XML–Processing Using Field Notation Grammars Applied to Tennis Data

Daniel Weidner[1] and Dietmar Seipel[1]

University of Würzburg, Department of Computer Science,
Am Hubland, D – 97074 Würzburg, Germany

[1]{daniel.weidner,dietmar.seipel}@uni-wuerzburg.de

Abstract. XML is a well–known and frequently used data format for semi–structured, complex and nested data. In this paper, we store tennis data in an XML format; a match is structured into sets, games, and points, and for every point the location of the tennis field and the trajectory of the ball is given in some way.
In earlier work we had implemented an analysis of tennis data, that were entered by hand, using the declarative logic programming toolkit Declare. Recently, we have added an automatic recognition of the tennis data from tennis videos using convolutional neural networks (CNNs).
Unfortunately, the automatically obtained XML files from the CNN cannot be used for the intended analysis: coordinates are not given in a bird's–eye perspective, but as recorded from a camera behind the court; moreover, the ball trajectories have to be reduced to the moments when the ball is hit – instead of one position per video frame, i.e. 24 positions per second (the ball is hit once about every 1.5 seconds, e.g. a rally with 9 hits can take 13 – 15 seconds); also, the CNN frequently fails in recognizing the ball or the tennis field.
In this paper, we present a process of XML transformation based on field notation grammars. We have transformed the data obtained from the convolutional neural network into structured *clean* data which can then be used for further processing, like data mining. Our transformation uses *field notation grammars (FNGs)* and solves linear equations to *project* `x`– and `y`– coordinates *into the plane*. FNGs have been implemented in Declare using Prolog; they extend the common path notation XPATH of XML, and they combine XML–processing with Prolog.

Keywords: Prolog · XML Processing · Field Notation Grammars

1 Introduction

The field of artificial intelligence (AI) can be divided into symbolic and subsymbolic approaches, e. g., [2,6,8,16]. *Symbolic, knowledge– or rule–based AI* models central cognitive abilities of humans like *logic*, deduction and planning in computers; mathematically exact operations can be defined. *Subsymbolic or statistical AI* tries to learn a model of a process (e. g., an optimal action of a robot or the classification of sensor data) from the data.

In this work, we consider a *data mining analysis* of data from sports, where the process contains both. A convolutional neural network generates XML data from a tennis video, which is then transformed using Field Notation Grammars (FNG's), a component of the XML query, transformation, and update language FNQUERY [11]. The video can be either a whole match, or just a single set or game – in the wording of tennis. After the transformation, the newly generated clean data can be used for further processing like for domain specific data mining and for visualization.

The convolutional neural network creates `x`– and `y`– coordinates of the player positions, the ball trajectories and the court. Figure 1 shows instances created by the CNN. It refers to the final of the Australian Open 2015, where Novak Djokovic (bottom player in blue shirt and white shorts) played against Andy Murray (top player dressed in black). The light blue boxes are the areas where players were found by the CNN, while in the green box the ball was found. The ball trajectory is visualized for the whole process, and it is given in orange/red. Where the ball hits the ground or is hit by a player's racket, the CNN creates a big orange dot.

Fig. 1. Instances of CNN Process

Note that the ball trajectory is very distorted, because of the camera position. Therefore, if the rally is long, the visualized orange/red ball trajectory is confusing, like in the first picture of Figure 2. One rally means all ball exchanges between two players in one scoring point, i.e. from one serve to another. To tackle this problem, the rally is visualized in bird's–eye perspective, see in right picture of Figure 3, and we will show in this paper how to obtain this bird's–eye perspective.

In a recent bachelor thesis [4], we have analyzed which kind of videos are suitable for the convolutional neural network, or which kind of videos do not yield XML data at all. We concluded that hard courts generate better results than grass courts. We will also show the result of the analysis in this paper.

The structure of the XML file obtained from the CNN and its content needs adjustment. Thus, we transform the XML file by defining grammars in field notation, and translating coordinates by solving linear equations, which can be seen in Listings 5, 6 and 7.

The cleaned data can then be visualized with a tennis tool implemented based on the declarative toolkit Declare [12,17]. Also further processing like data mining together with a domain expert can be done in an easier way.

The rest of this paper is structured as follows: Section 2 presents the context of field notation grammars, and Section 3 introduces the generated XML file structure and explains our transformation. Next, we present the mathematic coordination processing in Section 4. The case study of the usability is shown in Section 5. In Section 6, we demonstrate how the new data can be visualized and processed further. Finally, Section 7 concludes with a summary.

2 Processing XML Data Using the Package PL4XML

The *Extensible Markup Language* XML is a well–known standard data format for representing and exchanging semi–structured data and knowledge. Prolog on the other hand can handle complex tree–structured objects nicely, and has an easy–to–use meta–programming property. Therefore Prolog is very helpful to process XML [1]. In [11,13,14,15], the declarative language FNQUERY/PL4XML was developed for querying, transforming and updating XML data. The language uses two alternative representations for XML in Prolog, the field notation and the graph notation; in this paper, we will use field notation.

2.1 XML Objects in Field Notation

Field Notation (FN) is a generic Document Object Model (DOM), where complex objects are represented as triples `T:As:C` consisting of a tag name `T` (the XML element name), a list of attribute/value pairs `As`, and a list of such triples `C` (the contents of the XML element). This has several advantages: the sequence of attribute/value–pairs is arbitrary, values can be accessed by attributes rather than argument positions, null values can be omitted, and new values can be added at runtime.

E.g., the XML file of Listing 2 can be represented as shown in Listing 1:

Listing 1. Fragment of an XML File in Field Notation

```
match:[
   court:[xLT:455, ...]:[],
   frame:[id:49, ...]:[], ...,
   rally:[
     frame:[...]:[], ...] ]
```

Since `rally` has no attributes, the attribute/value–list can be omitted. The binary infix–predicate `:=/2` allows for accessing the sub–elements (e.g., `Frame := Match@frame`) and the values of the attributes (e.g., `Time := Frame@time`) of an Fn–triple, which corresponds to the child–axis and the attribute–axis, respectively, of XPATH. This query language passed on XPATH is called *FNQuery*.

2.2 Field Notation Grammars

As mentioned before, FNQUERY includes a package for transforming XML data. The available transformations are summarized in the call `fn_transform(Options, +X, ?Y)` for various forms of options and data `X,Y`. The data can be given as items in field or graph notation, or as files.

The transformation can also be done by using field notation grammars. A *field notation grammar* (FNG) is defined by rules for the binary, infix–predicate `--->/2`. The grammar is applied to an FN–triple or an XML file (left side of `--->`) and creates a new FN–triple or XML file (right side of `--->`), respectively. For example, for the following FNG transforms the file of Listing 1 into another XML file containing the number N of sub–elements of `match` as the value of a single attribute `subs`; the second rule leaves all other FN–triples unchanged:

```
match:_:Content ---> match:[subs:N]:[] :-
   length(Content, N).
X ---> X.
```

The transformed item represents an XML file; in the following, a header for the XML version has been added:

```
<?xml version="1.0"?>
<match subs:153/>
```

This FNG can be read as follows: transform all triples with tag `match` and sub–elements `Content` to new triples with tag `match` and no sub–elements; the single attribute/value pair has the attribute `subs` with value N, where N is the length of `Content`, meaning the number of sub–elements of the original tag `match`.

If an XML–file should be transformed, the order of FNG's is very important. If more than one FNG's can be applied to a FN–triple, the first one is used

and the new transformed FN–triple is never touched again, meaning the second (third, etc.) FNG is ignored for this FN–triple. It is also important to write at least one FNG for all triples in an XML–file. Therefore, the rule `X ---> X` specifies that all triples `X` remain untouched. To ensure that this FNG, is only applied to FN–triples that one wants to keep unchanged, the rule has to stand at the end of the program, since the order is important, as mentioned recently before.

3 Transforming Tennis Data Represented in XML

This section introduces the translation of tennis videos into suitable data. This process is divided into several single processes, while in this work we focus on the XML transformation. The first step starts with a tennis video which recognizes data by a convolutional neural network (CNN) [10,3]. The CNN saves `x`– and `y`–coordinates of the court, both player positions, and the ball trajectory. It further recognizes whether a ball trajectory got a sharp bend, which means, whether the ball hits the ground or is hit by a player, and in addition, whether a rally starts or ends. All these coordinates are not given in bird's–eye perspective, which means that we have to convert them.

3.1 Architecture of the CNN–Generated XML File

We want to introduce the structure of the CNN–generated XML file, based on how the CNN works. Initially the tennis court is searched, unless it is contained in a frame and the CNN recognizes it successfully. Then, a tag `court` with the coordinates of the four corner vertices is created. Next, it is analyzed for every frame whether a rally starts (or later ends). Till the start, every frame is saved with a tag `frame`.

At the same time, it is determined if the camera has moved, more specifically, whether the coordinates of the court have changed. Is this the case, a new tag `court` is created and the CNN continues for every frame with the search for the start of the rally. When the rally starts, a tag `rally` is opened. Now every `frame` is a sub–element and all coordinates, as mentioned above, are saved. The first and last `frame` contain the attribute/value pair `event`:*rally_start* or `event`:*rally_end* respectively, and the `rally`–tag is closed after the latter. Inside a rally, it is also analyzed whether the coordinates of the court change. In this case it is proceeded analogously, meaning a tag `court` is created.

For the transformation, not all frames are interesting. We want to save the start and the end of the rally, and all frames, which contain the time where the ball hits the ground, which we can detect by the sharp bend mentioned in the previous section. In the later case, the frame contains the attribute/value pair `event`:*contact*. The CNN continues after finding a rally in the same way it has worked initially, until the video ends; then an XML file closes the tag `match`. In Listing 2, we see a fragment of an example XML tennis file created by the convolutional neural network.

Listing 2. Fragment of a Tennis File represented in XML and created by a CNN

```
<?xml version="1.0"?>
<match>
  <court xLT="455" yLT="210" xRT="811" yRT="209"
    xLB="281" yLB="556" xRB="992" yRB="557"/>
  <frame id="49" time="1.69" event="none"
    xPosTopPl="510" yPosTopPl="206" xPosBotPl="0"
    yPosBotPl="0" xPosBall="0" yPosBall="0"/>
  ...
  <rally>
    <frame id="175" time="6.03" event="rally_start"
      xPosTopPl="0" yPosTopPl="0" xPosBotPl="668"
      yPosBotPl="475" xPosBall="712" yPosBall="367"/>
    ...
    <frame id="179" time="6.17" event="contact"
      xPosTopPl="492" yPosTopPl="96" xPosBotPl="697"
      yPosBotPl="506" xPosBall="708" yPosBall="338"/>
    ...
    <frame id="264" time="9.1" event="rally_end"
      xPosTopPl="0" yPosTopPl="0" xPosBotPl="0"
      yPosBotPl="0" xPosBall="0" yPosBall="0"/>
 </rally>
 ...
</match>
```

3.2 Transformation Process

Next, we want to explain the transformation of the XML file. Thus, we consider how we want the file to look like in the end:

Listing 3. XML File we can handle.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<match>
  <set id="1" score_A="5" score_B="3">
    <game id="1" service="A" score_A="0" score_B="0">
      <point id="1" top="B" service="A" score_A="0"
         score_B="0" winner="A" error="0">
         <hit id="1" hand="forehand" type="ground"
            time="00:00:42" x="0.17" y="-12.07"/>
         <hit id="2" hand="backhand" type="ground"
            time="00:00:44" x="-0.49" y="5.89"/>
         <hit id="3" hand="backhand" type="ground"
            time="00:00:46" x="-3.92" y="-3.42"/>
      </point> ...
    </game> ...
  </set> ...
</match>
```

To realize this transformation, we consider all single conversions, that are necessary for this step by step. First, we see that both nestings differ from each other. The match is nested by set, game, point, and hit. This kind of nesting will be done in future work, since we can not recognize when a game or even a set is finished. For this, we want to include either background knowledge to derive when a game is finished or a score board analyzer. Since a rally represents a single point, we can rename all `rally` tags into `point` tags.

Next, we consider all `frame` tags with *contact* as their `event`–value. Also, we save the start (serve) and the end of all rallys, i.e. frames with attribute/value pair `event`:*rally_start* and `event`:*rally_end*, respectively. Later they get the first and last hit id.

Hits also save the `hand`, meaning if the ball is played with the forehand or the backhand. This is currently calculated by comparing the `x`–coordinate of the ball and the corresponding player. To determine which player plays the ball, we use the `y`–coordinate of the ball. Together with the background knowledge of whether a player is left or right handed, we can get this information.

The type of the hit in Listing 3, currently always *ground*, is part of future work. Last, we see the `x`– and `y`–coordinates. These stand for the coordinates, when the ball hits the ground. For all these coordinates we have the same problem: they are not coordinates in bird's–eye perspective, but are those if one would draw `x`– and `y`–axes into the video. We need a mathematical process to transform these numbers, into to the plane. To convert an arbritary point into the plane, the process needs the currently used court coordinates. Therefore we will search for the last tag with the name `court`, by using an FnQuery axis. The mathematical transformation is explained in the next section.

3.3 Field Notation Grammars for Transforming Tennis Data

Alltogether, we can see in Listing 4 the field notation grammar which transforms all these frames; note that the mathematical process is done with the predicates `vanish_point`, `view_point` and `project_point`.

Listing 4. All FNG transformations togehter

```
Frame ---> Hit :-
% Just filter frames
   fn_item_parse(Frame, frame:_:_),
   Time := Frame@time,
   Event := Frame@event,
% Only frames where a rally starts or end
% or a contact happened are processed
   member(Event,[rally_start,rally_end,contact]),
% Ball and Playerpositions are read
   [XPosBall, YPosBall, XPosTopPl, XPosBotPl] :=
      Frame@[xPosBall, yPosBall, xPosTopPl, xPosBotPl],
   Source_Point = point(XPosBall,YPosBall),
% Determine the closest preceding court-tag
```

```
% and save all coordinates
   Court := Frame/preceding::court,
   [XLT, YLT, XLB, YLB, XRT, YRT, XRB, YRB] :=
      Court@[xLT, yLT, xLB, yLB, xRT, yRT, xRB, yRB],
   LT = point(XLT,YLT), LB = point(XLB,YLB),
   RT = point(XRT,YRT), RB = point(XRB,YRB),
% Calculate vanish- and viewpoint and
% project the ball-positioni into the plane
   vanish_point(LT,LB,RT,RB,Vanish),
   view_point(LB,RB,Vanish,View),
   project_point(Source_Point,
      LB, RB, Vanish, View, Target_point),
   Target_point = point(XPos,YPos),
% Determine whether the top or bottom player plays
% the ball and therefore calculate back- or forehand
   ( ( YPos < 0 ) ->
     ( XPosPl = XPosBotPl )
   ; ( XPosPl = XPosTopPl ) ),
   Handedness = right,
   determine_hand(XPosBall, XPosPl, Handedness, Hand),
   Hit = hit:[id:'1', hand:Hand, type:'ground',
              time:Time, x:XPos, y:YPos]:[].
```

After processing the `frame`–tags, we do not need the `court`–tags. We can also omit all `frames` outside the `rally`–tags, all inside with attribute/value pairs `event`:*none*. This can be done with an FNG rule of the form `T:As:Es ---> ''`, where `T = frame` or `T = court`. E.g., for `T = frame`, elements with an attribute `event` with the value `none` are removed by the first Prolog rule, since they are replaced by an empty string `''`.

```
Frame ---> '' :-
   fn_item_parse(Frame, frame:_:_),
   none := Frames@event .

Court ---> '' :-
   fn_item_parse(Court, court:_:_).

% Keep all other FN-triples untouched:
X ---> X .
```

The previous field notation grammars include the call `fn_item_parse/2`, which normalizes the two forms of Field Notation triples to the longer form. It transforms abbreviated Field Notation Grammars, i.e. tags without attributes `I1 = T:C`, to the long form `I2 = T:[]:C`, and it leaves `I3 = T:As:C` unchanged. We use this to ensure that it can not happen that both forms (the short and long form) of the FN–triples appear in one XML file; without this, the process would stop in a failed state.

4 Mathematical Coordinate Transformation

As mentioned in the previous section, the coordinates of the generated XML file are the `x`– and `y`–coordinates that can be seen in the video, and therefore not the `x`– and `y`–coordinates in the plane – the so called bird's–eye perspective.

To translate these coordinates, we need methods from *perspective geometry* [9]. Since symmetry is lost under perspective projection, a simple rotation is not valid. For the translation we need to calculate the vanish point and the view point.

- The *vanish point* is the point in a perspective drawing onto which parallel lines appear to converge.
- The *view point* is the reflection point of the perpendicular point from the vanish point on the ground line, in our case `x`–axis.

We know that the tennis court is a rectangle, and so we have parallel side lines. The intersection of the perspective side lines will meet in the vanish point. The CNN provides all four vertices of the court, so we can easily calculate the vanish point. This can be seen in Listing 5.

Listing 5. Calculate the Vanish Point from two lines

```
vanish_point(LT,LB,RT,RB,E) :-
   LT = point(xLT,yLT), LB = point(xLB,xLB),
   RT = point(xRT,yRT), RB = point(xRB,yRB),
   M1 is (yLB-yLT)/(xLB-xLT),
   M2 is (yRB-yRT)/(xRB-xRT),
   E1 is (M1*xLT-M2*xRT-yLT+yRT)/(M1-M2),
   E2 is M1*(E1-A1)+yLT,
   E = point(E1,E2).
```

From this, we can easily calculate the view point, by doubling the distance from the perpendicular point on the ground line to the vanish point. This calculation can be seen in Listing 6.

Listing 6. Calculate the View Point

```
view_point(A,C,E,V) :-
   A = point(A1,A2), C = point(C1,C2), E = point(E1,E2),
   M1 is (C2-A2)/(C1-A1), M2 is -1/M1,
   Z1 is (M1*A1-M2*E1-A2+E2)/(M1-M2),
   Z2 is M1*(Z1-A1)+A2,
   V1 is 2*E1-Z1, V2 is 2*E2-Z2,
   V = point(V1,V2).
```

An arbitrary point p can be projected into the plane as follows. First, we take a ray starting from the vanish point v through the point p. We calculate the intersection w between the ray and the ground line. Next, we take the perpendicular l through the point w on the ground line. A second ray starting from the view point through the point p intersects the line l in the target point in the plane p', and we are done. We see this calculation in Listing 7.

Listing 7. Project an Arbitrary Point into the plane

```
project_point(Starting_Point, A, C, E, V, Target_Point) :-
   Starting_Point = point(AA1,AA2),
   A = point(A1,A2), C = point(C1,C2),
   E = point(E1,E2), V = point(V1,V2),
   M1 is (E2-AA2)/(E1-AA1), M2 is (C2-A2)/(C1-A1),
   AC1 is (M1*AA1-M2*A1-AA2+A2)/(M1-M2),
   AC2 is M1*(AC1-AA1)+AA2,
   M3 is -1/M2, M4 is (AA2-V2)/(AA1-V1),
   Target_1 is (M3*AC1-M4*AA1-AC2+AA2)/(M3-M4),
   Target_2 is M3*(Target_1-AC1)+AC2,
   Target_Point = point(Target_1,Target_2).
```

In the left picture of Figure 2, we can see a shot of the CNN process. The green box is the current search space for the ball. The cyan and blue box show the search space for the bottom and top player, respectively. In orange the ball trajectory is saved. Note that the CNN displays the whole trajectory during the detection, while only the current searching boxes are displayed. The picture refers to the match of Novak Djokovic (top player with white shirt and green shorts) versus Daniil Medvedev (bottom player with white shirt and white shorts) in the final of the Australian Open in 2021, played on a light blue hard court.

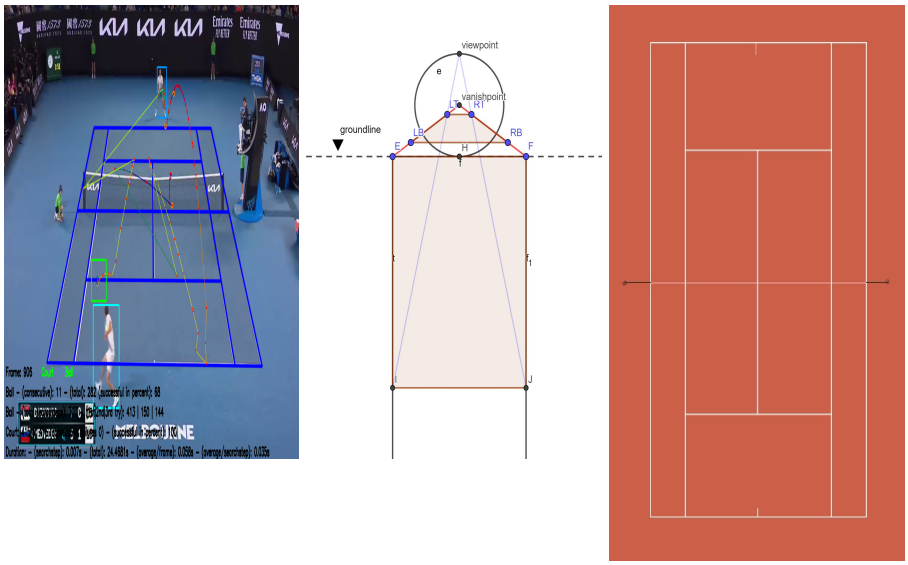

Fig. 2. Project the Court into the Plane

The center picture, which was created with the software Geogebra [7], visualizes how the projection into the plane is done. We can see the vanish point and the view point. Also, we can see parallel lines, which represent the tennis side lines. The visual rays show how an arbitrary point is projected into the plane. The projected tennis court starts on the ground line. Finally, on the right side we can see a final tennis court.

5 Case Study for Video Transformation with CNN

We have evaluated different tennis videos to determine properties in the videos, for which the CNN–generates results [4]. It is impossible to use a video from a personal device as an input. Since Australian Open, US Open, both having hard courts, French Open, having clay court and Wimbledon, having grass court (and all four years Olympic Games, where the court depends on the location) are the most popular tennis tournaments, the corresponding videos have the best camera resolution, and so we get the best results, see Table 1.

	# Videos	# Videos with correct detection	Rate in %
Hard Courts	49	26	53,06%
Australian Open	17	12	70,59%
US Open	19	13	68,42%
Clay Court	27	7	25,93%
French Open	20	7	35%
Grass Court	30	2	6,67%
Wimbledon	20	2	10%

Table 1. Rate of Correct Detection

Grass courts, for example at the Wimbledon tournament, are very hard to detect; only seven of the 20 Wimbledon videos and 0 of 10 other grass court videos have worked. Instead, hard courts like those at the Australian and US Open are detected correctly in about two of three times, i.e. 12 of 17 at the Australian and 13 of 19 at the US Open. Furthermore, one of 13 additional hard court videos has worked. For clay court, we have seven out of 27 videos, and all seven were filmed at the French Open. For videos which detect all attributes correctly, we get an Xml file, where our transformation works well.

6 Visualization and Further Processing of the Data

Once the data are in a useful structure with useful information, we want to visualize the data. For this, the declarative toolkit Declare offers a tennis tool. In Figure 3, we can see the graphical user interface for the tennis tool. The

example rally on the right side is the one of Figure 1, i.e. from the final of the Australian Open 2015, Novak Djokovic vs. Andy Murray.

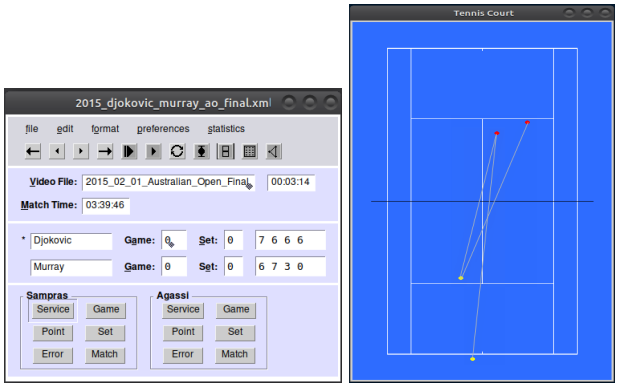

Fig. 3. Graphical User Interface of the Tennis Tool

In former work [17,12], we had worked with XML files entered by hand. In [18], we have presented a further processing of the data with methods of explorative data mining: we created association rules by generating spatio–temporal data from the XML data. Usually, the software Weka [5] generates many association rules, and a suitable *pre–* and *post–processing* by a *domain expert* is necessary to reduce the amount of generated rules. Finally, we were able to obtain suitable association rules. A further interpretation of the rules could for example analyze how often and under which circumstances players play the ball diagonal (called cross) or parallel to the sidelines (called longline).

7 Conclusions and Future Work

In this work, we introduce a tennis data transformation process. Starting with a video of a tennis match, we generate XML files with a convolutional neural network. *Logic programming* and *field notation grammars* were very useful for transforming these unclean XML data: we filter important data by removing useless data, and we put the data in a structure that can be used for further processing, for example data mining like we did in former works.

But by now, we only have stored the `x`– and `y`–coordinates and the type of hand, forehand or backhand. In the future, we plan to complete the data by using *meta data* and *background knowledge* (physical world). This knowledge can be used to generate more attributes and to detect anomlies, like excluding impossible data. For example, it is unlikely that a ball is hit perpendicular into the air.

We are also planning to implement intelligent heuristics in logic programming for (partially) determining the *score* – even for incomplete XML files – and for comparing it with the score from a detected score board.

References

1. Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.
2. Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational Inductive Biases, Deep Learning, and Graph Networks. *arXiv preprint arXiv:1806.01261*, 2018.
3. Michael Baumgart. Erkennung von Spielstand, Schlagposition und Spielertrajektorien beim Tennis. Master Thesis, University of Würzburg, 2019.
4. Moritz Braun. Analyse eines Tools zur Erkennung von Spielfeldern, Spielern und Balltrajektorien aus Tennisvideos. Bachelor Thesis, University of Würzburg, 2022.
5. Eibe Frank, Mark A. Hall, and Ian H. Witten. *The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, fourth edition, 2016.
6. Ben Goertzel. Perception Processing for General Intelligence: Bridging the Symbolic/Subsymbolic Gap. In *International Conference on Artificial General Intelligence*, pages 79–88. Springer, 2012.
7. Markus Hohenwarter. Geogebra. *Available online at https://www.geogebra.org/*, 2002.
8. Clayton McMillan, Michael C Mozer, and Paul Smolensky. Rule Induction through Integrated Symbolic and Subsymbolic Processing. In *Advances in Neural Information Processing Systems*, volume 4, pages 969–976, 1992.
9. Joseph L. Mundy and Andrew Zisserman. Appendix – Projective Geometry for Machine Vision. *Geometric Invariance in Computer Vision*, 1992.
10. Matthias Proestler. Erkennung und Nachverfolgung von Balltrajektorien bei 2D-Fernsehaufnahmen im Tennis. Master Thesis, University of Würzburg, 2017.
11. Dietmar Seipel. Processing XML–Documents in Prolog. In *Workshop on Logic Programming (WLP 2002)*, 2002.
12. Dietmar Seipel. Analyse von Tennismatches am Beispiel des Finales der US Open 2002: Pete Sampras – Andre Agassi, 2004.
13. Dietmar Seipel. *Pl4Xml–An Swi–Prolog Library for Xml Data Management (Manual)*, 2007.
14. Dietmar Seipel. Deductive Databases, Lecture Notes of a Course at the University of Würzburg, since 2015.
15. Dietmar Seipel, Falco Nogatz, and Salvador Abreu. Domain–Specific Languages in Prolog for Declarative Expert Knowledge in Rules and Ontologies. *Computer Languages, Systems & Structures*, 2018.
16. Paul Smolensky. Connectionist AI, Symbolic AI, and the Brain. *Artificial Intelligence Review*, 1(2):95–109, 1987.
17. Jürgen Wehner. Verwaltung und Analyse von Zeitreihen zu Videosequenzen. Diploma Thesis, University of Würzburg, 2003.
18. Daniel Weidner, Martin Atzmueller, and Dietmar Seipel. Finding Maximal Non-redundant Association Rules in Tennis Data. In *Declarative Programming and Knowledge Management*, pages 59–78. Springer, 2019.