Universiteit Leiden
The Netherlands

Opleiding Informatica

Analysis and visualisation

of spatio-temporal hockey data

Kevin Noordover

Supervisors:
Joost Kok & Arie Willem de Leeuw

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

22/01/2018

Abstract

In this thesis we discuss several aspects of spatio-temporal hockey data. We start with a brief introduction about Data Science in sports and we describe the available datasets. In particular, we focus on the information the datasets contain and the quality of the data. Also several heatmaps are made to visualize the data and we find clear patterns in the heatmap of the location of the ball. However, this is not the case for the positions where ball possession switches occur. Finally, we calculate the compactness of the teams in ball possession and without possession of the ball. We find that the compacness of a team increases marginally when a team is in ball possession.

Contents

Chapter 1

Introduction

Data science has an important part in computer science. Big data is a growing concern for many occasions because more and more data becomes available, which needs computing power to be processed. More sophisticated methods and algorithms are needed as well. The increase of available data also comes with a benefit: it becomes easier to analyse simple patterns like a hockey match.

The popularity of sports has led to the development of a new reasearch area that applies Data Science to sports. This research area is called Sports analytics and in recent years a lot of progress has been made in the analysis of team-based sports [RM16]. An analysis can be done for parameters like avarage running distances and speed [EAC+07], the centroid of a team [HKWJ14] and possession of the ball [JJM17].

Up to now hockey has not been investigated in much detail. In this research we will use position data to perform a tactical analysis of four hockey matches during the World Championship in 2014. The main research question is: What can we learn from data analysis of a hockey match given the coordinates of the players and the ball?

To answer this question, we start by giving some information about hockey and describing the position data in Chapter 2. Hereafter, in Chapter 3 we discuss the problems that come with the data and give a description of how we handled these problems. In Chapter 4 an analysis of the data is worked out. Finally, we discuss the results of this analysis, what we can learn from it and how this helps in feature analysis in Chapter 5.
This thesis is a bachelor thesis written for the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University. Supervisors for this thesis are Joost Kok and Arie-Willem de Leeuw.

Chapter 2

Definitions

Tactical analysis is performed in almost every sport. There are however big differences between each sport and how the data is obtained and stored. This means that most conclusions drawn in this thesis may be exclusive to hockey or the dataset that is used for the analysis, altough similar sports like football may see similar patrons.

2.1 Tactics and data analysis

In every sport, bringing a good tactic is a key aspect that can make or break a game. Even a bad prepared all-star team can be defeated by well prepared non-professionals. Chess even requires nothing but a good tactic. No wonder that much research is done in this area. Most research focusses on data analysis, since all trainers and coaches can use this to identify whether a certain tactic works. Although this gives a lot of valuable information, it is not (yet) possible to simulate a game. A short introduction to data analysis is given in this section.

Data analysis, as the name suggests, is a process of inspecting, modifying and modeling large amounts of data to discover useful informations like recurring patrons and conditions that lead to a desired result. In the case of team sports this means identifying the right tactic that leads to a dangerous attack or succesfull defense and the opposite: tactics that do not work. On a small scale this is nothing more than a process of trial and error, but at a large scale like international football trial and error is not enough, more data is needed and more has to be analysed.

2.2 Hockey

Hockey is one of the most popular team sports in the world, mostly played in Great Brittain and its former colonies, The Netherlands, Belgium and France. The game consists of two teams with eleven players each and

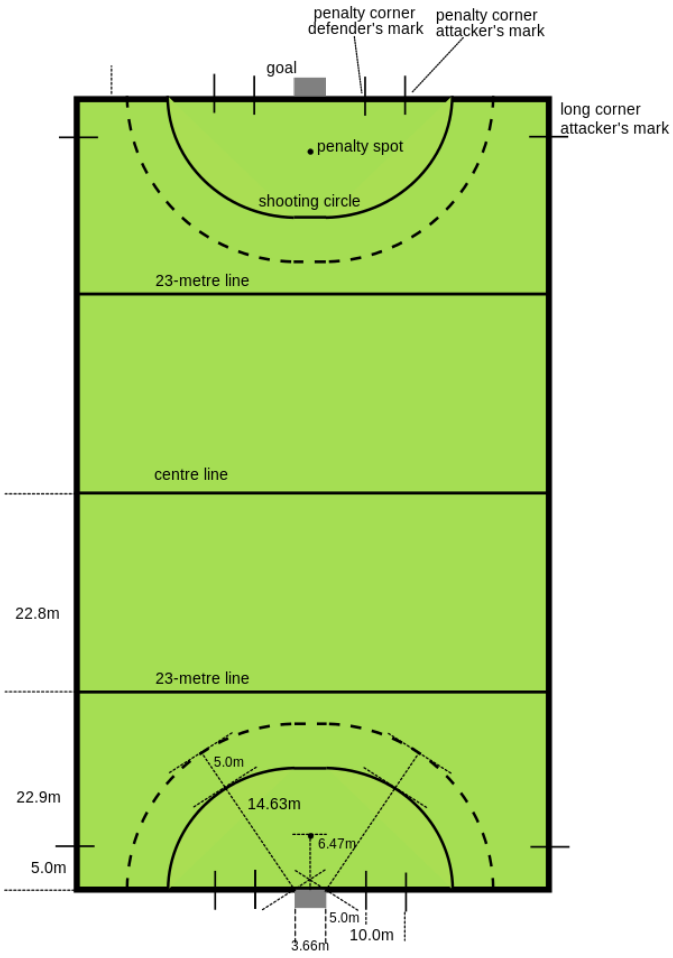

Figure 2.1: A diagram of a hockey field with all dimensions [Mer12]

the objective is to maneuver the ball into the opponent's goal using nothing but a hockey stick. Hockey can be played on both (articficial) grass fields and on ice, but this thesis only covers field hockey.

The hockey field (shown in figure 2.1) is rectangular, 91,4 meters long and 55,0 meters wide [Fed17]. The field contains two 23-meter lines that cross the field 23 meters from each backline, two shooting circles with a radius of 15 meter around each goal, a penalty spot within the circle and the center line with a spot in the middle where each game continues after each goal or after the start of a period. The goals are 3,66 meters wide and are placed at each end of the field.

When a defender makes an foul in the circle that does not prevent the attacking team from scoring a goal, a penalty corner is awarded. Five defenders must take place behind the back line and the other five players of the defending team must start behind the center line. The attacking team take place outside the circle, except for one who brings the ball into play from the penalty corner spot at the back line. Once the ball is played, the attacking team may not touch the ball inside the circle before it is touched outside and a shot on goal may not exceed the height of the backboard in the goal (460mm). A defending foul that does prevent a likely goal leads to a penalty from the penalty spot.

2.3 The dataset

The dataset is provided by Incatec, who have used their video recorded footage to create a dataset containing the coordinates of each player and the ball. Due to shortcomings the ball could not easily be determined by analysing just the footage. This meant that each frame consists of 40 potential coordinates for the ball. A big number of those 40 could easily be rejected just by looking at the score of each coordinate. The score is the likelihood of a certain ball to be the real ball and ranges anywhere from 0 to over 100.

The data of the players was given in an array of normal coordinates relative to the center of the field. The lines around the fields form the boundaries of the coordinates, what means that a player walks between the sidelines as long as the x and y coordinate were between -1 and 1. The length of this player array was not fixed like the ball array. The number of distinct coordinates - players may share one coordinate - is the number of entries in the player array.

If a player disappears at a frame, all players that came behind it in the array move up one place and the id of the disappeared player is added to an array containing the removed players of a certain frame. A special case is when a coordinate with a weight greater than 1 loses a player (two players shared this coordinate, but one leaves). In this case the removed players array is not filled, since the coordinate keeps its location in the array. The team of each player is just used to group players together, it does not stand for a certain team.

Each frame as seen in figure 2.2 has nine attributes: *PTZView*, *ballLineValues*, *ballLines*, *balls*, *frameNumber*, *mainBall*, *players*, *playersRemovedIndices* and *timeStamp*. The analysis is done using only the balls, the players and the removed player indices. The framenumber is used to determine the speed between two frames, since the number of skipped frames may vary. On first sight *mainBall* looked like an usable attribute, but in almost all frames it pointed to a coordinate that could impossibly be the ball.

```
{'PTZView': [0.19756992161273956, -0.2212752103805542, 7.049657344818115],
 'ballLineValues': '\x00\x00\x00\x00\x00',
 'ballLines': [[32.2504997253418, -12.279402732849121], [0.0, 0.0], [0.0, 0.0],
               [0.0, 0.0], [4.616629123687744, -5.379396438598633]],
 'balls': [[36.72950744628906, 19.298006057739258, 1.0, 44.0],
           [1.259029746055603, -25.322696685791016, 1.0, 43.0],
           [4.616629123687744, -5.379396438598633, 2.0, 45.0],
           [34.02127456665039, 17.68569564819336, 2.0, 37.0],
           [32.2504997253418, -12.279402732849121, 1.0, 176.0],
           [29.46883773803711, -1.216287612915039, 1.0, 42.0]]
 'frameNumber': 31259,
 'mainBall': 0,
 'players': [{'normPosition': [-0.7827243804931641, -0.05624507740139961],
              'team': 1,
              'weight': 1},
             {'normPosition': [0.9923557043075562, -0.06732448190450668],
              'team': 2,
              'weight': 1}]
 'playersRemovedIndices': [],
 'timeStamp': 531.373162257}
```

Figure 2.2: A slice of the dataset.

Chapter 3

Verification

One of the biggest tasks was to verify the data and make it useable for futher analysis. The right ball had to be determined and not all frames contained the right number of players. The process of verifying and modifying the data for further analysis is described in this chapter.

3.1 Determing the ball

As described earlier, the dataset contains a fixed fourty balls per frame. Each ball had its own coordinates relative to the center of the field. A negative x-coordinate meant that the ball was located on the left half of the field, a positive x-coordinate meant that it is located on the right half of the field, relative to the camera location. Apart from the coordinates, each ball also has two flags: radius and value. Radius is unused in this thesis, since we can assume that the ball has a fixed size during the match. Value is a validity score (higher is better). If a ball could not be set, random values from memory were taken along a low validity score. Since this number is just a probability, blindly using the ball with the highest score would not be enough.

A first step of the verification process was checking whether each ball followed a trustworthy path. The parse program was expanded with a function to export the path of one certain ball during the whole match to a csv file, so it could be plotted in a graph. The first noticable fact was that each ball jumped to the same random location far from the other coordinates at many occasions. By adding the field borders to the graph, it became clear that that coordinate meant that the ball was not used at that frame and that it could be rejected. The result of removing the bad coordinates is visible in the first two plots in figure 3.1.

The quality of the path was increased, but still not optimal. The ball still makes strange jumps from to fixed points and sometimes even travels 200km/h between two frames. Since those speeds are by far impossible to reach in a normal hockey match, a speed limit was added as extra filter. If the speed between two frames exceeded 100km/h, the new frame is skipped and the next frame is compared to the previous frame. This will eliminate strange jumps to fixed locations, so ball paths are more trustworthy. Paths with just noise can be

eliminated with this method as well.

The result of this filter is visible in the last plot of figure 3.1. All fake paths are removed and the paths that remain look like possible ballpaths. A similar graph could be plotted for all other balls with enough valid coordinates. All left for determing the mainball for each frame was comparing all balls for each frame and check which ball has a valid coordinate that does not violate the set filters.

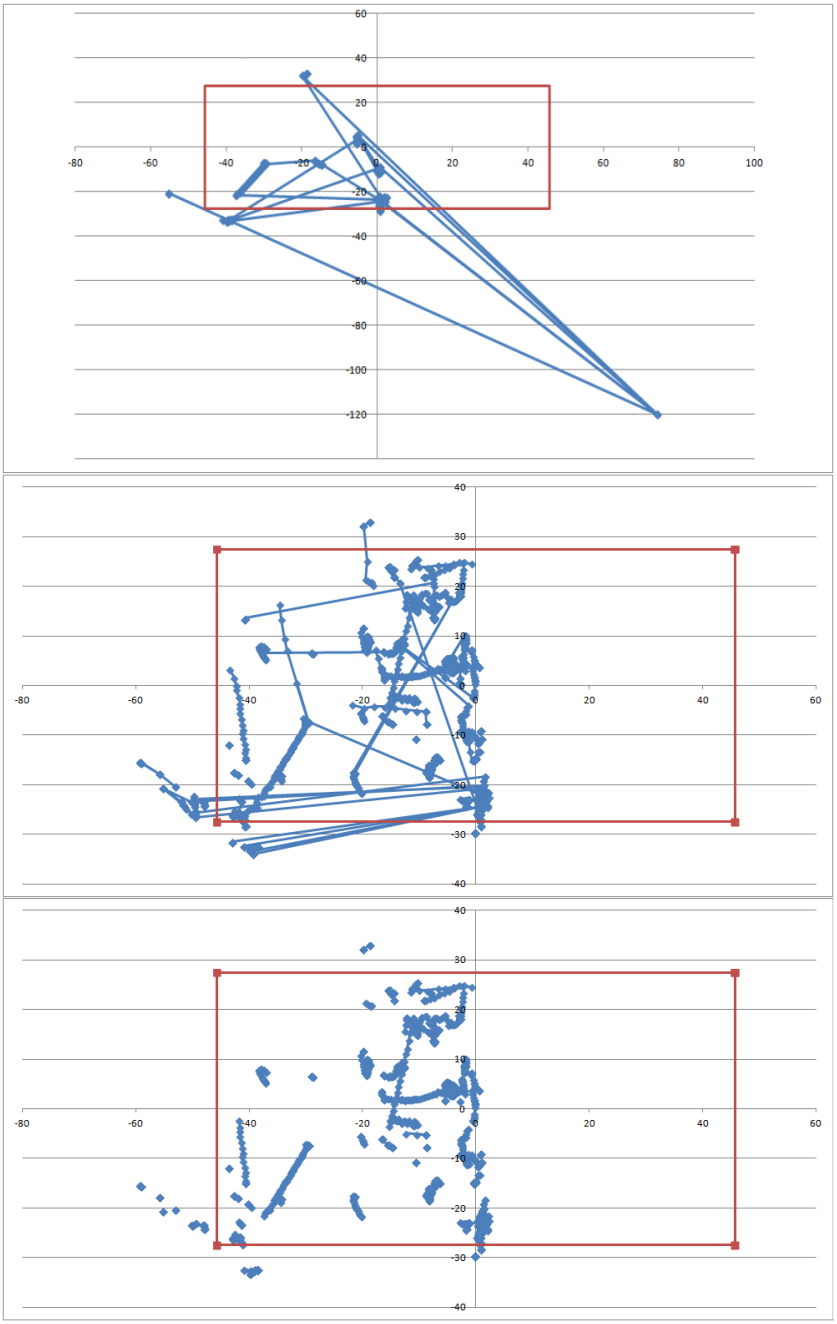

Figure 3.1: Three plots of the ball path. In red the outer lines of the fields, in blue the parsed path of the ball. The numbers on each axis stand for the coordinates. The first plot shows how an unedited path of a single ball looks. No switching takes place. The second plot uses a different ball, and adds a filter for coordinates far outside the fields. The final plot shows the same ball, but with the addition of a speed limit.

The plot of this single ball looked good enough to continue the process of determing the main ball. The filter used above was extended to all the balls in each frame. For the speed verification the last known frame was

used as comparison. On avarage 60% of the frames only had one ball candidate, while the majority of the other frames did not have any candidate at all that met above filters (fig. 3.2). The gaps that were created by this shortcomings were filled by avaraging the last known coordinate before it and the first coordinate after it, creating a straight line at a constant speed between the two.

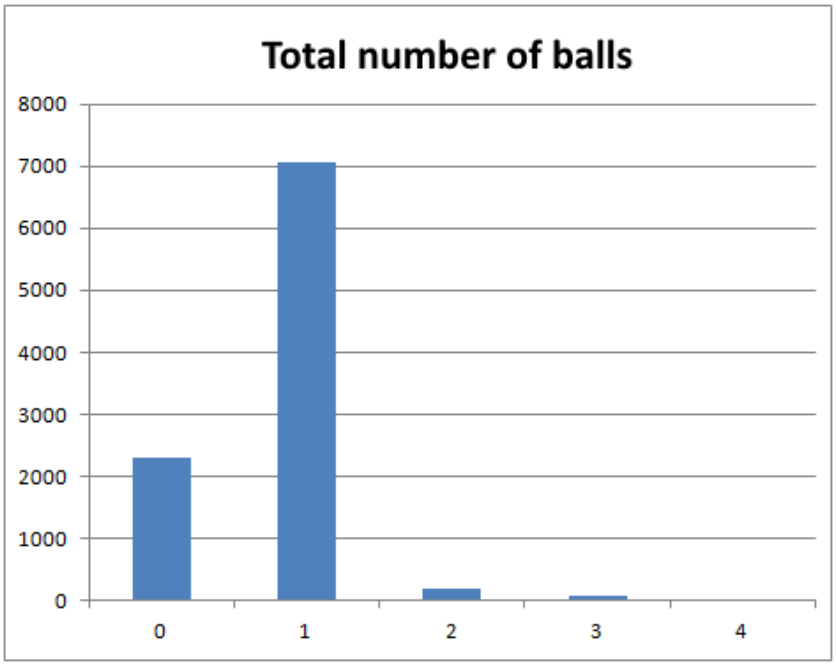

Figure 3.2: This plot shows how many ball candidates were left per frame after the addition of above filters. The high peak for 1 means that most frames only had one candidate. The best result would be one peak for 1, while all other columns are empty.

One thing that was not taken in consideration by using this technique was the occurance of local optima. In many occasions there was an option that the real ball disappeared from the dataset, while a false other ball candidate was nearby. If the real ball would not appear soon enough, the false ball could get through the speed filter and become the main ball. Once the real ball appeared again, the algorithm would not switch to it because the distance to the false bal is too big to cover within one frame.

One of the possible solutions to the problem of finding the ball after a number of frames without any candidate was backtracking a ball candidate for the next frame. If this ball was not given a validity score bigger than zero in the previous frame - where no ball candidate could be found - it can be assumed that this is not the actual ball but a false positive.

One of the possible solutions to this problem was by looking at the new ball itself, apart from the whole enviroment. If this new ball candidate was given a non-zero score between the last frame the real ball was visible and the frame that is being checked, one can assume that this is a fake ball. It already exists before it should exist and should therefore be excluded.

Adding this new check to the algorithm resulted in a clean game where the ball acted as it should. There were still some noticable weird jumps, but those could not be optimised any further because the constraints of the data were reached. The reading algorithm would be unable to decide if a change in direction or speed was

because an invalid ball was used as main ball or if it was normal behaviour.

3.2 The number of players

Just like the ball, the set of players was also not perfect. Many frames missed one or more players. Hockey is played with 22 players that should also all appear in most frames in the dataset. Players may be excluded for a certain time by the referee, which means that the team has to continue with less than 11 players. Surprisingly, not 22, but 19 was the most used number of players for a certain frame, as cen be seen in Figure 3.3. This was a big problem for further analysis, since the players are an important factor. Multiple missing players at a certain frame could drastically alter the outcome. Unlike the ball array however, there was no alternative available. If a player disappeared, there were no other players that could fill the gap. Altough not perfect in the case of more than one missing player, this would make filling in the gaps easier.

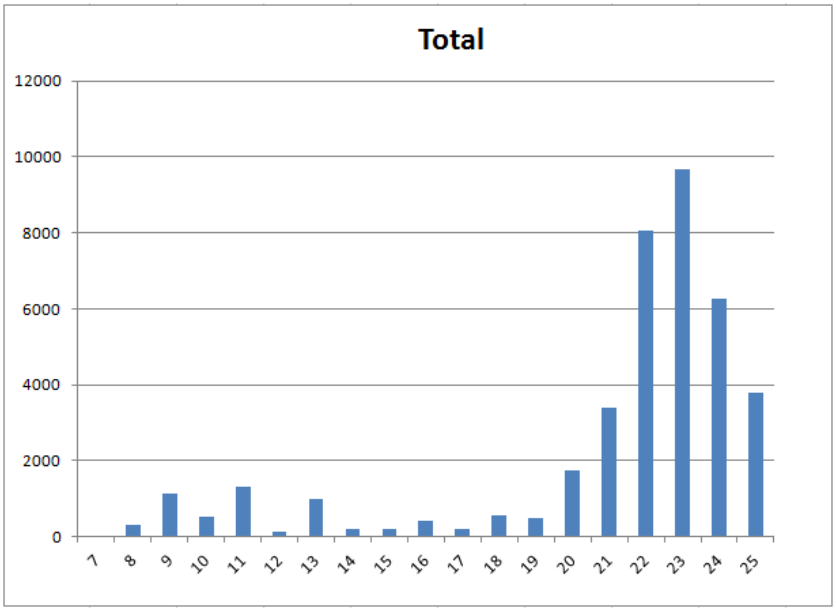

Figure 3.3: This plot shows how many players were on the field per frame. Just like the previous figure, the x-axis stands for the number of players for a frame and the y-axis stands for the number of frames with that quantity.

The players were stored in the dataset in two parts: an array with all players and an array that contained the ids of players that disappeared at a given frame. This means that a player at the 15th location in the array at frame x would win a place in this array if the $PlayersRemovedIndices$ array at frame $x+1$ contained an id lower than 15. Each player in the player array has a normal position relative to the field with the center of the field being zero, and a weight factor. This weight stands for the probable numer of players that shared a certain spot. So there is an option that, although the player array only has length 20, it was still possible to follow all players.

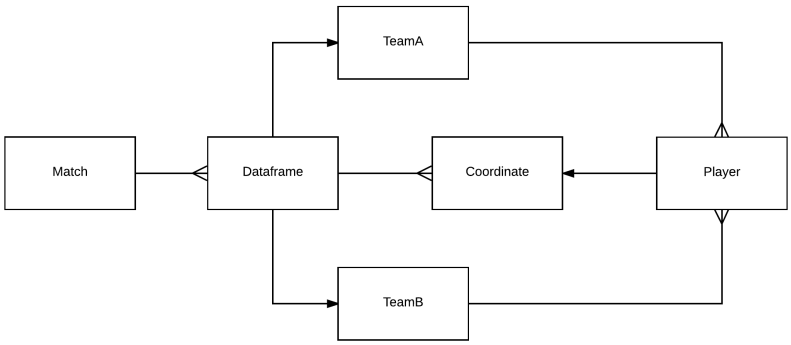

Figure 3.4: The datamodel of the match

3.3 Teams

Because most tactical analysis done in the past relies on knowing to which team each player belongs, team information would be a great addition to this dataset as well. Since the team-id that comes with the dataset is not the same for each frame (team A could have 1 as id at one frame and 3 as id at the next), a comparision had to be made with the grouping of the previous frame. The problem with this approach is that the same player can have different ids as well, what makes comparing two frames more difficult.

Tracking each player is done as follows: each frame gets assigned two teams with eleven players each. Each player has its own id, so tracking one player can be done by just looking at its id. Each frame, a player gets assigned another id as well, this one is the id of the corresponding coordinate in the dataset for that certain player. When parsing the dataset, the parser looks at the *PlayersRemovedIndices* when determing the new coordinate-id for a certain player. For each id in *PlayersRemovedIndices* that's below the coordinate-id that the player had in the previous frame, one is subtracted from this id. For example, when player A has the sixth coordinate at frame 200 and the *PlayersRemovedIndices* contains the number 5 at frame 201, player A will get coordinate-id 5 at this frame. Figure 3.4 shows a datamodel of how each Player is related to a frame.

Now each player has a fixed id, comparing the frames is much easier. First the two team-ids with the highest number of players are chosen, the third team may contain the referees and coaches. Next, for both team A and team B, a score of likelihood is generated for both remaining team-ids. Each player that is present in both the previous frame and the team that is compared to, adds one to the score. According to the score, the team-ids of team A and team B are filled in.

3.4 Ball possession

An important thing to know is which team is in ball possession at a given frame. This information can be used for many subjects like a heatmap of where the possession changes or for determing the compactness for teams with and without the ball. This information is not present in the dataset and thus had to be determined on the basis of the ball and player coordinates.

For most frames, simply chosing the player that is the closest to the ball and within 2 meters is enough. This may give false positives for frames where the player in ball possession is walking or running near an opponent, but in the majority of the frames this will not be the case. The algorithm does not do this when a ball is being passed between two players of one team or other occasions where the ball travels in a straight line. Instead, it uses the information of the previous frame. The other team after all is not in ball possession either, even if the ball is closer to one of its players.

Chapter 4

Tactical analysis

4.1 Heatmap

Heatmaps are a great tool to visualise a match. Not only do heatmaps help to make recurring patrons visible, it can also even out eventual errors that exist in the dataset since those errors will either happen only once, or happen so often that they can be identified and excluded for further research. Another advantage is that the possibilities are endless. It can be generated for the ball, for the players and even for an individual player. It's possible to create one for the exact coordinates and for a bigger portion of the field, the circle for example.

So heatmaps can be useful for data analysis in many ways. The most common kind of heatmap is the one used for density visualization. This is also the kind used for this analysis. For simplicity, the colors used are black, white and everything between those colors. Using shading instead of common used rainbows makes it less likely that the perception of the data is less prone to errors.

The first heatmap discussed is the one generated by the ball, visible in figure 4.1. The assumption is that all attacks carried out by a team create a clear heatmap on the half of the opponent. Stronger teams may use a certain tactic to get to the opponent's circle over and over again if a previous attempt was succesfull. Weaker teams on the other hand will have more failed attempts and thus a less clear heatmap. A clear structure is missing because this weaker team does not get many chances to bring their tactic into action.

That this is indeed the case follows from the heatmap generated from the first dataset. It is clear that the team playing from left to right is stronger than the opposing team. What it makes clear that this is not the opposing team playing the ball around at their own half of the pitch is the great density at both sides without much in the center. Teams passing the ball around at their own half will not do this exclusively vertically (left-right in the heatmap), but will also pass the ball horizontally. Attackers on the other hand will exclusively play vertically becasue the aim is to get the ball to the other team's circle as fast as possible while reducing the risk that the ball is lost.

The heatmap shows that the offending team will start each attack from either the left (top) or right (bottom)

side of the field and will move to the center once the circle is reached. The offending team enters this circle not trough the middle, from where scoring is easier, but from the harder sides. Also noticable is the position of all lines of the field because those lines were seen as the ball, while the ball itself was located somewhere else or not even on the field.

A similar pattern is visible in the heatmap of the second dataset. It is clear that the team playing towards the right goal is the stronger team, and just like the first heatmap, they also play from the outside towards the inside. Interesting is that the left half of the field has almost the exact opposite heatmap compared to the other half. When attacking, the stronger team plays the ball towards the outside of the field, but on their own half they keep the ball in the center. If the other team plays with the same tactic, it may be invisible in the heatmap, because their possession is low.

Both heatmaps also uncover a bug in the determination process of the ball. The center line, circles and even the penalty spot are visible. This means that once the algorithm used to determine the ball may sometimes switch to a line mistaken for a ball.

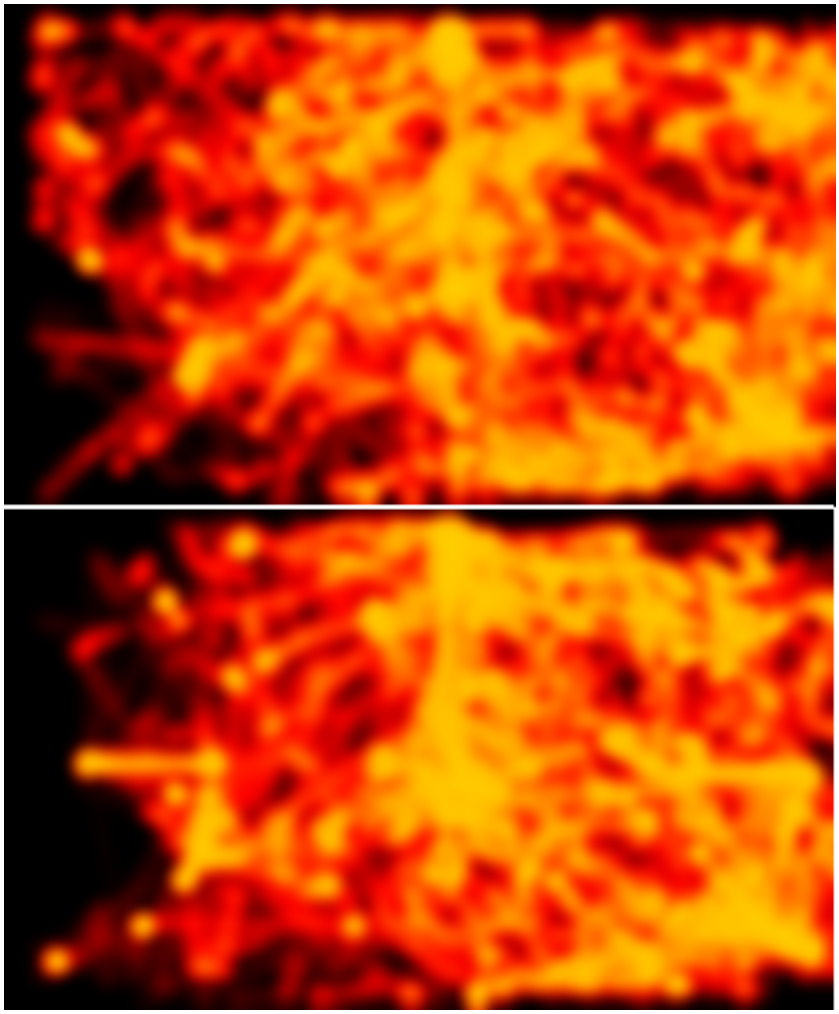

Figure 4.1: The heatmaps of the location of the ball in the two matches. Black means that the ball has never been in that area, yellow means that the ball has been there a lot. Red is the halfway point between the two other colors.

4.2 Takeover heatmap

Heatmaps are not only a great tool to plot the location of the ball, but can also be used to plot the locations where the possesion of the ball changes between the two teams. This heatmap will globally show at which locations the defending team stops the offending team. A correlation to above heatmaps is to be expected, since turnovers are more likely to occur at locations where the ball gets very often than at locations where this isn't the case.

This heatmap will therefore show similarities with the heatmaps above. However, it can still be used for further analysis. It is very unlikely the turnovers will occur evenly spread at the locations where the ball comes, so differences between above heatmaps and the heatmaps of the turnovers will provide the information that is needed. If the two heatmaps do end up almost the same, we can conclude that there are no specific locations where a takeover takes place.

The latter turned out to be the case for the first match. The generated heatmap looks the same as the general ball heatmap, which means that there is no correlation between a certain location and a change in posession. Losing the ball often means the team made a mistake of some sort, which allows the defending team to take the ball. One can assume that, because the dataset was recorded off a match on the highest level in hockey, teams won't make the same mistake over and over again.

As can be seen in figure 4.2, the second match also does not show a clear pattern. The heatmap resambles the location of the ball and no additional hot spots are present. This feeds the assumption that the level of skill is the most important factor in the generation of this heatmap. A game in a local competition may give different results.

4.3 Compactness

The past two techniques both analysed the location of the ball. In the next step, we look at the team currently in possession and the location of the players to analyse whether or not there exists a correlation between the two parameters. For this analysis the compactness of each team is calculated and plotted in a box plot, a graph that is used to plot groups of numerical data through their quartiles. The numerical data is devided between four groups: both teams and whether or not they are in possession of the ball.

The compactness of each team for each frame is calculated as follows: first we take the avarage position of all players currently on the field. Next, the avarage distance to that point is calculated, excluding the player standing the furthest away, so the goalkeeper is excluded when the team is not defending. The compactness is thus denoted by the avarage distance to the avarage position of the players. Lower numbers mean higher compactness.

A downside of using the compactness is that the result may be influenced by the fact that a field is not a perfect square. If all players are spread over the width of the field, but not over the length, the compactness

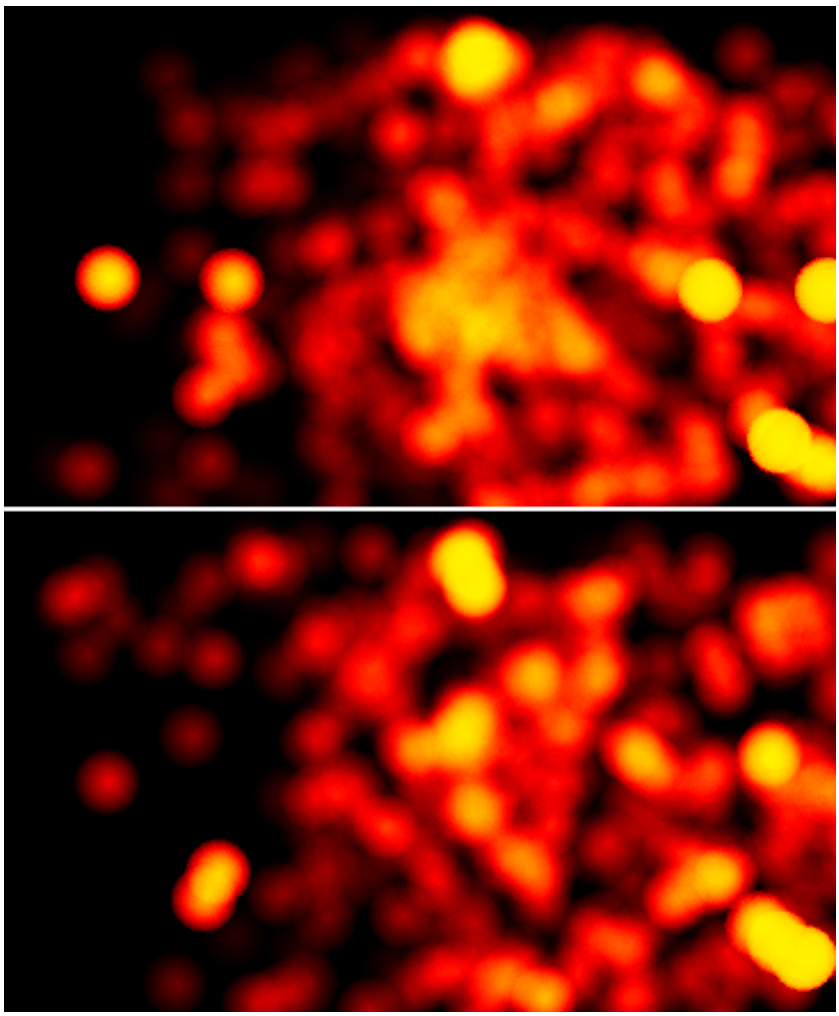

Figure 4.2: The change heatmaps of two matches, to illustrate where the two teams change ball possessions the most often. Just like the previous heatmaps, black means no possession changes at a certain location, yellow means a lot of possession changes.

may still be relatively low because a high spread over the length of the field has more impact on the total compactness. Because a plot of the general compactness does not tell the whole story, two additional plots are made for the compactness on both the X-axis (length) and the Y-axis (width).

In each attachment below, the box plot of team A is on the left, and the one of team B is on the right. The x-axis tells whether or not a plot is created from the frames where the team had the ball, the y-axis is the compactness as described above in meters. Two matches are plotted: a match where team B was the stronger team and where the ball was more often than not on the half of team A, and a match that was more equal.

The plots of the first match (figure 4.3) shows a small correlation between the team and the compactness. The plot of both axis shows that the stronger team B, the right plot, has a higher level of compactness than the other team, both with and without ball possession. A look at the two other graphs tells that the X-compactness is responsible for the difference between the two teams. This is suprising, since one would expect the weaker team to play more compact because that is the team that is defending more often.

In the second match (figure 4.4) it is clear that both team A and team B play more compact if they are in ball

possession. For team A this difference may not be very statistically significant, but it is for team B. The plots for both axes show that both contribute to this relationship. The plots of this match reveal the same result as the previous match: the team is less compact when defending.

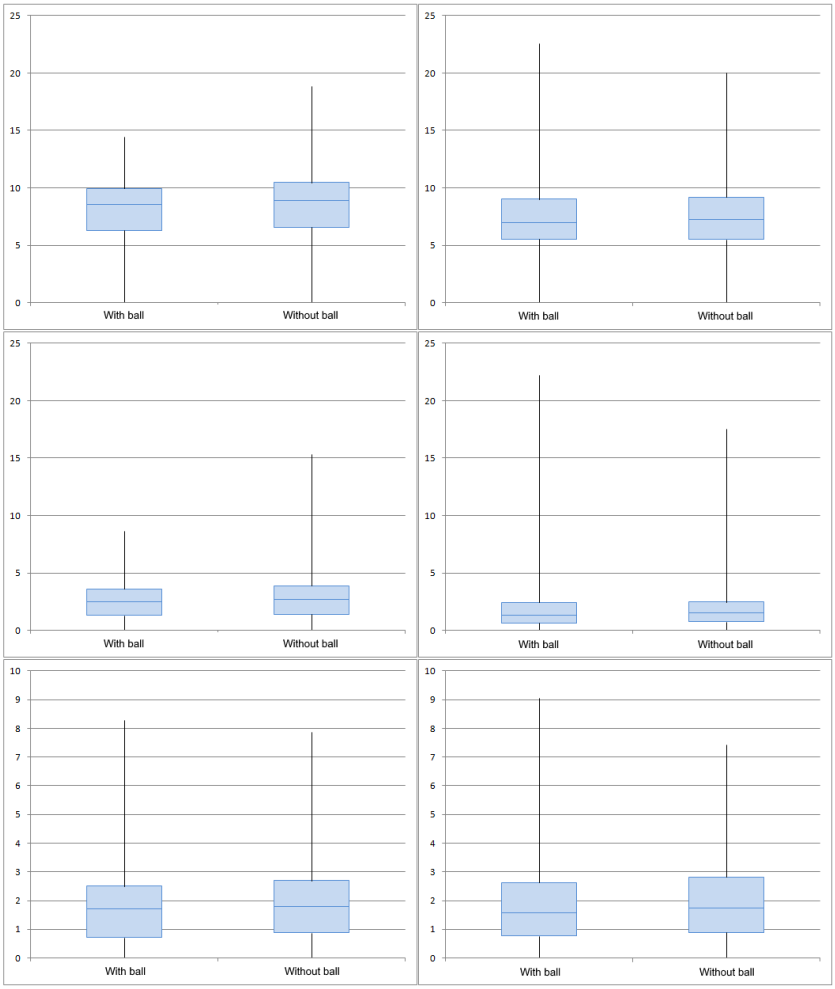

Figure 4.3: A box and whisker plot of the compactness of both teams in ball possession and without possession of the ball. The left column is team A, the right column team B. The first row is the total compactness, the second row the compactness on the X-axis, the third row the compactness on the Y-axis.

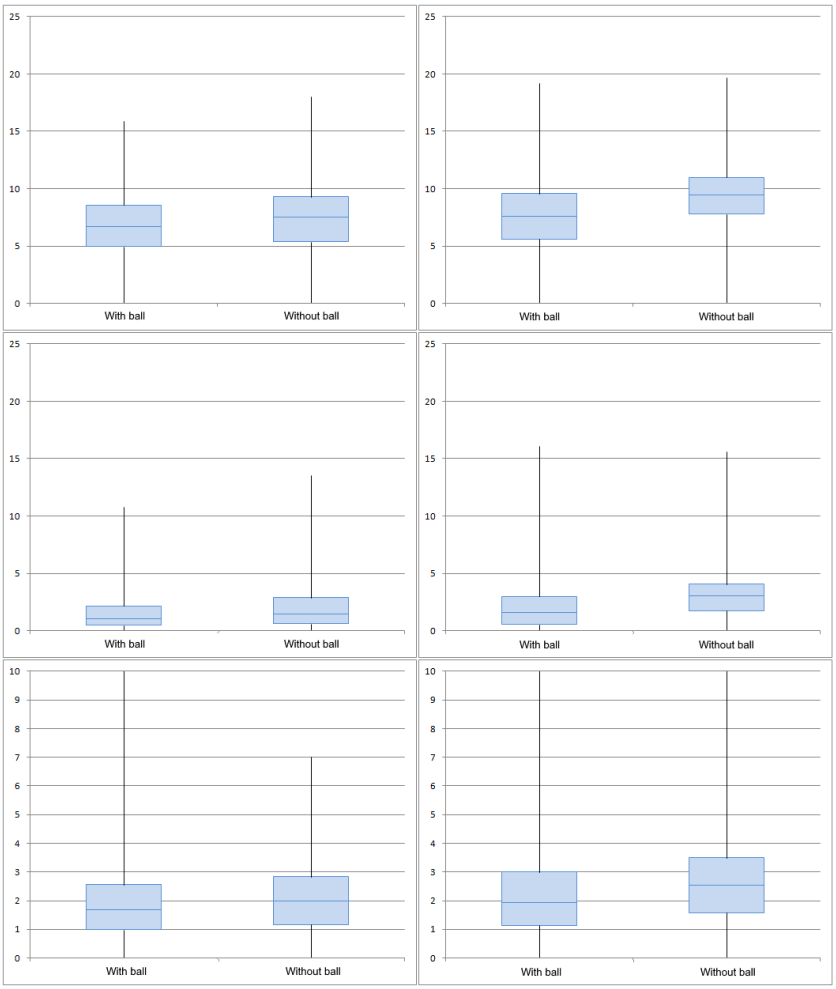

Figure 4.4: The box and whisker plot of the other match. Just like the previous plot, the left column is team A and the right column team B. The rows are total, X-axis and Y-axis compactness.

Chapter 5

Conclusions

The question that had to be answered is 'What can we learn from data analysis of a hockeymatch given the coordinates of the players and the ball?' In this thesis three methods have been dealt with: the heatmap of the ball position, the heatmap of the location where the team in ball possession changes and the connection between teams, ball possession and the player compactness. Although not all methods did provide a clear outcome, the results of this research are very promising. It is possible to learn from data analysis, especially from further research with data of higher quality.

What we learned from the ball location heatmaps is that the offensive team will play the ball according to a fixed pattern. In both heatmaps discussed, the stronger team playing from left to right burns a clear pattern. Playing the ball through the center towards the circle does not happen, instead the ball is played from the sides and curves towards the circle once it is reached. This pattern is very clear for both heatmaps and is also visible in the two excluded matches.

The results from the possession change heatmaps were less usefull. As expected the heatmaps did have much in common with the heatmaps of the ball location, but they did not show any clear pattern apart from it. This could have a few different reasons: either there is simply no pattern or the noise of the data had a too big impact on the ball possession algorithm, what renders it useless. For now, it is not possible to draw conclusions from these heatmaps.

The last part of the research - finding patterns in team compactness - also did not provide clear results. Just like above, noise could have attributed to this. However, there was a pattern that could be investigated further: a team seems more compact when offending than when defending. The difference between possession the ball and not was small, but it was visible for each team. This suprising fact - one might expect that a defending team is more compact - is worth further research as well.

A big drawback was the quality of the data. It was good enough to work with, but it was not perfect. What the heatmaps and a simulator showed was that the determination algorithm of the ball did not work pefectly. It was optimised in every way possible, but could not prevent some frames to have the ball at clearly wrong

locations. This can be attributed to the white lines of the field; on the heatmaps one can easily distinguish the lines of the circles and the penalty dots. This resulted in two matches being excluded from further analysis because those matches were influenced too much by this behaviour.

The number of possible tactics that can be analysed is much higher than the three covered in this thesis. This research showed that analysis is possible, even with a lower quality dataset. The low quality makes it possible to analyse a large number of games for not much money, but it has as disadvantage that some analyses are not possible or may give unclear results. For more reliable results, a dataset of better quality is needed, referably with a clear ball.

Bibliography

[EAC+07] Rampinini E, Coutts AJ, Castagna C, Sassi R, and Impellizzeri FM. Variation in top level soccer match performance. https://www.ncbi.nlm.nih.gov/pubmed/17497575/, 2007.

[Fed17] The International Hockey Federation. Rules of hockey. http://www.fih.ch/media/12236728/fih-rules-of-hockey-2017.pdf, 2017.

[HKWJ14] Folgado H, Lemmink KA, Frencken W, and Sampaio J. Length, width and centroid distance as measures of teams tactical performance in youth football. https://www.ncbi.nlm.nih.gov/pubmed/24444244/, 2014.

[JJM17] P. D. Jones, N. James, and S. D. Mellalieu. Possession as a performance indicator in soccer. http://www.tandfonline.com/doi/abs/10.1080/24748668.2004.11868295, 2017.

[Mer12] Robert Merkel. Field hockey diagram. https://commons.wikimedia.org/wiki/File:Hockey_field_metric.svg, 2012.

[RM16] Robert Rein and Daniel Memmert. Big data and tactical analysis in elite soccer: future challenges and opportunities for sports science. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4996805/, 2016.

Chapter 6

Appendix

This chapter provides code fragments that were used during the research. The language *C#* is used in these fragments.

6.1 Determing the main ball

In the code fragment below, i is used to identify the frame for which the ball has to be determined. The member *BallSize* in each MatchFrame stands for the validity score.

```
for (int ball = 0; ball < 40; ball++)
{
    //Is in field, validity above 5 and not lower than 75% of the highscore
    if (MatchFrames[i].Balls[ball].Y < -100 ||
        MatchFrames[i].BallSize[ball] < 5 ||
        MatchFrames[i].BallSize[ball] * 1.33 < highscore)
        continue;

    //Is under certain speed
    double thisSpeed = 0;
    int compFrame;
    for (compFrame = i; compFrame > 0; compFrame--)
    {
        if (MatchFrames[compFrame - 1].MainBall > -1)
        {
            int compBall = MatchFrames[compFrame - 1].MainBall;
            thisSpeed = Math.Pow(MatchFrames[i].Balls[ball].X -
                MatchFrames[compFrame - 1].Balls[compBall].X, 2) +
```

```
                Math.Pow(MatchFrames[i].Balls[ball].Y -
                MatchFrames[compFrame - 1].Balls[compBall].Y, 2);
            break;
        }
    }
    if (compFrame != 0 &&
        thisSpeed > Math.Pow(maxSpeed * (i - compFrame + 1), 2))
    {
        continue;
    }

    framecounter++;
    theBalls += ball + " ";
    if (MatchFrames[i].MainBall == -1)
    {
        //If past 1 frames were -1, and this ball already existed
        //without moving, it is probably a bad ball
        bool goodBall = false;
        PointF ballCoordinate = MatchFrames[i].Balls[ball];
        for(int j = i - 1; j >= i - 1 && j >= 0; j--)
        {
            if (MatchFrames[j].MainBall != -1 ||
                MatchFrames[j].BallSize[ball] < 5 ||
                MatchFrames[j].Balls[ball].X != MatchFrames[i].Balls[ball].X ||
                MatchFrames[j].Balls[ball].Y != MatchFrames[i].Balls[ball].Y)
            {
                goodBall = true;
            }
        }
        if (!goodBall)
            continue;

        MatchFrames[i].MainBall = ball;
        MatchFrames[i].mainBallSpeed = thisSpeed;
        //Draw line between this frame and last frame with known ball
        for (compFrame = i - 1;
            compFrame > 0 && MatchFrames[compFrame].MainBall < 0;
            compFrame--)
        {
```

```
            if (MatchFrames[compFrame - 1].MainBall > -1 ||
                MatchFrames[compFrame - 1].MainBall == ball)
            {
                int oldBall = MatchFrames[compFrame - 1].MainBall;
                double dX = (MatchFrames[i].Balls[ball].X -
                    MatchFrames[compFrame - 1].Balls[oldBall].X) /
                    (i - compFrame + 1);
                double dY = (MatchFrames[i].Balls[ball].Y -
                    MatchFrames[compFrame - 1].Balls[oldBall].Y) /
                    (i - compFrame + 1);
                double curX = MatchFrames[compFrame - 1].Balls[oldBall].X;
                double curY = MatchFrames[compFrame - 1].Balls[oldBall].Y;

                for (int j = compFrame; j < i; j++)
                {
                    curX += dX;
                    curY += dY;
                    MatchFrames[j].MainBall = ball;
                    MatchFrames[j].Balls[ball] = new PointF(curX, curY);
                    MatchFrames[j].BallSize[ball] = 200;
                    MatchFrames[j].mainBallSpeed = thisSpeed;
                    MatchFrames[j].BallCount = 1;
                    MatchFrames[j].BallCandidates = ball.ToString();
                }
            }
        }
    }
    else
        MatchFrames[i].MainBall = -2; //More than one candidate
}
```

6.2 Calculating the compactness

This function belongs to the *DataFrame* class, which is used in the previous fragment in the *MatchFrames* array. This class contains all information of one frame, including all calculated fields. Information from this class is used to plot each diagram.

```
public double getCompactness(bool teamA)
{
    PointF avg = getAvaragePosition(teamA);
    Team theTeam = (teamA ? this.TeamA : this.TeamB);
    int size = 0;
    double highest = 0;
    double tot = 0;
    foreach (Player player in theTeam.Players)
    {
        if (player.PlayerId >= 0)
        {
            size++;
            //Calculate distance to avarage point
            //Include conversion from normal to real coordinates
            double d = Math.Pow(
                (this.Players[player.PlayerId].X - avg.X) * 22.85 *
                (this.Players[player.PlayerId].X - avg.X) * 22.85 +
                (this.Players[player.PlayerId].Y - avg.Y) * 13.75 *
                (this.Players[player.PlayerId].Y - avg.Y) * 13.75, 0.5);
            if (d < highest)
                tot += d;
            else //Exclude furthest player
            {
                tot += highest;
                highest = d;
            }
        }
    }
    return tot / (size - 1);
}
```

6.3 Dividing the players into teams

This code fragment is taken from the function *fillTeams*(). This function contains additional code for the first frame, which is almost identical to this fragment.

```
for(int mId = 1; mId < MatchFrames.Count; mId++)
{
    DataFrame newFrame = MatchFrames[mId];
    DataFrame prevFrame = MatchFrames[mId - 1];

    //We fill in an array for each team-id (0-2)
    teams = new List<int>[3];
    teams[0] = new List<int>();
    teams[1] = new List<int>();
    teams[2] = new List<int>();

    for (int a = 0; a < newFrame.Players.Length; a++)
    {
        for (int p = 0; p < newFrame.PlayerCount[a]; p++)
            teams[newFrame.PlayerTeam[a]].Add(a);
    }

    //Compare each team-id with both teamA and teamB and give a score
    //Lowest (negative) -> A, highest -> B
    int[] score = new int[3];
    for(int j = 0; j < 3; j++)
    {
        if(teams[j].Count == 0)
        {
            score[j] = 0;
            continue;
        }
        foreach (Player compPlayer in prevFrame.TeamA.Players)
        {
            int removedUnder = 0;
            foreach(int rpId in newFrame.PlayersRemoved)
            {
                if (rpId < compPlayer.PlayerId)
                    removedUnder++;
            }
```

```
            if (teams[j].Contains(compPlayer.PlayerId - removedUnder))
                score[j]--;
        }
        foreach (Player compPlayer in prevFrame.TeamB.Players)
        {
            int removedUnder = 0;
            foreach (int rpId in newFrame.PlayersRemoved)
            {
                if (rpId < compPlayer.PlayerId)
                    removedUnder++;
            }
            if (teams[j].Contains(compPlayer.PlayerId - removedUnder))
                score[j]++;
        }
    }

    //For both best results, assign players from array to team,
    //(keeping the order of previous frame)
    int lowestId = 0, highestId = 0, lowest = score[0], highest = score[0];
    for (int j = 1; j < 3; j++)
    {
        if (score[j] < lowest)
        {
            lowestId = j;
            lowest = score[j];
        }
        if (score[j] > highest)
        {
            highestId = j;
            highest = score[j];
        }
    }
    newFrame.TeamA = new Team();
    newFrame.TeamB = new Team();
    newFrame.TeamA.TeamId = lowestId;
    newFrame.TeamB.TeamId = highestId;
    for(int pID = 0; pID < newFrame.TeamA.Players.Length; pID++)
    {
        if (pID < teams[lowestId].Count)
```

```
        {
            newFrame.TeamA.Players[pID] = new Player(teams[lowestId][pID]);
            newFrame.TeamA.Players[pID].team = false;
        }
        if (pID < teams[highestId].Count)
        {
            newFrame.TeamB.Players[pID] = new Player(teams[highestId][pID]);
            newFrame.TeamB.Players[pID].team = true;
        }
    }

    MatchFrames[mId] = newFrame;
}
```