XML Processing for Ball Trajectories Extracted from Tennis Videos

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 widely recognized as a prevalent and extensively employed data format for representing semi–structured, complex and nested data. In previous work, we have worked with XML files of tennis games. Starting with a video file, an XML file is created which contains the x– and y–coordinates of the court, players and ball trajectories. This is done using a *neural network* and image recognition. Furthermore, we have processed the data in the previous work, as it is unclean after creation.
In the present paper, we present an update on the *transformation approach*. We improve the transformation process without losing the old transformations. In contrast to old work, where we brought data recognized by the neural network into a clean structure, we now correct errors that occur when creating the data.
In addition, we update the *mathematical transformation* of the tennis data that occurs in x– and y–coordinates. The conversion of the data into bird's eye view is now more accurate and faster, and the new recursive approach is declarative.

Keywords: Prolog· XML Processing · Field Notation Grammars · Graph Notation Grammars ·

1 Introduction

The field of artificial intelligence (AI) can be divided into symbolic and subsymbolic approaches, e.g. [1, 4, 5, 10]. *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 paper, we will show that using both approaches is a good way to achieve hybrid intelligence. We use a neural network to generate data and then we use declarative methods such as rules and grammars to cleanse the extracted data. The resulting data can then be used for data mining. In previous work, we have applied this data mining to data that was entered by hand. So we want to move step by step towards an automated process that generates data that no longer differs from the hand data.

The *domain–specific, declarative tool* FnQuery/Pl4Xml was developed for *querying, transforming and updating* Xml data, cf. [8]. It contains two Prolog document object models: the field notation (FN) and the graph notation (GN), which represent Xml data as terms and facts in Prolog, respectively, and languages for querying, transforming and updating Xml data. In previous work, we have used this tool to clean unclean raw data provided by a neural network. We have written grammars that bring the Xml data into a clean structure. This can then be used for data mining. Nevertheless, errors occur when creating the original Xml file. We want to recognize some of these automatically based on the data and clean them up in a post-process step.

In addition, we have implemented a new mathematical transformation updating our previous work of [13]. The task is to convert coordinates into a bird's eye view. While these were previously estimated using projective geometry, we have now approximated the transformation declaratively and recursively. We will explain the advantages of this transformation and explain the individual mathematical steps in a comprehensible manner.

The rest of this paper is structured as follows: Section 2 provides an overview of the tennis tool and where we have worked in this paper, and Section 3 presents the context of Declare and field and graph notation grammars. Section 4 demonstrates the post-processing steps of the tennis tool and how the Xml tennis data is subsequently transformed. In Section 5, we show the new mathematical conversion of the coordinates into bird's eye view. Finally, Section 6 concludes with a summary.

2 Current Status of the Tennis Tool

Our current version of the tennis tool is packed in two Docker containers: the *Convolutional Neural Network (CNN)* Tennis Recognition [2] and a dockerized version of the Prolog–based *logic programming* tool Declare [15]. These containers will interact through a shared volume, with Declare accessing Xml files generated by the CNN [2, 6].

Declare uses Field and Graph Notation Grammars to transform the unstructured Xml representing ball trajectores into a clean and structured XML format. The resulting clean file can be visualized in a web browser using a Prolog web application. Additionally, we intend to incorporate data mining capabilities into the browser–based tool, similar to our previous work in [12].

The presented study builds upon our prior research conducted in [13, 14]. The present paper serves as a continuation of our previous work with significant updates: In the previous paper, we had discussed the process of transforming tennis data stored in Xml format using field and graph notation grammars. We had addressed the need for cleaning and structuring Xml files, generated by a convolutional neural network (CNN), for analysis. This is because the automatically generated Xml files had been found to be unsuitable.

The coordinates are recorded from a camera behind the court, rather than from a bird's–eye perspective. Additionally, not all frames of the video are interesting for the further process. Therefore, only important frames should be filtered.

While our earlier work had focused purely on cleaning up the data, we now also want to correct the errors in the CNN. This means that we are upgrading the transformation step, which will bring the data closer to the data we worked with in the first data mining process, but which had been entered by hand.

Secondly, we approximate the coordinates in a bird's eye view. While we previously calculated these imprecisely using focal and vanishing points, we will now approximate them recursively and declaratively. The new conversion of the data is faster and more accurate in comparison.

3 XML Transformations in Declare with FNQUERY

In this section, we will discuss the Declare toolkit and integrated XML maintenance. We will also describe the technique we use to write grammars to transform the XML data obtained.

3.1 The Declarative Toolkit Declare

The *declarative logic programming toolkit* Declare [7] is an open–source library developed in SWI–Prolog containing a deductive database system DDBASE that can access hybrid databases (e.g., relational, XML, or semantic web) to produce complex structured answers using a query language DATALOG* for the stratified evaluation of logic programs with embedded Prolog calls.

Declarative logic programming in Declare can be used as a *mediator tool* for managing and connecting hybrid database knowledge. For XML knowledge, we use the toolkit FNQUERY [8]. This integrated library of Declare for maintaining XML data can also be used via an integrated GUI [15].

3.2 XML Transformation using Field Notation Grammars

Field Notation (FN) is a generic Document Object Model (DOM) for representing objects in XML. It represents objects as Prolog triples `T:As:C` comprising a tag name `T`, attribute/value pairs `As`, and sub-elements `C`, named content. This hierarchical representation enables seamless navigation and manipulation of XML data.

PL4XML introduces Field Notation Grammars (FNGs) to transform XML data into field notation. These grammars use rules defined by the binary infix predicate `-->/2`, with the left side representing the input FN–triple or XML file and the right side representing the transformed output. By applying these rules in sequence, XML data can be modified, enriched, or restructured efficiently to meet specific needs.

Conversely, by storing XML objects as Prolog facts using an object-relational mapping, FNQUERY can implement the backward axes of XQuery [3, 11]. This allows us to define graph notation as a relational representation of XML.

The graph notation database can simultaneously store numerous XML elements. In this method, unique identifiers are assigned to elements and their descendants. As a result, XML elements stored in the graph notation database can be reconstructed.

To transition to the Graph Notation, set the Declare variable `fn_mode` to `gn`. An FN–triple in field notation can be stored in the GN database using the predicate `fn_to_gn/2`, which will return an identifier for the stored FN–triple. The stored FN–triple can be reconstructed with the inverse predicate `gn_to_fn/2`. Items in the graph notation database can be loaded from and saved to a file, provided `fn_mode` is set to `gn`.

4 Post–Processing of the Tennis Data in XML

Analogous to our work in [13, 14], Field and Graph Notation Grammars [9] are used to post–transform the XML tennis data. In the previous work we have transformed unclean data in XML form into a structure and cleanliness with which we can then apply data mining, similar to what we had presented in [12]. In that work the data was still entered by hand and therefore already in the correct format for the KDD–process.

In contrast, the problem with the data obtained after the automatic transformations from [13, 14] is that there are domain–specific errors and inaccuracies, e.g. in the case of a faulty ball hit. Either the CNN did not recognize a ball hit or the CNN recognized a hit that was not one.

In this paper we want to tackle these problems by writing field notation grammars that correct the data. For this purpose, we focused on three error classes that occurred when evaluating the tennis transformations. In addition to the two incorrect ball hit detections mentioned above, the following problem also occurred: In one video the CNN lost the court, player, and ball recognition. Accordingly, a kind of pause was detected for individual frames. This was linked to the fact that the CNN recognized the end of a rally, a sequence of back and forth shots between players, and subsequently recognized the start of a new one, which was actually a single rally in the original. In the following we will describe the field notation grammars, that tackle these problems.

4.1 An Erroneously Recognized Hit

This case is easy to check. We test whether the `y`–coordinates of two consecutive hits are equal. At the same time, the time difference must also be small enough, so that case 2, which we will deal with later, does not occur. In this case we calculate the center of both points, update the original hit and delete the surplus hit. In Listing 1 we parse all hits for this. We start by normalizing the hit tag using `fn_item_parse/2`. Since the attributes of all hits consist of six argument–value pairs `[id:ID, hand:Hand, type:Type, time:Time, x:X, y:Y]` we can unify them. Using the axes for the follower node, we can determine two consecutive hits with their `x`– (`XCoo, XNext`) and `y`–coordinates (`YCoo,YNext`) and their times

(Time1, Time2). Since the origin of the coordinate system lies in the center of the tennis court, a rally also means that the sign of the y- coordinates always changes. Or to put it another way mathematically, the multiplication of the y-coordinates of two consecutive hits is always negative. A violation would be a positive product. In this case we calculate the center point and update the x– and y coordinates of the original point using the command As*[/entry::X]. Finally, we delete the second hit using the infix–predicate <->/2 and update the IDs of the other hits in this point.

Listing 1. Field Notation Grammar in Graph Notation to Remove an additional Hit

```
Hit ---> Hit :-
    fn_item_parse(Hit, hit:As1:_),
    As1 = [id:ID, _, _, time:Time1, x:XCoo, y:YCoo],
    As2 := Hit/following/attributes::'*'
    As2 = [_ID, _Hand, _Type, Time2, XNext, YNext],
    YCoo*YNext > 0,
    Time2-Time1 < 2,
    NewX is (XCoo+XNext)/2,
    NewY is (YCoo+YNext)/2,
    As1*[/entry::[@x=NewX]],
    As1*[/entry::[@y=NewY]],
    Point := Hit/parent::point <-> [/hit::[As2],
    update_hit_ids_after_certain_id(Point,ID,New_Point,-1),
    Point := Point * New_Point.

X ---> X.
```

As with any grammar, the rule X --> X is used to leave FN–Triples unchanged if they are not considered. In this case, all FN–Triples that do not have hit as a tag [13, 14].

4.2 A Missing Hit

This case is of course much more difficult to consider, as we cannot calculate the coordinates of a hit that was not recognized. Furthermore, we cannot only look at the y–coordinates, because assuming two hits are missing, they are still alternating. However, we can use the time differences within a ball change to determine whether one or more hits were not recognized. We now want to hypothesize a point. In this paper we start with the hypothesis that the ball was hit in approximately the same neighbourhood as the previous hit by the same player. This means that assuming there is too much time difference between the hits with Id 2 and Id 3, we use the hit with Id 1 to hypothesize new coordinates for a point with a new Id 3. This procedure is automatically repeated until the time difference between all hits is small enough. Exception: Suppose the hit with Id 2 was not recognized. In this case we have the problem that we cannot use a previous hit of the player. However, to ensure that we still have a complete rally, we place a point in the court. Note that if the serve comes from the left, the point is in the middle of the right-hand fair court and vice versa, since this is the area for an error-free serve.

```
Point ---> New_Point :-
    fn_item_parse(Point, point:_:Hits),
    Hit1 := Hits/hit,
    Hit1 = [id:ID, _, _, time:Time1, x:XCoo, y:YCoo],
    Hit2 := Hit1/following_sibling::hit,
    Hit2 = [_, _, _, time:Time2, _, _],
    Time2-Time1 > 3,
    ( ID = 1  ->
      ( NewID = 1,
        ( XCoo < 0 ->
          NewX = 2.0575
        ; NewX = -2.0575 ),
      Sign(YCoo,Sign), NewY = -Sign*3.2 )
    ; ( NewID = ID,
        NewX := hit::preceeding/x,
        NewY := hit::preceeding/y ) ),
    NewTime is Time+3,
    New_Hit = [
       id:NewID, hand:forand, type:ground,
       time:NewTime, x:NewX, y:NewY ],
    split_list(Hits,ID, [List1,List2]),
    append([List1,New_Hit,List2],List3),
    update_hit_ids_after_certain_id(List3,ID,New_Point,1).

X ---> X.
```

4.3 A Rally Split into Two

As a final adjustment, we would like to merge two rallies into one if they were split into two due to incorrect recognition. We hope that in future work the CNN will have progressed so far that this phenomenon will no longer occur. It is difficult to tell from the data alone whether this occurs. Parallel viewing of the video is almost necessary, so the following grammar is applied afterwards. Since we want to merge two rallies, we have to adapt the XML structure. A rally symbolizes a scored point within a game. Using the axis search, we search for the successor of the first point with the tag `point`. We then merge the content (i.e. the rallies) of both points and then delete the next point.

```
Point ---> Point :-
   fn_item_parse(Point, point:_:Hits),
   NextPoint := Point/following_sibling::point,
   fn_item_parse(NextPoint, point:_:NextHits)
   Hits*[NextHits],
   delete(NextPoint).
```

5 Updating the Mathematical Process

In [13], we have described the mathematical process used to convert tennis coordinates into a bird's eye view. The underlying coordinates are the distorted ones you see on the screen. We will present a different mathematical transformation in this paper. We will recursively approximate the desired coordinates. This has several advantages: 1. our implementation is declarative, meaning the source code is significantly shorter, 2. the calculation is faster in runtime and 3. the approximated data is more accurate, since in the old calculation a vanishing point had to be approximated.

5.1 Problem Description

Given distorted coordinates of the corner points of the tennis court, as well as the coordinates of an arbitrary point. We are looking for the coordinates of the point in bird's eye view. The dimensions of the tennis court, i.e. the coordinates of the tennis court from a bird's eye view, are predefined. We place the point of origin $(0, 0)$ in the center of the tennis court. This means that the corner points have the coordinates $(\pm 4.115, \pm 11.885)$.

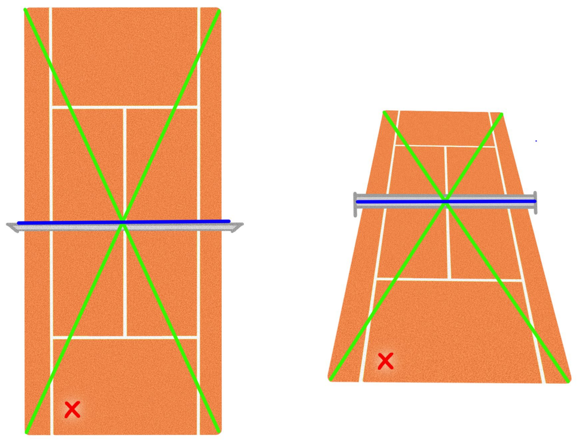

Fig. 1. Visualization of the coordiante calculation part 1

For the new mathematical calculation, we follow the following idea: We calculate parallel points in the distorted image and the corresponding point in bird's eye view. The tennis court in the distorted image resembles a parallelogram, but according to the coordinates it is not an exact parallelogram but a *trapezoid*. This is why we will refer to it as a trapezoid in the rest of the paper. We start with the four corner points of the tennis court. We first calculate the intersection point of the diagonals and place a horizontal line through this point and calculate the intersection point of this horizontal line with the two outer lines. Next, we check whether the y–coordinate of the point we are looking for is above or below the point of intersection, see Figure 1.

Depending on this, we look at the upper or lower trapezoid and proceed recursively. We end when the y–coordinates of the intersection point and the point we are looking for no longer differ, see Figure 2.

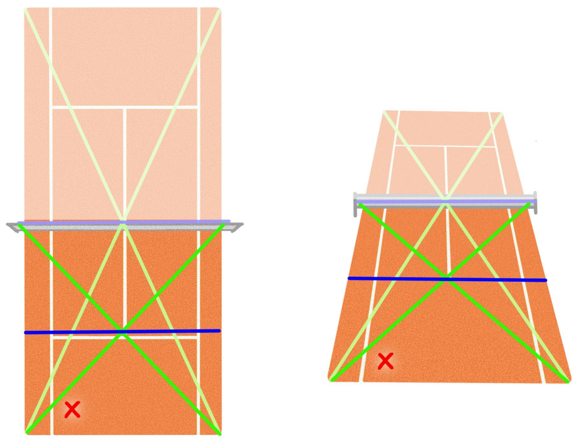
Fig. 2. Visualization of the coordiante calculation part 2

Now that we have calculated the y–coordinate, we still need the x–coordinate. To do this, we use the intersection points of the side line with a horizontal line and calculate the distance. Using cross–multiplication, we calculate the ratio between the arbitrary point and the outer points in both models. This allows us to calculate the x–coordinate.

5.2 Implementation

In this section we will explain the implementation. We start with pre–processing to check whether a ball is outside the field, because we cannot take this into account in our transformation. But the information *ball is outside the field* is sufficient, so the exact coordinates outside the field are irrelevant. Furthermore, the tennis court must be brought into the correct starting position. Then we can start calculating the coordinates, starting with `y` and then `x`. We have provided mathematical linear algebra calculations at the end of the chapter.

Ball is Out of Bounds First check if a ball is out of bounds (outside the court). Since the tennis court has four outer lines, we have to check whether a point is above or below each of them. As in all calculations, we take the corner points of the tennis court to calculate the corresponding straight line. By inserting the respective `x`– and `y`–coordinates, we can determine whether a point is below, exactly on or above the straight line.

```
check_ooB(Trapezoid) :-
   Trapezoid = (LB, RB, LT, RT, Source_Point),
   LB = point(XLB,YLB),
   RB = point(XRB,YRB),
   LT = point(XLT,YLT),
   RT = point(XRT,YRT),
   Source_Point = point(XS,YS),
   YS =< (XS-XLB) * (YLB-YRB) / (XLB-YRB) + YLB,
   YS >= (XS-XLT) * (YLT-YRT) / (XLT-XRT) + YLT,
   YS >= (XS-XLB) * (YLB-YLT) / (XLB-XLT) + YLB,
   YS >= (XS-XRB) * (YRB-YRT) / (XRB-XRT) + YRB.
```

Mirroring and Rotation (Normalization) of the Trapezoid Second, normalize and rotate the court such that its a parallelogram parallel to the `x`–axis. When creating coordinates using the CNN, the `y`–coordinates are mirrored on the `x`–axis. To normalize, we reverse this mirroring by negating the coordinates. For the rotation, we first check whether there is any rotation at all. To do this, we check whether the `y`–coordinates of the lower and upper points coincide, respectively. If this is not the case, we rotate by an angle that can be determined analogously to the calculation of a straight line gradient. The fixed point of the rotation is the point at the bottom left. Note, in the implementation, we stick to the speaking style of a trapezoid.

```
normalize_Trapezoid(Trapezoid, New_Trapezoid) :-
   Trapezoid = (
      point(XLB,YLB), point(XRB,YRB), point(XLT,YLT),
      point(XRT,YRT), point(XBall,YBall) ),
   New_Trapezoid = (
      point(XLB,-YLB), point(XRB,-YRB), point(XLT,-YLT),
      point(XRT,-YRT), point(XBall,-YBall) ).

rotate_left_bottom(Old_Trapezoid, Old_Trapezoid) :-
   Old_Trapezoid = (
      point(_,Y1), point(_,Y1), point(_,Y2)
      point(_,Y2), point(_,_) ).

rotate_left_bottom(Old_Trapezoid, New_Trapezoid) :-
   Old_Trapezoid = (
      point(XLB,YLB), point(XRB,YRB), point(XLT,YLT),
      point(XRT,YRT), point(XBall,YBall) ),

   Ang is -atan( (YRB-YLB) / (XRB-XLB) ),

   XRBnew is XLB + cos(Ang)*(XRB-XLB) - sin(Ang)*(YRB-YLB),
   YRBnew is YLB + sin(Ang)*(XRB-XLB) + cos(Ang)*(YRB-YLB),

   XLTnew is XLB + cos(Ang)*(XLT-XLB) - sin(Ang)*(YLT-YLB),
   YLTnew is YLB + sin(Ang)*(XLT-XLB) + cos(Ang)*(YLT-YLB),

   XRTnew is XLB + cos(Ang)*(XRT-XLB) - sin(Ang)*(YRT-YLB),
   YRTnew is YLB + sin(Ang)*(XRT-XLB) + cos(Ang)*(YRT-YLB),

   XBallnew is XLB + cos(Ang)*(XBall-XLB) -
               sin(Ang)*(YBall-YLB),
   YBallnew is YLB + sin(Ang)*(XBall-XLB) +
               cos(Ang)*(YBall-YLB),

   New_Trapezoid = (
      point(XLB,YLB), point(XRBnew,YRBnew),
      point(XLTnew,YLTnew), point(XRTnew,YRTnew),
      point(XBallnew,YBallnew) ).
```

Calculation of the `Y`–Coordinate We have now prepared the tennis court to calculate the coordinates from a bird's eye view. We first calculate only the `y`–coordinate and then use it to determine the `x`–coordinate. The calculation is recursive in that we iteratively halve the playing field, so to speak. First we make some preliminary considerations. We look at the distorted image and the bird's eye view at the same time. We know the coordinates of the ball in the distorted image, called `Ball = point(XBall,YBall)` (line 2 and line 13) and want the coordinates in bird's eye view called `NewBall = point(NewXBall,NewYBall)`. We also know the coordinates of the tennis corner points. We know these in both viewing perspectives. If we now intersect the diagonals, we determine a point that corresponds one to one in both images. We examine whether the `y`–coordinate of point Ball is above or below the diagonal intersection point. Since we normalized the distorted trapezoid or parallelogram at the beginning and prepared it for `x`–axis parallelism, we can now draw a horizontal line through the intersection of the diagonals. Now we calculate two more points, namely the intersection points of the outer lines with the horizontal line. In this way, we can divide the playing field into a lower and an upper half. We now continue working with the half in which point P lies. We repeat this process until we have approximated the y–coordinate of point Ball and thus also that of point NewBall.

```
 1  determine_Y_coordinate(Trapezoid,(YCourt1,YCourt2),YCourt) :-
 2     Trapezoid = (LB,RB,LT,RT,point(_,YBall)),
 3     LB = point(XLB,YLB),
 4     RB = point(XRB,YRB),
 5     LT = point(XLT,YLT),
 6     RT = point(XRT,YRT),
 7     intersect_two_lines_by_points(LB,RT,RB,LT,point(_,Y)),
 8     abs(YBall-Y) < 1,
 9     YCourt is (YCourt1+YCourt2)/2,
10     !.
11
12  determine_Y_coordinate(Trapezoid,(YCourt1,YCourt2),Y) :-
13     Trapezoid = (LB,RB,LT,RT,point(_,YBall)),
14     LB = point(XLB,YLB),
15     RB = point(XRB,YRB),
16     LT = point(XLT,YLT),
17     RT = point(XRT,YRT),
18     intersect_two_lines_by_points(
19        LB, RT, RB, LT, point(_,YIntersec) ),
20     abs(YBall-YIntersec) >= 1,
21     XS1 is ( YIntersec - YLB + (YLT-YLB) / (XLT-XLB)*XLB) /
22            ( (YLT-YLB) / (XLT-XLB) ),
23     XS2 is ( YIntersec - YRB + (YRT-YRB) / (XRT-XRB)*XRB) /
24            ( (YRT-YRB) / (XRT-XRB) ),
25     YCourt is (YCourt1+YCourt2)/2,
26     ( YBall < YIntersec ->
27       ( determine_Y_coordinate( ( LB, RB,
28            point(XS1,YIntersec),point(XS2,YIntersec),
29            point(_,YBall) ),
30            ( YCourt1, YCourt ), Y ) )
31     ; ( determine_Y_coordinate(
32            ( point(XS1,YIntersec),point(XS2,YIntersec),
33              LT, RT, point(_,YBall) ),
34            ( YCourt, YCourt2 ), Y ) ) ).
```

Calculation of the x–Coordinate To calculate the x–coordinate, we use the following coordinates and points: The last calculated intersection points of the horizontal line and the tennis sideline and our starting point of the ball `Ball = point(XBall,YBall)`. The side outlines are defined by the court corner points, see line 2-9. Since the tennis court is the same width at every point in real life, we can work with a tennis width of 8.23 meters. Figuratively speaking, the width of the tennis court gets smaller the further you go in the distorted image above. However, we can calculate this width w = `XR-XL` using the two intersection points. We can also calculate the distance d = `XBall-XL` from the left intersection point to the ball point. This distance is horizontal due to our previous calculation. We can now carry out the cross-multiplication: $\frac{\texttt{NewXBall}}{8.23} = \frac{d}{w}$

The distance `NewXBall` is now the `x`–coordinate if the origin is at the height of the left tennis outline. However, since we have placed the origin in the center of the field, we now have to take this shift into account (−4.115 in line 10) and we are done.

```
1  determine_X_coordinate(Trapezoid, NewXBall) :-
2     Trapezoid = (
3        point(XLB,YLB), point(XRB,YRB),
4        point(XLT,YLT), point(XRT,YRT),
5        point(XBall,YBall) ),
6     intersect_with_side_line(YBall,
7        point(XLB,YLB), point(XLT,YLT), XL),
8     intersect_with_side_line(YBall,
9        point(XRB,YRB), point(XRT,YRT), XR),
10    NewXBall is 8.23 * (XBall-XL) / (XR-XL) - 4.115.
```

A helping math method to calculate the intersection point of two lines, where both are defined by two points and a method to calculate the intersection point of one line and a horizontal line can be seen in the appendix. There you will also find the method that updates the IDs after a point has been added or removed (add or subtract 1).

6 Conclusion and Future Work

In this paper, we have presented our current work on a tennis tool. The tool takes a video of a tennis match as input and converts it into a XML document using a *neural network*. This first saves `x`– and `y`–coordinates of the player, tennis court and ball trajectories. However, the data is very unclean at the beginning. In an earlier work, we had started to transform them. However, in order to use the data for data mining, further post–processing steps have to be performed on the ball trajectories.

In this work, we have addressed the biggest errors that the neural network makes. For both transformations, we use the library PL4XML and grammars for maintaining XML data in field notation. Furthermore, we have introduced a new *mathematical transformation* to approximate coordinates in bird's eye view. In the future, we want to test different videos to get an overview of which errors in the data still need to be corrected.

Moreover, we hope to be able to analyze and evaluate a *data warehouse* with videos of different players and tournaments over the years and thus complete the entire loop of the tennis tool.

References

1. 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.
2. Michael Baumgart. Erkennung von Spielstand, Schlagposition und Spielertrajektorien beim Tennis. Master Thesis, University of Würzburg, 2019.
3. Chamberlin, Don and Florescu, Daniela and Robie, Jonathan and Simeon, Jerome and Stefanescu, Mugur. XQuery: A Query Language for XML. In *SIGMOD Conference*, volume 682, page 50, 2003.
4. Ben Goertzel. Perception Processing for General Intelligence: Bridging the Symbolic/Subsymbolic Gap. In *International Conference on Artificial General Intelligence*, pages 79–88. Springer, 2012.
5. 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.
6. Matthias Proestler. Erkennung und Nachverfolgung von Balltrajektorien bei 2D-Fernsehaufnahmen im Tennis. Master Thesis, University of Würzburg, 2017.
7. Dietmar Seipel. Declare – A Declarative Toolkit for Knowledge–Based Systems and Logic Programming, 2024.
8. Seipel, Dietmar. Processing XML–Documents in Prolog. In *Workshop on Logic Programming (WLP 2002)*, 2002.
9. Seipel, Dietmar. *PL4XML– An Swi–Prolog Library for XML Data Management (Manual)*, 2007.
10. Paul Smolensky. Connectionist AI, Symbolic AI, and the Brain. *Artificial Intelligence Review*, 1(2):95–109, 1987.
11. Walmsley, Priscilla. *XQuery*. O'Reilly Media, Inc., 2007.
12. Weidner, Daniel and Atzmueller, Martin and Seipel, Dietmar. Finding Maximal Non–Redundant Association Rules in Tennis Data. In *33th Workshop on (Constraint) Logic Programming*, pages 59–78. Springer, 2019.
13. Weidner, Daniel and Seipel, Dietmar. XML–Processing Using Field Notation Grammars Applied to Tennis Data. In *36th Workshop on (Constraint) Logic Programming*, 2022.
14. Weidner, Daniel and Seipel, Dietmar. An HTML 5–Based Graphical User Interface and a Transformation for Tennis Data in XML. In *37th Workshop on (Constraint) Logic Programming*, 2023.
15. Weidner, Daniel and Waleska, Marcel and Seipel, Dietmar. Interfacing the declarative toolkit declare using python and docker. In *35th Workshop on (Constraint) Logic Programming*, 2021.

A Appendix

```
update_hit_ids_after_certain_id(Point, ID) :-
   fn_item_parse(Point, point:_:Hits),
   last_element(Hits,hit:[]:[id:ID|_]).

update_hit_ids_after_certain_id(
      Point, ID,New_Point,Add_Sub) :-
   fn_item_parse(Point, point:_:Hits),
   split_list(Hits,ID, [List1,List2]),
   maplist( add(Add_Sub), List2, List3 ),
   append(List2, List3, New_Point).

intersect_two_lines_by_points(P1, P2, P3, P4, S) :-
%  Line 1 through P1 & P2
   P1 = point(X1,Y1),
   P2 = point(X2,Y2),
   M1 is (Y2-Y1) / (X2-X1),

%  Line 2 through P3 & P4
   P3 = point(X3,Y3),
   P4 = point(X4,Y4),
   M2 is (Y4-Y3) / (X4-X3),

   X is (M1*X1-M2*X3+Y1-Y3)/(M1-M2),
   Y is M1*(X-X1)+Y1,
   S = point(X,Y).

intersect_with_side_line(
      YBall, point(XB,YB), point(XT,YT), X) :-
   X is ( YBall - YB + XB * (YT-YB) / (XT-XB) ) /
        ( (YT-YB) / (XT-XB) ).
```