IDP documentation: analysis and visualization of position data in professional football

Andreas Schmelz

Technische Universität München

Abstract. In order to track the running performance of football players during official Bundesliga matches, the Deutsche Fußball Liga (DFL) has deployed a video based tracking system. This tracking system collects location and speed of every individual player with a frequency of 25 hertz. An analysis of this data allows clubs, trainers, and scientists to conclude about accomplished performance of players and their team. To make use of this data it has to be analyzed and visualized. This requires the transformation into a diagnostic relevant Data metrics.
The purpose of this interdisciplinary project(IDP) is to provide a basis of scientific research on a collection of relevant metrics based on given data sets. In especial we want to put exhaustion in correlation with speed, acceleration and additional parameters. The goal is to implement a data analysis tool with an easy to use interface, uses the provided DFL data sets and generate visualizations that also can be export the results in a reusable data format for further processing. ...

Keywords: football, analysis, visualization, big data

1 Problem

1.1 Requirements

As part of the IDP we want to develop an application that can

- Read the provided position data sets
- Set this data in relation to associated data files match-information and event-data
- Analyze the data according to relevant defined metrics and formulas
- These Formulas should have partially adjustable parameters
- The tool should immediately present results
- It should be possible to export the results for post processing and statistically analysis in external tools
- This analysis should be applicable to many (100+) files in one export-run

1.2 Given data

As given Data we were provided data directly from the DFL. The data is in a "self-describing" xml data format. By the nature of xml it contains a lot of

repeating meta data. Every node contains the structure of itself. Also because the format is string based instead of binary the data results in a large file size overhead. For every match, there are three files created: position data, match information and event data.

Position data stores the data of the position of every participating player, as well as the ball. Some data sets even contained officials and trainers.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <PutDataRequest MessageTime="2014-03-08T17:30:15.70+01:00"
       RequestId="1" TransmissionComplete="true"
       TransmissionSuspended="false">
3    <MetaData MatchId="DFL-MAT-0001Q1" Type="pitch-size">
4      <PitchSize X="105.00" Y="68.00"/>
5    </MetaData>
6    <POSITIONS EventTime="2014-03-08T15:31:45.20+01:00"
         MatchId="DFL-MAT-0001Q1">
7      <FrameSet Club="BALL" GameSection="firstHalf" Match="DFL
           -MAT-0001Q1" Object="DFL-OBJ-0000XT">
8        <Frame BallPossession="2" BallStatus="1" M="1" N="1" S
             ="0.37" T="2014-03-08T15:31:45.200+01:00" X="2.59"
             Y="-11.30" Z="0"/>
9        <Frame BallPossession="2" BallStatus="1" M="1" N="2" S
             ="0.37" T="2014-03-08T15:31:45.240+01:00" X="2.58"
             Y="-11.29" Z="0"/>
```

Listing 1.1: sample of the positions data set

The data is grouped into so called framesets that stores the position of a single participant (either player ball or official) internally referenced as DFL-Object of one half Time. Including exchange players there are around 50 frameSets for every match. In every frameset are around 68000 singe Frames. Every 40ms a frame is recorded and contains information about location, speed, time, game-/ and ball status. On total every game consists of around 3 Million Frames.

Match information contains the description of the participating parties, the match and a description of the players.

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <PutDataRequest MessageTime="2014-03-10T16:22:50.79+01:00"
       RequestId="1" TransmissionComplete="true"
       TransmissionSuspended="false">
3    <MatchInformation>
4      <General AwayTeamId="DFL-CLU-00000G" AwayTeamName="FC␣
           Bayern␣München" Competition="Bundesliga"
           CompetitionId="DFL-COM-000001" DL-PROVIDER-ID="
           g699283" GameTitle="VfL␣Wolfsburg:FC␣Bayern␣Munchen"
           HomeTeamId="DFL-CLU-000003" HomeTeamName="VfL␣
           Wolfsburg" Host="Die␣Liga␣-␣Fu ß ballverband␣e.V.␣(
           Ligaverband)" KickoffTime="2014-03-08T15:31:55.00+01
           :00" MatchDay="24" MatchId="DFL-MAT-0001Q1"
```

```
        PlannedKickoffTime="2014-03-08T15:30:00.00+01:00"
        Season="2013/2014" Type="Ligabetrieb" TypeOfSport="Fu
        ß ball"/>
5     <Environment Address="In␣den␣Allerwiesen␣1,␣38446␣
        Wolfsburg" AirHumidity="46" AirPressure="1033"
        Country="Germany" Floodlight="on" NumberOfSpectators=
        "30000" PitchErosion="none" PitchX="105.00" PitchY="
        68.00" Precipitation="none" Roof="open" SoldOut="true
        " StadiumCapacity="30000" StadiumId="DFL-STA-000003"
        StadiumName="Volkswagen␣Arena" Temperature="12"/>
6     <Teams>
7       <Team PlayerMainColorOfShirt="#FFFFFF"
          PlayerOtherColorOfShirt="#339933" Role="home"
          TeamId="DFL-CLU-000003" TeamName="VfL␣Wolfsburg">
8         <Players>
9           <Player FirstName="Diego" LastName="Benaglio"
              PersonId="DFL-OBJ-0000U1" PlayingPosition="TW"
              ShirtNumber="1" Shortname="D.␣Benaglio"
              Starting="true" TeamLeader="true"/>
10          <Player FirstName="Patrick" LastName="Ochs"
              PersonId="DFL-OBJ-0000U2" PlayingPosition="RV"
              ShirtNumber="2" Shortname="P.␣Ochs" Starting="
              true" TeamLeader="false"/>
```

Listing 1.2: sample of the match information

Match information gives clue about general information about the match as stated above. It also introduces and declares the DFL-objects closer. This contains such information as names, playing position, their associated team and whether a player is starting in the match.

Events contains the occurring events during the match like passes, fouls, free kicks shots and more.

```
1  <PutDataRequest MessageTime="2014-04-01T09:57:29.32+02:00"
       RequestId="1" TransmissionComplete="true"
       TransmissionSuspended="false">
2    <Event EventId="6992830015600006" EventTime="2014-03-08
         T15:31:56.95+01:00" InternalMatchId="DFL-MAT-0001Q1">
3      <Pass Blocked="false" Distance="medium" Height="flat"
         PassOrigin="ownHalf" PenaltyBox="false" Player="DFL-
         OBJ-0000IC" Successful="true" Team="DFL-CLU-00000G"/>
4    </Event>
5    <Event EventId="6992830015600007" EventTime="2014-03-08
         T15:32:00.62+01:00" InternalMatchId="DFL-MAT-0001Q1">
6      <Pass Blocked="false" Distance="long" Height="high"
         PassOrigin="ownHalf" PenaltyBox="false" Player="DFL-
         OBJ-0000O4" Successful="false" Team="DFL-CLU-00000G"/
         >
7    </Event>
```

Listing 1.3: sample of the events information

Events have a heterogeneous data format. Dependent on the type of event additional information needs to be added. In most cases it references the currently played match, the participating player or players and the time it occurred. As the format does not only change in the amount of parameters declared but also can have descendant children it makes it hard to get them parsed into a common structure. It also makes it hard to use a generic processing on these data without many exceptions. Those data sets use ids as references. This means the dataset is relational like and must be joined over these foreign key in order to get the entire un-partitioned data.

The position-data is mostly homogeneous, but we are provided by datasets of multiple seasons, in which the data format is different. This is not a showstopper, but requires additional effort in order to normalize and compare across these seasons. Another issue is that the xml format is self-describing, but no real documentation of the scheme is provided. This gives a basis for a reasonable guess but remains uncertainty about exact units, especially can we never be 100% sure all the data formats are the same for every attribute across all matches. While we will assume about the data validity upon some manual observations we implemented some tests to verify those assumptions are correct. With these checks we observed that:

- We cannot assume the ball to always be the first frameSets but rather we have to find the ball at arbitrary index.
- Not all Frames are continuous. The dataset sometimes has gaps in the dataset. This means we have to readjust framesets.
- Not all frameSets consist of a strictly monotonous increasing sequence of Frames. Correction was to sort the frames after importing them.
- Not all matches have the ball status and ball possession. An information we used to determine the active play information
- Some match-information do only list the starting players. We needed to make the program fail resistant against not assignable objects to a 'real' player.

2 Implementation

The program was implemented in the java programming language. Java programs are platform and operating system independent. The program can run on any computer providing a java virtual machine.

The class structure of the application can be seen in figure 1. The application follows the model view controller (MVC) design pattern. It is divided into the model, the data layer that defines a common data structure for the used datasets. The view is the visual representation of the currently loaded data. Those are the generated charts and images produced from data. The control is responsible for handling input of the user and updating the model and view as necessary.

This abstraction and decoupling follows the single responsibility principle. It defines clear closures and provides an reusable interface. This made it easy to have a visual in-application representation but also a batch export without

major adjustments. The exports are only a predefined set of configurations that control which settings should be applied to the data and the visuals. It also has the advantage to be extendable easily by calculated properties on the dataset, charts that are based on the data, or control elements that affect the view or the data.

2.1 Model

The data model is derived from the structure of the given file structure but sets the individual files in context with each other. The hirachy can bee seen on the right side of the UML diagram in figure 1 A `Game` defines to be all information that is available for a football match. It consists of the players position data (including speed and acceleration) and the `Match` information as defined in the match file (player and team description, officials, arena and other meta information) A game can not only access the players of its current match, but has access to a global repository of all players. This makes it possible to fill partial gaps of exchange players as some datasets only define the starting players. The position class consists of multiple frameSets, which is the collection of frames of a single player during one half time.

The multiplicity is. one IDP application can store multiple games. Every `Game` holds exactly one `Position` and one `Match` object. Every `Position` object is made of typically 50 `FrameSets`. One `FrameSet` holds 68000 `Frames`. A Match has at least 22 `Player` (usually more including exchange player, officials and trainers). Out of convenience and performance reason the reusable `MeanData` object was introduced. It wraps the functionality of creating gathered information about the single frames. For this always one minute of frames (1500) are gathered and provide an interface to merge and filter individual `MeanData` sets without having to access many single frames on every access. This `MeanData` object are created for properties such as velocity, acceleration, sprints, energy consumption, and relative velocity within a speed zone.

2.2 View

The views show the analyzed data. Usually properties are mapped by a function and then a group of values are reduced to one single data point. `MeanData` provides this for every given formula the information of sum, count, average and square sum, used for calculating the standard deviation on multiple `MeanData` objects.

This data is sued to show the average velocity (`visMean`) the players relative distance within a speed zone (`visZones`) and the average sprints of players (`visSprints`), the a textual representation is provided by the `Table`. Other views use different data. The animated football field (`visualField`) directly accesses the frames of every player. Naturally it is only possible to display one single game at a time. The `visSpeed` class displays the speed, acceleration and energy consumption of one currently selected `FrameSet`.

2.3 Controller

The `Config` class provides functionality to define which information should be filtered or shown. It therefore sets the current state into a global variable that can be read from all other classes.

2.4 Lifecycle

The lifecylce of the program states as follows

The program finds all Match-position file pairs inside a user selected root directory. The user can select pairs to be loaded into memory. The match and position xml files are red and their schema is detected. With this information the files are parsed and put into the Data model as described in 2.1. Additional the data is validated with predefined constrains. By this approach we abstract the inconsistencies of files away and do not have to check about different structures from now on. This data is then pre-analyzed and put into the `MeanData` slots that will later be reused by most of the visualizations. The `Meandata` only needs to be recomputed in case certain settings are changed. These settings should rarely change because it requires an entire recomputation of the original dataset. For this case the model will not only save the derived value but also the original one.

The user then can view the analyzed dataset in the visualizations such as the charts. He also is able to adjust settings during the run time and get almost immediate feedback resulting an updated view. For the currently loaded data set it is possible to export them. The current user settings will be used for exporting.

The batch transfer works on all files found by the tool in the user defined root directory. Being limited in amount of available memory it is only possible to hold a limited number of dataset. The application will sequentially read all files but after each file pair of match and position datas. The tool will run the analysis. Exports that rely on the explicit frames such as the position field or the speed of individual players within a half time will immediately be exported. Then the position frames are unreferenced in order to be freed by the java garbage collection. As a result of this only information containing in the `MeanData` object will be available until the end of the analysis and thus only those information can be used to make an analysis across multiple matches. The memory used to hold the analyzed data is sufficient to provide all data for the final analyze run that merges multiple (or all) into one final result..

3 Final Program

3.1 How to use

On start of the program the user is prompted with a user interface that consists of two main parts (figure 2). On the left side a view is present that will show the analyzed information of the data on the right side is a control panel that its main functionality is to provide a way of changing the settings. The left side is

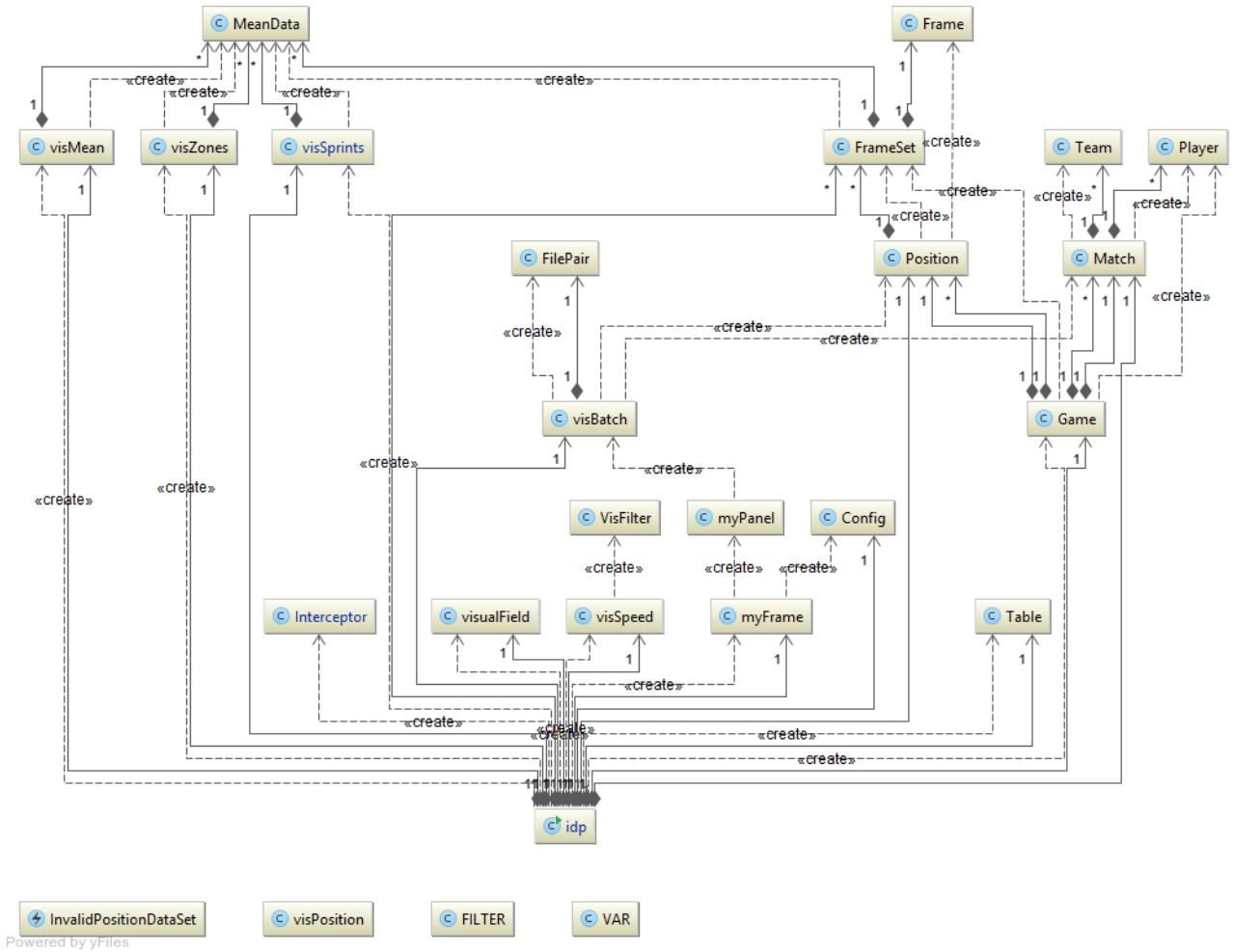


Fig. 1: program structure.

a tabbed layout where every view can show its information and is independent of other tabs. On start up time these tabs do not yet provide any information as long as no data is loaded.

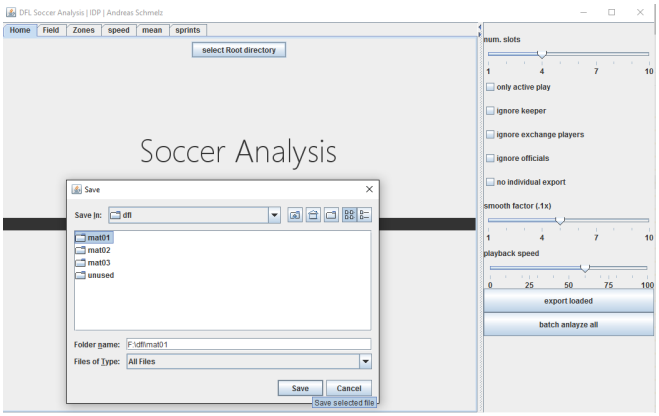


Fig. 2: start screen of the program.

The user has to select a root directory that stores the DFL data sets. After an analysis of this folder the tool reports back which matches could be read. On the tab ("batch") the user can load individual data sets into the program. He has to ensure that the amount of loaded data sets does not exceed the memory available on his machine. One match requires on average 300 megabyte of memory. After a match being loaded the views will reflect this data and can interactively be analyzed by the user.

3.2 Settings

The configuration of the currently filtered data can be adjusted by the control elements of the GUI on the right side.

Slider of time slots This slider defines the number of slots a half time will be divided into. The default value is 3, which means every half time is sliced into chunks of 15 minutes each.

Active play check box activated only the active play will be shown in the analysis. This means the interruptions during the game will be filtered out. We assume that the players during a match will only exert themselves during this time. Some matches do not have the match interrupted flag set. Those matches will not be included in case the active play check box is activated.

Exclude exchange player exchange player will not show the same exhaustion pattern as players starting. Players that do not start from the first minute will not be taken into account in case this check box is activated.

Exclude keeper The keeper will most likely not exhaust during the match. For this case the keeper role can be excluded.

Exclude officials Some data sets have also position data of officials. This check box will exclude them if present.

slider of smoothing factor This slider adjusts the factor, that will be used in order to smooth the acceleration. It uses the formula stated in equation 2. Note that it will not update the currently loaded datasets, as this this require an re-computation of the entire data sets. It does have impact on freshly loaded datasets, as well as the data export and the visualization of speed acceleration and energy consumption of individual framesets (in `visSpeed`). Ensure to adjust this slider if needed before loading datasets.

3.3 Calculated properties

All properties were computed on the aggregated data on a per match basis. It is also possible to merge the results of multiple matches in the export functionality. A football halftime is usually 45 minutes + overtime, as this makes it harder to compare and merge data from multiple matches we only focus on the first 45 minutes of a match each. Additionally we split this time in equally length time slices so we can eliminate temporary fluctuations. We call these time frames slots. The user can during runtime of the program adjust the amount of slots in real time.

The calculated properties are

Average (mean) velocity

$$mean_v = 1/n * \sum_{i=1}^{n} v_i$$

Average velocity in a defined speed zone How fast, on average all player run during the time slot Average velocity within a speed zone z formula

$$speedzone_z = 1/n * \sum_{i=1}^{n} v_i * \sigma(v, z) \tag{1}$$

with

$$\sigma(v, z) = \begin{cases} 1 & \text{v is in zone z} \\ 0 & \text{else} \end{cases}$$

The average velocity of the players during a time slot that they spend in a specific speed zone As speed zones were predefined the intervals $(0, 2, 4, 5.5, 7)$ m/min

Sprints per minute we count all sprints and divide by the time passed. A sprint is an uninterrupted sequence of one second during which the player is at least 7 m/min fast. If a sprint lasts 2 seconds or more it still is counted as one.

Energy Cost For calculating the energy cost we use the wide spread formula by Pietro E. di Prampero.[1] His formula calculates energy cost in joules per kilogram per meter based on velocity and acceleration. Our initial data set does not provide acceleration data. As acceleration we used the delta of two successive speed measurements. The formula of di Prampero is a polynomial of degree 5. The measured data had some errors, in some samples the velocity did not change linearly but step wise. This lead to a very high, unnatural computed energy consumption. The value exploded at those points. In order to minimize this error we smoothed the derived acceleration value. Successive values are blended with their neighbors. This smoothing can be adjusted by an factor, that describes of how much influence the previous acceleration has.

$$A_i = (1 - p) * A_i * p * A_{i-1} \tag{2}$$

Conclusively this parameter solely has influence on the magnitude of the energy consumption. We recommend this parameter not to take for scientific analysis but rather as an indicator to compare with different data sets.

Calculation of these values can be easily implemented by a function in the program, but we are not only interested in the calculation of a single data, but rather the mean of many of them. Not only should it be possible to gather the mean of multiple primitives, but also do we want to partition across several parameters. Also should it be possible to filter out certain framesets under dynamic conditions. For this we partition the data into small buckets and combine them on demand. This partitioning could be done for every parameter (dimension), but would increase memory consumption. We decided to split data time-wise into buckets of 1 minute chunks.

For each of these chunks we collect the sum, count and square sum (additional min and max value). This information is sufficient to calculate average and standard deviate, also is it possible to merge multiple buckets. It also has the advantage, that it allows calculate standard deviate with one single pass rather than the naive approach of calculating the average by the formula $E_v = 1/n * \sum_{i=1}^{n} v_i$ and then the standard deviate with $\sigma = 1/n * \sum_{i=1}^{n} (E_v - v_i)$.

Our approach is $E_v = sum/n$ and $\sigma = sq_{sum}/n - E_v * E_v$

with sum and sq_{sum} being calculated in one pass. [2] This results in a close to real-time aggregation of pre-calculated mean data. It yet is unavoidable to do an entire recalculation in case a parameter is changed that can not be assembled from these chunks, e.g. the smoothing factor. most of the visualization can be built from this data. VisualField (fig. 3a) and visualSpeed (fig. 3b) are some exceptions.

[1] Pietro Enrico di Prampero et al: A simple method for assessing the energy cost of running during incremental tests

[2] https://www.strchr.com/standard_deviation_in_one_pass

3.4 Presented data

From these calculated properties some charts and visual representations are created.

The visField (fig. 3a) displays the position of the active players on a football field. It is animated and should give a broad understanding over what kind of the underlying position data is used. The playback speed can be adjusted. Every object has a short fading line behind itself. The length of this line is an indication of the speed.

The visSpeed (fig. 3b) visualizes the speed, acceleration and energy consumption of an object within a halftime. They are all displayed in one page as they are directly derived only from raw measured speed data. It should be used to reason about measurement errors in the original dataset. For this purpose the graph is zoom-able in the center to get a more detailed view. The first two lines both show the speed. The color schema shows in which speed zone the object currently is. The the third and fourth the acceleration gray is acceleration below 2ms2, green acceleration between 2 and 4 ms2, red anything above. The final two charts are the energy consumption according to di Prampero. We avoided adding labels here, as the magnitude is too dependent on the smoothing factor.

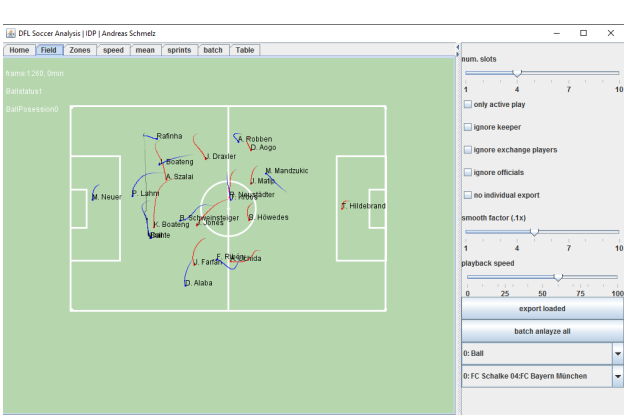
The visSprints (fig. 3c) Here are shown the amount of sprints within a time slot. The graph shows the number of sprints per minute and the count of samples this data point bases on. Toggling any of the check-boxes results in an immediate result here.

The visMean (fig. 3d) This shows the average (mean) velocity of all players taken into account with the current configuration. It shows the average velocity in [m/min] and the number of samples.

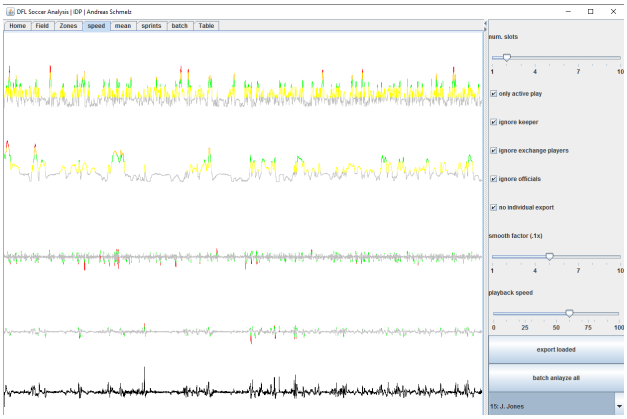
The visZones (fig. 3e) This visualization shows the average distance (or: relative distance) of the players within a specific speed zone. The shown value is in m/min and also the sample count is shown. For calculation the equation from 1 is used.

3.5 Export

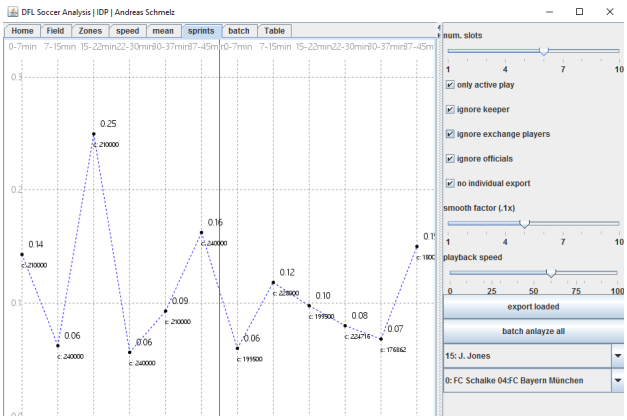
The export functionality does export all the data as shown in the visualizations. Additionally some more visualizations are taken. A very important feature of the export is that the numbers of the analysis are exported into a reusable csv file. This allows the user to make his own computation on prepared data. Also is the data of all the MeanData objects merged and exported as one entity. csv Files are created for every match and show all properties calculated for every participant for every time slot. So even this export is already a high reduction of the initial data it can still be too much for a human to read and interpret. Those files are intended for further processing and interpretation.



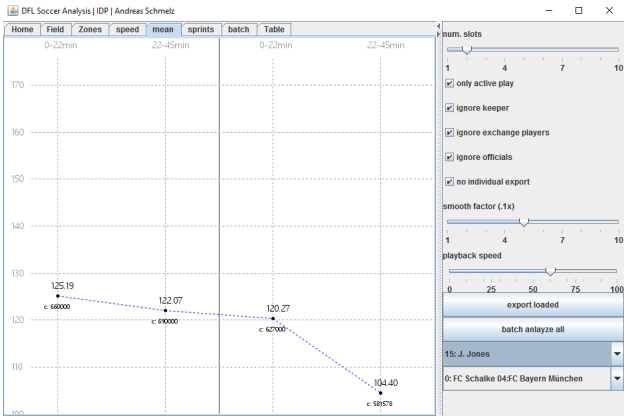
(a) visual Field



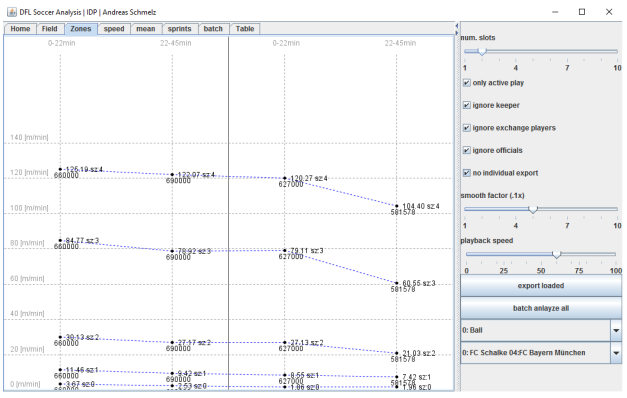
(b) single player speed



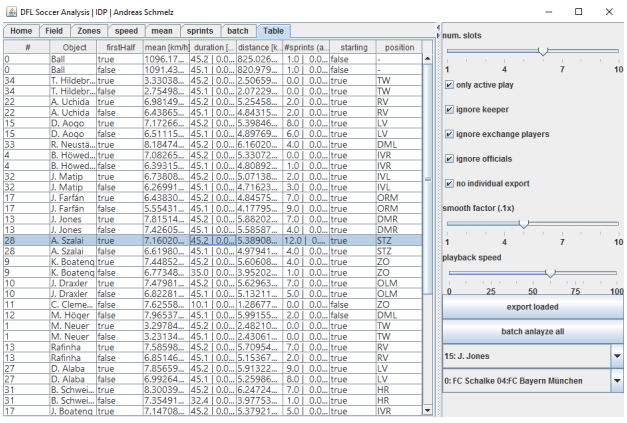
(c) sprints



(d) average speed (one match)



(e) speed zones



(f) table

Fig. 3: visualizations in the program

4 Evaluation

Due to the nature of Java being a managed language, the program cannot allocate and free memory itself. So the program had to be programmed in a way that the program cannot assume memory will be available. Rather it had to use the least amount memory necessary and unset references to any objects that no longer are crucially required. This approach profits of the java internal garbage collection mechanism. An unmanaged language like C would have an advantage here as we could reason about the available memory and explicitly free memory iff necessary. It though would require more programming effort and still be bound to the maximum available memory.

Currently the program uses only one thread. The computation are comparatively low and most of the waiting time in the program is spent waiting for the hard disk. Even large iterations over an entire data set are fast enough that multithreadig was not prioritized higher. Yet it should be introduced as the main tasks in the program do not directly interfere with each other. Especially a decoupling of the user interface-thread from input/ output activity should be favored. The lack of feedback during the reading and exporting phase is inconvenient to the user. It is not user friendly, that for long jobs like the export no progress is communicated to the user.

The good parts of the program are, that it provides a clean architecture that easily can be extended by new visualizations or mathematical equations. The application allows to define new analyzed variables simply by defining a java method. The framework provides a generic interface, that will care about iterating over the data set and applying the existing filter as necessary. This approach is yet only that easy if the formula can directly be derived from a single sample point. An analysis over multiple data samples is still possible but requires more effort.

The approach of merging multiple frames into time slots allows a almost real-time adoption to user input. On the other hand this is only possible for time slices along the time axis and for certain attributes that span across an entire frame set. Calculating subsets for those slices require an re-computation of the entire data set. Creating those slices could also be done for multiple dimensions similar to a data cube. But this approach does require a exponentially growing amount of memory.

4.1 Alternative viable implementation approach

As an alternative to the presented application there are several other possibilities. One that was considered during implementations was, to load the entire data set into a relational data set and implement a UI that consumes the data through the SQL interface. This approach has the benefit, that once the import is done, a guaranteed sanitized data set is provided. Most of the analyzed formula could be expressed by SQL-queries. So very clean abstraction of data layer (database), application logic (SQL query) and graphical user interface (web-or application GUI) can be provided. The downside of this approach is, that the original setup

is non- trivial, especially how the data import to the db could look alike, as the data is not perfectly clean. Also is it less portable, as the application requires a running database with potential data in it. The assumption to the requirements of this project were that the project was used sparsely and then probably for newly recorded matches. So the data set is not final and must be extendable by the user through the program. A program that loads the data once is sufficient and the additional re-storing would be unnecessary effort.

5 Conclusion

The software implemented during this IDP fulfills the requirements as described in 1.1. We were able to plan and implement a scientific relevant data analysis and visualization tool within a limited time frame. This application can be used by scientists to get a better understanding of the collected data by DFL, create, reason and validate about predicates of dependencies of parameters in sports science. The exported analyzed csv files allow for statistically analysis and can support proving statements on the basis of a very large data basis.

The data analysis relies on the quality of the input data. We must assume data is collected consistently across multiple games and seasons. A systematic error in the input data will result in the output data and increase with the degree of the applied function.

The program we delivered gives insight to a broad spectrum of variables. It is possible to observe the single positions and speed of an individual player in the visField or visSpeed, but also to create the average parameter of the hundred given matches. We so can provide statistics on a data set of up to 300 million unique data points.

With this large amount of data it still remains challenging to set up the right hypothesis in order to prove it statistically. Many dependencies and situations can not yet be queried from the program. E.g. "do players run faster in case their team is ahead or behind in the game?" or "How much does the ball possession influence the players performance?". The DFL data provides sufficient data to create those statistics, but would require additional programming effort.

We provide a generic application that can be easily altered and extended. It provides a broad spectrum of analyzed properties and exposes them in a raw, reusable data format together with visualizations.