510 245>
</item>
<item>
<time beg="209" end="209" framerate="50" start="00:00:04:09">
<data multiInterpretation="0 1 HF 0.9 BOF 0.1" topPlayer="12
143" ball="12 125" event="HF" bottomPlayer="514 238">
</item>
appendix
```

Figure 3.6: Example of the 3DCues XML log file

Looking closely at the XML log file, one may notice the extra elements and attribute-value pairs. Every event starts with an element <item>. The element <time> contains the temporal information. The attributes "beg" and "end" denote the framenumbers in which the event occurs. The attribute "framerate" denotes the frames per second of the movie and the "start"-attribute gives a representation of time in hundreds of a second. The spatial information is stored in the next element. This element contains the coordinates of both the players and the ball. The first number defines the y -coordinate and the second number represents the x coordinate. The first three elements of this log file represent the beginning of a rally in which the bottom player hits a serve from the right side (deuce side). The ball bounces in the left far serve box and then the notation of the third event contains something interesting. The temporal information in this element is the same as the two above. Something divergent occurs at the event recognition. The 3DCues module is not totally sure what kind of event has occurred. The solution here is to add a multiInterpretation attribute in which possible events are recorded with their corresponding chance. In this example it is far more likely to assume that the event HF has occurred since it is awarded with a 0.9 chance compared to the 0.1 of the BOF event.

Data conversion

To quickly search through the spatial-temporal data stored in the XML file, the data is converted and stored into a relational database. A tennis match generates vast amounts of spatial-temporal data and the bigger the amount of data becomes the more a relational database becomes a suitable solution.
By means of a XML parser and a small script the full XML file is read and converted to SQL `INSERT` queries. For the purpose of this work the following events are distinguished: SPB (*Serve Player Bottom*), SPT (*Serve Player Top*), HPB (*Hit Player Bottom*), HPT (*Hit Player Top*), BB (*Ball Bounce*). Every record represents one event. An example of a XML file converted to a relational database is shown in Figure 3.7.



id	time	event	bottomPlayerX	bottomPlayerY	topPlayerX	topPlayerY	ballX	ballY
1	32080	SPB	238	452	130	42	238	452
2	32095	BB	220	495	127	45	167	200
3	32103	HPT	220	501	120	39	120	39

Figure 3.7: XML log file converted to relational database records

MySQL is chosen for the implementation of the RDBMS. In the database dump the column "id" is the primary key. Since only one event can occur at a time, the time column could also fulfill as the primary key index. When building a tactical tennis application one needs to know where a player is located at any time. Every record in the table contains the location of the players and the ball even if the player is not involved in the event, i.e. it is important to know where the top player is located at the moment the bottom player hits the ball.

3.3 Spatial Temporal Language

In order to derive useful tactical information from the spatial-temporal data a query language need to be constructed. Obtaining high-level information from a database containing spatial-temporal data can eventually result in large SQL queries. Next to the large queries, you can not expect a tennis coach to learn a complex language like SQL first before he can use the software. For those two reasons a language is developed in which the tennis coach can express what he wants to see. The first subsection elaborates extensively on the developed language and its associated context free grammar. The next subsections explain how a relatively simple definition defined in this language can be transformed to a complex SQL query.

3.3.1 The Context Free Grammar (CFG)

A context free grammar (CFG) is a quadruple $< \mathcal{S}, \mathcal{N}, \mathcal{A}, \mathcal{R} >$, where $\mathcal{S}$ is the start symbol, $\mathcal{N}$ is the set of non-terminals, $\mathcal{A}$ the alphabet and $\mathcal{R}$ the set of rules. A CFG defines a formal language, i.e. the set of all sentences (strings of words) that can be derived by the grammar. The CFG should define a first order logic programming language that can derive information about a set of the quintuples $< \mathcal{F}, \mathcal{E}, \mathcal{L}_{PlayerBottom}(\text{x,y}), \mathcal{L}_{PlayerTop}(\text{x,y}), \mathcal{L}_{Ball}(\text{x,y}) >$, where $\mathcal{F}$ is the frame number, $\mathcal{E}$ is the event occurred (serve, hit or bounce), $\mathcal{L}$ is respectively the location of the bottom player, the top player and the ball expressed in (x,y) coordinates. The most important elements of the CFG are discussed in this section. For a complete overview of the CFG, the reader is referred to Appendix A.
The alphabet $\mathcal{A}$ is made up of a set of characters, which can be combined to form a string of symbols. In this grammar the alphabet consists of 83 terminal symbols. Next to these symbols, the language distinguishes certain combination of symbols. These combinations are called *keywords* and are treated as single entities. For the understanding of this section it suffices to know that the alphabet contains symbols and keywords. Section 3.3.3 discusses in more detail how both types are recognized, analyzed and processed by the system. The alphabet containing the *symbols* is defined as:

$\mathcal{A}_{symbol}$ `=` `{` $\forall\ \exists\ \neg\ \wedge\ \vee\ <\ \leq\ >\ \geq$ `== + - * / ( ) , $ : := ?`
`0...9 A...Z a...z }`

The alphabet can roughly be divided in three parts. The first part contains the logic operator to express logic formulas. The second part consists of the computation symbols in order to make computations on the spatial-temporal data. The last part of the alpabet contains the alphanumerical symbols in order to construct formula names and variables.
The alphabet containing the *keywords* is defined as:

$\mathcal{A}_{keyword}$ `=` `{ LPAREN, RPAREN, FDEF, NEGATION, NUMBER,`
`TIMEPARAMETER, TIMEFRAMEPARAMETER, ...}`

Note that this alphabet only contains a subset of the *keywords*. The complete representation of *keywords* can be found in Appendix A. The alphabet containing the symbols, $\mathcal{A}_{symbol}$, can also be called the abstract mathematical

alphabet. It contains symbols that can be found directly back in the grammar (e.g. the digits and the parentheses), but it also contains mathemetical symbols that should be rewritten to keywords first, before they can actually be used (e.g. ∀ should be rewritten to `FOR_ALL`).
From the set of production rules, $\mathcal{R}$, the most important are discussed in the boxes below.

```
<definitions> ::=   <definitions> <function_definition> |
                    <function_definition>
```

Every program written in the spatial-temporal language consists of a list of definitions. This production rule defines that a list of `<definitions>` consists of one or multiple `<function_definition>`s.

```
<function_definition> ::=   <definition_name> LPAREN
                            <definition_argument_list> RPAREN
                            FDEF <definition_formula>
```

The production rule `<function_definition>` describes how a function definition should look like. Every function definition starts with a name to label the definition. The function name is a unique string consisting of the characters A-Z. After the `<definition_name>` a left parenthesis, indicated by the terminal character `LPAREN`, is placed to indicate the beginning of the argument list. The argument list is either empty or is made up of a list of parameters separated by a comma. The argument list is ended with a right parenthesis. The terminal `FDEF` is used to separate the function header from the function body - the formula. The terminal `FDEF` corresponds with the symbol `:=`.

```
<definition_formula> ::=    <formula> | <term>
```

The `<definition_formula>` consists of two production rules. The production rule `<formula>` is mainly concerned with the construction of logic formulas, the `<term>` rule is mostly concerned with the computational formulas.

```
<formula> ::=      LPAREN <formula> RPAREN |
                   NEGATION <formula> |
                   <formula> <logic_operator> <formula> |
                   <formula> <logic_operator> <term> |
                   <standard_formula> |
                   <quantifier> |
                   <definition_inner_definition> |
                   ...

<term> ::=         <term> <computation_operator> <term> |
                   NUMBER |
                   TIMEPARAMETER |
                   TIMEFRAMEPARAMETER |
                   ...
```

The production rule `<formula>` is an extensive one. Every possible combination of operators, terms, quantifiers, standard formulas and inner definitions is described here. The production rule starts with the possibility to embrace a `<formula>` with parentheses. This comes in useful when dealing with formulas in which the sequence of parsing can be ambiguous. A `<formula>` can also be preceded with a logical negation in which the truth value is inverted. The next two statements make it possible to interconnect `<formula>`s with the logic operators `AND` and `OR`. The sequence of `<term>`s and `<formula>`s can be mutually switched without consequences. In the rule `<standard_formula>` the user is enabled to ask the system if one of the five events has occured on a particular point in time. It also offers the possibility to request a coordinate of an event. An example of the use of standard formulas is given in Table 3.1.

Standard formula	Semantics	Result
`SPB(t1,$,$,$,$)`	`Has the bottom player hit a serve at time t1?`	`True/False`
`SPB(t1,?,$,$,$)` `SPB(t1,$,?,$,$)`	`Request respectively the x and y coordinate of the bottom player at time t1 and event SPB`	`Integer`
`SPB(t1,$,$,?,$)` `SPB(t1,$,$,$,?)`	`Request respectively the x and y coordinate of the top player at time t1 and event SPB`	`Integer`
`BB(t1,?,$)` `BB(t1,$,?)`	`Request respectively the x and y coordinate of the ball at time t1 and event BB`	`Integer`

Table 3.1: Standard formula examples. The use of `SPT`, `HPB` and `HPT` are analogous to the use of `SPB`.

The `<quantifier>` makes it possible to reason about complete sets. First order predicate logic knows two quantifiers: the existential and the universal. The existential quantification is used to indicate that a predicate is true for at least one member of the domain. The existential quantification is distinct from universal quantification, which asserts that a predicate is true for any member of the domain. The `<definition_inner_definition>` makes it possible to use a definition within another definition. The production rule `<formula>` is further explained with examples in Table 3.2. The `<formula>` rules also deal with the compare operators. Replacing the `<logic_operator>` with a `<compare_operator>` makes it possible to compare `<standard_formula>`s - given that the outcome of the `<standard_formula>` is a coordinate not a thruth value - and `<term>`s in every possible sequence.
The production rule `<term>` enables the user to add, subtract, divide and multiply integers. The outcome of a computation is always an integer. The other

elements in this production rule are mostly terminal characters. A `NUMBER` corresponds to a digit ranging from 0 to 9. A `TIMEPARAMETER` is used to label a point in time. It can be used as an argument in a `<function_definition>`, `<definition_inner_definition>`, `<standard_formula>` and in a `<quantifier>`. The `TIMEFRAMEPARAMETER` is a special terminal character that gives the user the possibility to indicate a point in time in terms of seconds. A possible application for this feature is:

- `t2 - t1 < s(2)`, meaning that the time between `t2` and `t1` is less than 2 seconds.

`s(2)` is converted to the corresponding number of frames. The argument in the `TIMEPARAMETER` is multiplied by the framerate, which can be determined when the video file is uploaded in the software.

`<formula>`	Example
`LPAREN <formula> RPAREN`	`( 1+2 < 4 )`
`NEGATION <formula>`	`!  (3 < 2)`
`<formula> <logic_operator> <formula>`	`(3 + HPT(t1,?,$,$,$,$) < 6) && (HPB(t1,?,$,$,$,$) - 1 > 0)`
`<formula> <logic_operator> <term>`	`(3 + 2 < 6) && (2 > 1)`
`<quantfier>`	`FOR_ALL:t1(100 > BB(t1,$,?))`
`<definition_innner_definition>`	`IN(4,5) || RALLY(t1,5)`

Table 3.2: Syntax examples of the production rule `<formula>`

3.3.2 Spatial-Temporal Definitions Based on the CFG

Having discussed the most important features of the grammar, this section shows examples of the powerful possibilities this language offers. In the first example is showed how a 'rally' can be defined. A rally in tennis always starts with a serve, followed by a number of hit events from both players - possibly alternated with ball bounces - and the rally always ends with either a fault or a winner from both players.

```
1  RALLY(t1,t2) :=   (SPB(t1,$,$,$,$) || SPT(t1,$,$,$,$) &&
2                    (
3                     OUT(BB(t2,?,$),BB(t2,$,?))  ||
4                     NET(BB(t2,$,?))  || WINNER(t2)
5                    ) &&
6                    t1 < t2 &&
7                    !EXISTS:t3((SPB(t3,$,$,$,$) ||
8                               SPT(t3,$,$,$,$) &&
9                               t1 < t3 && t2 > t3)
10
11 OUT(x,y) :=       x < 127 || x > 289 || y < 50 || y > 518
12
13 NET(y) :=         y == 208
```

Line 1 holds the `<function_definition>` `RALLY` which comes with two `TIMEPARAMETER`s. The first `TIMEPARAMETER`, `t1`, is used to indicate a point in time where a serve is hit by either the bottom or the top player. In lines 2-5 the stop condition of the rally is defined. A rally ends when a ball is hit wide or long, when a ball collides with the net or when one of both players hits a winner. Checking if a ball is in or out, if it hits the net or determining a winner are constantly recurrent functions, therefore they are defined as independent functions definitions (line 11 and 13) and used as `<definition_inner_definition>`s. By saying that `t1` has to occur before `t2`, line 6, only rallies starting with a serve event are selected. Line 7 defines a quantifier with a bounded variable `t3`. It says: "*there doesn't exist a t3 where a serve is hit while the rally is not finished yet. Since t1 is a serve event and t2 the end condition of the rally, there can't exist a t3 between t1 and t2*".
The function definition of `OUT` and `NET` need a bit more explanation with respect to the numbers used to compare x and y with. These numbers correspond to the coordinates of the court used by Vampire when an image is mapped from the real projection to the 2D projection. In the 2D projection, one pixel on the screen matches to 2 inches on the real court. The resulting coordinates can be found back in Figure 3.8. The x and y coordinates of the `OUT` function formula are respectively based on the coordinates of the single sidelines and both baselines. The y coordinate of the `NET` function formula corresponds with the y coordinate of the net in the scaled court. A bounce event at this coordinate can be twofold. On the one hand the ball can can actually be hit in the net resulting in the end of a point. On the other hand the ball can also *touch* the net. When the ball touches the net the rally is not necessarily finished instantly. When the ball touches the net and bounces on the opponent's half the rally continues. Though, it's not always evident for the system to see where the ball bounces

since it can drop just behind the net. This inaccuracies can be partly solved by looking at the next event. If the next event is a HPB or HPT, the ball evidently has touched the net and dropped on the opponent's half. If the next event is a SPB or SPT the ball touched the net and either landed untenable in the opponent's half or the ball touched the net and bounced on the same half of the court. This ambiguity currently can't be solved.

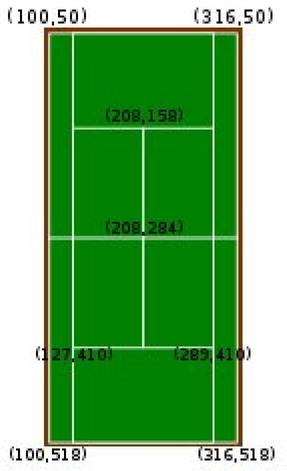


Figure 3.8: Coordinates of the scaled tennis court used by Vampire

In the next example the system is going to look for double faults hit by the bottom player from the deuce side. A double fault is hit when the serving player misses his first serve as well as his second serve. In this case the point is directly awarded to the opponent. In terms of tennis, a player serves from the deuce side if the player is serving at the right of the center mark.

```
1  DOU_FAU(t1,t2,t3,t4) :=   SPB(t1,$,$,$,$) && BB(t2,$,$) &&
2                            SPB(t3,$,$,$,$) && BB(t4,$,$) &&
3                            t1 < t2 && t2 < t3 && t3 < t4 &&
4                            t2-t1 < s(2) && t4-t3 < s(2) &&
5                            t4-t1 < s(20) &&
6                            !INLFSBOX(BB(t2,?,$),BB(t2,$,?))
7                            &&
8                            !INLFSBOX(BB(t4,?,$),BB(t4,$,?))
```

In pseudo language, this definition reasons over four points in time and it says: "*At t1 a serve is hit by the bottom player, at time t2 the ball has bounced, at time t3 a serve is hit by the bottom player and at time t4 the ball has bounced (line 1 and 2), such that the ball doesn't bounce in the left far serve box after both* *the first serve and the second serve (line 6-8).*" The lines that are not mentioned in the pseudo sentence are used to control the sequence of events. Line 3 restricts the sequence that first a serve should be hit before it can bounce. The second serve can only occur after the first serve has been hit and the second bounce can only occur after the second serve has been hit. Line 4 restricts the system not to search for arbitrary serve and bounce events but only the serve-bounce events that occur within two seconds from each other. Line 5 defines another time restriction and indicates that the whole double fault has to happen within 20 seconds.

3.3.3 Parsing Definitions

For the understanding of the definitions the system has to syntactically analyze, i.e. *parse*, the string of symbols that make up a definition. Before the parser can do its job, the string of symbols has to be examined by the *lexical analyzer* in order to generate understandable bits of data, i.e. *tokens*, for the parser. The objective of the parser is to create the concrete syntax tree (CST), or just parse tree. A parse tree is an ordered, rooted tree that represents the syntactic structure of a string according to a formal grammar. From the parse tree, the abstract syntax tree (AST), or just syntax tree, is derived. The syntax is 'abstract' in the sense that it does not represent every detail that appears in the real syntax. The usefulness of the syntax tree will become clear in the next subsection.
Having defined the grammar of the developed spatial-temporal language and elaborated on two sample definitions of the language in the previous two subsections (3.3.1 and 3.3.2), this section shows how the definitions are further processed by the lexical analyzer and the parser.

Lexical Analyzer

The input to the lexical analyzer is a string of symbols, i.e. the definitions, from an alphabet of characters. The lexical analyzer groups together certain terminal characters into single syntactic entities, called tokens. A token is a string of terminal symbols, with which we associate a lexical structure consisting of a pair of the form (token type, data). The first component, the token type, is a syntactic category such as "constant" or "identifier," and the second component, the data, is a pointer to data that has been accumulated about this particular token. For a given language the number of token types will be presumed finite [2].

The implementation of the lexical analyzer is done with JLex. JLex is a lexical analyzer generator, written for Java, in Java and is based on the well known LEX analyzer which was written for the UNIX operating system. The following example elaborates on the working of JLex. Consider the string of symbols:

- `s(2)`

When this string is offered to JLex it is examined one symbol at a time. When a potential pattern is recognized JLex is able to look further ahead in order to see whether a string of symbols matches a pattern.

-
  ```
  s\(([0-9]+)\) { return new Symbol(sym.TIMEFRAMEPARAMETER,
                  new ParameterToken(new String(yytext()))); }
  ```

In this example, JLex recognizes an `s` followed by a left parentheses, followed by one or more digits varying from 0 till 9, followed by a right parentheses. When the pattern is matched JLex exectues the code between the accolades. The code between the accolades contains the output of JLex. As said before, the output of the lexical analyzer serves as the input for the parser. The interface between the lexical analyzer and the parser is implemented using the class `sym`. The `sym` class is generated by CUP. CUP is a system for generating parsers from simple specifications and will be discussed in detail next.

Parser

Parsing, or syntax analysis, is a process in which the string of tokens (which is the output from the lexical analyzer) is examined to determine whether the string obeys certain structural conventions explicit in the syntactic definition of the language. It is also essential in the code generation process to know what the syntactic structure of a given string is. For example, the syntactic structure of the expression $A + B * C$ must reflect the fact that B and C are first multiplied and that then the result is added to A. No other ordering of the operations will produce the desired calculation. From a set of syntactic rules it is possible to automatically construct parsers which will make sure that a source program obeys the syntactic structure defined by these syntactic rules [2].
CUP (Constructor of Useful Parsers) is chosen for generating the parser. CUP is written in Java and produces parsers which are implemented in Java [6]. For CUP to produce a parser, the CFG described in Section 3.3.1 should be rewritten to the specific CUP syntax. Table 3.3 shows an example of this conversion. One can notice the similarity between the CFG and the CUP syntax. In CUP labels can be placed on various symbols in the right hand side of productions. In Table 3.3 the `definitions` symbol is labeled with `d` and the `function_definition` is labeled with `fd`. Labeling symbols makes it possible to dynamically execute JAVA code between the accolades.

CFG	`<definitions> ::=`	`<definitions> <function_definition>` | `<function_definition>`

CUP	`definitions ::=`	`definitions:d function_definition:fd`
		`{:`
		`  RESULT = d.addDefinitionNode(fd);`
		`:} |`
		`function_definition:fd`
		`{:`
		`  RESULT = new Definitions(fd);`
		`:};`

Table 3.3: Transforming the CFG to CUP syntax: `<definitions>`

By means of an example with the `<standard_formula>` `HPT` the total flow of the parsing process is depicted in Table 3.4. An `HPT` formula is used to request whether a serve has been hit at time `t1`. This string of symbols is analyzed by JLex and it recognized the `HPT` prefix. It then turns the `definition` into a token using the `sym` interface. CUP then analyzes the token according to its definition about how a `HIT_PLAYER_TOP` should look like. If the definition adheres to the syntax, the JAVA code between the accolades is executed.

definition	`HPT(t1,$,$,$,$)`
JLex	`"HPT" {return new Symbol(sym.HIT_PLAYER_TOP);}`

CUP	`HIT_PLAYER_TOP LPAREN`
	`definition_standard_formula_arg:p1 COMMA`
	`definition_standard_formula_arg:p2 COMMA`
	`definition_standard_formula_arg:p3 COMMA`
	`definition_standard_formula_arg:p4 COMMA`
	`definition_standard_formula_arg:p5 RPAREN`
	`{:`
	` RESULT = new DefinitionStandardFormulaNode`
	`           ("HPT",p1,p2,p3,p4,p5);`
	`:}`

Table 3.4: The interpretation of a `standard_formula` by JLex and CUP

In the introduction of this section the issue of ambiguity was mentioned that all parsers have to cope with. CUP has the ability to specify `PRECEDENCE` rules in which one can indicate to precede, e.g. an add operation before a multiply operation. Also the code between the accolades needs a bit more explanation. The objective of the code is to build up the parse tree. Every token generates a node that is connected to a previous node and hence a directed and rooted graph evolves. This subsection ends with a part of the eventual syntax tree (see Figure 3.9) of the `RALLY` definition as given in Section 3.3.2 that is generated after the lexical analyzer and the parser have done their jobs.

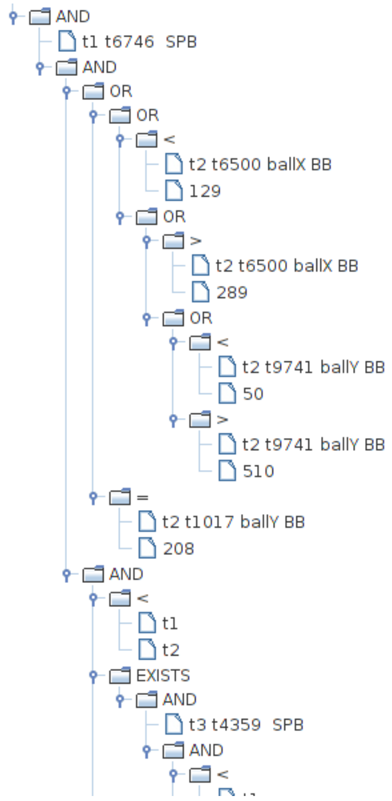


Figure 3.9: Syntax tree of the `RALLY` definition including the `definition_inner_definitions` `OUT` and `NET`.

3.3.4 Database Query Conversion

The importance of the syntax tree that is generated by the parser will become clear in this section. This section explains how the system converts a definition to a SQL query by means of the syntax tree. First, the conversion from a formula to a SQL query is formally presented using the abstract mathematical alphabet $\mathcal{A}_{symbol}$ as introduced in section 3.3.1. By using the $\mathcal{A}_{symbol}$ alphabet one can get a grasp more easily of the mappings in a single overview. The second part of this section shows the implementation details and examples.

Formal Conversion Rules

Mapping is the essential part of the conversion step. Every production rule of the grammar has to be correctly mapped to the corresponding SQL statement. In the table below the formal mappings of the most common production rules are given to show how definition formulas should be mapped to SQL.

Formula	SQL Mapping	Formula	SQL Mapping
ϕ	M(ϕ)	(ϕ)	(M(ϕ))
$\neg$ ϕ	NOT (M(ϕ))		
$\forall$ (ϕ)	NOT EXISTS NOT (M(ϕ))	$\exists$ (ϕ)	EXISTS (M(ϕ))
ϕ && ψ	M(ϕ) AND M(ψ)	ϕ || ψ	M(ϕ) OR M(ψ)

Table 3.5: Formally mapping production rules to SQL

Since SQL is able to do basic mathematic operations like adding, subtracting, mulitplying and dividing, computations from the grammar can be mapped one-to-one to a SQL statement.
The mapping of the production rule `<standard_formula>` is more complex since multiple arguments are involved. First of all, the `<standard_formula>` has to be examined on its return type (see Table 3.1). When the return type is an truth value, the database has to be searched to see if the event has occured on a specified point in time. When the return type is an integer, the user requests a coordinate. To obtain this request an additional statement has to be included with the desired column (e.g. `playerBottomX`, ballY, etc.). The implementation of the mapping of `standard_formula`s will be discussed in more detail later.

Implementing the Conversion Rules

When converting a definition to a SQL query, the two most vital parts are the syntax tree and the conversion rules. The former specifies *what* should be converted, the latter specifies *how* it should be converted. The advantage of the syntax tree is the possibility to use recursion in the SQL query implementation.

The implementation of every SQL query is initiated with a `SELECT` clause. Since the user is able to specify one or more `timeparameters` (e.g. `t1` and `t2`) in a definition, the `SELECT` clause always retrieves the `time` column(s) from the rows filtered by the `WHERE` clause.
An example of the mapping of the ¬ statement is shown below. When a negation is recognized by the `run` method, it adds the SQL `NOT` statement to the query. The next step is to go deeper and to decompose the rest of the formula. The result of this decomposition will be placed between the parentheses of the `NOT` clause.

$\neg\ \phi$	NOT (M(ϕ))	`if(formula.negation)`
		 `  return "NOT ( " + run(formula) + " ) ";`

Table 3.6: Left column: mathematical representation. Center column: mathematical conversion to SQL. Right column: mapping to SQL implementation

The next example shows how a sample definition formula is transformed to a valid SQL query. Suppose the following definition:

```
OUTSIDESINGLELINE(t1) := HPT(t1,?,$,$,$) < 130 ||
                         HPB(t1,?,$,$,$) > 289
```

The purpose of this definition is to find all `HIT` events by either the top or the bottom player that occured beyond the right single side lines. Since this definition has one `timeparameter`, the SQL statement to start with is:

```
SELECT t1.time AS t1 FROM events AS t1 WHERE
```

The box below shows the implementation steps that are taken when the formula part of this definition is transformed to SQL.

1	`SELECT t1.time AS t1 FROM events AS t1 WHERE`

2	`SELECT t1.time AS t1 FROM events AS t1 WHERE (`
	`M(HPT(t1,?,$,$,$) < 130) OR`
	`M(HPB(t1,?,$,$,$) > 289) )`

3	`SELECT t1.time AS t1 FROM events AS t1 WHERE (`
	`(t1.event = ’HPT’ AND t1.topPlayerX < 130) OR`
	`M(HPB(t1,?,$,$,$) > 289))`

4	`SELECT t1.time AS t1 FROM events AS t1 WHERE (`
	`(t1.event = ’HPT’ AND t1.topPlayerX < 130) OR`
	`(t1.event = ’HPB’ AND t1.bottomPlayerX > 289))`

A definition containing a quantifier is `VOLLEY`. It shows all the moments in the video where a volley has been hit by the bottom player. A volley is defined as an `HIT` event with the condition that the previous event was an `HIT` event by the opposite player. In other words, a volley is a stroke that does not precede a ball bounce. The definition is showed in the box below:

```
VOLLEY(t1,t2) := HPT(t1,$,$,$,$) &&
                 HPB(t2,$,$,$,$) &&
                 t1 < t2 &&
                 !EXISTS:t3(BB(t3,$,$) && t1 < t3 && t2 > t3)
```

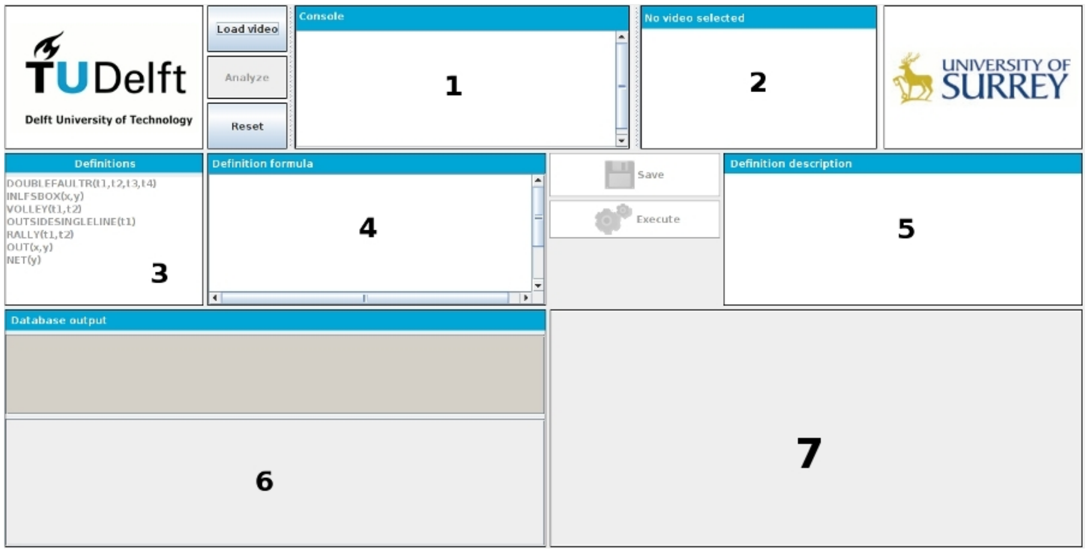
The transformation to SQL is given as:

```
SELECT t1.time AS t1, t2.time AS t2 FROM events AS t1, events
AS t2 WHERE ((t1.event = ’HPT’ AND (t2.event = ’HPB’ AND
((t1.time < t2.time) AND NOT ( EXISTS ( SELECT t3.time FROM
events AS t3 WHERE ((t3.event = ’BB’ AND ((t1.time < t3.time)
AND (t2.time > t3.time)))))))))
```

Interesting to see is the transformation efficiency. From a relatively simple and short definition a relatively complex and large SQL query is automatically produced.

3.4 Graphical User Interface

A Graphical User Interface (GUI) is built around the pipeline to integrate the video processing features together with the editing and execution of definitions in one system. The initial screen of the GUI is depicted in Figure 3.10.



DOUBLEFAULTR(t1,t2,t3,t4)
INLFSBOX(x,y)
VOLLEY(t1,t2)
OUTSIDESINGLELINE(t1)
RALLY(t1,t2)
OUT(x,y)
NET(y)

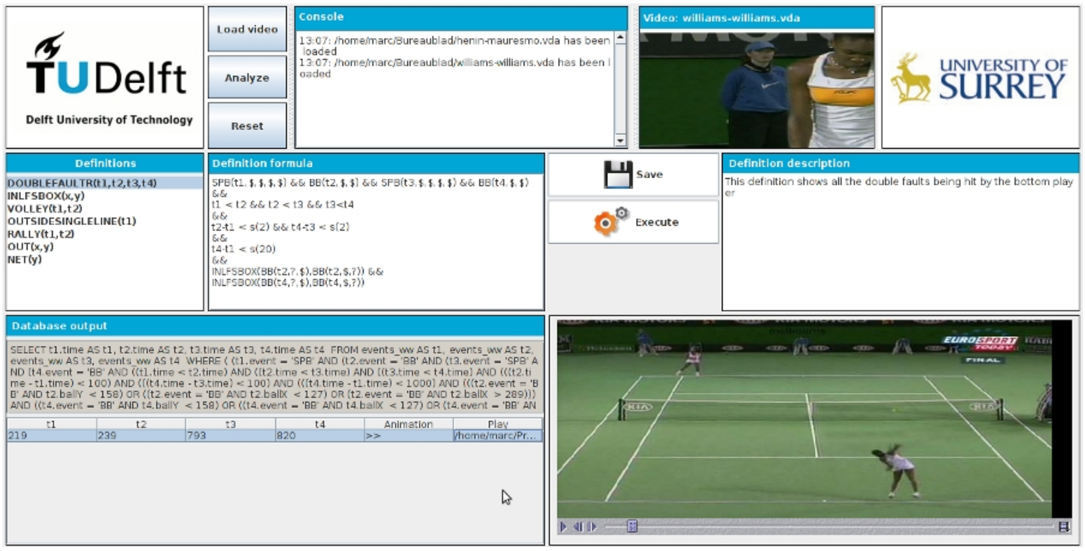
Figure 3.10: Initial screen of the GUI

The labeled boxes will be explained to show how the pipeline is integrated in the GUI:

1. **Console** a terminal screen in which the user is being informed about the status of the system.
2. **Video preview** showing a preview image of the selected video.
3. **Definitions** box used to list all the definitions that are either predefined by the system or added by the user
4. **Definition formula** box used for editing definitions.
5. **Definition description** box used for writing an informal description of the definition.
6. **Query output** box used to show the SQL query and a list of fragments that meet the query's requirement.
7. **Video box** box used as media player to play the video from the list of fragments.

A few items on the GUI screen are left undiscussed yet. The buttons in the top half of the screen are used to respectively load a video in the system (Load video), analyze the inserted video (Analyze) and to reset the whole system to its start state (Reset). The two buttons located at the center of the screen are used to save the edited definition (Save) and to execute the definition (Execute).

In Figure 3.11 the GUI is showed when a video is loaded. The user has written a definition that filters out all the double faults hit at the deuce side of the court of the tennis video. The eventual query together with the points in time that matches the requirements of the definition are showed in the bottom left corner of the screen. The extraction of the movie fragment from the tennis video may need a little more explanation.
When Vampire analyzes a tennis video, the video is decomposed in single frames resulting in a set of JPG images. The pictures matching the points in time, and all the pictures in between, are concatenated and the resulting file is decoded with an AVI codec. For the playback framerate the framerate of the original tennis video is used.



```
13:07: /home/marc/Bureaublad/henin-mauresmo.vda has been
 loaded
13:07: /home/marc/Bureaublad/williams-williams.vda has been l
oaded
```

```
DOUBLEFAULTR(t1,t2,t3,t4)
INLFSBOX(x,y)
VOLLEY(t1,t2)
OUTSIDESINGLELINE(t1)
RALLY(t1,t2)
OUT(x,y)
NET(y)
```

```
SPB(t1, $, $, $, $) && BB(t2, $, $) && SPB(t3, $, $, $, $) && BB(t4, $, $)
&&
t1 < t2 && t2 < t3 && t3<t4
&&
t2-t1 < s(2) && t4-t3 < s(2)
&&
t4-t1 < s(20)
&&
INLFSBOX(BB(t2,?,$),BB(t2,$,?)) &&
INLFSBOX(BB(t4,?,$),BB(t4,$,?))
```

This definition shows all the double faults being hit by the bottom play
er

```
SELECT t1.time AS t1, t2.time AS t2, t3.time AS t3, t4.time AS t4  FROM events_ww AS t1,  events_ww AS t2,
events_ww AS t3, events_ww AS t4  WHERE ( (t1.event = 'SPB' AND (t2.event = 'BB' AND (t3.event = 'SPB' A
ND (t4.event = 'BB' AND ((t1.time < t2.time) AND ((t2.time < t3.time) AND ((t3.time < t4.time) AND (((t2.ti
me - t1.time) < 100) AND (((t4.time - t3.time) < 100) AND (((t4.time - t1.time) < 1000) AND (((t2.event = 'B
B' AND t2.ballY  < 158) OR ((t2.event = 'BB' AND t2.ballX  < 127) OR (t2.event = 'BB' AND t2.ballX  > 289)))
AND ((t4.event = 'BB' AND t4.ballY  < 158) OR ((t4.event = 'BB' AND t4.ballX  < 127) OR (t4.event = 'BB' AN
```

t1	t2	t3	t4	Animation	Play
219	239	793	820	>>	/home/marc/Pr...

Figure 3.11: Sample screenshot of the GUI when a tennis video is analyzed and evaluated.

Chapter 4

Evaluation

To evaluate the system and to get a 'succes' indication, measurable conditions have to be specified. As mentioned in the introduction, three requirements were important for this system. The system should be portable, flexible and user-friendly. Each requirement will be briefly discussed to see to what extent they have been met.

Portability

The starting requirement was to develop a system that could perform its tasks with only the video data from one camera behind the tennis court. Having tried multiple footage, the accuracy of the video analysis software (Vampire) can only be guaranteed using broadcast tennis footage. When using amature video footage the accuracy rate drops drastically to, at most, 30%. The reason for this seems to lie in the use of professional cameras resulting in higher quality video footage. Also the angle of projection plays a role in the accuracy. Broadcast videos are generally recorded from a greater height. The effect of perspective and hence the occlusion effect is diminished.
Right now the system runs on Linux Ubuntu and can not easily distributed to another workstation, let alone another operating system. The reason for this is the Vampire software that runs on RAVL (Recognition Audio Video Library). In order to use the system, a workstation first have to be equipped with the RAVL libraries which , at the moment, have trouble working under Microsoft Windows operating systems.

Flexibility

The user should be able to specify what moments he wants to extract from a tennis video. The developed spatial-temporal language is flexible enough to enable the user to define formulas with a certain degree of freedom. However, looking at both the developed spatial-temporal language and the SQL query language there is still space left for further improvements. SQL has the ability to define queries in which the number of a condition is counted (using the `SUM()` clause). This gives more freedom to the user, e.g., right now it is possible to define a query saying *show all rallies where the ball has been hit two times cross and the next ball is hit down the line.* However, currently it is not possible to express queries that say *show all rallies where the ball is **at least** hit two times cross and then down the line.*

User-friendly

To measure the usability of the system the system should be be used by a group of tennis coaches. This method, panel testing, exposes the vulnerabilities of the system. During the project the system has not been subject to a formal panel testing method, however the system has been presented to tennis coaches. The returned faadback was:

- **Complex spatial-temporal language** Defining formulas in the developed language is too complex for the average tennis coach.
- **Vampire's processing time** After a video has been loaded in the system, the analyzing step of Vampire takes a long time. A one minute tennis video takes about half an hour to be fully processed on an Intel dual-core machine.
- **User interface** The graphical user interface was experienced as organized and intuitive.

In conclusion, what is achieved by the research and the developed prototype? The main objective was to develop tactical tennis software that supports the tennis coach in practice sessions. The prototype has been developed and can be, to a certain extent, be used in practice sessions.
The developed spatial-temporal language and the conversion to SQL has shown to be effective. A relatively short and easy to understand formula in the spatial-temporal language can be successfully converted to a rather large and complex SQL query.
As discussed the system has also downsides. For the system to become more flexible, the language should be extended to give the tennis coach the possibility to express whatever he wants. Also the input of definitions must be made easier to make it understandable for every tennis coach. Next to the language, the processing time of the tennis video analysis software must improve to make the system widely applicable.

Chapter 5

Discussion and Future Work

When this project started the main objective was to design a system that enables the tennis coach to analyze tennis matches using the footage of only one video camera. Having developed the system's prototype and after some extensive discussions with authorities in the tennis sport the system might be better off when it is equipped with the possibility to use the data of multiple cameras. Since the system is intended for players at the professional level, a setup using multiple cameras with the possibility to replay situations from multiple angles and to increase the accuracy of the spatial-temporal data is preferred.
For a tennis coach to specify what he or see wants to extract from the tennis footage an additional layer must be made on top of the language. The objective of this layer is to guide the user in expressing a definition in the language. A possibility is to create a graphical wizard with a number of steps in which the user can graphically express the situation he wants to extract. This wizard must be capable of converting the user input to a definition in the spatial-temporal language. To increase the portability and usability of the system, further work can be done in making the system work on portable tablet devices with a touch screen.
The video analysis software might be rewritten to deal with data from multiple cameras. The main bottleneck right now is the processing time. The time to convert a tennis video to a database filled with spatial-temporal data takes too long. The revision of Vampire or writing new tennis video analysis software should reduce the processing time drastically.

The work of this thesis started in february 2010. With a global overview of the desired outcome and a set of requirements the design and implementation started. Every two weeks there was a discussion with the supervisor in which the prototype was discussed. June 30, the prototype finished and the writing of the thesis began. The planning was tight, but certainly achievable. After this Master's thesis project the project might be continued since there is interest from national and international tennis federations.

Bibliography

[1] Abiteboul, S., Herr, L., and den Bussche, J. V. Temporal versus first-order logic to query temporal databases. In *PODS* (1996), ACM Press, pp. 49–57.

[2] Aho, A. V., and Ullman, J. D. *The theory of parsing, translation, and compiling.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[3] Chomicki, J., and Toman, D. Temporal logic in database query languages. In *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer US, 2009, pp. 2987–2991.

[4] Dang, B., Tran, A., Dinh, T., and Dinh, T. A real time player tracking system for broadcast tennis video. In *ACIIDS (2)* (2010), N. T. Nguyen, M. T. Le, and J. Swiatek, Eds., vol. 5991 of *Lecture Notes in Computer Science*, Springer, pp. 105–113.

[5] Huang, Y.-P., Chiou, C.-L., and Sandnes, F. E. An intelligent strategy for the automatic detection of highlights in tennis video recordings. *Expert Syst. Appl. 36*, 6 (2009), 9907–9918.

[6] Hudson, S. E. Cup user's manual, September 2010. `http://www2.cs.tum.edu/projects/cup/manual.html`.

[7] Jiang, Y.-C., Lai, K.-T., Hsieh, C.-H., and Lai, M.-F. Player detection and tracking in broadcast tennis video. In *PSIVT '09: Proceedings of the 3rd Pacific Rim Symposium on Advances in Image and Video Technology* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 759–770.

[8] Kittler, J., Christmas, W. J., Kostin, A., Yan, F., Kolonias, I., and Windridge, D. A memory architecture and contextual reasoning framework for cognitive vision. In *SCIA* (2005), H. Kälviäinen, J. Parkkinen, and A. Kaarna, Eds., vol. 3540 of *Lecture Notes in Computer Science*, Springer, pp. 343–358.

[9] Kolonias, I. Cognitive vision systems for video understanding and retrieval. Master's thesis, University of Surrey, UK, 2007.

[10] Konur, S. A survey on temporal logics. *CoRR abs/1005.3199* (2010).

[11] Lepetit, V., Shahrokni, A., and Fua, P. Robust data association for online applications. In *CVPR (1)* (2003), IEEE Computer Society, pp. 281–288.

[12] Leucker, M., and Sánchez, C. Regular linear temporal logic. In *ICTAC* (2007), C. B. Jones, Z. Liu, and J. Woodcock, Eds., vol. 4711 of *Lecture Notes in Computer Science*, Springer, pp. 291–305.

[13] Mahmood, N., Burney, A., and Ahsan, K. A logical temporal relational data model. *CoRR abs/1002.1143* (2010).

[14] McIlroy, P. Hawk-eye: Augmented reality in sports broadcasting and officiating. In *ISMAR* (2008), IEEE.

[15] Miyamori, H. Improving accuracy in behaviour identification for content-based retrieval by using audio and video information. In *ICPR (2)* (2002), pp. 826–830.

[16] Miyamori, H., and ichi Iisaku, S. Video annotation for content-based retrieval using human behavior analysis and domain knowledge. In *FG* (2000), IEEE Computer Society, pp. 320–325.

[17] Pingali, G. S., Jean, Y., and Carlbom, I. Real-time tracking for enhanced tennis broadcasts. In *CVPR* (1998), IEEE Computer Society, pp. 260–265.

[18] Pingali, G. S., Opalach, A., and Jean, Y. Ball tracking and virtual replays for innovative tennis broadcasts. In *ICPR* (2000), pp. 4152–4156.

[19] Rashbass, C. The relationship between saccadic and smooth tracking eye movements. *J. Physiol. abs/1005.3199* (1961), 159, 326–338.

[20] Shaeib, A., Conroy, K., and Roantree, M. Extracting tennis statistics from wireless sensing environments. In *DMSN* (2009), M. A. Nascimento and N. Tatbul, Eds., ACM International Conference Proceeding Series, ACM.

[21] Tien, M.-C., Wang, Y.-T., Chou, C.-W., Hsieh, K.-Y., Chu, W.-T., and Wu, J.-L. Event detection in tennis matches based on video data mining. In *ICME* (2008), IEEE, pp. 1477–1480.

[22] Tynan, R., Schoofs, A., Muldoon, C., O'Hare, G. M. P., Conaire, C. O., Kelly, P., and O'Connor, N. E. Intelligent middleware for adaptive sensing of tennis coaching sessions. In *CSE (2)* (2009), IEEE Computer Society, pp. 891–896.

[23] Yan, F., Christmas, W. J., and Kittler, J. Layered data association using graph-theoretic formulation with application to tennis ball tracking in monocular sequences. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 10 (2008), 1814–1830.

[24] Yan, F., Kostin, A., Christmas, W. J., and Kittler, J. A novel data association algorithm for object tracking in clutter with application to tennis video analysis. In *CVPR (1)* (2006), IEEE Computer Society, pp. 634–641.

[25] Yu, X., Sim, C.-H., Wang, J. R., and Cheong, L. F. A trajectory-based ball detection and tracking algorithm in broadcast tennis video. In *ICIP* (2004), pp. 1049–1052.

[26] Zhu, G., Xu, C., Huang, Q., Gao, W., and Xing, L. Player action recognition in broadcast tennis video with applications to semantic analysis of sports game. In *ACM Multimedia* (2006), K. Nahrstedt, M. Turk, Y. Rui, W. Klas, and K. Mayer-Patel, Eds., ACM, pp. 431–440.

Appendix A

Context Free Grammar

The full Context Free Grammar (CFG) of the designed spatial-temporal language is provided in this appendix. A context free grammar (CFG) is a quadruple $< \mathcal{S}, \mathcal{N}, \mathcal{A}, \mathcal{R} >$, where $\mathcal{S}$ is the start symbol, $\mathcal{N}$ is the set of non-terminals, $\mathcal{A}$ the alphabet and $\mathcal{R}$ the set of rules. A CFG defines a formal language, i.e. the set of all sentences (strings of words) that can be derived by the grammar.

```
A =         FOR_ALL, EXISTS, NEGATION, AND, OR, LESS, GREATER,
            LESSOREQUAL, GREATEROREQUAL, EQUALS, PLUS,
            MINUS, TIMES, DIVISION, LPAREN, RPAREN, FDEF,
            COLON, COMMA, HIT_PLAYER_BOTTOM, HIT_PLAYER_TOP,
            SERVE_PLAYER_BOTTOM, SERVE_PLAYER_TOP, BOUNCE_BALL,
            QUESTION_MARK, WILDCARD, NUMBER, STRING, PARAMETER,
            TIMEPARAMETER, TIMEFRAMEPARAMETER
```

```
P =
startParser ::=             start_def
start_def ::=               definitions
start_def ::=               definitions
definitions ::=             definitions function_definition |
                            function_definition
function_definition ::=     definition_name LPAREN
                            definition_arg_list RPAREN FDEF
                            definition_formula
definition_name ::=         STRING
definition_arg_list ::=     definition_arg_list COMMA
                            definition_argument |
                            definition_argument
definition_argument ::=     PARAMETER | TIMEPARAMETER
```

```
def_std_form_arg ::=        WILDCARD | QUESTION_MARK |
                            TIMEPARAMETER
definition_formula ::=      formula | term
formula ::=                 LPAREN formula RPAREN |
                            NEGATION formula |
                            formula logic_operator formula |
                            formula logic_operator term |
                            term logic_operator formula |
                            term compare_operator term |
                            quantifier |
                            standard_formula compare_operator
                            standard_formula |
                            standard_formula compare_operator
                            term
                            term compare_operator
                            standard_formula
                            term logic_operator term
                            standard_formula
                            def_inner_def
def_inner_def ::=           definition_name:dn LPAREN
                            def_arg_list_id RPAREN
def_arg_list_id ::=         def_arg_list_id COMMA def_arg_id |
                            def_arg_id
def_arg_id ::=              standard_formula |
                            TIMEPARAMETER |
                            PARAMETER |
                            NUMBER |
standard_formula ::=        HIT_PLAYER_BOTTOM LPAREN
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg RPAREN |
                            HIT_PLAYER_TOP LPAREN
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg RPAREN |
                            SERVE_PLAYER_BOTTOM LPAREN
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg COMMA
                            def_std_form_arg RPAREN |
                             SERVE_PLAYER_TOP LPAREN
                             def_std_form_arg COMMA
                             def_std_form_arg COMMA
                             def_std_form_arg COMMA
                             def_std_form_arg COMMA
                             def_std_form_arg RPAREN |
                             BOUNCE_BALL LPAREN def_std_form_arg
                             COMMA def_std_form_arg COMMA
                             def_std_form_arg RPAREN
quantifier ::=               FOR_ALL COLON TIMEPARAMETER LPAREN
                             formula RPAREN |
                             FOR_ALL COLON TIMEPARAMETER LPAREN
                             term RPAREN |
                             EXISTS COLON TIMEPARAMETER LPAREN
                             formula RPAREN |
                             EXISTS COLON TIMEPARAMETER LPAREN
                             term RPAREN
term ::=                     standard_formula
                             computation_operator
                             standard_formula |
                             term computation_operator term |
                             LPAREN term:t RPAREN |
                             NUMBER |
                             TIMEPARAMETER |
                             PARAMETER |
                             TIMEFRAMEPARAMETER
logic_operator ::=           AND | OR
compare_operator ::=         GREATER | LESS |
                             GREATEROREQUAL | LESSOREQUAL |
                             EQUALS
computation_operator ::=     PLUS | MINUS |
                             TIMES | DIVISION

𝒮 =                          start_def
```