Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica

**Tesis Doctoral**
Identification of Optimal Strategies in Tennis using Broadcast Video Sequences and Machine Learning

Antonio Terroba Acha
*Dirigida por:*
Dr. Javier Varona Gómez
Dra. Margarita Miró Julià

2013
A mi mujer María y mis tres hijos, Carla, Nina y Jamie.
Acknowledgements

I would like to thank my supervisors Dr. Javier Varona and Dra. Margarita Miró, for their guidance throughout the course of this work, which has been especially challenging at times due to its long-distance nature. I would also like to thank Dr. Walter Kosters for his help on the data mining articles and for the always stimulating technical discussions on this work. Special thanks also go to Dr. Bill Christmas for supporting me at the initial stages of this thesis and letting me use their computer vision framework and libraries. I would also like to thank Dr. Jean-Bernard Hayet for his support during the implementation of the Fast-Projection algorithm and for being so accessible when needed. Many thanks also go to Dr. David Sanz who has reviewed all the tennis-specific aspects of this thesis and has introduced me to several professional tennis coaches who were able to provide invaluable input into this work. Finally, and most importantly, I would like to express my deepest gratitude to my wife Maria for the love and support she has shown to me during this long, and sometimes complicated journey.
Abstract

The analysis of tennis sequences from broadcast videos has been studied before with an aim to automatically annotate the score or to classify the content for later retrieval, but seldom with a tactical aim.

This work focuses therefore in designing a system able to perform tactical analysis in tennis games. The main objectives of such a system include the identification of the most frequent patterns displayed by a player in a game, and the discovery of optimal sequences of strokes that maximize the probability for a player to beat the opponent. The proposed system must also be able to translate the results into a high-level and user-friendly language.

To do so, it is fundamental to start off with an accurate notational representation of a tennis game. This representation must incorporate the majority of variables that intervene in the game, such as the players positions, the ball bouncing positions, the strokes performed, the ball effects, the distances run, the score, etc. In order to obtain this information, a semi-automatic system is proposed and implemented.

Once this accurate data has been gathered, and with a view to achieve the objectives aforementioned, this thesis explores this information from two different angles. A first one, based on multivariate sequential data mining, in an attempt to discover game patterns and player characterization; and a second one, based on reinforcement learning and artificial intelligence with an aim to identify optimal potential sequences of strokes that would improve the chances of winning. In this way, the system is able to suggest the most appropriate strategy for each of the players.

To showcase the tactical conclusions and the overall process, this work analyzes several real tennis matches and displays graphically the results obtained.

Key words: Sports Broadcast Video Analysis; Tennis; Markov Decision Processes; Monte Carlo Simulations; Optimal Policies; Sequential Data Mining.
Resumen

La anotación automática de secuencias de videos deportivos partiendo de retransmisiones televisivas ha sido estudiada con anterioridad con el fin de conseguir sistemas capaces de detectar y clasificar las secuencias, pero pocas veces con una finalidad táctica.

Este trabajo se centra por tanto en conseguir un sistema capaz de realizar análisis tácticos de partidos de tenis. Los objetivos de dicho sistema incluyen la identificación de las secuencias de golpes más utilizadas por los jugadores, así como el descubrimiento de las secuencias óptimas de golpes que maximicen las probabilidades de un jugador de ganar a su contrincante. Por último, el sistema debe ser capaz de trasladar estos resultados a un lenguaje sencillo y fácilmente visualizable.

Para ello, y como condición sine qua non, es preciso partir de una representación notacional muy aproximada y completa de un partido de tenis. Una representación que incorpore con exactitud la mayoría de las variables que intervienen en el desarrollo del juego (posiciones de los jugadores, posición del bote de la pelota, golpe efectuado, efectos sobre la pelota, distancias recorridas, marcador, etc.). Para conseguir esto, ha sido necesario en primer lugar desarrollar un sistema semi-automático de visión por ordenador que capture la información relevante.

Una vez obtenida dicha representación, y para poder conseguir los objetivos señalados anteriormente, esta tesis explora dichos datos desde dos perspectivas distintas. Una primera, basada en la minería de datos secuencial, cuyo objetivo es el descubrimiento de patrones de juego y la caracterización de los jugadores. Y una segunda, basada en el aprendizaje por refuerzo y en la inteligencia artificial, centrada en identificar potenciales secuencias óptimas de golpes. De esta forma, el sistema es capaz de sugerir la estrategia más apropiada para cada uno de los jugadores.

Para comprobar las conclusiones tácticas, así como el procedimiento seguido, este trabajo analiza varios partidos reales de tenis y muestra gráficamente los resultados obtenidos.
Palabras Claves: Procesado de Videos Deportivos; Tenis; Procesos de Decisión de Markov; Simulaciones de Monte Carlo; Políticas Óptimas; Minería de Datos Secuencial.
# Contents

1 Introduction ................................................. 1  
   1.1 Related Work ........................................... 3  
   1.2 Motivation ............................................... 5  
   1.3 Scientific Contribution .................................. 5  
   1.4 Overview of this Thesis ................................. 6  
   1.5 Overview of Publications ............................... 7  

2 Annotation System ........................................... 9  
   2.1 System Description ...................................... 10  
      2.1.1 Introduction ....................................... 10  
      2.1.2 System Design ..................................... 10  
      2.1.3 High Level Overview ............................... 15  
   2.2 Pre-Processing .......................................... 15  
      2.2.1 Luminance Filter .................................. 17  
      2.2.2 Texture Filter ..................................... 19  
      2.2.3 Linearity Filter ................................... 20  
      2.2.4 Probabilistic Hough Filter .......................... 21  
      2.2.5 Selection of the Main Lines ....................... 22  
   2.3 Automatic Calibration of Broadcast Images ............... 25  
      2.3.1 Previous Work ..................................... 25  
      2.3.2 Homography ........................................ 26  
      2.3.3 Fast Projection Algorithm ......................... 28  
      2.3.4 Hypothesis Generation .............................. 32  
      2.3.5 Hypothesis Evaluation .............................. 33  
      2.3.6 Homography Matrix Calculation ..................... 35
3 Data Mining and Pattern Discovery

3.1 Formalization
  3.1.1 Definitions
  3.1.2 Reference Model
  3.1.3 Attributes Considered

3.2 Multi-variate similarity-based sequential pattern mining
  3.2.1 Similarity measure
  3.2.2 Visualization
  3.2.3 Similarity thresholds
  3.2.4 Mining Problems

3.3 Mining Problem 1
  3.3.1 Sequential Pattern Mining
  3.3.2 Clustering
  3.3.3 Weighted centroids

3.4 Mining Problem 2
  3.4.1 Identification of unbalancing events
  3.4.2 Winning patterns
  3.4.3 Completion of attack patterns
  3.4.4 Pattern prequel and sequel

3.5 Mining Problem 3

3.6 Results
  3.6.1 Mining Problems 1 and 2
  3.6.2 Mining Problem 3

3.7 Conclusions
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>xi</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 Monte Carlo Tree Search (MCTS)</td>
<td>86</td>
</tr>
<tr>
<td>4.3.1 Algorithm structure</td>
<td>87</td>
</tr>
<tr>
<td>4.3.2 Simulation</td>
<td>88</td>
</tr>
<tr>
<td>4.3.3 Back-propagation</td>
<td>88</td>
</tr>
<tr>
<td>4.3.4 Selection</td>
<td>90</td>
</tr>
<tr>
<td>4.3.5 Expansion</td>
<td>90</td>
</tr>
<tr>
<td>4.3.6 Algorithm</td>
<td>90</td>
</tr>
<tr>
<td>4.4 Experiments</td>
<td>91</td>
</tr>
<tr>
<td>4.4.1 Initial Model</td>
<td>91</td>
</tr>
<tr>
<td>4.4.2 Full Tennis Model</td>
<td>93</td>
</tr>
<tr>
<td>4.5 Conclusions and further research</td>
<td>98</td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>101</td>
</tr>
<tr>
<td>Bibliography</td>
<td>103</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Screenshot of <em>Dartfish</em> software</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Data Capture and Annotation System</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Ball position in broadcast image and in reference court</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Annotation system overview</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>High level flowchart for the homography calculation and tracking</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>Histograms of the images’ central area for a clay court (left) and a one-color hard court (right)</td>
<td>18</td>
</tr>
<tr>
<td>2.6</td>
<td>Histograms of the images’ central area for a two-color hard court (left) and a grass court (right)</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>Luminance threshold calculation</td>
<td>18</td>
</tr>
<tr>
<td>2.8</td>
<td>Image after applying luminance filter (left) and the result after applying a texture filter with $s_p = 7 \times 7$ (right)</td>
<td>20</td>
</tr>
<tr>
<td>2.9</td>
<td>Image after applying a texture filter with $s_p = 3 \times 3$ (left) and image after applying the Canny filter (right)</td>
<td>20</td>
</tr>
<tr>
<td>2.10</td>
<td>Image after applying the linearity filter. $\tau = 1$ (left), $\tau = 3$ (center), and $\tau = 5$ (right)</td>
<td>21</td>
</tr>
<tr>
<td>2.11</td>
<td>Lines detected by the probabilistic Hough transform</td>
<td>22</td>
</tr>
<tr>
<td>2.12</td>
<td>Detected segments along the lines</td>
<td>23</td>
</tr>
<tr>
<td>2.13</td>
<td>Main line selection after applying different methods</td>
<td>25</td>
</tr>
<tr>
<td>2.14</td>
<td>Transformation between the image plane and the reference plane</td>
<td>27</td>
</tr>
<tr>
<td>2.15</td>
<td>Vanishing points for a typical broadcast perspective</td>
<td>28</td>
</tr>
<tr>
<td>2.16</td>
<td>Cross Ratio definition</td>
<td>29</td>
</tr>
<tr>
<td>2.17</td>
<td>Terminology used in fast projection</td>
<td>30</td>
</tr>
<tr>
<td>2.18</td>
<td>Line $(p, p_v)$ on the projected image</td>
<td>31</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.19</td>
<td>Terminology used to denote the intersections on the reference model</td>
<td>32</td>
</tr>
<tr>
<td>2.20</td>
<td>Shapes allowed to classify an intersection as type “A”</td>
<td>32</td>
</tr>
<tr>
<td>2.21</td>
<td>Hypothesis evaluation algorithm</td>
<td>34</td>
</tr>
<tr>
<td>2.22</td>
<td>Algorithm to find the optimum four correspondences</td>
<td>36</td>
</tr>
<tr>
<td>2.23</td>
<td>Reference model projected back onto the broadcast image</td>
<td>37</td>
</tr>
<tr>
<td>2.24</td>
<td>Court detection: grass court with shades</td>
<td>38</td>
</tr>
<tr>
<td>2.25</td>
<td>Court detection: clay court with different lighting conditions</td>
<td>38</td>
</tr>
<tr>
<td>2.26</td>
<td>Court detection: indoors hard court</td>
<td>39</td>
</tr>
<tr>
<td>2.27</td>
<td>Court detection: outdoors hard court</td>
<td>39</td>
</tr>
<tr>
<td>2.28</td>
<td>Pyramidal Lucas-Kanade optical flow. Taken from [5]</td>
<td>41</td>
</tr>
<tr>
<td>2.29</td>
<td>Original frame ( n ) and frame ( n + m )</td>
<td>41</td>
</tr>
<tr>
<td>2.30</td>
<td>Displaying the vector flows</td>
<td>42</td>
</tr>
<tr>
<td>2.31</td>
<td>Algorithm to find the homography matrix in frame ( n + m )</td>
<td>43</td>
</tr>
<tr>
<td>2.32</td>
<td>Final result of the tracked court</td>
<td>44</td>
</tr>
<tr>
<td>3.1</td>
<td>Structured Information</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Tennis court reference model</td>
<td>48</td>
</tr>
<tr>
<td>3.3</td>
<td>Sequence similarity visualization</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>Virtual rallies identification</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>PREFIXSPAN Algorithm</td>
<td>56</td>
</tr>
<tr>
<td>3.6</td>
<td>Pattern diagrams</td>
<td>57</td>
</tr>
<tr>
<td>3.7</td>
<td>Event similarity representation of events shown in Figure 3.6</td>
<td>58</td>
</tr>
<tr>
<td>3.8</td>
<td>Similarity-based DBSCAN Algorithm</td>
<td>59</td>
</tr>
<tr>
<td>3.9</td>
<td>DBSCAN. Core points and border points. Taken from [18]</td>
<td>60</td>
</tr>
<tr>
<td>3.10</td>
<td>NFA for the winning pattern sequel</td>
<td>65</td>
</tr>
<tr>
<td>3.11</td>
<td>Winning pattern similarity</td>
<td>66</td>
</tr>
<tr>
<td>3.12</td>
<td>Algorithm — Winning patterns identification</td>
<td>68</td>
</tr>
<tr>
<td>3.13</td>
<td>Service winning patterns</td>
<td>70</td>
</tr>
<tr>
<td>3.14</td>
<td>Groundstroke attacking winning patterns</td>
<td>71</td>
</tr>
<tr>
<td>3.15</td>
<td>Four subrallies from the Roland Garros 2009 match</td>
<td>72</td>
</tr>
<tr>
<td>4.1</td>
<td>Simple tennis model</td>
<td>77</td>
</tr>
<tr>
<td>4.3</td>
<td>Simulation results</td>
<td>80</td>
</tr>
<tr>
<td>4.4</td>
<td>Tactical model</td>
<td>81</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.5</td>
<td>Tennis court grid</td>
<td>82</td>
</tr>
<tr>
<td>4.6</td>
<td>Some possible trajectories: (left) serving to the deuce court, (right) hitting from the baseline</td>
<td>84</td>
</tr>
<tr>
<td>4.7</td>
<td>Example of a rally with three strokes</td>
<td>85</td>
</tr>
<tr>
<td>4.8</td>
<td>Phases of the Monte Carlo Tree Search</td>
<td>88</td>
</tr>
<tr>
<td>4.9</td>
<td>PNFA and Shortest Path</td>
<td>89</td>
</tr>
<tr>
<td>4.10</td>
<td>Shortest-path MCTS Algorithm</td>
<td>91</td>
</tr>
<tr>
<td>4.11</td>
<td>Comparison of algorithms tested</td>
<td>92</td>
</tr>
<tr>
<td>4.12</td>
<td>Points won profile comparison between original match results, Q-Learning and MCTS</td>
<td>94</td>
</tr>
<tr>
<td>4.13</td>
<td>Successful strategies displayed by Soderling against Nadal</td>
<td>96</td>
</tr>
<tr>
<td>4.14</td>
<td>Successful strategies displayed by Djokovic against Nadal</td>
<td>97</td>
</tr>
<tr>
<td>4.2</td>
<td>Algorithm — Unbalancing model</td>
<td>99</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Types of strokes annotated ......................................................... 13
2.2 Ball status ............................................................................. 13
2.3 Different cases when calculating the cross ratio .................... 31
2.4 Classification of intersections according to their shape on the reference model 33

3.1 Data evaluated ........................................................................ 69
3.2 NaLi Service winning patterns ................................................ 69
3.3 Serena Williams Service winning patterns ............................... 69

4.1 Policy for Player P1 ................................................................. 79
4.2 Policy for Player P2 ................................................................. 80
4.3 Actions for the service ............................................................ 84
4.4 Actions for the rest of strokes ................................................ 84
4.5 State-action sequence for Figure 4.7 ........................................ 85
4.6 Best strategies after 10,000 episodes ............................... 93
4.7 Best strategies after 10,000 episodes ........................................ 95
4.8 Best strategies after 10,000 episodes for Soderling ................ 96
4.9 Best strategies after 10,000 episodes for Djokovic .................. 97
Chapter 1

Introduction

Do not fear going forward slowly; fear only to stand still.
Chinese Proverb

The use of video analysis in sports is an emerging area of research. It allows both coaches and athletes to identify and correct the athlete’s technique and improve the biomechanics of the movements. Specialized video capturing software packages, which usually include drawing and measuring tools, can highlight key movements on the video and compare them with reference clips. This helps improve athletic performance by providing corrections and adjustments as needed, thanks to its instant visual feedback. Although most of these packages are geared towards biomechanical excellence and injury rehabilitation, they now allow to label different match situations, and in this way, they have also begun to contribute to tactical analysis that can help identify behavior patterns from those video sequences.

Given the difficulty to apply labels automatically, most of the software packages need an operator to organize those entries manually. This is the case of Dartfish\(^1\) a software used in many tennis academies that not only studies the mechanical aspect of the stroke but also permits to label sequences for further study. This way, at the end of a match, the player can see those sequences that he is interested in, without having to watch the whole match, whilst also combining qualitative and quantitative analysis of the number of repetitions of certain important actions. Over time, more and more computer programs are emerging in the market that will help coaches create these records for further analysis\(^2\).

\(^1\)http://www.dartfish.com
\(^2\)See for example, InterplaySports, GPSports, NACSport, Sportcode Gamebreaker, among others
CHAPTER 1. INTRODUCTION

However, after conversations with professional tennis coaches\textsuperscript{3} they indicate that players or coaches seldom use these tools due to their lack of familiarity with the technology and the need for an expert manual supervision of the sequences to be able to label them correctly. These professional coaches also indicated that the only two tools used to prepare the tactics for an upcoming match are the following:

- The viewing of recorded past matches, where the coach would comment with the player the strengths and weaknesses observed and would analyze movement and behavior in specific situations.

- The review of the official statistics provided by the ATP (Association of Tennis Professionals) or WTA (Women’s Tennis Association) during the competitions.

These two tools are far from ideal. In the first place, they imply long hours watching recorded games, only to manually identify a few tactical insights, and secondly, the interpretation of the official statistics is harsh since they do not provide context information and are difficult to use in order to prepare tactically for the following match.

This thesis addresses, therefore, the problem of creating a computer vision system that will facilitate high-level, user-friendly interpretation of an observed scene, providing a set of optimal strategies based on a real match under analysis.

To achieve this goal, a semi-automatic computer vision system is proposed. The system is intended for use with low-quality, off-air video from a single camera (unlike, for

\textsuperscript{3}Davis Cup captains, Albert Costa and Jordi Arrese, as well as Fed Cup captains, Mico Margets and Pancho Alvariño
example\cite{51}) and employs a number of low-level image processing tasks to extract relevant information. The extracted set of data enables, in turn, the creation of a cognitive framework where patterns can be identified. This framework is initially exposed to data mining techniques in order to obtain knowledge on the most frequent patterns employed by either of the players under analysis.

Following this knowledge discovery, our work builds on it to identify the optimal strategies that would allow the players improve their game and beat the opponent. This is done by exposing the framework to states, actions and rewards and transforming the tennis game into a Markov Decision Process (MDP), which is able to successfully describe the dynamic interaction between the players. This approach allows to apply the concepts of game theory, reinforcement learning or artificial networks in order to find optimal policies.

This thesis, therefore, presents a combination of ad-hoc computer vision modules that feed off real video sequences along with data mining and machine learning algorithms in an attempt to gain tennis tactical insights that could be used by tennis players or coaches alike.

1.1 Related Work

The research community has tackled the analysis of tennis tactics from two angles. The first one is by using an observational methodology in order to record and statistically analyze certain aspects of a tennis game, providing measurable, objective and quantifiable information. Under this perspective, we can find that various authors have investigated elite tennis strategy using timing factors\cite{24, 61}, shot details\cite{30, 4, 47}, positional play, distance covered\cite{31}, surface of game\cite{30, 49} and point profiles\cite{49, 46, 53, 48}. These studies help determine whether the variables considered have an effect on the strategies of elite tennis players. It is interesting to mention the work done by Barnett et al.\cite{3, 4} where he develops a tennis model based on a Markov chain model to predict outcomes of tennis matches. This model allows for players that are ahead on sets, to increase their probability of winning the set, compared to their probabilities of winning the first set. This is then followed by a revised model for games in a match that has an additive effect on the probability of the server winning a point.

The other perspective employed when tackling tennis tactics has been undertaken by the computer vision research community, feeding off in many cases from broadcast video sequences. In this case, inherent limitations such us the use of a single camera position
and standard speed/resolution characteristics impose many restrictions for an accurate analysis. In this way, broadcast tennis sequences have also been studied before with an aim to either automatically annotate the score or to classify the content for later retrieval, see Refs. [59, 6, 11, 78].

Wang et al. [68] use data mining techniques to discover patterns and playing styles in tennis videos, but they only consider relative player movements for this analysis, so other variables such as actions or absolute player positions are not taken into consideration. Wang et al. [67] take into account 58 possible patterns and try to find them in the footage using Bayesian networks. However, this pattern classification method is very error prone due to the difficulty to accurately track the tennis ball in standard broadcast images. Lames [38] focuses on relative phases of lateral displacements, but it neither includes vertical movements nor represents the reality of a professional tennis game where players do not move back to the center of the court every time.

Schroeder et al. [57] use a framework based on short term and long term memory layers that allows an incremental processing of data streams. However, the tennis model described only includes one variable: the ball landing position with essentially only two different locations (right or left). Therefore the strategic conclusions are rather simplistic. Chu et al. [12] use symbolic sequences to tackle tactics analysis. They use four locations for the players’ position, five players movement directions (up, down, left, right, still) and three player’s speed indicators (fast, medium, still) to find frequent moving patterns. However, their main objective consists of identifying match situations (passing shots, volleys and unforced errors) for the purpose of game abstraction. Player actions are not taken into consideration and there is no indication about successful or winning strategies.

Jager et al. [33] analyze the players movements in volleyball using self-organizing artificial neural networks that allow them to cluster and visualize configurations and trajectories. Terroba et al. [63] apply the concepts of pattern masks and unbalancing events to multivariate data mining in order to discover successful strategies in tennis. Vis et al. [66] apply data mining to find frequent sequences between individual players and across matches. Nevertheless, the need to obtain accurate ball landing positions for the analysis of these two latter papers makes this methodology rather impractical.

Zhu et al. [78] propose a system that is able to recognize the player’s actions such as forehand and backhand, as well as tracking the player’s position. Nevertheless the reported results only show statistics with no tactical conclusions. Recently, research on decision strategy has been applied to table tennis [43, 69, 70] covering both opponent modeling and
1.2. MOTIVATION

learning anticipation policies by means of reinforcement learning. Although there is certain similarity between this work and ours, theirs do not show any model or framework for table tennis and their results are only applied to the case of learning a policy to improve the response to the opponent’s serves, not a full game.

Other interesting applications of broadcast tennis video analysis have been proposed recently. Chang et al.\[8\] implement a system to insert advertising messages and logos projecting them back onto the court in an effective and non intrusive manner, whereas Han et al.\[25\] propose a system able to generate a mixed-reality presentation of tennis videos on mobile devices by transmitting camera parameters.

1.2 Motivation

The previous section highlights the lack of research centered around tactical discovery in tennis. The few papers that cover the subject compromise the accuracy of the data in order to fully automate the system, resulting in very simple tennis models and conclusions. The main objectives pursued on this thesis are the following:

1. Improve upon the current way of preparing a match by tennis coaches.
2. Identify the most frequent patterns displayed by a player in a game.
3. Identify optimal sequences of strokes that will improve the chances of winning.
4. Translate the results into high-level, user-friendly interpretation.

Therefore, our final goal is to be able to characterize a player by means of his most frequent patterns and also to obtain a number of successful strategies for a real tennis game, that would highlight why player 1 is beating player 2, and ultimately, identify what player 2 needs to do to reverse the situation.

1.3 Scientific Contribution

In this work, a number of techniques for effectively capturing relevant information and extracting strategic knowledge are going to be discussed and evaluated, leading to some scientific contributions in the areas of computer vision, data mining and machine learning. The major contributions achieved in this work are summarized below:
• A robust computer vision system able to record relevant information directly from the broadcast video and transform it into a notational stream of data that is incorporated into the framework. Previous work compromises the accuracy of the data and the set of recognized actions in lieu of fully automating the system, resulting in very simple tennis models. In this work, we propose a semi-automatic system that complements the computer vision automatically extracted information with a manual supervision of a rich set of actions that cover up to 168 possible strokes. We believe that this set represents the majority of actions displayed by a player in any match allowing valid strategic conclusions.

• We establish a framework for multivariate data mining based on distances and thresholds where data mining algorithms can be explored. We examine several mining problems and introduce the concept of pattern masks as a means to mine regular patterns. By splitting patterns into a prequel and a sequel, we propose an efficient algorithm to mine winning patterns, anchored on so-called unbalancing events. For the prequel we consider a distance notion based on event similarities, whereas the sequel has to comply with a Probabilistic Non-deterministic Finite Automata (PNFA).

• We create a separate framework for the game of tennis based on states, actions and rewards that provides a strategic insight into the game. Previous work focuses on event classification or simple tactical analysis based on coarse players positions. Instead, we applied the concepts of game theory and reinforcement learning to establish a comprehensive framework where optimal policies can be found.

• The Monte Carlo Tree Search (MCTS) method is adapted to the Probabilistic Non-deterministic Finite Automata (PNFA) case and we introduce a back-propagation mechanism based on the shortest path. This modification is evaluated against other algorithms and leads to a higher rate of points won.

1.4 Overview of this Thesis

The rest of the thesis is organized as follows. Chapter 2 describes the multimedia system implemented to collect tennis data from broadcast images. We present all the computer vision and camera calibration algorithms used to obtain accurate positions and actions. This system, which entails the following steps: pre-processing, court detection, court tracking,
player tracking and event annotation, is used throughout this thesis as an input to the pattern discovery algorithms.

Chapter 3 explores the use of multivariate sequential data mining techniques along with a comprehensive set of tennis-specific spatiotemporal attributes. This proves to be an effective approach in order to discover successful tennis strategies within a tennis match. To this purpose, we introduce the concepts of event thresholds, rally similarities and pattern masks and define several data mining problems. The results demonstrate that this analysis can help tennis professionals focus on the successful sequences of strokes that led to winning points during the match, as well as the discovery of frequent rallies.

Chapter 4 focuses on the identification of optimal strategies by means of artificial intelligence, artificial networks and reinforcement learning. In this chapter, we develop a Markov Decision Process-based framework where machine learning algorithms can be executed in order to identify optimal policies. In this sense, we also present a novel modification to the Monte Carlo Tree Search algorithm that proves to behave better than other popular Temporal Difference algorithms. We apply this model to several real life matches and identify the most successful policies using Monte Carlo simulations. The results show that the identified policies do allow for an increase in the percentage of points won during simulations, offering a new insight into tennis tactical analysis. The examples include recent famous tennis matches with clear tactical conclusions.

1.5 Overview of Publications

Parts of this thesis are published in the form of papers. Following we provide a list of the papers on which the different chapters are based.

Chapter 3. Data Mining and Pattern Discovery

- *Tactical Analysis Modeling through Data Mining: Pattern Discovery in Racket Sports*, International Conference on Knowledge Discovery and Information Retrieval (KDIR 2010), Valencia, Spain, 25-28 October 2010. [63]

- *Tennis Patterns: Player, Match and Beyond*, 22nd Benelux Conference on Artificial Intelligence (BNAIC 2010), Luxembourg, 25-26 October 2010. [66]
Chapter 4. Identification of optimal strategies

Chapter 2

Annotation System

It isn't the mountain ahead that wears you out; it’s the grain of sand in your shoe.

Robert W. Service

Existing notational tennis analysis usually consists of coarse measurements that summarize point outcomes: forced or unforced errors, aces, break points won, length of rallies, and so on. However, a finer analysis is needed in order to identify game patterns and successful strategies.

This detailed analysis must include information such as ball trajectories, positions of the players and the particular strokes used. In order to obtain and use this information, a transformation between the broadcast image coordinates and a reference court is needed.

In this chapter, we present the system developed and the algorithms used to capture and annotate all needed variables during a tennis game. However, the detection algorithms could be easily adapted to a number of sports such as football, badminton or volleyball.

Once all the data is captured, further analysis based on reinforcement learning, data mining or artificial networks is applied in order to identify tactical information and game patterns, as described in Chapters 3 and 4.
CHAPTER 2. ANNOTATION SYSTEM

2.1 System Description

2.1.1 Introduction

The initial approach was to design and develop a system able to capture automatically all needed variables. The University of Surrey had developed a similar system [11]. However, after using it for a short while, it was concluded that in order to obtain reliable and robust data, we would need to implement purpose-built system. The reasons for this are listed below:

• Even though the University of Surrey’s system was able to recognize and classify the forehand and backhand strokes [50, 54], the results were only correct less than 60% of the time. Also, the granularity offered was not enough for a serious tactical analysis, which would need at least ten categories.

• The ball trajectory results [74, 75] were better than the stroke detection ones, although it wasn’t 100% accurate due to the fact that only one camera was used and some occlusions were unavoidable.

• The processing speed was very slow, taking more than one hour to analyze a few points.

For these reasons and in order to ensure almost 100% data accuracy, that would enable us to identify correct strategic patterns, it was decided to implement our own system.

2.1.2 System Design

The system has been designed to work with normal speed, low resolution cameras and broadcast sequences, as opposed to other systems that employ several high speed cameras [51]. On an average, the video sequences to analyze show the following characteristics:

• Image: 640 x 480 pixels

• Frame Rate: 25 frames per second

The following figure depicts the main modules of the annotation system:

A brief description of these modules can be found below. The Compute Homography module, however, will be further explained in the following sections.
2.1. SYSTEM DESCRIPTION

![Diagram of Data Capture and Annotation System]

Figure 2.1: Data Capture and Annotation System

**Video Control**

This module is in charge of playing the recorded broadcast video as well as offering an interactive interface with the user. This interface enables the user to play frame by frame, to fast-forward, to play backwards, control the playing speed and pause the recording.

This module is also responsible for sending the selected frame to the Compute Homography module, which will process it and will return the real coordinates on the reference court model of the mouse position clicked on the broadcast image. This position is also visualized on the computer screen in a small representation of the tennis court for a quick correctness check.

**Annotate Result**

This module assigns a winner to the point just played and registers the reason for its completion. The system allows the following options:
CHAPTER 2. ANNOTATION SYSTEM

Figure 2.2: Ball position in broadcast image and in reference court

- Winner. Shot that is not reached by the opponent and wins the point.

- Forced error. Error caused by the opponent’s good play or attack.

- Attacking Error. Error caused when the player attempts to attack the opponent.

- Unforced error. Error that cannot be attributed to any factor other than poor execution by the player.

Annotate Action

This module assigns a label or tag to the annotated position. Depending on the situation, different tags have been created. Table 2.1 shows the tags allowed by the system for the different types of strokes, and Table 2.2 shows the tags used to describe the status of the ball.

Update Score

On of the most important variables to characterize the behavior of a player at a specific point in time is the score at that very moment. The implemented system uses this module to keep track of the score and automatically updates it after each point. The only information passed to this module is whether the match will be played to a maximum of 3 sets or 5 sets, and whether a tie-break will be permitted in the last set.
2.1. **SYSTEM DESCRIPTION**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>First Serve</td>
</tr>
<tr>
<td>SS</td>
<td>Second Serve</td>
</tr>
<tr>
<td>FH</td>
<td>Forehand (top spin or flat)</td>
</tr>
<tr>
<td>FHS</td>
<td>Forehand sliced</td>
</tr>
<tr>
<td>BH</td>
<td>Backhand (top spin or flat)</td>
</tr>
<tr>
<td>BHS</td>
<td>Backhand sliced</td>
</tr>
<tr>
<td>VOL</td>
<td>Volley (forehand or backhand)</td>
</tr>
<tr>
<td>SM</td>
<td>Smash</td>
</tr>
<tr>
<td>LOB</td>
<td>Lob (forehand or backhand)</td>
</tr>
<tr>
<td>DSH</td>
<td>Dropshot (forehand or backhand)</td>
</tr>
</tbody>
</table>

Table 2.1: Types of strokes annotated

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NET</td>
<td>Ball hits the net</td>
</tr>
<tr>
<td>BIN</td>
<td>Ball bounces inside the court</td>
</tr>
<tr>
<td>BOUT</td>
<td>Ball bounces outside the court</td>
</tr>
<tr>
<td>HBIN</td>
<td>Hypothetical position where the ball would bounce</td>
</tr>
<tr>
<td>ME</td>
<td>Ball has not been hit correctly</td>
</tr>
</tbody>
</table>

Table 2.2: Ball status

**Generate XML**

At the end of each game, the system generates an XML file with all the captured information. The reason to generate a file for each game is because it allows easily to resume the annotation where we stopped it and avoids a possible loss of data in case of an unexpected error.

On the other hand, we decided to separate the capture and annotation functionality from the analysis and data mining function. This way, one team could focus on the data gathering function on relatively standard PCs, where another team can explore the pattern identification in more powerful servers.

The XML structure is as follows:

```xml
<game>
  <sets_won_by_1></sets_won_by_1>
  <sets_won_by_2></sets_won_by_2>
</game>
```
CHAPTER 2. ANNOTATION SYSTEM

Compute Homography

This module is responsible for calculating the reference court coordinates of any location selected on the broadcast image. It also performs the player and court tracking. Its functioning will be described in detail in the following section.
2.2. PRE-PROCESSING

2.1.3 High Level Overview

As mentioned above regarding the analysis of sports videos, the players’ positions as well as the ball bouncing positions have a great semantic importance, since these positions and their tracked movements provide information about the actions displayed on the court. For this analysis, it is necessary to calculate these positions on a reference model.

In the next sections, we present a number of algorithms to achieve automatic calibration and tracking. These algorithms could apply to several sports providing a sufficient number of lines or references are present on the image. Figure 2.3 shows the main stages of the annotation system implemented.

Figure 2.4 shows some of the logic and module dependencies from a high level point of view.

2.2 Pre-Processing

The main purpose of this step is to extract a number of straight lines from the input frame, resulting in a set of court line candidates. The International Tennis Federation (ITF) dictates\textsuperscript{1} that all lines of the court shall be of the same color clearly contrasting with the color of the surface. In our analysis, all court lines studied were white. However, since they were not the only white objects in the images, a set of additional filters was needed. These filters will be described in the following sections and include a luminance filter to identify the white pixels on the image and discard the rest, a texture filter to remove white textured areas not belonging to court lines, i.e. white logos or white t-shirts, a linearity filter that ensures that all court lines candidates are nor wider that certain threshold and finally, we apply the probabilistic Hough transform due to its smaller computational cost and robustness on low quality images to find all the lines and intersections present in the image.

\textsuperscript{1}http://www.itftennis.com/technical/rules/index.asp
Figure 2.4: High level flowchart for the homography calculation and tracking
2.2. PRE-PROCESSING

2.2.1 Luminance Filter

Since the tennis court lines are white or light color, we apply an initial filter to remove the image pixels that do not hold this property. For this, as suggested by [23], we classify each image pixel as a possible line candidate based on its luminance, which is defined as:

\[ Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \]  \hspace{1cm} (2.1)

In this way, if the value of the luminance on a pixel, \( Y(x, y) \) is greater than a specific threshold \( \sigma_y \), then this pixel is considered a line candidate \( l(x, y) = 1 \), otherwise the pixel is discarded:

\[ l(x, y) = \begin{cases} 1 & \text{if } Y(x, y) \geq \sigma_y \\ 0 & \text{otherwise} \end{cases} \]  \hspace{1cm} (2.2)

Farin et al. [19, 20] propose a fixed threshold \( \sigma_y = 128 \) for all types of images. However, our tests have proved that an adaptive threshold value was fundamental in order for this filter to work properly on different light conditions, surface types of court colors.

To this end, we first calculate the histogram of the image on a gray scale, but to avoid that the public or the stadium contaminates the result, we crop the image and only use a central section over which we calculate the histogram. The following Figures 2.5, 2.6 show several histograms for different surfaces and different lighting conditions.

These histograms, which are calculated using 256 bins, show the presence of one or more peaks (the presence of two peaks is always a signature of grass courts) that indicate the most frequent luminance colors on the image processed. Therefore, the identification of an appropriate luminance threshold cannot be fixed for all frames in order to obtain good results.

Our tests indicate that the following threshold definition allows for a luminance filter that works correctly under any lighting condition or court surface, making in turn, the calculation of the adaptive threshold an automatic process:

\[ \sigma_y := \arg \max_y \{ y \mid h(y) \leq 0.7 \times h_{\text{max}}(y_{\text{max}}) \land y \geq y_{\text{max}} \} \]  \hspace{1cm} (2.3)

where \( h(y) \) is the histogram value for the bin \( y \) and \( y_{\text{max}} \) is the value of \( y \) for the maximum histogram value. See Figure 2.7.
Figure 2.5: Histograms of the images’ central area for a clay court (left) and a one-color hard court (right)

Figure 2.6: Histograms of the images’ central area for a two-color hard court (left) and a grass court (right)

Figure 2.7: Luminance threshold calculation
2.2. PRE-PROCESSING

2.2.2 Texture Filter

The previous luminance filter successfully removes the majority of the colors that compose the tennis court surface. However, the pixels that belong to the players’ light colored t-shirts, or the white areas of the stadium or the sponsors logos are still considered to be line candidates. An initial attempt to detect tennis court lines through the Canny filter resulted in a number of false positives (see Figure 2.9 right).

To avoid this, we had to remove the white pixels in all textured regions, and this was done by means of the structure tensor. The structure tensor is a tool that encodes the structure information around a point and provides information about the maximum signal variation direction and its orthogonal direction [7]. The tool has been used in all kind of applications, such as: edge detection, corner detection, structure analysis, optical flow estimation and especially texture analysis. In the 2D case, the structure tensor $S$, also known as autocorrelation matrix, calculated for each pixel candidate around a window of pixels of dimension $s_p$ is defined as follows:

$$S = \frac{\sum_{s_p} (\frac{\partial I}{\partial x})^2 \sum_{s_p} \frac{\partial I}{\partial x} \frac{\partial I}{\partial y}}{\sum_{s_p} \frac{\partial I}{\partial x} \frac{\partial I}{\partial y} \sum_{s_p} (\frac{\partial I}{\partial y})^2}$$

(2.4)

In our case, we have calculated the structure tensor for each pixel candidate around a window of pixels of dimension $s_p = 7 \times 7$. The calculation of the derivatives is done through a Sobel operator of size $3 \times 3$ pixels which combines a Gaussian filter to smooth out the result and make it more robust against noise. Other methods exist to estimate the structure tensor based on a set of quadratic filters along with an analysis in the frequency domain, or more recently, non-linear versions of gradient based methods [7].

The calculation of the eigenvalues and eigenvectors of this matrix provides information about the local structure of the image. Specifically, if both eigenvalues $\lambda_1, \lambda_2 (\lambda_1 \geq \lambda_2)$ are big, this means a textured area. If one eigenvalue is big and the other one small, it means that an edge with a certain orientation exists [20, 72].

We will use these properties as the criteria to remove textured areas and allow only the pixels that belong to straight lines. Our tests have shown that the next condition satisfies all requirements:

$$\lambda_1 > 5 \cdot \lambda_2$$

(2.5)
Figure 2.8: Image after applying luminance filter (left) and the result after applying a texture filter with $s_p = 7 \times 7$ (right)

Figure 2.9: Image after applying a texture filter with $s_p = 3 \times 3$ (left) and image after applying the Canny filter (right)

Figures 2.8, 2.9 shows the result after applying this filter.

### 2.2.3 Linearity Filter

After the previous filters have been processed, we apply a further constraint. We exploit the property of court lines not measuring more than $\tau$ pixels wide. This way, we check if there are pixels at a distance of $\tau$ pixels around the candidate pixel whose luminance is darker than the pixel under analysis. As suggested by Farin et al. [20], the threshold value for the luminance difference was set as $\sigma_d = 20$. Therefore, following Equation 2.2, we further consider a pixel as a line candidate $l(x, y) = 1$ or not $l(x, y) = 0$, according to the
2.2. PRE-PROCESSING

Figure 2.10: Image after applying the linearity filter. \( \tau = 1 \) (left) \( \tau = 3 \) (center) and \( \tau = 5 \) (right)

following condition:

\[
l(x, y) = \begin{cases} 
1 & \text{if } Y(x, y) - Y(x - \tau, y) > \sigma_d \land Y(x, y) - Y(x + \tau, y) > \sigma_d \\
1 & \text{if } Y(x, y) - Y(x, y - \tau) > \sigma_d \land Y(x, y) - Y(x, y + \tau) > \sigma_d \\
0 & \text{otherwise}
\end{cases}
\]  

(2.6)

Although Farin et al. suggests a value of \( \tau = 8 \) pixels, our tests showed that the optimum value was \( \tau = 3 \) pixels. Smaller values resulted in the removal of the top court lines (due to the camera perspective) and larger values brought about wider lines that would be treated by the Hough filter as multiple lines. Figure 2.10 shows the results for various values of \( \tau \).

2.2.4 Probabilistic Hough Filter

The last pre-processing step involves parametrizing the lines identified on the image so they can be further analyzed. Since this analysis is based on line segments, it was decided to use the probabilistic Hough transform [36]. Other advantages of this method over the standard Hough transform include a smaller computational cost, more robustness on low quality images and a more control parameters [36, 5, 41].

The control parameters chosen for the probabilistic Hough transform for our particular scenario are the following:

- Segment length \( \geq 37 \) pixels

- Space between segments of the same line so they can be considered as one line \( \leq 10 \) pixels
These parameters cover the requirements for tennis court lines, although they could be easily tuned to detect the lines in other sports such as soccer or volleyball. Figure 2.11 shows the result of applying the probabilistic Hough transform to Figure 2.8 on the right. Note. each segment has been displayed with a different color.

2.2.5 Selection of the Main Lines

Usually, the result of applying the Hough transform is not going to be as clean as Figure 2.11. Most of times, specially in clay courts, the result obtained will consist of a number of segments with different lengths along the direction of the lines that define the tennis court (see Figure 2.12).

However, it is necessary to extract the main lines from these segments for further analysis. A first step to obtain the main lines is to find groups of similar segments. To do so, we loop through all segments returned by the Hough filter and find groups of segments that verify the following condition:

- The angle between the segments is $< 3^\circ$
- The distance between segments is $\leq 10$ pixels

This way, all the segments that belong to the same line are grouped together. The next step is to extract a single line from each group of segments. To do so, we have considered two options and both have been implemented. A first approach consists of locating the
2.2. PRE-PROCESSING

Figure 2.12: Detected segments along the lines

farthest points of the segments of a group and then join them together to produce a single line. The main advantage of this method, which was the first one to implement, is its speed. However, it has a weakness, since it needs the farthest points to be part of the main line and this was not always the case.

Therefore, the second approach uses all the segments of the group in order to finding the main line. We do so by means of least squares linear regression analysis as described below.

**Ordinary Least Squares Linear Regression**

In this method, we try to find a line that minimizes the distance from each of the segment’s farthest points. We do this by resolving a system of linear equations for each of the segments of each group:

\[
\sum_{j=1}^{n} X_{ij} \beta_j = y_i, \quad (i = 1, 2, \ldots, m) \quad (2.7)
\]

where \(X_{ij}\) and \(y_i\) represent the \(m\) coordinates of each of the segment’s farthest points, and \(m = 2 \times n_s\), being \(n_s\) the number of segments found for each group. \(\beta_j\) represent the parameters of the approximation function and since we are trying to find a line, then \(n = 2\).
The previous equation can be written in matrix notation as:

\[ X\beta = y \]  (2.8)

The result of solving this equation system by singular value decomposition is a line that takes into account all segments’ farthest points and minimizes their distance to it. Nevertheless, it became apparent that not all segments should contribute equally to the definition of the line. Usually, the longest segments are the most representative and the ones that should have a larger weight when identifying the resulting line.

**Weighted Least Squares Linear Regression**

The previous method assumes that all residuals resulting from each of the equations that define the segments have the same variance and they are uncorrelated \[45\]. However, as mentioned above, the longest segments should contribute more towards the definition of the main line. To achieve this, we use weights adjusted depending on the length of each segment.

There are two possible options:

- To use a weighted matrix and then solving the system of equations. In this case, the diagonal matrix \( W \) is defined using the length of each segment:

\[(X^T W X) \hat{\beta} = X^T W y \]  (2.9)

- Alternatively, we could use equidistant points in each segment depending on its length, so that longer segments contribute with more points to the definition of the line. In our tests, a sampling of 30% of the total length of a segment provided good results without penalizing the process.

As a summary for all these approaches, Figure 2.13 shows the results of applying the previous methods. The figure on the left shows the segments obtained after applying the Hough filter. The figure on the right shows the outcome of the main line selection process: lines in red were obtained joining directly the segment’s farthest points, whereas white lines were obtained after applying weighted least squares. It is noticeable on the vertical line in the center that the regression method outperforms the first initial approach.
2.3 Automatic Calibration of Broadcast Images

The main purpose of this stage is to implement the appropriate algorithms to: a) relate the main lines detected during the preprocessing step to the court lines of a reference model, and b) find the geometric transform that converts the broadcast image under analysis into a projected one so that any point on the original image can be referenced back to a court model.

2.3.1 Previous Work

The initial proposed systems [59] needed to manually select four lines in order for the process to identify the rest. Also, the geometric transform were based on a simplified camera model that only considered the tilt, the distance from the camera to the court and the focal distance. Calvo et al. [6] used the Hough transform to detect the court lines but they employed heuristics to assign the detected lines to reference model based on several assumptions. Recently Farin et al. [20] proposed a system more robust to occlusions also based on the Hough transform. His method used a combinatorial search to establish the

Figure 2.13: Main line selection after applying different methods

Line Classification

Once all the main lines on the broadcast image have been identified, they are sorted into two sets. One set contains all the horizontal lines, while the other set consists of the vertical lines. Candidate lines are classified as vertical if their angle is less than 25° and horizontal otherwise.
correspondence between the image lines and the reference model. However, both the Hough transform and the combinatorial search were computationally expensive. For this reason, Farin proposed later a modification to his system [19], where he substituted the Hough transform by RANSAC-based line detector and he tried to establish the correspondences between the image lines and the reference model using two line segments instead of the four segments used originally.

Finally, Hayet et al. [29] suggested an interesting improvement on the correspondence mapping based on geometric invariants. This allowed first to calculate the homography matrix using only two points (instead of four) and secondly, to provide more robustness to large occlusions.

The system implemented in this thesis is mostly based on the projective geometry system proposed by Hayet et al.

### 2.3.2 Homography

The camera model for perspective images of planes has been studied in detail before [16, 27]. Points on a reference plane are mapped to the points on the image by a plane projective transformation or homography. If we denote the coordinates of the points on the reference plane as homogeneous vectors $P = (X, Y, 1)^T$, and the coordinates of the points on the image plane as homogeneous vectors $p = (x, y, 1)^T$, then this homography relationship can be described as:

$$P = Hp$$ (2.10)

where $H$ is a $3 \times 3$ homogeneous matrix which includes a scale factor.

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ (2.11)

In order to resolve 2.11, a system of equations with eight variables\(^2\) it is sufficient to find four correspondences between the image and the reference model with the condition that three of them cannot be collinear points.

\(^2\)Since this is a scale invariant homogeneous matrix, and only ratios are important, the number of variables can be reduced from nine to eight.
2.3. AUTOMATIC CALIBRATION OF BROADCAST IMAGES

However, those correspondences are not known a-priori. Therefore, a list of hypothesis will have to be generated and evaluated by the system. Unfortunately, this process of generation/evaluation has two major weaknesses:

- If only one of the four correspondences is wrong, the system will have to re-calculate the matrix $H$. Also, the system will not know which them is the wrong one, allowing this correspondence in further hypothesis evaluations.

- The solution to the system of equations to obtain $H$ is computationally expensive (it is done by singular value decomposition), especially if we need to make this calculation for each hypothesis evaluation.

Hayet et al. [28, 29] proposal avoids the calculation of $H$ as an evaluation method if we know the vanishing points. In this way, only two correspondences are needed instead of four, increasing the speed and the efficiency of the algorithm. We will describe this algorithm in the next section.

Radial Distortion

Although the matrix $H$ does take into account both the internal parameters of the camera (such as the focal length, the position of the camera’s principal point, the aspect ratio and the skew) and the external ones (such as rotation and translation) [13], it does not compensate the nonlinear radial distortion which becomes apparent when the focal length
where $k_i$ are the radial distortion coefficients and $r$ is the distance of the image coordinates to the center $r^2 = x^2 + y^2$. Therefore, an estimation of the $k_i$ coefficients is enough to compensate camera lens distortions for the majority of practical applications. Moreover, in cases where narrow-angle cameras capture scenes far from the camera (which is our case), an approximation of the radial distortion can be obtained by simply calculating the $k_1$ parameter. Previous work on tennis broadcast videos [11] suggest a value for $k_1$ as:

$$k_1 = 2 \times 10^{-7}$$

This value has been used for all sequence frames giving satisfactory results.

### 2.3.3 Fast Projection Algorithm

Since we are working with a set of orthogonal lines, all lines on the tennis court converge either on the horizontal vanishing point or the vertical vanishing point. However, usual tennis broadcast videos place the camera at the end of one of the sides. This makes that the service and base lines are almost horizontal (see Figure 2.15).

![Figure 2.15: Vanishing points for a typical broadcast perspective](image)

Therefore, the vertical lines identified at the end of the pre-processing phase (side lines and center lines) converge on a vertical vanishing point $p_v$, whereas the horizontal lines...
2.3. AUTOMATIC CALIBRATION OF BROADCAST IMAGES

Figure 2.16: Cross Ratio definition

(base lines and service lines) converge ad infinitum, $p_h \sim \infty$.

**Calculation of $p_v$.**

The calculation of the vertical vanishing point is a RANSAC-based algorithm [21] which finds all vertical lines that converge “near” a possible vertical vanishing point. After that, an overdetermined system of equations is resolved using least squares.

Because not many vertical lines are usually detected in our application, rather than choosing random vertical lines for a number of times (generic RANSAC algorithm), we iterate all possible combinations of vertical lines. The result of this algorithm is a set of vertical lines whose intersections are within a range of 3 pixels. All vertical lines that do not comply with this condition are removed from the final consensus (this eliminates all the vertical lines detected by the Hough algorithm which are not court lines).

Once we have selected the set of vertical lines which do converge within 3 pixels, we use least squares to determine the most accurate vanishing point. The system of equations includes the linear equations of each of the lines that are part of the final consensus. The method used to resolve the linear system $Ax = B$ was through singular value decomposition (SVD).

**The Cross Ratio**

Projective transformations do not maintain distances or distance ratios, however, a ratio of ratios or cross ratio of lengths on a line is maintained. This is the most important invariant in projective transformations [27].

---

3Even if we had 20 vertical lines, the number of possible combinations would be just 190. However, usually, no more than 6 vertical lines are detected after the pre-processing step.
The cross ratio can be defined for four collinear points or four concurrent lines. For the points displayed in Figure [2.16] the cross ratio is defined as:

\[
X_{\text{Ratio}} = \frac{|AB||CD|}{|AC||BD|} = \frac{|A'B'||C'D'|}{|A'C'||B'D'|}
\] (2.14)

As mentioned above, it is advisable not to compute the homography matrix H to evaluate each hypothesis because of its performance implications. Now, and once \( p_v \) and \( p_h \) are known, we can use the invariance property of the cross ratio to calculate the projection of any point on the reference model on to the image projective perspective.

Let us consider Figure [2.17] let us assume we want to evaluate hypothesis \( h = (p_1, P_1; p_2, P_2) \), which includes two correspondences between the image plane and the reference model. The objective is to find the position of point \( p \) on the image plane, knowing its position \( P \) on the reference model. To do so, we calculate the cross ratio of the line which includes the four points \( (Y, Y_2, Y_1, P_v) \) and its projection \( (y, v_2, v_1, p_v) \), where \( Y \) is the ordinate of point \( P \) on the y-axis, \( Y_2 \) is the ordinate of point \( V_2 \) on the y-axis, \( Y_1 \) is the ordinate of point \( V_1 \) on the y-axis. We obtain:

\[
X_{\text{Ratio}}_A(p, v_2, v_1, p_v) = \frac{p v_2 v_1 p_v}{p v_1 v_2 p_v}
\] (2.15)

\[
X_{\text{Ratio}}_B(Y, Y_2, Y_1, P_v) = \frac{Y - Y_2}{Y - Y_1}
\] (2.16)

\[
X_{\text{Ratio}}_A = X_{\text{Ratio}}_B
\] (2.17)

On the other hand, Figure [2.18] shows the points \( (p, v_2, v_1, p_v) \) on the projected image:
If we apply the Thales theorem, we obtain:

\[
\frac{pu_2}{pu_1} = \frac{p_y - v_{2y}}{p_y - v_{1y}} = \frac{u_{2x} - p_x}{u_{1x} - p_x}
\]  

(2.18)

By solving these equations, we obtain the value of \( p = (p_x, p_y) \) for the initial hypothesis \( h = (p_1, P_1; p_2, P_2) \).

Finally, it is worth mentioning that even though we have taken the cross ratio of four points as a single value, it varies depending on the order in which we take the points. There exist \( 4! = 24 \) possible permutations, although due to symmetries, there are only six unique values of the cross ratio. Table 2.3 shows the different possibilities.

<table>
<thead>
<tr>
<th>Case A</th>
<th>Case B</th>
<th>Case C</th>
<th>Case D</th>
<th>Case E</th>
<th>Case F</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_v )</td>
<td>( P_v )</td>
<td>( P_v )</td>
<td>( P_v )</td>
<td>( P_v )</td>
<td>( P_v )</td>
</tr>
<tr>
<td>( V_1 )</td>
<td>( V_1 )</td>
<td>( V_2 )</td>
<td>( V_2 )</td>
<td>( P )</td>
<td>( P )</td>
</tr>
<tr>
<td>( V_2 )</td>
<td>( P )</td>
<td>( V_1 )</td>
<td>( P )</td>
<td>( V_1 )</td>
<td>( V_2 )</td>
</tr>
<tr>
<td>( P )</td>
<td>( V_2 )</td>
<td>( P )</td>
<td>( V_1 )</td>
<td>( V_2 )</td>
<td>( V_1 )</td>
</tr>
</tbody>
</table>

Table 2.3: Different cases when calculating the cross ratio

Therefore, the calculation of the cross ratio for each of these cases allows to obtain the value of \( p = (p_x, p_y) \) for each couple of points taken as a hypothesis \( h \). We can denote this relationship as:

\[
p = \Pi_h(P)
\]  

(2.19)

The next section will detail how the system generates these hypothesis.
2.3.4 Hypothesis Generation

Once we have the main line segments parametrized from the pre-processing phase and classified as vertical or horizontal, the first step to generate a list of hypotheses is to calculate all possible intersections between the vertical and the horizontal segments. Figure 2.19 shows the terminology used to denote each of the intersections on a tennis court.

Figure 2.19: Terminology used to denote the intersections on the reference model

However, looking at each of these intersections on Figure 2.19, it is noticeable that some of them share the same shape. We can, therefore, classify them accordingly as per Table 2.4.

Table 2.4 shows that there are only eight unique shapes or categories. The next step is to associate these eight shapes with each of the intersections found on the image, allowing for a small error margin. For instance, to find the corner “A”, we allow for the following possibilities:

Figure 2.20: Shapes allowed to classify an intersection as type “A”

Let us denote a correspondence as \((p_i, P_i)\), where \(p_i\) represents a point on the image, and \(P_i\) its correspondent point on the reference model. After the classification process, we can generate two lists of correspondences. The first one will include the correspondences gener-
2.3. AUTOMATIC CALIBRATION OF BROADCAST IMAGES

<table>
<thead>
<tr>
<th>Intersection</th>
<th>Shape</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>⌞</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>⌜</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>⌝</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>⌞</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>⌜</td>
<td>$T_t$</td>
</tr>
<tr>
<td>F</td>
<td>⌝</td>
<td>$T_t$</td>
</tr>
<tr>
<td>G</td>
<td>⌜</td>
<td>$T_b$</td>
</tr>
<tr>
<td>H</td>
<td>⌝</td>
<td>$T_b$</td>
</tr>
<tr>
<td>I</td>
<td>⌜</td>
<td>$T_t$</td>
</tr>
<tr>
<td>J</td>
<td>⌝</td>
<td>$T_b$</td>
</tr>
<tr>
<td>K</td>
<td>⌜</td>
<td>$T_r$</td>
</tr>
<tr>
<td>L</td>
<td>⌝</td>
<td>$T_t$</td>
</tr>
<tr>
<td>M</td>
<td>⌜</td>
<td>$T_r$</td>
</tr>
<tr>
<td>N</td>
<td>⌝</td>
<td>$T_t$</td>
</tr>
</tbody>
</table>

Table 2.4: Classification of intersections according to their shape on the reference model

ated by the actual intersections detected on the image, in other words, those intersections to which we could assign a shape or category. To generate the list of correspondences, we just need to take into account the different possibilities between intersections and categories shown in Table 2.4.

On the other hand, and in order to make the system more robust, when none of the correspondences of the first list yields adequate results, a second list of correspondences is generated. This second list includes all possible intersections between all horizontal and vertical segments from the image. Note that this second list in only used in case of error since its evaluation is a lot more costly.

Finally, we construct a list of hypotheses, where each hypothesis $h_k = (p_i, P_i; p_j, P_j)$ is made of each of the possible combinations between two correspondences:

$$H = \{h_k\}$$  (2.20)

2.3.5 Hypothesis Evaluation

The hypothesis evaluation consists of checking its ability to justify the results observed, i.e., the detected lines and intersections on the image. In this way, if $\{p_i\}$ is the set of intersections on the image and $\{P_j\}$ is the set of intersections on the reference model, each
hypothesis \( h_k = (p_i, P_i; p_j, P_j) \) is evaluated comparing the set \{\( p_i \)\} with the set \{\( \Pi_h(P_j) \)\}.

Since the number of hypotheses to evaluate can be considerable, a RANSAC-based algorithm has been implemented. See Figure 2.21.

---

**input**
- \( H \), set of hypotheses;
- \( \{p_i\} \), set of intersections on the image;
- \( \{\Pi_h(P_j)\} \), set of projections of the model’s intersections \( \{P_j\} \) on the image under hypothesis \( h \);

**output**
- \( h_k \), hypothesis that better supports the results observed (larger number of coincidences found);
- \( \{p_i, P_j\} \), set of correspondences that hold under hypothesis \( h_k \);

**begin**

\[
\begin{align*}
\text{coincidences} & := 0 \\
\text{iterations} & := 0 \\
\text{best}\_\text{number}\_\text{of}\_\text{coincidences} & := 0 \\
\text{while} \quad \text{iterations} < \text{max}\_\text{iterations} \quad \text{and} \quad \text{coincidences} < \text{min}\_\text{coincidences} \\
& \quad \text{Choose randomly a hypothesis } h \text{ with two non-collinear correspondences} \\
& \quad \text{foreach intersection } \{P_j\} \text{ on the reference model} \\
& \quad \quad \text{Calculate its projection } \{\Pi_h(P_j)\} \text{ on the image} \\
& \quad \quad \text{foreach intersection } \{p_i\} \\
& \quad \quad \quad \text{if projection is within margin allowance error from any of the intersections } \{p_i\} \text{ and their categories coincide} \\
& \quad \quad \quad \quad 1. \text{ coincidences } + + \\
& \quad \quad \quad \quad 2. \text{ keep the correspondence} \\
& \quad \quad \text{if coincidences } > \text{best}\_\text{number}\_\text{of}\_\text{coincidences} \\
& \quad \quad \quad 1. \text{ best}\_\text{number}\_\text{of}\_\text{coincidences} = \text{coincidences} \\
& \quad \quad \quad 2. \text{ keep this hypothesis } h \\
& \quad \quad \quad 3. \text{ keep the correspondences} \\
& \quad \quad \quad \text{iterations } + + \\
& \quad \quad \quad \text{coincidences} = 0 \\
& \quad \quad \text{reset correspondences found} \\
& \quad \text{return } h_k \\
& \quad \quad \{p_i, P_j\} \\
\end{align*}
\]

**end**

---

Figure 2.21: Hypothesis evaluation algorithm

Our experimental tests have shown than the following values yield a good trade-
off between speed and efficiency: \( \text{max\_iterations} = 200, \text{min\_coincidences} = 8 \) and \( \text{margin\_allowance} = 3 \) pixels.

Therefore, the output of the algorithm would be the best two correspondences-hypothesis that produces the largest number of coincidences between the reference model and the intersections detected on the broadcast image. The output of the algorithm also provides a list with all the coincidences found under this hypothesis, i.e., all the correspondences \( \{p_i, P_j\} \) verified. This way, if the number of verified coincidences is \( n \), then we will obtain \( n + 2 \) verified correspondences \( (n < 12) \).

### 2.3.6 Homography Matrix Calculation

As explained in section 2.3.2, the calculation of the homography matrix \( H \) will allow us to obtain the position on the reference model of any point selected on the broadcast image in a single operation. This is needed for the notational analysis introduced at the beginning of this chapter. Additionally, the matrix \( H \) also allows to obtain the projection of the reference model on the broadcast image, and therefore, enabling the system to be able track the court throughout a sequence of frames.

Once the best hypothesis and the list of verified correspondences have been found as detailed in the previous section, we can now calculate the homography matrix. As mentioned in section 2.3.2, only four correspondences are needed in order to resolve the resulting system of equations, providing that three of them are not collinear. Thus, we need to select the four correspondences to be used out of the list of all possible correspondences found.

These four points shall be chosen in a way that maximize the resulting polygon area to compensate for distortion errors. The method used to maximize the area and find the four correspondences which will be used to calculate the matrix \( H \) can be seen in Figure 2.22, which uses Jarvis’ algorithm [34] to compute the convex hull of a given set of points.

Once the homography matrix has been calculated, we can use the projective transform \( P = Hp \) to find out the position on the reference model of any point selected on the broadcast image during the video playback. The resulting point, \( P = (X, Y, 1)^T \) will be passed over to the video control module for its annotation and further analysis.

We will also use the homography matrix to calculate the projection of the reference model intersections on the image frame. This will be used by the court tracking module to quickly calculate the new homography matrix in later frames. This projection calculation
CHAPTER 2. ANNOTATION SYSTEM

input
\{p_i, P_j\}, set of verified correspondences under hypothesis \(h_k\);

output
\{p_k, P_l\} \in \{p_i, P_j\}, set of four correspondences for which \(\{p_k\}\) produces a max area;

begin
\text{max_area} := 0

foreach possible combination of four correspondences

1. Find the convex hull of the four points chosen, numbering the vertices
2. Find the area by calculating its cross product
   \(\text{area} = \frac{1}{2} \sum_{i=0}^{3} (x_i y_{i+1} - x_{i+1} y_i)\)

   \text{if area} > \text{max_area}
   \begin{align*}
   1. & \quad \text{max_area} = \text{area} \\
   2. & \quad \text{Keep this combination of correspondences}
   \end{align*}

return
\{p_k, P_l\}

end

Figure 2.22: Algorithm to find the optimum four correspondences

is also direct:

\[ p = H^{-1} P \quad (2.21) \]

Figure 2.23 shows the result of the identifying all court points (in green) and projecting them along with the court lines back onto the broadcast image using the homography matrix.

2.3.7 Results

The detection algorithms have been applied to a total of 2,143 frames from several TV DVB broadcast tennis games converted to avi format using the MPEG-4-based XviD codec. The analyzed games cover all surface courts (hard court, grass court and clay court) and under several lighting conditions. Figures 2.24, 2.25, 2.26, 2.27 show several specially difficult frames. These figures show the use of the homography matrix to project (warp) the image on the reference model. Figure 2.24 shows a large shade that darkens most of the court line pixels, thus making difficult to detect the lines. In this case, the system calculates all possible intersections and successfully identifies a correct hypothesis. Figure 2.25 shows
2.4. COURT TRACKING

Figure 2.23: Reference model projected back onto the broadcast image

A clay court with different luminance levels according to the humidity of the clay, which varies from one area to another. This is typical from clay court images that also experience a further challenge: players, when sliding on the court, place some of the clay onto the lines and corners, removing this way those references. However, the proposed system is able to correctly perform under these constraints due to its robustness.

In our tests, the system is able to detect and perform the tracking in a very effective way. In most cases, detection is performed in 10ms, needing up to 50ms in the most complicated cases under a quad core processor at 2.49GHz.

2.4 Court Tracking

Once the homography matrix has been calculated for the initial frame and in order to perform a fast calculation in the following frames, the reference model points are projected on frame $t+1$ using frame’s $t$ homography. This way, the set of points $\{H_t^{-1}(P_j)\}_{t+1}$ is compared with the resulting intersections detected in frame $t+1$, $\{p_i\}_{t+1}$.

The comparison is similar to the one used during hypothesis evaluation, i.e., the intersections must be within a margin allowance distance and their categories must coincide. If the result of this comparison is greater than a certain threshold, the system takes the correspondences in $t+1$ as correct and calculates a new $H(t+1)$ with these values. The threshold established was 8 correct coincidences. If this number is not reached, more
Figure 2.24: Court detection: grass court with shades

Figure 2.25: Court detection: clay court with different lighting conditions
2.4. COURT TRACKING

Figure 2.26: Court detection: indoors hard court

Figure 2.27: Court detection: outdoors hard court
hypotheses get generated and evaluated as described above.

In section 2.2.2 the structure tensor was introduced as a tool that allows to encode the structure information around a point and obtain information such as signal variation, both in the direction of maximum variation as well as in its orthogonal direction. Back then, we were interested in identifying textures in the image.

Likewise, in order to perform the tracking of the tennis court, we need to identify the points that display changes in two orthogonal directions. The use of second derivatives proves useful since it discards areas with uniform gradient. This way, as mentioned before, corners characterize for presenting a structure tensor with two large eigenvalues. However, it would not be efficient to calculate the autocorrelation matrix around each point of the image, since spectators, logos, etc will produce corners that we do not want to track. Also, we now “know” where the court lines area, so we should focus on those points.

To do so, we create a trapezoid-like mask around the court so only the image area that contains the court is analyzed. Then, we identify the image corners using the autocorrelation matrix. To avoid candidates coming from the players or other objects inside the court, we perform a second filter of the resulting points. As we know the homography and know where the court corners are located in the initial frame, we select those pixels as the only candidates to calculate the tracking.

The tracking is performed by identifying the corners displacements between the previous frame and the current one. This is usually known as sparse optical flow methods, as they track an already specified subset of points rather than the whole image.

For our application, we will use the Lucas-Kanade algorithm [39]. This algorithm assumes small and coherent movements from one frame to the next. However, this does not hold for areas as large as the tennis court. Also, we might track the court between frame \( n \) and frame \( n+m \), \( m \geq 1 \), rather than consecutive frames, so the corner displacement could be potentially considerable (\( m \) could be as large as 30). To overcome these constraints, we make use of a pyramidal technique.

Figure 2.28 depicts the use of the pyramidal approach working with the Lucas-Kanade algorithm, which tracks starting from the highest level of an image pyramid, i.e. lowest resolution and reduced size and working down to lower levels, i.e. the image with the original pixels. The image pyramids are produced using a Gaussian kernel and then removing every even-numbered row and column [55]. This results in an image that is exactly a quarter of its predecessor’s area.
This approach enables to minimize the movement assumptions and calculate the pixel displacements faster and allowing larger movements. The following Figures show an example of the court tracking. The frames are non-consecutive, which imply large pixel displacements.

Figure 2.29 shows the initial frame $n$ and the frame $n + m$ that we want to track. Let us assume that the homography in frame $n$ is known.

Figure 2.29 left shows the identified points worth tracking after applying the autocorrelation matrix to the left image of Figure 2.29. We can see that the identified points (in red) are within the tennis court limits. Figure 2.30 on the right shows the displacement of these points. Note the two points on this image that display different vector flows to the rest of the tracked points.
It is interesting to note that the Lucas-Kanade algorithm is able to track both translation and rotation movements of the camera, which is something noticeable in this type of application.

As seen in the right Figure 2.30, not all points and vector flows returned by the algorithm are necessarily correct. To resolve this problem, a RANSAC-based algorithm has been implemented. The main objective of this algorithm is to identify the best four points that will allow to calculate the homography matrix in frame $n + m$. See Figure 2.31.

Figure 2.32 shows the results of applying this algorithm over the points obtained in Figure 2.30 right. The robustness of this algorithm removes the outliers that can appear as a result of the autocorrelation matrix results.

### 2.5 Conclusions

In this chapter, we have detailed a system able to annotate positions selected on the perspective broadcast image and automatically convert them into reference coordinates on a 2D model without manual intervention. These annotations will be used later in the Thesis for further analysis. The system is also able to efficiently track the court movements along sequences for a fast court detection. The main contribution of this work is the robustness and improvement over previous systems. Future work on this part includes a more intuitive GUI (graphical user interface), and computational optimizations on the line detection module. The Hough algorithm, as
2.5. **CONCLUSIONS**

input

\{p_{LK}\}, set of points returned by the Lucas-Kanade algorithm, representing:

the tracked corners from frame \(n + m\)

\{p_n\}, set of points that represent the court corners to be tracked from frame \(n\);

output

\(H_{n+m}\), homography matrix in frame \(n + m\);

begin

\textit{coincidences} := 0

\textit{iterations} := 0

\textit{best\_number\_of\_coincidences} := 0

\textbf{while} \textit{iterations} < \textit{max\_iterations} \textbf{and} \textit{coincidences} < \textit{min\_coincidences}

Choose randomly four points from \(\{p_{LK}\}\) (three of them non-collinear)

Calculate \(H\) with those four points

\textbf{foreach} point \(p \in \{p_n\}\)

Obtain \(P\) from the reference model

Calculate \(d = H^{-1}P - p\)

\textbf{if} \(d < \text{margin\_allowance}\)

\textit{coincidences} + +

\textbf{if} \textit{coincidences} > \textit{best\_number\_of\_coincidences}

1. \textit{best\_number\_of\_coincidences} = \textit{coincidences}

2. keep this four points

3. keep \(H\)

\textit{iterations} + +

\textit{coincidences} = 0

\textbf{return}

\(H_{n+m}\)

end

Figure 2.31: Algorithm to find the homography matrix in frame \(n + m\)

well as the texture analysis in the pre-processing state, although robust, can be slow in certain situations.

Further improvements could include the ball tracking and the automatic action detection, to make the system fully automatic.
Figure 2.32: Final result of the tracked court
Chapter 3

Data Mining and Pattern Discovery

Do not go where the path may lead, go instead where there is no path and leave a trail.
Ralph Waldo Emerson

The analysis of tennis sequences has been studied before with an aim to either automatically annotate the score or to classify the content for later retrieval [11, 50, 78, 51, 6]. This analysis and the methods to recognize and classify the images have been usually undertaken by the computer vision research community, as stated in Chapter 2. However, the study of the captured data in order to find patterns and relationships between variables [56, 73, 62] so that tactical and strategic knowledge of a tennis game can be extracted is relatively novel. The objective of this chapter is to establish a framework and methods that allow us to obtain such knowledge. This framework is based on mining structured data, taking into consideration a comprehensive set of influencing variables captured during a tennis game.

Several contributions can be drawn from this chapter. We establish a framework for multivariate data mining based on distances and thresholds. We examine several mining problems and introduce the concept of pattern masks as a means to mine regular patterns. By splitting patterns into a prequel and a sequel, we propose an efficient algorithm to mine winning patterns, anchored on so-called unbalancing events. For the prequel we consider a distance notion based on event similarities, whereas the sequel has to comply with a nondeterministic finite automaton. Finally, we apply the algorithms and methods to real
world examples and extract novel knowledge in the sports strategy arena. In this way, where current analysis simply states winner percentages, we are able to indicate how these winners were performed and how they relate to each other.

The rest of the chapter is organized as follows. In Section 3.1 we formalize a tennis game and present some definitions used in the sequel. In Section 3.2 we define the concepts of multivariate similarity, similarity thresholds and pattern masks, as well as the mining problems to solve. In Sections 3.3, 3.4, 3.5, we present the algorithms used to solve these mining problems. Finally, we present the results obtained in Section 3.6 and the conclusions to the study in Section 3.7.

3.1 Formalization

In this section we explain how we formalize a tennis match between two players, 1 and 2. For the rules of tennis, the reader is referred to [32].

Although many computerized systems exist for collecting and managing observational data[35], our need to record the exact position of the players and the ball on the court, forced us to develop a stand alone application as explained in Chapter 2. This allowed us to calculate those positions on a reference court model by means of computer vision algorithms and camera calibration techniques. Along with the players and ball position, other relevant variables were also collected as part of our sequential data.

As a first observation, our data can be structured at several levels, see Figure 3.1.

![Figure 3.1: Structured Information](image)

Each level has its own different attributes, for instance:

---

1 This is especially relevant in the area of behavior research analysis.
3.1. **FORMALIZATION**

1. Tournament level: category of the event, surface, altitude, indoor/outdoor, etc.

2. Match level: players, weather, ATP points at stake, round, etc.

3. Set level: number of sets needed to win match, etc.

4. Game level: player serving, serving/returning for the set, serving/returning for the match, etc.

5. Point level: game points, set points, match points, number of strokes during the rally, distance covered by player, length of the point, speed of player, outcome of the point, winner of the point, etc.

6. Stroke level: stroke type, position of players, speed of the ball generated after the stroke, landing position of the ball, change in the trajectory of the ball, etc.

Note that some of the attributes like speed, distance covered, etc. are not captured directly at the time the data is being gathered, but rather calculated later on during a post-processing stage.

### 3.1.1 Definitions

We will consider an event as a single stroke episode. This event will contain all attributes that characterize the stroke, i.e., the player that hits the stroke, the type of stroke, the position of both players at the time of hitting the ball, the position of the landing ball on the opponent’s side after the stroke, the generated speed of the ball, etc. A rally, on the other hand, refers to the sequence or series of events that completely describe the strokes exchanged by the players during a game point. In other words, a rally will always start with a service and will end with the final stroke that leads to the conclusion of the point.

We will also define a partial rally as a subsequence of a rally. Partial rallies are made of consecutive events, with players alternating. For instance, looking at rally \( (A, B, C, D, E) \), then \( (B, C, D) \) is a partial rally, whereas \( (B, D) \) is not.

We will also define the player’s strategy as a subsequence of consecutive events for just that one player. Therefore, it will only include the events generated by one player, regardless of the opponent’s. This way, a serve and volley strategy will include these two events regardless of the service return by the opponent.

Finally, we define the stroke exchange equilibrium as a partial rally that includes the series of events that have no interest to our study, since they do not force or put the players
under any kind of pressure. During a point, one can observe easily phases of attack/defense and equilibrium that interchange.

3.1.2 Reference Model

The coordinates of events are as suggested in Figure 3.2. All integer coordinate pairs are in $C = \{0, 1, \ldots, 316\} \times \{0, 1, \ldots, 768\}$. The positions between $(0, 0)$ and $(316, 768)$ represent coordinates both inside and outside of the court, being $(50, 150)$ and $(266, 618)$ the coordinates of the top left corner and the bottom right corner of the doubles court respectively. This reference system gives us $2.5\,m$ of space at each side of the doubles sidelines and $7.5\,m$ at each side of the baselines which is plenty to capture all the action within a match.

It is worth noting that scenarios where the player hits the ball before the bounce, i.e., a volley or a smash, will include a hypothetical bouncing ball position in order to guarantee that all sequences have the same structure. For instance, let’s consider the following scenario: Player 1 goes to the net, Player 2 tries to pass him with a passing shot, but Player 1 is able to volley. In this case, the passing shot by Player 2 will not have a corresponding ball bouncing position. However, we will record a ball position that represents a rough approximation of where the ball would have bounced.

\footnote{The study of these phases makes also an interesting research topic.}
Also, note that because the players change sides every couple of games, a transformation in the coordinates is needed so that the data to be mined is always coherent.

### 3.1.3 Attributes Considered

We will now first focus on the **rally or stroke level**. An event is a 7-tuple \((pl, st, P_1, P_2, P_3, sb, us)\) describing a single stroke, its attributes being:

- **pl**: player hitting the ball, \{1, 2\};
- **st**: stroke type, \{FS, SS, FH, FHS, BH, BHS, VOL, SM, LOB, DSH\}\(^3\)
- **\(P_1 = (x_1, y_1)\)**: position of the player when the ball is hit, \(C\);
- **\(P_2 = (x_2, y_2)\)**: position of the opponent when the ball is hit, \(C\);
- **\(P_3 = (x_3, y_3)\)**: position of the ball when it bounces on the opponent’s half of the court, \(C\);
- **\(sb\)**: speed of the ball generated after the stroke, \{slow, normal, fast\};
- **\(us\)**: unbalancing stroke that breaks the exchange equilibrium, \{0, 1, 2, 3\}.

Most attributes are self-explanatory, however, attribute \(us\) is worth mentioning as it represents the intention of one player to attack and destabilize the rally with his/her stroke. The non-zero values indicate whether it is a first, second or third attack. Unbalancing events will be explained in detail later in the chapter as they are the key to winning patterns.

As an example, a sequence including the first events within a rally might look like this:

\[
\langle (2, \text{FS}, (142, 618), (231, 56), (163, 267), \text{fast}, 1), \\
(1, \text{BHS}, (191, 64), (134, 610), (103, 566), \text{slow}, 0), \\
(2, \text{FH}, (78, 608), (173, 55), (108, 239), \text{fast}, 2), \ldots \rangle
\]

Other attributes at the **point** and **game** level worth considering are the following: (Note. In this thesis, they have been used to filter out datasets depending on the objectives pursued).

\(^3\)They correspond to: first serve, second serve, forehand, forehand sliced, backhand, backhand sliced, volley, smash, lob and drop shot, respectively as per Table 2.1
• $ps$: player serving, $\{1, 2\}$;
• $gp$: game points, $\{-6, \ldots, 6\}$;
• $sp$: set points, $\{-6, \ldots, 6\}$;
• $mp$: match points, $\{-6, \ldots, 6\}$;
• $op$: outcome of the point$^4$, $\{1, \ldots, 4\}$;
• $wp$: winner of the point, $\{1, 2\}$.

The negative values of the game, set and match point attributes refer to points against, for instance, $gp = -2$ means 2 breakpoints against the player that is serving. This convention eliminates the need to use the normal tennis score mechanism and provides all needed information to include the score as an input parameter.

Note that the series will be of different lengths. In the following sections we will try to mine sequences of these events.

3.2 Multi-variate similarity-based sequential pattern mining

A rally consists of a series of single events, as defined in the previous section. Every rally has at least one event. In the case of an ace, the rally indeed consists of only one event.

We are interested in finding patterns in these sequences. We can consider sequences within one point, one game, one set, one match, and even between matches. In the latter case, we might consider the same players, but also different ones.

3.2.1 Similarity measure

First, we need to define a similarity measure $\text{sim}$ between individual events. In this case, an event is represented like this:

$$e = (pl, st, P_1, P_2, P_3, sb, us) = (pl, st, (x_1, y_1), (x_2, y_2), (x_3, y_3), sb, us)$$

Then, if $e' = (pl', st', P_1', P_2', P_3', sb', us')$ and $pl = pl'$, a first proposal for a similarity measure between events $e$ and $e'$ is:

$^4$These values are mapped to: Winner, Forced error, Attacking Error and Unforced error respectively.
3.2. MULTI-VARIATE SIMILARITY-BASED SEQUENTIAL PATTERN MINING

\[ sim(e, e') = \text{simplayer}(P_1, P_1') + \text{simplayer}(P_2, P_2') + \text{simball}(P_3, P_3') \]
\[ + \text{simstroke}(st, st') + \delta(sb, sb') + \delta(us, us') \]  
\[ (3.1) \]

where each function determines the similarity between the corresponding attributes. We state initially that \( sim(e, e') = 0 \) if \( pl \neq pl' \), although we will revisit this assertion in Section 3.5.

If \( dist(P, Q) \) represents the Euclidean distance between points \( P \) and \( Q \) then:

\[ \text{simplayer}(P, Q) = f(dist(P, Q)) \in [0, 1] \]  
\[ (3.2) \]
\[ \text{simball}(P, Q) = g(dist(P, Q)) \in [0, 1] \]  
\[ (3.3) \]
\[ \text{simstroke}(st, st') = \delta(st, st') + \epsilon(st, st') \in [0, 1] \]  
\[ (3.4) \]
\[ \delta(u, v) = \begin{cases} 
1 & \text{if } u = v \\
0 & \text{otherwise}
\end{cases} \]  
\[ (3.5) \]

Here we have used suitable monotonically decreasing functions \( f \) and \( g \) with \( f(0) = g(0) = 1 \). The function \( \epsilon \) allows for additional weight in the case of near equal stroke types. All of the six terms can get their own weight, if necessary (See Section 3.4.2). Note that \( 0 \leq sim(e, e') \leq sim_{\text{max}} \) for suitable \( sim_{\text{max}} \leq 7 \).

Now that we have defined the similarity between events, we can easily determine the similarity \( sim(seq_1, seq_2) \) between same-length sequences (or partial rallies) \( seq_1 \) and \( seq_2 \) of single events as follows. If the length of both sequences equals \( n \) and \( seq_1 = \langle e_1, \ldots, e_n \rangle \) and \( seq_2 = \langle e'_1, \ldots, e'_n \rangle \), then:

\[ sim(seq_1, seq_2) = \sum_{i=1}^{n} sim(e_i, e'_i) \]  
\[ (3.6) \]

If the sequences are of unequal length, we define initially their similarity to be 0. However, as we will see in Section 3.4, two sequences of different lengths can still be similar when we extend the similarity definition to include winning patterns by means of pattern masks.

Finally, in order to find frequent patterns shared by both players, we define the following
CHAPTER 3. DATA MINING AND PATTERN DISCOVERY

function:
\[ R_{180}(x,y) = (316 - x, 768 - y) \quad \forall (x,y) \in C \] (3.7)

as a 180° rotation around the center of the court. We can further extend this concept to the event definition:
\[ R_{180}((pl, st, P_1, P_2, P_3, sb, us)) = (3 - pl, st, R_{180}(P_1), R_{180}(P_2), R_{180}(P_3), sb, us) \] (3.8)

and extend it also to the sequence definition in the similar fashion.

3.2.2 Visualization

Since we are interested in finding proximities in multivariate data, any of the techniques used in multidimensional scaling aiming to produce a geometric representation of this data will allow us to visualize the similarities or dissimilarities present.

In order to do this, we can define the distance between two events \((e, e')\) as follows:
\[ \text{dist}(e, e') = \text{sim}_{\text{max}} - \text{sim}(e, e') \] (3.9)

and
\[ \text{dist}(seq_1, seq_2) = (\text{sim}_{\text{max}} \times n) - \text{sim}(seq_1, seq_2) \] (3.10)

where \(n\) is the sequence length.

Figure 3.3 shows a proximity example for a small data sample containing 3 rallies and 32 events using torus multi-scaling dimensioning analysis techniques. Each event is represented by a dot.

3.2.3 Similarity thresholds

Once we know the similarity value between events \(\text{sim}(e, e')\) and sequences \(\text{sim}(seq_1, seq_2)\), we need to establish the criteria by which we will consider two events or two sequences as similar in order to proceed with the pattern mining. We will use the thresholds \(\text{event}_{\text{thr}}\) and \(\text{series}_{\text{thr}}\) for this matter. Note that we are defining two different thresholds to allow greater flexibility. This way, two events \(e\) and \(e'\) will be considered similar if and only if:
3.2. MULTI-VARIATE SIMILARITY-BASED SEQUENTIAL PATTERN MINING

Figure 3.3: Sequence similarity visualization

\[ \text{sim}(e, e') \geq \text{event}_\text{thr} \]  \hspace{1cm} (3.11)

and likewise, two sequences \( seq_1 \) and \( seq_2 \) of length \( n \) will be considered similar if and only if:

\[ \text{sim}(seq_1, seq_2) \geq \text{series}_\text{thr} \times n \]  \hspace{1cm} (3.12)

The concept of a \( \text{series}_\text{thr} \) makes it possible to find long similar sequences which might present higher event similarity at some points than others, and that a simple \( \text{event}_\text{thr} \) filter would block.

Given these thresholds, we can now define the so-called support in a match \( M \) for a sequence \( seq \):

\[ \text{support}_M(seq) = \sum_{\text{seq}'\in M} \text{sim}(seq, seq') \]  \hspace{1cm} (3.13)

Being similar requires the rallies to satisfy inequalities \[3.11\] and \[3.12\]. Note that it is
not necessary that seq is a subrally from M itself, it can even be any synthetic sequence. If we want to find common patterns for both players, as opposed to the normal support, the rotated sequences must also be taken into account. In that case the most similar sequence, i.e., the one having the maximum similarity, contributes to the support computation. So we define:

\[
\text{support}_{\text{rot}}_{M}(\text{seq}) = \sum_{\text{seq}^{'}\in M} \max(\text{sim}(\text{seq}, \text{seq}^{'}), \text{sim}(R_{180}(\text{seq}), \text{seq}^{'}))
\]

(3.14)

### 3.2.4 Mining Problems

Now that we have a notion of distance between sequences, we can define our mining problems:

**Mining problem 1: Frequent Rallies** Given a match between two players, determine the (partial) rallies that occur most. More precisely, given an event threshold \( \text{event}_{\text{thr}} \), a series threshold \( \text{series}_{\text{thr}} \), and a minimum support threshold \( \text{min}_{\text{support}} \), determine those partial rallies \( r \) in the match that have at least \( \text{min}_{\text{support}} \) partial rallies \( r^{'} \) which satisfy that \( \text{sim}(r, r^{'}) \geq \text{series}_{\text{thr}} \times n \) (where \( n = \text{length}(r) = \text{length}(r^{'}) \)). Such a rally is called a frequent rally.

**Mining problem 2: Winning Rallies** Given a match between two players, determine the partial rallies that lead to winners or forced errors. In this case, we are not so interested in finding very close similar partial rallies like in the previous case, but similar attacking patterns that may bring about different defensive responses that do not have to be exactly similar. More precisely, given a pattern mask \( p_{\text{mask}} \) of a certain length \( n \) with corresponding thresholds, and a minimum support threshold \( \text{min}_{\text{support}} \), determine those partial rallies \( \text{seq}_1 \) in the match that end with an unbalancing event, and for which there are at least \( \text{min}_{\text{support}} \) partial rallies \( \text{seq}_2 \) that satisfy Equation [3.21] Such a rally \( \text{seq}_1 \) is called a winning (partial) rally.

**Mining problem 3: Anonymous Frequent Rallies** Given a minimum support threshold \( \text{min}_{\text{support}} \), an event threshold \( \text{event}_{\text{thr}} \) and a series threshold \( \text{series}_{\text{thr}} \), determine those partial rallies \( \text{seq} \) of length \( n \) in the match \( M \) for which \( \text{support}_{\text{rot}}_{M}(\text{seq}) \geq \text{min}_{\text{support}} \).
3.3 Mining Problem 1

The main process to resolve Mining Problem 1 can be described with the pseudo-code shown in Algorithm 3.4 below. First, an adaptation of the PrefixSpan algorithm \[52\] called Similarity-based PrefixSpan (see Section 3.3.1) is applied. The output of this algorithm is a set of similar partial rallies as per definitions in Section 3.2.3 of different sequence lengths, each one with a corresponding support value.

Finally, an iterative process extracts the centroids of those partial rallies and clusters them until a set of centroids that are not similar between each other is found. We call these patterns virtual partial rallies as they are an approximation of the real partial rallies occurring within the match.

---

input

\( R \), a set of rallies;
\( \text{event\_thr} \);
\( \text{series\_thr} \);
\( \text{min\_support} \);
n, sequence length;

output

\( V \) a set of virtual rallies along with their support;

begin

PrefixSpan\( (R, \text{min\_support}) \)

foreach group of similar patterns found

Calculate initial centroids \( C \)

Record the support for each of the groups, \( \text{weights} \)

Similarity-based DBSCAN\( (C, \text{series\_thr}, n) \)

while \( \text{cluster\_size} \neq 1 \forall \text{clusters} \)

Calculate new centroids, \( C' \), using the \( \text{weights} \)

Similarity-based DBSCAN\( (C', \text{series\_thr}, n) \)

end

---

Figure 3.4: Virtual rallies identification

3.3.1 Sequential Pattern Mining

As mentioned in Section 3.1.1 we are interested in finding partial rallies made of consecutive events. Sequential pattern mining, which discovers frequent subsequences as patterns
in a sequence database [1] is an important data mining problems with broad applications. Although several algorithms have been proposed (see [1, 58, 26, 76]), we have adapted the PrefixSpan algorithm [52], which is more efficient pattern-growth algorithm for mining frequent patterns without candidate generation.

PrefixSpan recursively projects a sequence database into a set of smaller projected sequence databases and grows sequential patterns in each projected database by exploring only locally frequent fragments[52]. This algorithm also defines the concept of pseudoprojection to reduce the number of physical projected databases to be generated. The algorithm is described in Figure 3.5, where the main idea is to divide the sequence database by frequent prefix and to grow the prefix-spanning patterns in depth-first search fashion.

Before presenting the algorithm, a few definitions are needed. If we denote \( \alpha \) as a sequential pattern in the sequence database \( R \), then we define the \( \alpha \)-projected database, \( R|_{\alpha} \), as the collection of postfixes of sequences in \( R \) with regards to prefix \( \alpha \). Note that this definition includes similarity-based comparisons. Even though the PrefixSpan algorithm only needs two inputs (the sequence database, \( R \) and the minimum support threshold \( \text{min_support} \)), its recursive behavior is controlled by three parameters (\( \alpha \) which is the sequential pattern; \( l \), which is the length of \( \alpha \); and \( R|_{\alpha} \) which is the \( \alpha \)-projected database if \( \alpha \neq \{\} \), otherwise, it is the sequence database \( R \)). The algorithm is started by calling it initially \( \text{PrefixSpan} (\{\}, 0, R) \).

---

**Figure 3.5: PrefixSpan Algorithm**

<table>
<thead>
<tr>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R ), a set of rallies;</td>
</tr>
<tr>
<td>( \text{min_support} );</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>The complete set of sequential patterns;</td>
</tr>
</tbody>
</table>

| function PrefixSpan \( (\alpha, l, R|_{\alpha}) \) |
|-----------------------------------------------|
| begin |
| \( B \leftarrow \{ b | (r \in R|_{\alpha}, b \in r) \land (\text{support}_{R|_{\alpha}}(b) \geq \text{min_support}) \} \) |
| foreach \( b \in B \) |
| Append it to \( \alpha \) to form a sequential pattern \( \alpha' \) |
| foreach \( \alpha' \) |
| Construct a \( \alpha' \)-projected database \( R|'_{\alpha} \) |
| Call PrefixSpan \( (\alpha', l + 1, R|'_{\alpha}) \) |
| end |
Let’s consider the three rallies represented in Figure 3.6:

\[ r_1 = \langle ABCDE \rangle \]
\[ r_2 = \langle FGHIJ \rangle \]
\[ r_3 = \langle KLMNOP \rangle \]

The calculation of the similarity matrix between all events contained in those three rallies allows us to obtain the visual representation shown in Figure 3.7.

In this example, the steps to mine the rallies are as follows:

1. Scan the database once to find all the frequent events based on the \( \text{event}_{thr} \) and count the support for each one.

2. Construct recursively the projected databases, checking both the individual event distance through the \( \text{event}_{thr} \) and the sequence distance through the \( \text{series}_{thr} \) every time we append a new event to the pattern. For instance, event A is similar to K,
so we project the database. Now, $B$ is also similar to $L$ and the sequences $\langle AB \rangle$ and $\langle KL \rangle$ also satisfy equation 3.12, so we append it to the pattern and continue projecting the database until the thresholds are not reached or $support < min\_support$.

3. Continue with the next frequent event and repeat step 2.

### 3.3.2 Clustering

It is apparent from Figure 3.3 that clusters of similar patterns can be identified. Those clusters represent similar sequences as per definition of equation 3.12. Therefore, if we are able to find the centroids of these clusters, we will identify the partial rallies that best represent those repetitive patterns.

Although other clustering algorithms exist, we decided to use the DBSCAN algorithm [18] because it does not require to know a priori the number of clusters in the data as opposed to the k-means algorithm. It can also work with arbitrary shaped clusters, performs well with noisy data and requires only two parameters to function.

As the algorithm had to work with multi variate similarity-based constraints, a modified version of the algorithm devised by Ester et al. [18] is depicted in Figure 3.8, where we impose no condition over the minimum number of points to form a cluster (i.e. $MinPts = 0$).

We run the clustering algorithm for each group of partial rallies identified as similar. To find a cluster, we start with an arbitrary sequence from the input set of rallies and we calculate all its similar sequences as per the definition in Equation 3.12. If this sequence is what Ester at al. denote a core point (see Figure 3.9), this process create a cluster. If the
3.3. MINING PROBLEM 1

input
- \( R \), a set of rallies;
- \( \text{series}_{\text{thr}} \);
- \( n \), sequence length;

output
- \( C \) a set of clusters found;

begin
\( C = 0 \)
\( \text{foreach} \) unvisited sequence \( s \) in \( R \)
  \( \text{Mark} \) \( s \) as visited
  Find the set of neighbors \( J = \{ j \} \) / \( \text{sim}(j, s) \geq \text{series}_{\text{thr}} \times n \)
  \( C \) = next cluster
  Add \( s \) to \( C \)
\( \text{foreach} \) sequence \( s' \) in \( J \)
  \( \text{if} \) \( s' \) is not visited
    \( \text{Mark} \) \( s' \) as visited
    Find the set of neighbors \( J' = \{ j' \} \) / \( \text{sim}(j', j) \geq \text{series}_{\text{thr}} \times n \)
    \( J = J \cup J' \)
  \( \text{if} \) \( s' \) does not belong to any cluster
    Add \( s' \) to cluster \( C \)
end

Figure 3.8: Similarity-based DBSCAN Algorithm

sequence is a border point, then the algorithm visits the next sequence in the database.

As mentioned above, the algorithm’s two needed parameters \( \epsilon \) and \( \text{minPts} \) were set to:

\[
\epsilon = \text{series}_{\text{thr}} \times n \\
\text{minPts} = 0 \quad (3.15)
\]

where \( n \) represents the length of the sequence. These settings indicate that we are not requiring a minimum number of points to form a cluster and that the condition for a sequence to be considered part of a cluster is that it satisfies Equation 3.12.
3.3.3 Weighted centroids

As stated earlier, when clusters of similar sequences are formed, the identification of their centroids will provide us with the virtual sequences that best represent them. However, the support of each cluster, defined by the number of points that belong to the cluster, must be kept for subsequent clusterings. This way, a centroid found in a cluster of, say, ten points should have more importance or weight that another centroid found in a cluster of just two points.

The centroid calculations shown below take into account these weights during the algorithm’s iterative process, until a set of centroids that are not similar between each other is found.

Due to the heterogeneous nature of the data, the calculation of the centroid of a set of sequences is not obvious. The formal description of the problems is as follows. Be $G = \{r_1, r_2, \ldots, r_m\}$ a group of $m$ similar partial rallies of length $n$, each group with a support $w_i, 1 \leq i \leq m$. The partial rallies satisfy:

$$r_i = \langle e_{i1}, e_{i2}, \ldots, e_{in} \rangle / \sim(r_i, r_k) \geq \text{series}_{\text{thr}} \times n \text{ for all } i, k = 1, \ldots, m$$

(3.16)

where $e_{ij} = (p_{ij}, s_{ij}, P_{1ij}, P_{2ij}, P_{3ij}, b_{ij}, c_{ij}).$

We are interested in finding the $n$-length rally $r' = \langle e'_1, e'_2, \ldots, e'_n \rangle$ such that each of its attributes represent the centroid of each of the events’ attributes for all $m$ partial rallies provided.

If $e'_j = (p'_j, s'_j, P'_{1j}, P'_{2j}, P'_{3j}, b'_j, c'_j, f'_j)$, then the centroids are calculated as follows:
The process of finding centroids of similar partial rallies and clustering is repeated in a recursive fashion until a set of centroids that are not similar between each other is found. This final set can be depicted graphically and truly represents the most frequent set of rallies found in the input dataset. Note that this input dataset can be narrowed out to filter only certain types of events, i.e., we could be interested in finding out return patterns, or longer-than-three strokes patterns or patterns when a player is facing a breakpoint, etc.

### 3.4 Mining Problem 2

The key to finding similar winning patterns is to identify similar attacking events. These events will act therefore as fingerprints in the process. The identification of these unbalancing events can be done in two ways: manually, by recognizing the actions that put one of the players in the defense, or automatically, by analyzing the variables involved. The former method requires domain knowledge so we have followed the second approach.

#### 3.4.1 Identification of unbalancing events

The logic put in place to recognize an event as an unbalancing event is based on the following model:

1. Stroke hitting place. Since we know the player’s position when he/she hits a ball and also the trajectory of the incoming ball, we can determine the distance between

\[
\begin{align*}
P'_{1j} &= \frac{\sum_{i=1}^{m} p_{1ij} * w_i}{\sum_{i=1}^{m} w_i} \\
P'_{2j} &= \frac{\sum_{i=1}^{m} p_{2ij} * w_i}{\sum_{i=1}^{m} w_i} \\
P'_{3j} &= \frac{\sum_{i=1}^{m} p_{3ij} * w_i}{\sum_{i=1}^{m} w_i} \\
s'_j &= \max(s_i * w_i), \forall i \ 1 \ldots m \\
c'_j &= \max(c_i * w_i), \forall i \ 1 \ldots m \\
f'_j &= \text{median}(f_i * w_i), \forall i \ 1 \ldots m \\
b'_j &= \text{median}(b_i * w_i), \forall i \ 1 \ldots m
\end{align*}
\]
the players position and the actual hitting place. We know that for a player to hit comfortably the ball, this distance should be around 1m for a forehand and 0.5m for a backhand [2]. Therefore, if this distance is outside this range, we can assume that the player was forced or reached the ball in a precarious way. Based on this distance, we set a probability value.

2. Change in the ball trajectory. Depending on the resulting angle we also set a probability value. Our analysis has shown that angles $\geq 15^\circ$ usually bring about unbalancing events.

3. Speed of the ball. A fast ball followed by a slow ball increases as well the probability value.

Each of these criteria has its own weight. Based on the sum of all probability values, we deem one event to be an unbalancing event or not. Mathematically, we express this in the following way. If $SHP(t, x, y)$ represents a time-dependent Stroke Hitting Place function, $DC(t, x, y)$ a Direction Change function and $BSC(t, x, y)$ a Ball Speed Change function and if we ignore the Cartesian coordinates dependency for clarity reasons, we can write:

$$f(t) = 0.5SHP(t) + 0.2DC(t) + 0.3BSC(t)$$ (3.18)

$$UE(t) = \begin{cases} 
1 & \text{if } f(t) \geq 0.8 \\
2 & UE(t - 1) = 1 \land f(t) \geq 0.8 \\
3 & UE(t - 2) = 1 \land UE(t - 1) = 2 \land f(t) \geq 0.8 \\
0 & \text{otherwise} 
\end{cases}$$ (3.19)

3.4.2 Winning patterns

It will be shown later on that we might want to compare two sequences that do not correlate exactly. A typical example will be the response to an attack that may produce different answers. For instance, a fast first serve to the same corner may result in 1) an ace, 2) a forced error or 3) a short ball that will trigger a winner. All these cases have one thing in common: the initial attacking service. However, the short ball in case 3) might bounce in many areas and therefore the similarity measure defined above cannot be used.

Thus, in this case, the sequence similarity will be more relaxed at certain points than
others, and only some events will enforce a high similarity condition. In other words, we are trying to identify sequential patterns with constraints.

Before we define the winning pattern similarity measure, we need to introduce the concept of a pattern mask. Given a pattern \( p = \langle e_1, e_2, \ldots, e_n \rangle \), we define a pattern mask \( p_{mask} \) as follows:

\[
p_{mask} = \langle \text{sim}_1(e_1), \text{sim}_2(e_2), \ldots, \text{sim}_n(e_n) \rangle
\]  

(3.20)

where each \( \text{sim}_i(e_i) \) represents a particular similarity measure for each event in the pattern. This definition implies that a variety of different similarity measures for each event within the sequence could be used. Some similarity functions will indeed favor some attributes over others in order to fully characterize each pattern.

In this case, a sequence \( seq_1 = \langle e_1, \ldots, e_n \rangle \) will be considered similar to a sequence \( seq_2 = \langle e'_1, \ldots, e'_n \rangle \) (with respect to \( p_{mask} \) and corresponding thresholds \( \text{event}_{thr,i} \) \( (i = 1, 2, \ldots, n) \)), if and only if:

\[
\text{sim}_i(e_i, e'_i) \geq \text{event}_{thr,i} \quad \text{for} \quad i = 1, 2, \ldots, n
\]  

(3.21)

Therefore, for a particular event, the similarity threshold could be very low or even 0, meaning that event wildcards could effectively be allowed. Note also that this similarity implies the sequence similarity concept defined in Section 3.2.3, when the pattern mask is made of equal similarity functions, all sharing the same threshold \( \text{series}_{thr} \). Instead of adjusting the thresholds, it is also possible to rescale the similarity functions; however, the current approach seems to have a better underlying intuition.

### 3.4.3 Completion of attack patterns

We first establish the following equivalences. If we call 1 a first attacking event and \( FE \) a possible forced error as a consequence of 1, then depending on whether the first attack results in a winner (meaning a stroke that will not get a response from the opponent) or in a forced error, we state that:

\[
1, EOR \equiv 1, FE
\]  

(3.22)

where \( EOR \) denotes the end of a rally. Note that \( FE \) automatically includes this last event.
The implication of the previous equation is that two sequences of different lengths can be similar and will represent nonetheless the same winning pattern. Similarly, if 2 represents a second attacking event performed by the player that produced event 1, then:

\[ 1, \succ, 2, EOR \equiv 1, \succ, 2, FE \] (3.23)

where \( \succ \) indicates an event (not being \( FE \)) that does not carry strategic information, as it is a forced defensive response, and therefore no similarity constraint should be enforced. It will usually be a soft ball that can be attacked.

Finally, following with the same logic:

\[ 1, \succ, 2, \succ, 3, EOR \equiv 1, \succ, 2, \succ, 3, FE \] (3.24)

The three equivalences above represent the basic patterns to finish an attack depending on whether the attacking player needed 1, 2 or 3 strokes to finalize the point.

In this case, we might want to enforce a high similarity measure for the first attacking event, a medium similarity event for the second attacking event and a low similarity measure for the third attacking event. This can be done with the pattern mask.

### 3.4.4 Pattern prequel and sequel

For each winning pattern, we define its **prequel** as the sequence of events that appear in the pattern up to the first attacking event. Similarly, we define its **sequel** to be the remaining events in the pattern. We consider the first unbalancing event as being part of both prequel and sequel.

For the remainder of the section, and in order to describe a winning pattern, we will use the following convention. We will continue to use 1, 2 and 3 to indicate the first, second and third unbalancing event, \( FE \) to indicate a forced error event and \( \succ \) to indicate any event (again not being \( FE \)). We will also use \( X, Y, Z \) to indicate a particular event on which we may enforce a similarity function.

Take, for example, the following pattern. The two players are exchanging crosscourt strokes keeping the ball deep until one player gets a short ball that triggers an attack.

---

5The authors believe that very rarely a player will need more than three strokes to finish an attack, and in such a case, one could argue that the opponent did recover from the initial attack and lost the point later on due to a new and different attack.
changing the direction and driving the ball down the line.

If \( X \) represents the crosscourt stroke and assuming that we do not want to impose any similarity check on the response to the attack, then the pattern of the prequel could be represented as: \( p = \langle X, \triangledown \triangledown, 1 \rangle \). \(^6\)

In this case, the possible sequels would be \( \langle 1 \rangle \), \( \langle 1, FE \rangle \), \( \langle 1, \triangledown \triangledown, 2 \rangle \), \( \langle 1, \triangledown \triangledown, 2, FE \rangle \), \( \langle 1, \triangledown \triangledown, 2, \triangledown \triangledown, 3 \rangle \) or \( \langle 1, \triangledown \triangledown, 2, \triangledown \triangledown, 3, FE \rangle \). \(^7\)

![Figure 3.10: NFA for the winning pattern sequel.](image)

The sequel can be represented by a nondeterministic finite state machine or nondeterministic finite automaton (NFA) which can be dealt with in the pattern mining computation. See Figure 3.10, where \( S_0 \) is the initial state, and \( S_F \) represents the final state.

Although in theory we could impose similarity checks to the NFA via the pattern mask, so that a further classification can be obtained, the authors do not believe that this extra granularity brings about any extra worthwhile knowledge. The current automaton only uses the unbalancing event attribute.

**Algorithm**

In order to clarify the algorithm, and to explain the different choices made so far, we begin with an example. Figure 3.11 below shows a variation on the pattern just mentioned. Here, we are interested in studying three events prior to the attacking one. In this case, we use the pattern \( \langle X, Y, Z, \triangledown \triangledown, 1 \rangle \) to try to find a similar sequence of three events \( \langle X, Y, Z \rangle \) that will allow the first player to attack the ball and unbalance the opponent. The use of the pattern mask allows to select which events in the pattern should have a high similarity. This figure also takes into account both the prequel and sequel of the winning pattern.

\(^6\)This represents pattern 19 from \([64]\).

\(^7\)Note that as per the previous section, \( p \equiv \langle X, \triangledown \triangledown, 1, FE \rangle \) for winning the point using one stroke only, \( p \equiv \langle X, \triangledown \triangledown, 1, \triangledown \triangledown, 2 \rangle \) for two strokes, and so on.

\(^8\)Note that in fact the \( X \) in pattern \( \langle X, \triangledown \triangledown, 1 \rangle \) represents not only a crosscourt stroke but any stroke that will allow the player to trigger an attack on the next ball after stroke \( X \).
In this example, if we assume for simplicity that all similarity functions in the pattern mask are the same, and \( \text{event}_{\text{thr}X} \) represents the event similarity threshold for the event \( X \), \( \text{event}_{\text{thr}Y} \) for event \( Y \) and so on, and \( e_i.us \) represents the unbalancing stroke attribute of event \( i \), then the two rallies:

\[
\begin{align*}
 r_1 &= (e_{11}, e_{12}, e_{13}, e_{14}, e_{15}) \\
r_2 &= (e_{21}, e_{22}, e_{23}, e_{24}, e_{25}, e_{26}, e_{27})
\end{align*}
\]

where \( e_{15} \) and \( e_{27} \) are both last events, will be similar and belong to the same winning
3.5 MINING PROBLEM 3

pattern \((X, Y, Z, ∞, 1)\) if all the following conditions are true:

\[
\begin{align*}
sim(e_{11}, e_{21}) & \geq \text{event}_X \\
sim(e_{12}, e_{22}) & \geq \text{event}_Y \\
sim(e_{13}, e_{23}) & \geq \text{event}_Z \\
sim(e_{15}, e_{25}) & \geq \text{event}_{1}
\end{align*}
\]

\[
\begin{align*}
e_{15.us} &= 1 \\
e_{25.us} &= 1 \\
e_{27.us} &= 2
\end{align*}
\]

The algorithm implemented to identify the winning patterns is described in the pseudocode below (see the algorithm in Figure 3.12) and can be summarized as follows. Firstly, we locate events that verify the condition of being first attacking events (see Section 3.4.1). Then for each pattern, we expand the projected database (see Section 3.3.1) in depth-first search fashion checking from the pointer to the left using the similarity mask. For each sequence found, we expand likewise the sequel to the right checking the NFA as well. Several optimizations are possible, e.g., pruning of the search space, but the current implementation does not focus on this issue, the datasets being of relatively small size.

3.5 Mining Problem 3

This mining problem is a special case of the first mining problem. Even though it seems that there are two supports that are of interest for any given sequence, there are in fact three: for any event sequence \(seq\), we are interested in \(support_M(seq)\), \(support_M(R_{180}(seq))\) and \(support_{rot_M}(seq)\). But since (except for rare cases, if both a subrally and its rotated version are similar to the given one):

\[
support_{rot_M}(seq) = support_M(seq) + support_M(R_{180}(seq))
\]  \hspace{1cm} (3.25)

we will indeed restrict our attention to two of these.

A different approach has been used to explore this last mining problem. Since the tennis matches under consideration are all in the order of magnitude of \(\leq 2,000\) events, it is possible to use a brute-force algorithm. However, in other situations it will be useful to
CHAPTER 3. DATA MINING AND PATTERN DISCOVERY

input
- $R$, a series of rallies;
- $pmask$, a pattern mask (with thresholds);
- $NFA$, an automaton;
- $min\_support$, a threshold

output
- $W$, a set of winning patterns along with their support

begin
  Put all events $e$ from $R$ with $e.us = 1$ into set $S$
  foreach $e \in S$
    $support \leftarrow 0$
    foreach $e' \in S$ with $e \neq e'$
      if prequels similar according to $pmask$
          and sequels satisfy $NFA$
          $support \leftarrow +$
      if $support \geq min\_support$
        Add prequel and support to $W$
  return $W$

end

Figure 3.12: Algorithm — Winning patterns identification.

rely on more powerful (i.e., efficient) algorithms for frequent itemset mining, like the ones explained in Section 3.3 and 3.4. On the other hand, we will restrict ourselves to subrallies of length $n = 3$ for this study.

3.6 Results

Over 5,000 events with all their attributes (stroke type, players positions, ball bouncing places, speed of the ball, unbalancing strokes, score, outcome of the point, etc.), and more than 10 hours of recordings were captured and analyzed, covering men’s and women’s matches in both hard courts and clay courts. Table 3.1 shows the matches analyzed.

3.6.1 Mining Problems 1 and 2

As a first experiment, we tried to analyze the successful service winning patterns displayed by the players. Depending on the court surface, these points can account for more than half
### 3.6. RESULTS

<table>
<thead>
<tr>
<th>Tournament</th>
<th>Surface</th>
<th>Round</th>
<th>Duration</th>
<th>Events</th>
<th>Players</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roland Garros 2009</td>
<td>Clay</td>
<td>R16</td>
<td>210 min</td>
<td>1,628</td>
<td>Söderling vs</td>
<td>6–2, 6–7, 6–4, 7–6</td>
</tr>
<tr>
<td>ATP Paris 2007</td>
<td>Hard</td>
<td>F</td>
<td>70 min</td>
<td>518</td>
<td>Nadal vs</td>
<td>6–4, 6–0</td>
</tr>
<tr>
<td>Australian Open 2010</td>
<td>Hard</td>
<td>SF</td>
<td>122 min</td>
<td>831</td>
<td>S. Williams vs</td>
<td>7–6, 7–6</td>
</tr>
<tr>
<td>ATP Madrid 2011</td>
<td>Clay</td>
<td>F</td>
<td>138 min</td>
<td>1066</td>
<td>Djokovic vs</td>
<td>7–5, 6–4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.1: Data evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>18%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.2: NaLi Service winning patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>30%</td>
</tr>
</tbody>
</table>

the total points won (i.e., excluding unforced errors by the opponent), thus, its importance.

The winning pattern here is simply \( \langle 1 \rangle \), where the prequel is empty (no prior events, since the unbalancing stroke belongs to a service) and the usual sequel of \( \langle \rangle \), \( \langle FE \rangle \), etc. The following Figure 3.13 shows a few examples of successful service winning patterns found during the 2010 match between Na Li and Serena Williams:

Each section of Figure 3.13 represents the same winning pattern (service to the T on the Deuce court), being completed by one, two or three strokes, respectively. Overall, the analysis of the successful winning patterns displayed by both players during the match yields the following in Tables 3.2 and 3.3:

A second experiment was set up to try to find groundstroke attacking patterns. The
CHAPTER 3. DATA MINING AND PATTERN DISCOVERY

Figure 3.13: Service winning patterns

winning pattern here was set to be \( \langle X, Y, 1 \rangle \), where the prequel is \( \langle X, Y \rangle \). This prequel represents two events, \( X \) produced by the same player that hits the unbalancing stroke, and \( Y \) produced by the opponent. The similarity mask is set in such a way that the event threshold at the first unbalancing event (or pointer) and the event \( Y \) is fairly high, but it is lower at the event \( X \). Note that by lowering these thresholds or even eliminating the event \( X \) from the winning pattern, we would get more results. The outcome of this search over the 2009 match produced the following results: 11 groundstroke attacking winning sequences by Nadal all show the same pattern. Figure 3.14 shows a few examples of successful groundstroke attacking winning patterns found during the 2009 match between Rafael Nadal and Robin Söderling:

In this case, we have not shown the completion of the attack (i.e., the sequel) in order to make the figures clearer.

If we consider that only 21 non-service related points were won by Nadal on that game due to winners or forced errors, the fact that half of them were won with this sequence of events, shows that this was a clear strategy: to exchange strokes with Söderling until he was displaced to the left hand side of the court so Nadal could attack an eventual short ball with a powerful down the line forehand.
3.6. RESULTS

3.6.2 Mining Problem 3

By setting the thresholds to certain fixed values: $event_{thr} = 4$ and $series_{thr} = 12/3$ (the similarity between two 3-rallies is at most $3 \times 6.0 = 18.0$ and the whole sequence satisfies a stricter threshold condition than its three individual events combined), while adjusting the $min\_support$ value so that precisely 20 patterns of length 3 were generated for each match, we obtained the following results.

Figure 3.15 shows some interesting partial rallies that are present for both players, as per the definition of this particular Mining Problem 3, once we allow for rotation. These subrallies might be candidates for being more general patterns, but might also be just exemplary for this 2009 match. Some of the other frequent subrallies found for each player (Mining Problem 1) were not interesting anymore: they did not meet the higher support threshold, and were clearly too player specific.

The relative number of frequent patterns that start with a serve is much higher in this match than in the other two matches. In fact, 4 out of the 20 interesting patterns for this match start with a serve (making use of the rotated support), versus none of the interesting patterns for the other matches. In the Australian Open match, the application
of the rotation substantially increases the support (justifying the high min support needed for this match), contrary to the situation for the other matches. These results suggest that the two female players in the Australian Open match do act more like one another than the players in the rest of the matches.

### 3.7 Conclusions

In this chapter, we have applied the framework of pattern mining to the game of tennis. We have defined similarity measures, thresholds, support, and the corresponding mining problems. Experiments show that the approach works in practice and several tactical patterns are discovered.

Further research includes the search for more general patterns by constructing so-called pseudo-rallies (merging or generalizing several frequent subrallies) and proper (statistical) analysis of more datasets. For specific players strengths and weaknesses could be explored, and patterns for matches and more general ones should be further examined.
Chapter 4

Identification of optimal strategies

No man, for any considerable period, can wear one face to himself, and another to the multitude, without finally getting bewildered as to which may be the true.

Nathaniel Hawthorne

As repeated throughout this thesis, the identification of tactical information in tennis from broadcast video usually consists of simply providing general statistical information. In this chapter, we present a tennis model based on Markov Decision Processes (MDPs), which describes the dynamic interaction between the players and we introduce a novel Monte Carlo based method with an aim to extract optimal strategic information. Finally, we test the approach with real tennis data. We show that this framework, based on states, actions and rewards, allows for the identification of optimal strategies.

Tennis is a dynamic and complex game where the players are continuously making decisions in a short space of time. Variables as diverse as players positioning, stroke biomechanics, score or even weather conditions are constantly evaluated. One of the aims of the research about player behavior is to gain an understanding of the patterns exhibited by successful players.

One problem that affects the analysis of sports data resides in its variability. Hence, it is difficult to evaluate a strategy and obtain consistent results based solely on partial data with imperfect information. One way to solve this problem is by creating a simulated model. In this chapter, we present one fed with real game data from broadcast video.

The rest of the chapter is organized as follows. We first introduce our Markov Decision
Process-based tennis model in Section 2. In Section 3, we present the structure of the MCTS algorithm and define the concept of shortest-path back-propagation. In Section 4, we describe the multimedia system implemented. We then apply the MCTS algorithm to real tennis broadcast videos and show the results in Section 5. Finally, in Section 6, we conclude with a summary and future work.

4.1 Change of paradigm

In Chapter 3, we explored the raw data extracted from the computer vision system directly, i.e., without further processing. We defined the concept of an event as a single stroke episode, which contained all attributes that characterized the stroke, i.e., the player that hit the stroke, the type of stroke, the position of both players at the time of hitting the ball, the position of the ball landing on the opponent’s side after the stroke, the generated speed of the ball, etc. This allowed us to define in turn, the concept of rally, as the sequence of events that completely described the strokes exchanged by the players during a game point. In this way, we were able to characterize and annotate the whole tennis game (see 3.1.1).

Although the similarity function described in 3.2.1 could be customized to provide different influence areas around the exact court coordinates of the either the ball landing position or the players position, it was felt that a change of paradigm was needed in the research. After several conversations with professional tennis coaches, it was confirmed that ball trajectories should prevail over exact ball landing positions and that certain strategic court regions are more important than comparing exact players positions when trying to gain tactical insights into the game of tennis.

These changes led to a state-action framework that suited perfectly to the research field of Reinforcement Learning and Markov Decision Processes, where an agent (player about to hit the ball) and the environment (situation in the match: position in the court of opponent, position of the player, trajectory of the coming ball, score, weather conditions, etc) interact at discrete time steps. At each of these steps, the agent assesses the environment’s state and selects an action. A step later, as a consequence of his action, the agent receives a numerical reward and finds himself in new state (see [60]).

1The author experimented with elliptical areas, rather than circular ones to better characterize the ball trajectory, as well as with other geometries.

2David Sanz Rivas, Director of Coaches Education and Tennis Performance Research Group for the Spanish Tennis Federation (RFET) and captain of the Spanish National Wheelchair Tennis team; Tati Rascon, former coach of Fernando Verdasco, World No. 7 in 2009.
This chapter explores this new paradigm and finds optimal strategies in real tennis matches.

4.2 Tennis as a Markov Process

Although Markov chains have been previously applied to tennis in an attempt to either predict the outcome of a game or to find out the probability for a player to win a match [10, 44], the application of a full Markov Decision Process (MDP) to model the dynamics of tennis rallies is novel. In order to be able to use this model, we must define a series of states, a series of actions, the probability of moving from one state to the next and finally the rewards for a particular action. The sequence of events is as follows. Let $s_t$ denote the state at time $t$, and $a_t$ the action chosen at that time from the state $s_t$. Then, the system transitions from state $s_t$ to state $s_{t+1}$ with probability $P(s_{t+1}|s_t, a_t)$ and a reward $R(s_t, a_t)$ is obtained. Once the transition to the next state has occurred, a new action is chosen and the process is repeated.

Formally, a Markov Decision Process is defined by the following 4-tuple $(S, A, P(.,.), R(.,.))$, where:

- $S$ is the set of possible states. It also includes the final states that represent the end of the point due to a mistake or a winner.

- $A$ is the set of possible actions.

- $P(.,.)$ represents the set of probabilities that define the transition model [56]. They are defined as follows:
  - $P(a|s)$ defines the chance of playing an action $a$, given a certain state $s$.
  - $P(s'|s, a)$ defines the probability of reaching state $s'$ if action $a$ is done in state $s$.

- $R(.,.)$ is the set of rewards received in each state for each particular action. The reward scheme used is as follows: -1, if the player loses the point, 1, if the player wins the point, and 0, in all other cases.

---

3 A rally in tennis is defined as a sequence of shots within a point. A rally starts with serve and return of the serve, and a sequence of shots until the point is won by one of the players.
The sequence of events described above that map states $s_i \in S$ and actions $a_i \in A$ to the probability of taking action $a_i$ when in state $s_i$ is called a policy $\pi$. The goal is to find a policy that maximizes the expected return given by (see Ref. [60]):

$$V^\pi(s) = E_\pi \{ R_t | s_t = s \} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\}$$

(4.1)

where $V^\pi(s)$ is the expected return when starting in state $s$ and following policy $\pi$, $0 < \gamma < 1$ is the discount factor and $E_\pi \{ \}$ denotes the expected value if the agent follows policy $\pi$. For finite MDPs, the optimal value is defined as:

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s), \forall s \in S$$

(4.2)

where $\Pi$ represents the set of all possible Markovian policies $\pi$. An optimal policy in $\Pi$ that achieves $V^*(s)$ for all $s \in S$ will be denoted by $\pi^*$. Note that this model assumes Markovian transitions and for this reason, states do not carry information about the history of previous visits. This implies that historic information like player’s tiredness or number of times the state has been visited is not considered.

In Section 3, we will introduce a Monte Carlo method used to estimate the value function and identify an optimal policy $\pi^*$.

### 4.2.1 Initial Model

In order to establish a test bed where various algorithms can be tried, a simple tennis model has been devised. In Reinforcement Learning literature (see [60]), it is usual to define a grid world with simplified rules where algorithms are tested on. So, from this point of view, we first define a simple model that represents an initial approach to modeling a tennis game. This model, although simplistic, also incorporates random errors, unbalancing scenarios, risk assessment and action selection. Figure 4.1 shows the layout for a tennis match between two players P1 and P2 under this model.

For this model, the following rules have been defined:

1. Only two player’s positions are allowed: left (L) and right (R).
2. Only two strokes are allowed: down-the-line (DL) and cross-court (CC).
3. Also, a player can decide to take risks with the stroke. For this purpose, we introduce
4.2. TENNIS AS A MARKOV PROCESS

Figure 4.1: Simple tennis model

a Boolean variable (risk), to indicate whether the player is taking risks or not. A risky shot is represented with the letter (R) added to the shot. This way (CCR) indicates a cross-court risky shot. Note that the more risk a player takes, the higher are the chances to win the point, but the higher are also the chances of making a mistake.

In order to make this model more real, we use the concept of unbalancing or forced situations. When one player plays a risky shot, then the opponent will have to play in a forced position which will increase the chances to make a mistake. This is a fundamental concept in tennis and has been modeled as per the rules defined below. For more details of this model, see [63]. F indicates a forced situation for the player who is going to hit the ball, whereas S indicates if the ball coming is a soft ball so that the player could attack. Note that the algorithm takes into account the number of consecutive times forced of soft situations occur (see Figure 4.2).

We can represent this model as a Markov Decision Process (MDP) by the following 4-tuple \((S, A, P(\cdot, \cdot), R(\cdot, \cdot))\), where:

- \(S=\{Rl, RlF, RlS, Rr, RrF, RrS, Ll, LlF, LlS, Lr, LrF, LrS, M, W\}\) is the set of possible states. Apart from the last two final states M and W, each state is represented by two letters and an optional one: the first, in upper case, representing the position of the opponent (R or L), the second letter, in lower case, representing where the ball from the opponent will bounce (r or l). The third optional letter represents the
possible unbalancing situation (F, for forced) and (S, for soft) for the player hitting the ball. The final states M and W represent the end of the point due to a mistake or a winner respectively.

- **A**={DL,DLR,CC,CCR} is the set of possible actions. Each action is represented by two letters and an optional third. The first two letters indicate the stroke and the third one, if present, indicates if the player is taking risks with the stroke.

- **P(.,.)** represents the set of probabilities that define the transition model\[56\]. They are defined as follows:
  - \(P(A|S)\) defines the chance of playing an action \(A\), given a certain state \(S\). In this way, the probability of playing a risky cross-court shot from state \(L_r\) will be represented by \(P(CCR|L_r)\).
  - \(P(S'|S,A)\) defines the probability of reaching state \(S'\) if action \(A\) is done in state \(S\). Note that under this model, only the final states M and W are stochastic. The remaining states are deterministically based on \(A\) and \(S\). For this reason, the two conditional probabilities \(P(M|S,A)\) and \(P(W|S,A)\) will fully define the modeled behavior.

- **R(.,.)** is the set of rewards received in each state for each particular action. The reward scheme used is as follows: -1, if the player loses the point, 1, if the player wins the point, and 0, in all other cases.

We will now apply these rules to a policy that characterizes a possible profile for P1 and P2 in a certain match. The objective is to build a policy that purposely favors one of the players over the other. We postulate:

1. P1 makes more mistakes from the left position when he tries a CC and the opponent is on the R side.

2. P2 likes to copy the trajectory of the incoming ball.

3. P1 has a very good attacking DL shot from the R side.

4. P2 has a very good attacking CC shot from the R side.
5. P1 does not take many risks and therefore does not make many mistakes. On the contrary, P2 likes to take risks and therefore makes more mistakes than P1 but he also achieves many more winners.

A possible policy implementation that complies with the five criteria enumerated above is shown in Tables 4.1 and 4.2, where we can see that, for example, P1 has a 40% chance of making an unforced error when the opponent is on the Right side and the ball is coming to P1’s left side, so being in state Rl, and he plays a normal cross-court shot (no risk taken), i.e., \( P(M|Rl,CC) = 0.4 \).

| STATE | ACTION | P(A|S) | P(M|S,A) | P(W|S,A) |
|-------|--------|-------|---------|---------|
| Rl    | DLR    | 0.05  | 0.30    | 0.10    |
| Rl    | DL     | 0.45  | 0.00    | 0.00    |
| Rl    | CCR    | 0.05  | 0.30    | 0.10    |
| Rl    | CC     | 0.45  | 0.40    | 0.00    |
| Rr    | DLR    | 0.05  | 0.30    | 0.10    |
| Rr    | DL     | 0.45  | 0.00    | 0.00    |
| Rr    | CCR    | 0.05  | 0.30    | 0.10    |
| Rr    | CC     | 0.45  | 0.00    | 0.00    |
| Ll    | DLR    | 0.05  | 0.30    | 0.10    |
| Ll    | DL     | 0.45  | 0.00    | 0.00    |
| Ll    | CCR    | 0.05  | 0.30    | 0.10    |
| Ll    | CC     | 0.45  | 0.00    | 0.00    |
| Lr    | DLR    | 0.05  | 0.30    | 0.10    |
| Lr    | DL     | 0.45  | 0.00    | 0.00    |
| Lr    | CCR    | 0.05  | 0.30    | 0.10    |
| Lr    | CC     | 0.45  | 0.00    | 0.00    |

Table 4.1: Policy for Player P1

Under this probabilistic model, when both P1 and P2 choose their actions randomly, P1 should always beat P2 in the medium/long term due to the higher mistake rate of P2. In order to prove this hypothesis, an experiment consisting on running 10,000 simulations has been conducted. Figure 4.3 plots the number of points won by each player during 10,000 simulations. It can be seen that after all the simulations, the Player 1 has won 41% more points than Player 2.
CHAPTER 4. IDENTIFICATION OF OPTIMAL STRATEGIES

| STATE | ACTION | P(A|S) | P(M|S,A) | P(W|S,A) |
|-------|--------|-------|---------|----------|
| Rl    | DLR    | 0.24  | 0.50    | 0.20     |
| Rl    | DL     | 0.56  | 0.30    | 0.00     |
| Rl    | CCR    | 0.06  | 0.50    | 0.40     |
| Rl    | CC     | 0.14  | 0.30    | 0.00     |
| Rr    | DLR    | 0.06  | 0.50    | 0.20     |
| Rr    | DL     | 0.14  | 0.30    | 0.00     |
| Rr    | CCR    | 0.32  | 0.50    | 0.20     |
| Rr    | CC     | 0.48  | 0.30    | 0.00     |
| Ll    | DLR    | 0.06  | 0.50    | 0.20     |
| Ll    | DL     | 0.14  | 0.30    | 0.00     |
| Ll    | CCR    | 0.24  | 0.50    | 0.20     |
| Ll    | CC     | 0.56  | 0.30    | 0.00     |
| Lr    | DLR    | 0.24  | 0.50    | 0.20     |
| Lr    | DL     | 0.56  | 0.30    | 0.00     |
| Lr    | CCR    | 0.08  | 0.50    | 0.40     |
| Lr    | CC     | 0.12  | 0.30    | 0.00     |

Table 4.2: Policy for Player P2

Profile of points won

Figure 4.3: Simulation results
4.2. Full Tennis Model

Figure 4.4 shows the overall model used. Given a number of states and actions captured during a match, we obtain a distribution model that is used as an input in order to simulate the real experience. This sample model is likely not to represent the complete state-action space, and for this reason, the model could incorporate data for the same players from other matches, as suggested by the dotted lines. This model incorporates probabilistic error functions to take into account shots that are targeted near the lines. In order to make this model more real, we use the concept of unbalancing or forced situations. When one player plays a risky shot, then the opponent will have to play in a forced position which will increase the chances to make a mistake. This is a fundamental concept in tennis (for more details of this model, see Ref. [63]). We use variable $F$ indicates a forced situation for the player who is going to hit the ball, whereas $S$ indicates if the ball coming is a soft ball so that the player could attack. Once the model is built, a number of simulated runs are executed in order to extract optimal policies using the Monte Carlo methods explained in the next sections.

![Tactical model diagram](image)

Figure 4.4: Tactical model

Ideally, a feedback loop, as shown in the diagram, would represent the attempt to influence the player’s experience and decision-making process by feeding up the results of the policy identification.

Firstly, we will introduce the set of actions, states and probabilities that characterize the sample model MDP. They have been defined in an attempt to very closely describe the variables processed and acted upon by a tennis player in a real life situation. We incorporate all possible strokes, a player’s position grid based on known tactical tennis
principles\[2\]. Other human constraints such as the player’s ability or speed are not factored into the model, however, they are implicitly accounted for as a result of the set of actions displayed for that particular player. For instance, a good player will be able to consistently hit winners from a particular position whereas a normal player will not. This will be extracted from the game data and fed into the simulation model for that player. All player’s positions will be in the grid depicted in Figure 4.5 and they are defined in the set $C = \{1, 2, 3, 4, 5, 6\} \times \{R, A, N, V, W\}$.

![Figure 4.5: Tennis court grid](image)

We also incorporate the information about the offensive or defensive momentum by the Boolean variables $F$ and $S$ described above. If the player hitting the ball is facing a forced situation, then $F = 1$. On the contrary, if the player is facing a soft ball that can be attacked, then $S = 1$. Otherwise, during normal stroke exchanges where no player is unbalancing the opponent, then $F = S = 0$.

In order to distinguish between serving and returning-the-serve states with rally states that can display the same players positions, we have added a few extensions:

- If the player is serving, the opponent’s position is set to $R$ (Return) and the forced
4.2. TENNIS AS A MARKOV PROCESS

flag is set to \( \{F,S\} \) to indicate a First or a Second serve. The soft flag is also set to \( \{D,A\} \) to indicate which part of the court the player is serving to the Deuce or Advantage court, respectively. The player’s position is as per Figure 4.5 and will always be \( \{3A,4A\} \).

- If the player is returning a serve, the player’s position is set to \( R \) (Return), and the opponent’s position is set to \( \{FSDC, FSDB, FSDT, FSAC, FSAB, FSAT, SSDC, SSDB, SSDT, SSAC, SSAB, SSAT\} \) to indicate the type of serve that the player is about to respond to (see next section to understand the notation used). The soft and forced flags are set as per their definition.

- In all other cases, i.e., during rallies, we defined the central horizontal position, \( C \), to indicate that either the player or the opponent is placed in positions 3 or 4, as at this point, this granularity is not needed. The soft and forced flags are set as per their definition.

In summary, the state will be defined by the following variables:

- Player position, \( PPOS \), position of the player hitting the ball;
- Opponent position, \( OPOS \), position of the opponent when the player hits the ball;
- Forced state indicator, \( F \);
- Soft state indicator, \( S \);

The player actions have been characterized as follows. Every time the player hits the ball, he/she will be able to decide the type of stroke to use, the direction of the ball trajectory, the effect to be given to the ball and finally whether risk should be taken or not. We have distinguished between serving actions and the rest. For the service, the possible options are described in Table 4.3.

Note that the effect has not been taken into account for the serve as the type of stroke provides enough information about the action. As for the rest of strokes, the actions are described in Table 4.4.

The criteria used so that the direction of the ball can be categorized is based on the angle defined between the place where the ball is hit and the place where the ball bounces (or where the ball is targeted if the ball does not bounce, i.e., the opponent plays a volley). Figure 4.6 shows some of these trajectories.
CHAPTER 4. IDENTIFICATION OF OPTIMAL STRATEGIES

<table>
<thead>
<tr>
<th>Stroke</th>
<th>Direction</th>
<th>Effect</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS (First Serve)</td>
<td>DT (Deuce court to the T)</td>
<td>N/A</td>
<td>1 (Yes)</td>
</tr>
<tr>
<td>SS (Second Serve)</td>
<td>DB (Deuce court to the Body)</td>
<td></td>
<td>0 (No)</td>
</tr>
<tr>
<td></td>
<td>DC (Deuce court to the Corner)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AT (Advantage court to the T)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AB (Advantage court to the Body)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AC (Advantage court to the Corner)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Actions for the service.

<table>
<thead>
<tr>
<th>Stroke</th>
<th>Direction</th>
<th>Effect</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>FH (Forehand)</td>
<td>C (Centre)</td>
<td>TS (Top Spin)</td>
<td>1 (Yes)</td>
</tr>
<tr>
<td>BH (Backhand)</td>
<td>XL (Cross-court to the left)</td>
<td>SL (Sliced)</td>
<td>0 (No)</td>
</tr>
<tr>
<td>VOL (Volley or Smash)</td>
<td>XXL (Cross-court to the left eXtreme)</td>
<td>DS (Drop shot)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XR (Cross-court to the right)</td>
<td>LB (Lob)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XXR (Cross-court to the right eXtreme)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DL (Down the Line)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Actions for the rest of strokes.

Figure 4.6: Some possible trajectories: (left) serving to the deuce court, (right) hitting from the baseline

In summary, the action will be defined by the following variables:
4.2. **Tennis as a Markov Process**

- Stroke, \( STR \), type of shot used by the player hitting the ball;
- Direction, \( DIR \), trajectory of the target ball;
- Effect, \( E \);
- Risk indicator, \( R \);

Figure 4.7 shows a classic three stroke serve pattern, where the player serves wide to open the court and hits a second shot looking for a winner or an approach. The corresponding state-action sequence annotation is denoted in Table 4.5.

![Figure 4.7: Example of a rally with three strokes](image)

<table>
<thead>
<tr>
<th>Player</th>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4A-R-F-D</td>
<td>FS-DC-NA-1</td>
</tr>
<tr>
<td>2</td>
<td>R-FSDC-1-0</td>
<td>FH-XXR-TS-0</td>
</tr>
<tr>
<td>1</td>
<td>5N-5A-0-1</td>
<td>FH-DL-TS-1</td>
</tr>
</tbody>
</table>

Table 4.5: State-action sequence for Figure 4.7

As per the tactical model introduced above, all the state and action data coming from broadcast videos of real tennis games is fed into the system using the computer vision module. This sequential information allows to identify for each player the list of states
visited and the actions taken from each state, as well as the next state following that action. What we need to do now is to find an estimation of the probability distribution for the actions and next states based on the sample data.

As stated by Wang et al.\[69\], the assumption of a stationary strategy is often unrealistic. In tennis, it is very typical for a player to change his/her strategy after losing a set or a service game. Therefore, we are assuming local stationarity, i.e., the strategy is assumed to be stationary within a number of games.

If we denote $S$ as the finite set of states visited by one player and $A$ as the finite set of actions performed, let us define $N_s$ as the number of times a player has visited the state $s$, and $N(s,a)$ as the number of times a player has performed action $a$ from state $s$. Let us also define $N(s',a,s)$ as the number of times that taking action $a$ from state $s$ results in state $s'$. In order to estimate the probability distributions, we have used the method of maximum likelihood estimation (MLE) since the sample size is large, and under this condition the variance of any estimated parameter is nearly as small as can be achieved by any other estimator (see Ref. [17]).

It is therefore possible to estimate the probability for a player to take action $a$ from a particular state $s$ by the following formula:

$$P(a|s) = \frac{N(s,a)}{N_s} \quad (4.3)$$

Likewise, the probability of reaching next state $s'$ after taking action $a$ from state $s$ can be estimated as:

$$P(s'|a,s) = \frac{N(s',a,s)}{N(s,a)} \quad (4.4)$$

### 4.3 Monte Carlo Tree Search (MCTS)

While an optimal policy for a Markov Decision Process can be obtained by the methods of dynamic programming, they rely on a well-defined evaluation function that has to be computed for every possible state of the system. However, such computations become unfeasible when the system involves a large state space or a large action space or when the transition probabilities and reward function are not known a priori. As a result, dynamic programming algorithms are often not a viable option for a given Markov Decision Process.

One approach is to estimate the evaluation function and find optimal policies by approximating the solution based on randomized/simulation-based methods\[9\], thus avoiding
4.3. MONTE CARLO TREE SEARCH (MCTS)

the need for a model of the environment’s dynamics. Reinforcement Learning methods employ this approach and use the experience to solve the prediction problem. In our case, we will see in later sections that although the state-action space for the full model is not huge ($\sim 10^5$), the lack of knowledge about the opponent’s strategy makes reinforcement learning a suitable approach.

Preliminary analysis on other reinforcement learning methods such as Q-Learning, Sarsa and Sarsa($\lambda$) with eligibility traces (see Ref. [60]) have shown worse performance than simulation-based Monte Carlo methods. The first experiment in Section 5 shows a comparison between MCTS and Q-Learning against the results in a real match.

In this section, we will introduce a variation on the Monte Carlo Tree Search (MCTS) method (see Refs. [22, 15, 10]), which uses a selection strategy based on the Upper Confidence Bound (UCB) sampling algorithm, derived from the n-Armed Bandit problem [60]. This method estimates the optimal value function under the constraint of a finite number of simulations per stage by using statistical sampling techniques.

4.3.1 Algorithm structure

The structure of the MCTS algorithm consists in iteratively running random simulations to evaluate candidate actions from the nodes of a search tree. The values in the nodes are used to select the best action during subsequent simulations.

Figure 4.8 outlines the four MCTS phases as implemented in our tennis model, where $S = \{s_1, s_2, \ldots, s_m\}$ is the finite set of states, $A = \{a_1, a_2, \ldots, a_n\}$ is the finite set of actions, and $a_i = a_i(s) \in A$ is a legal action from state $s$. $N$ is the search tree which contains the nodes that have been evaluated during the simulations. Each node $n(s)$ in the tree contains the total visit count for the state $s$ and the action value and count for each of those actions $a_i(s) \in A$.

Simulations start from a random starting state $s_0 \in S$ and each possible action $a_i(s_0) \in A$ from this state is evaluated by playing simulated games until the final state $s_f \in \{W, M\}$ is reached and the winner can be determined. The actions that lead to winning random play-outs are considered the good moves in the current game position and receive a positive reward. However, where standard MCTS algorithms update all nodes traversed during each simulation, we only update the nodes that define the shortest path from the initial state to the final state. The rationale of this modification is explained in Section 4.3.3.

In general, the accuracy of these estimates depends on the number of random play-outs
(usually > 1,000 games). Once the best action is selected, the game continues until the player encounters another state not evaluated before and repeats the process again. The following sections briefly detail how the algorithm works.

**Figure 4.8: Phases of the Monte Carlo Tree Search**

### 4.3.2 Simulation

If the current node has not been evaluated before, actions are selected at random until the end of the game. Adequate weighting of action selection probabilities has proven very important on the level of play in Go\[10\]. Likewise, in real tennis, the use of certain shots in some positions will lead to weak strategies. In this case, heuristics can be used to give larger weights to actions that look more promising. However, for the simple tennis model, we have implemented a uniform action selection distribution that allows all possible actions to be evaluated equally.

### 4.3.3 Back-propagation

After reaching the end of each of the simulated games, the (node, action) pairs that were traversed during that game are updated. The pair visit counts are increased and the win/loss ratio is modified according to the outcome. The result of a simulated play-out \( k \) starting from node \( i \) is counted with a positive reward \( (R_k = +1) \) if the game is won, and with a negative one if the game is lost \( (R_k = -1) \). The value action \( v_i \) for the (node, action) pair is computed by taking the average of the results of all simulated games made through this pair, i.e., \( v_i = \sum_k R_k \).

However, the nature of the tennis game implies that states can be recurrent, so we will never have a standard search tree. Also, the same action taken from the same node
can lead to different states, i.e., it is non-deterministic. Finally, different actions can take you to the same state. For these reasons, our tennis game can be modeled as a PNFA (Probabilistic Non-deterministic Finite Automaton), with probability transition functions for the actions.

Figure 4.9: PNFA and Shortest Path

Figure 4.9 shows an example of a typical point during a tennis. $s_0 \in S$ represents the initial node and $s_f \in S$ is the final one, which in this case, indicates a win. Before reaching the final state, the player traverses these states:

$$s_0 \xrightarrow{a_2} s_0 \xrightarrow{a_2} s_1 \xrightarrow{a_3} s_0 \xrightarrow{a_2} s_0 \xrightarrow{a_2} s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_4} s_2 \xrightarrow{a_5} s_f$$

If, as mentioned earlier, tiredness or other time-dependent factors are not taken into account, returning to a previously visited state should be equivalent to the first visit\(^4\). Also, from a Markovian point of view, we should not care about the whole route; just the shortest path from the initial node to the final one should matter. For this reason, our MCTS implementation uses the shortest path to back-propagate the result up to the starting node. In this example, only the (node, action) pairs $(s_0, a_4)$ and $(s_2, a_5)$ are updated with a value $R = 1$:

$$s_0 \xrightarrow{a_4} s_2 \xrightarrow{a_5} s_f$$

We have chosen to update all (node, action) pairs found in the shortest path for each of the simulated play-outs, so that there exists information in these pairs for later simulations. The back-propagation, however, finishes at the evaluated node and not at the very first initial node.

\(^4\)In long rallies during a tennis game, it is noticeable how certain positions and patterns repeat, giving the impression that the same point starts all over again.
4.3.4 Selection

Selection is the task that chooses the best action of a given node. It controls the balance between exploration and exploitation. Choosing a good balance between exploration and exploitation is very important. Because if, on the one hand, the selection is too explorative the tree will become too shallow to allow tactical play. If, on the other hand, it is too exploitative the algorithm might get stuck in a local maximum.

The strategy UCT (Upper Confidence Bound applied to Trees) from Kocsis et al.\[37\] is used to balance exploration and exploitation. UCT uses Equation 4.5 to select which action \( k \) of node \( p \) should be selected, where \( A \) is the set of actions that can be chosen from node \( p \). Furthermore, \( v_i \) represents the value of action \( i \), \( n_i \) is the visit count of action \( i \) and \( n_p \) is the visit count of \( p \). \( C \) is a coefficient to be tuned experimentally that determines the balance between exploration and exploitation (the higher \( C \) is set, the more explorative the selection will be).

\[
k = \arg\max_{i \in A} (v_i + C \sqrt{\frac{\ln n_p}{n_i}})
\]

(4.5)

4.3.5 Expansion

When the game reaches another state that cannot be found in the tree, the state is added as a new node to the tree and it is evaluated through a simulation phase. If the game reaches a state that has been already evaluated, the algorithm bypasses the simulation and back-propagation phases and goes directly to the selection phase. This ensures a fast execution once the tree gets larger. Note this is acceptable since the simple tennis model is stationary. This heuristic could not be used in a real tennis game.

4.3.6 Algorithm

Finally, the pseudo-code of our MCTS algorithm can be seen in the following iterative algorithm, see Figure 4.10.
4.4 Experiments

4.4.1 Initial Model

In this case, we are need to find a strategy for P2 and see if our player can adapt and win the opponent by choosing the right actions based on some policy. The objective is also to translate this policy into human understandable terms. Please note that the transition probabilities are unknown to the players, so that the use of backward dynamic programming is not an option.
To put our work in better perspective, we have compared our approach with three popular Temporal-Difference Learning algorithms\cite{60}. To do so, we have tested all the algorithms against the simple model experiment described in Section 4.2.1. The TD algorithms chosen are: Sarsa, Q-Learning and Sarsa($\lambda$) with eligibility traces.

Figure 4.11 shows the comparison between the algorithms and the random policy after 10,000 episodes. The particular values used in the algorithms are as follows: $\epsilon$-greedy action selection $\epsilon = 0.1$, discount rate $\gamma = 1$, learning rate $\alpha = 0.1$ and the trace-decay parameter $\lambda = 0.5$. Sarsa($\lambda$) uses replacing eligibility traces. As for the MCTS implementation, we have used $C = 1.0$ and $t = 1,000$. The reward scheme used in the TD methods is as follows: -1, if the player loses the point, 1, if the player wins the point and 0 in all other cases.

These plots show that all assessed algorithms are able to make the player P2 win very early during the episodic runs, improving over the random policy results. The on-policy Sarsa($\lambda$) algorithm shows the best results out of the three TD algorithms. However, MCTS shows the best winning rate depicting an improvement percentage of more than 45% over the random policy and almost 10% over the Sarsa($\lambda$) algorithm.

As for the winning strategy, Table 4.6 shows the best strategies found after running MCTS and Sarsa($\lambda$) algorithms for 10,000 episodes. It is interesting to note that MCTS
found that the most successful policy for P2 consists in taking risks in all occasions.

<table>
<thead>
<tr>
<th>State</th>
<th>Action MCTS</th>
<th>Action Sarsa(λ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ll</td>
<td>DLR</td>
<td>CCR</td>
</tr>
<tr>
<td>RrS</td>
<td>DLR</td>
<td>CCR</td>
</tr>
<tr>
<td>Rl</td>
<td>DLR</td>
<td>DL</td>
</tr>
<tr>
<td>Lr</td>
<td>CCR</td>
<td>DLR</td>
</tr>
<tr>
<td>Rr</td>
<td>DLR</td>
<td>DLR</td>
</tr>
<tr>
<td>LrS</td>
<td>CCR</td>
<td>CCR</td>
</tr>
<tr>
<td>RrS</td>
<td>CCR</td>
<td>DLR</td>
</tr>
<tr>
<td>RrF</td>
<td>CCR</td>
<td>CC</td>
</tr>
<tr>
<td>LrF</td>
<td>CCR</td>
<td>DL</td>
</tr>
<tr>
<td>RlF</td>
<td>DLR</td>
<td>CC</td>
</tr>
<tr>
<td>LrS</td>
<td>DLR</td>
<td>CCR</td>
</tr>
<tr>
<td>LrF</td>
<td>CCR</td>
<td>DL</td>
</tr>
</tbody>
</table>

Table 4.6: Best strategies after 10,000 episodes

It is also interesting to see that both strategies do not converge. Runs of over $10^8$ episodes using a greedy policy in the limit ($\epsilon = 1/t$) for the Sarsa($\lambda$) algorithm were executed. The results showed that the optimal strategy found was never identical to the MCTS one, which proves that an on-policy TD control algorithm, such as Sarsa, learns the safest policy, although in this case it might not be the best one.

4.4.2 Full Tennis Model

Over 5,000 actions from more than 10 hours of recordings were captured and analyzed, covering men’s and women’s matches on both hard and clay courts. In this chapter, we will present the results from three professional matches. The first one presents an interesting challenge because of its close result: the 2010 US Open Women’s Semi Final between Na Li and Serena Williams. The latter won the match by 7-6, 7-6. Our goal is to try to find a policy for Na Li that could enable her to beat Serena Williams.

Once all the state and action information for each of these players has been captured and recorded using the computer vision system, the transition probabilities are then estimated as per Section 4.2.2. The information fed into the system corresponds to the full match under analysis and does not include data from other matches. At this point, a mathematical model can be built, which is used as a framework where Monte Carlo simulations can be
executed.

Figure 4.12 shows the profile of points won by each player during the original match, as well as the results for Na Li after executing the MCTS and Q-Learning\textsuperscript{[71]} algorithms. The particular values used in the algorithms are as follows: $\epsilon$-greedy action selection $\epsilon = 0.3$, discount rate $\gamma = 1$, learning rate $\alpha = 0.1$. As for the MCTS implementation, we have used $C = 1.0$ and $t = 1,000$.

The reward scheme used in both algorithms is as follows: $-1$, if the player loses the point, $1$, if the player wins the point and $0$ in all other cases.

These plots show that both algorithms are able to make the player under analysis (Na Li) win, improving over the original results. MCTS, however, shows the best winning rate depicting an improvement percentage of more than 33% over the original points won by Na Li.

![MCTS Results](image)

Figure 4.12: Points won profile comparison between original match results, Q-Learning and MCTS

As for the winning strategy, the Table 6 shows the best strategies found after executing the MCTS algorithm for 10,000 episodes. This table shows the actions and states as well as the value of the action taken and the visit count for each of those states.

It is interesting to note that the two most successful strategies provide a 100% win ratio as the input data shows the same behavior. Nevertheless, these states were only encountered once during the original match, so probably its importance should be somehow
4.4. EXPERIMENTS

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Value</th>
<th>Visited</th>
</tr>
</thead>
<tbody>
<tr>
<td>2R2R01</td>
<td>BHD LTS1</td>
<td>12647</td>
<td>12647</td>
</tr>
<tr>
<td>5NCR01</td>
<td>FHD LTS1</td>
<td>10287</td>
<td>10287</td>
</tr>
<tr>
<td>2ACR01</td>
<td>BHD LTS1</td>
<td>9891</td>
<td>14938</td>
</tr>
<tr>
<td>CRCA00</td>
<td>BHX LTS1</td>
<td>5325</td>
<td>51513</td>
</tr>
<tr>
<td>CACA01</td>
<td>BHX LTS1</td>
<td>5295</td>
<td>10614</td>
</tr>
<tr>
<td>RFSAC10</td>
<td>BHX LSL0</td>
<td>4922</td>
<td>10000</td>
</tr>
<tr>
<td>CACR01</td>
<td>FHX RTS1</td>
<td>4009</td>
<td>20120</td>
</tr>
<tr>
<td>RFSDC00</td>
<td>FHX RTS1</td>
<td>3337</td>
<td>10000</td>
</tr>
<tr>
<td>3ARFA</td>
<td>FSAT NA1</td>
<td>2152</td>
<td>10000</td>
</tr>
<tr>
<td>RSSAC00</td>
<td>BHX LTS1</td>
<td>2032</td>
<td>10000</td>
</tr>
<tr>
<td>5RCA00</td>
<td>FHX RTS0</td>
<td>1499</td>
<td>16070</td>
</tr>
</tbody>
</table>

Table 4.7: Best strategies after 10,000 episodes

lessened. As for the rest of the actions, we can see that the best results for Na Li respond to the following patterns: extreme cross court backhand risky shots are used to unbalance the situation that can be finished off by cross court forehands. When serving, first serves to the body in the deuce court and to the T in the advantage court produced the best results. When returning S. Williams’ first serves, the best results were obtained slicing the ball to her backhand.

The second match being analyzed is the 2009 French Open 4th Round match between Rafael Nadal and Robin Soderling. This match is also very interesting since it has been the only match lost by Nadal in the French Open since 2005.

This match displayed 4 sets covering 271 rallies and 1628 state-action pairs. In order to identify the strategy that led Soderling to win this match, we have executed the MCTS algorithm from this player’s point of view, i.e., we have fixed Nadal’s strategy to what was recorded with the vision module, but have expanded Soderling’s strategy to maximize his strengths. The following table shows the best strategies found after running 10,000 episodes:

The rallies depicted in Figure 4.13 help visualize some of the strategies identified in the table above (Note: Nadal is depicted on the top court and Soderling on the bottom court).
CHAPTER 4. IDENTIFICATION OF OPTIMAL STRATEGIES

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Value</th>
<th>Visited</th>
</tr>
</thead>
<tbody>
<tr>
<td>2R2R00</td>
<td>BHXLT0S</td>
<td>624</td>
<td>1120</td>
</tr>
<tr>
<td>CVCR00</td>
<td>FHXXLT0S</td>
<td>753</td>
<td>1461</td>
</tr>
<tr>
<td>5R1R00</td>
<td>FHDLT0S</td>
<td>553</td>
<td>1094</td>
</tr>
<tr>
<td>CN2R00</td>
<td>FHXXLT0S</td>
<td>642</td>
<td>1374</td>
</tr>
<tr>
<td>CNCR00</td>
<td>FHXXLT0S</td>
<td>694</td>
<td>1696</td>
</tr>
<tr>
<td>1RCR00</td>
<td>FHDLT0S</td>
<td>548</td>
<td>1388</td>
</tr>
<tr>
<td>2A5R00</td>
<td>FHDLT0S</td>
<td>479</td>
<td>1728</td>
</tr>
</tbody>
</table>

Table 4.8: Best strategies after 10,000 episodes for Soderling

From the analysis, it is clear that one of the main strategies employed by Soderling was to attack the left side of Nadal’s court, either using cross court backhands or forehands down the line. Nadal being left-handed usually gets attacked on his backhand, but Soderling consistently attacking his forehand was able to unbalance him frequently. According to the table above, this represents the most successful strategy employed in this match.

The final match analyzed is the 2010 Masters 1000 Madrid Open Final between Novak Djokovic and Rafael Nadal. This is also an interesting match because it represents another loss for Rafael Nadal also in clay, his most successful surface. In this case, the game was finished in 2 sets with 133 rallies and 1065 state-action pairs.

Similarly to what we did with the previous match, we will try to identify Djokovic’s
most successful strategies in an attempt to find out what worked best for Djokovic in that match. The Table 8 shows the best strategies found after running 10,000 episodes.

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Value</th>
<th>Visited</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR5A00</td>
<td>BHXXLTS0</td>
<td>860</td>
<td>1609</td>
</tr>
<tr>
<td>CNCR01</td>
<td>BHXXLTS1</td>
<td>792</td>
<td>2017</td>
</tr>
<tr>
<td>CA5R01</td>
<td>FHXXRTS1</td>
<td>728</td>
<td>1233</td>
</tr>
<tr>
<td>5R5R00</td>
<td>FHD LTS1</td>
<td>634</td>
<td>4196</td>
</tr>
<tr>
<td>CACR01</td>
<td>FHXRTS1</td>
<td>520</td>
<td>2848</td>
</tr>
<tr>
<td>CRCR00</td>
<td>BHXXLTS1</td>
<td>456</td>
<td>8320</td>
</tr>
<tr>
<td>5A5R00</td>
<td>FHD LTS1</td>
<td>376</td>
<td>1603</td>
</tr>
</tbody>
</table>

Table 4.9: Best strategies after 10,000 episodes for Djokovic

Figure 4.14 visualizes some of the strategies identified in the previous table (Note: Djokovic plays on the top court and Nadal on the bottom court).

Comparing the results from these two last matches, it is noticeable that both exhibit a similar pattern: to attack the left side of Nadal’s court, either using cross court backhands or forehands down the line. Of course, in order to carry out successfully this strategy, you need to be a very good player, attack the ball on the rise and get Nadal slightly off-court. However, these experiments highlight the right strategy to use to be able to try to beat
4.5 Conclusions and further research

Automatically extracting tactical information from broadcast sports videos is gaining popularity. Additionally, it also makes for an interesting topic in multimedia processing because it combines computer vision techniques on real video sequences with data mining and machine learning.

In this chapter, focused on tennis analysis, we have developed an MDP-based framework where machine learning algorithms can be executed in order to identify optimal policies. In this sense, we have presented a novel modification to the Monte Carlo Tree Search algorithm that proved to behave better than other popular Temporal Difference algorithms.

We have also presented two tennis models and have applied the full model to several real life matches where the most successful policies have been identified using Monte Carlo simulations. The results show that the identified policies do allow for an increase in the percentage of points won during simulations, offering a new insight into tennis tactical analysis. The examples include recent famous tennis matches with clear tactical conclusions.

We can therefore infer that the described paradigm consisting of building a simulation model for a match or a player through computer vision data, along with the use of Monte Carlo methods reveals an appealing proposition.

Future work is divided into two categories. Firstly, further work will be carried out to enhance the current simulation model by taking other variables such as the score and previous results into consideration, as well as incorporating audio clues for more accurate detections. Secondly, we would like to close the operational model loop by providing feedback to the players in real time to see if they can indeed improve their results. For this purpose, we will attempt to perform field-testing.
input
  F, forced situation indicator for the player receiving the ball;
  S, soft ball indicator for the player receiving the ball;
  R, M, W, random risk, mistake and winner percentage values respectively;
  \( \pi \), percentage increment;
  P, policy;
output
  \( O \in \{ \text{Winner, Forced Error, Unforced Error} \} \) if point finishes,
  \( O = \text{Next player state indicator} \ F' \) and \( S' \) otherwise
begin
  if \( S = 0 \) and \( F = 0 \)
    if \( M \) triggers an error as per \( P \), then return \( O = \text{Unforced Error} \). end
  else
    if \( W \) triggers a winner as per \( P \), then return \( O = \text{Winner} \). end
    else if \( R \) has triggered a risky shot as per \( P \), then \( F' = 1; S' = 0 \) //Forced ball
    else if \( R \) has not triggered a risky shot as per \( P \), then \( S' = 0; F' = 0 \) //Balance
  else if \( F > 0 \) and \( S = 0 \)
    R \leftarrow R - F \ast \pi
    M \leftarrow M + F \ast \pi
    if \( M \) triggers an error as per \( P \), then return \( O = \text{Forced Error} \). end
  else
    W \leftarrow W - F \ast \pi
    if \( W \) triggers a winner as per \( P \), then return \( O = \text{Winner} \). end
    else if \( R \) has triggered a risky shot as per \( P \), then \( F' = 0; S' = 0 \) //Balance
    else if \( R \) has not triggered a risky shot as per \( P \), then \( S' = F; F' = 0 \) //Soft ball
  else if \( S > 0 \) and \( F = 0 \)
    R \leftarrow R + S \ast \pi
    M \leftarrow M - S \ast \pi
    if \( M \) triggers an error as per \( P \), then return \( O = \text{Unforced Error} \). end
  else
    W \leftarrow W + S \ast \pi
    if \( W \) triggers a winner as per \( P \), then return \( O = \text{Winner} \). end
    else if \( R \) has triggered a risky shot as per \( P \), then \( F' = S+1; S' = 0 \) //Forced ball
    else if \( R \) has not triggered a risky shot as per \( P \), then \( S' = 0; F' = 0 \) //Balance
return \( O=(S',F') \)
end

Figure 4.2: Algorithm — Unbalancing model.
CHAPTER 4. IDENTIFICATION OF OPTIMAL STRATEGIES
Chapter 5

Conclusions

In any field, find the strangest thing and then explore it.

John Archibald Wheeler

The evaluation of tennis tactics is one of the least developed disciplines of analysis when compared to physiological, kinematic and technical analysis among others. Previous work focuses on providing a statistical and/or probabilistic analysis or otherwise, rely on computer vision algorithms to try to identify the player’s movements and actions, follow the ball trajectory and/or automatically annotate the score. Very few papers focus on tactical discovery and the ones that cover the subject, they treat it in a very simple way with over-simplified models.

In this thesis, we have first developed a computer vision system able to record relevant information directly from the broadcast video and transform it into a notational stream of data that is incorporated into a framework for the game of tennis.

The proposed computer vision system expands on previous work to automatically detect the tennis court in various lighting conditions and court surfaces as well as performing well with big occlusions. Several algorithms were presented and developed to identify and track the court in real time. The result is a stream of sequential data which includes meta data to augment the richness of the information. This approach allows to define a comprehensive framework where the concepts of game theory, reinforcement learning, data mining or artificial intelligence can be applied.

The use of multivariate sequential data mining along with a broad set of spatiotemporal
attributes proves to be an effective approach in order to discover successful tennis strategies within a tennis match, as discussed in Chapter 3. To this purpose, we introduced the concepts of event thresholds, rally similarities and pattern masks so that any winning pattern can be defined and mined. We also defined several mining problems in a similar fashion to other state-of-the-art data mining research papers. We also introduce the novel concept of prequels and a sequels of a pattern, that are characterized by a pattern mask and an nondeterministic finite automaton, respectively. These results demonstrate that this analysis can help tennis professionals focus on the successful sequences of strokes that led to winning points during the match or focus on the frequent rallies displayed across one or several matches, so they can better prepare for future games.

Finally, in order to explore adaptive strategies, we developed a Markov Decision Process-based machine learning framework in Chapter 4 were we tested several well known algorithms and developed a novel extension to the Monte Carlo Tree Search algorithm that proved to behave better than other popular Temporal Difference algorithms. We applied this framework to real tennis games and the results identify both the strategies used to outbalance the opponent as well as how to counteract this tactically.

Therefore, the combination of computer vision on real video sequences along with data mining and machine learning techniques confirms a definite avenue when it comes to sports tactical analysis. The objectives set at the beginning of the thesis have been accomplished throughout this work with several papers and novel algorithms published along the way. First, the proposed system does not requires an expert supervision to categorize and label video sequences. Only certain meta data is needed such as the type of stroke or the winner of the point to complement the automatically extracted positional information to fully describe a tennis game. From this moment onwards, the system is able to work autonomously.

This way, the system is able to identify the most frequent patterns displayed by either of the players during a tennis game and this information can be easily depicted in a diagram for a coach to quickly understand the results. Once the repetitive patterns are understood, the system is also able to identify the most successful tactics by using Monte Carlo simulations. The results can be translated again into high-level, user-friendly diagrams that portray the tactical information sought after.

Unavoidably, the work proposed here is by no means complete. There are a number of areas in which it can be extended and improved. Further improvements could be done to the computer vision module to include ball tracking and automatic action detection.
In the evaluation of the tennis tactics, other interesting areas for further work include the discovery of unforced-error and losing patterns or the effect of the score during the game.
Bibliography


