ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Investigating a Trajectory Smoothing Technique for Performance Analysis in Football

Bachelor Thesis

Adam Suma

20th of August, 2024

Advisors: Prof. Dr. Ulrik Brandes, Yannick Herkommer
Department of Computer Science, ETH Zürich

Abstract

Tracking data in football has gained a lot of importance in the last few years, as it plays a pivotal role in performance analysis and tactical planning. Gathering such data can be done using different systems, which vary a lot in accuracy. For this purpose, smoothing filters can be applied to reduce differences across systems and provide analysts with more truthful data. In this thesis, we will study the impact of a novel geometric smoothing technique that uses circle splines to filter trajectories. We will provide an in-depth explanation of how the method works and a thorough analysis of its impact on tracking data and robustness to noise. As part of our investigation, we assess the differences between two different data sources and present how the smoothing affects them. Our results show, that the novel method manages to bring the physical metrics extracted from the data closer together and that it yields better speed approximations.

Contents

Chapter 1

Introduction

1.1 Motivation

A growing number of methods for soccer analytics rely on the availability of tracking data, in which the location of all players and the ball is recorded at high temporal resolution (multiple frames per second). Such data is obtained in a variety of ways, including GPS devices, fixed-installation camera systems, and broadcast video, and varies in accuracy. Accuracy is particularly relevant for analyses that involve speed, acceleration, and coordination of players. Momentous speed, for instance, may be approximated from subsequences of locations, often followed by a smoothing of the speed time series thus obtained. Accurate speed approximations are of particular relevance, as measuring sustained speeds helps in detecting the number and distance of sprints, which are important metrics for assessing the performance of soccer players during training and matches. Movements which present sudden high variations in acceleration (large values of jerk [3]) are avoided in general by athletes and can be seen as signs of imprecise measurements when looking at tracking data.

1.2 Research Goals

In this thesis, we study a novel geometric smoothing method for tracking data, which limits jerk in player movements and yields consistent results across different data sources by filtering existing trajectories using circle splines.
Firstly, we will look into the different methods of extracting the tracking data and their differences when it comes to commonly used metrics in soccer analytics. Afterwards, we will go through the definition of the technique, its implementation and speed approximations of filtered trajectories. This is followed by the main part of the thesis, where the impact of the smoothing

technique will be shown in a mix of experiments and visualizations that are meant to back its effectiveness and robustness. Finally, we will mention a few recommendations and future work that could help in understanding this technique even better.

1.3 Related Work

Quite a bit of research has already been done for assessing errors and removing noise from different tracking systems. One publication of significant interest to us is the "Correction of systematic errors in electronic performance and tracking systems" [13]. It is interesting to note, their clustering technique for the classification of multidimensional validation outputs. Intuitively, they assume that different tracking systems produce different errors for different movement patterns. These errors are then modeled according to a gold standard ground truth. Afterwards, regression models, created using the modeled errors, are used to map the raw data which results in better comparability between EPTS.
In the world of football, tracking data is seen as a new way to change the game. "Data Analytics in Football"[5], presents the importance of positional data in many areas of performance analysis and how Big Data is influencing decisions made in the sport. In this article, we are also introduced to Key Performance Indicators(KPI's), which are evaluated from positional data and are vital for modern soccer analytics. More about these indicators can be found in this pilot study (Perl, J., & Memmert, D.)[8].
There has been extensive work in smoothing and spline interpolation of signals. The Savitzky-Golay[11] and cubic spline[10] smoothing are some of the most prominent methods for this purpose and could be applied in practice for improving data quality.
With our work, we also aim to bring these fields together and examine how they can be applied to improve football metrics.

Chapter 2

Tracking Data

In the past few years, the emergence of tracking data has allowed for more comprehensive performance analysis, tactical planning and injury prevention in football. This type of data by itself is hard to analyze, but there are different analytical tools that can turn it into valuable insights which are further used to make coaching decisions. Furthermore, it can be used for automated event detection [17], which shows that event data could be derived almost completely from positional data of the players and ball, thus adding to its importance. Tracking data is acquired in many different ways and it is crucial in the evaluation of player performance. The various technologies used for this purpose, vary a lot in accuracy and as we will see later, they have a large impact into the final insights. In this chapter, we will discuss the different methods used for extracting player positions and the data we have used for analysing the smoothing technique.

2.1 Methods for extracting tracking data from soccer games

The four main technologies currently used for gathering this type of data are [14][7]:

- OT (Optical Tracking) Systems
- Vision-Based Models
- GPS (Global Positioning Systems)
- LPS (Local Positioning Systems)

They work in fundamentally different ways and we will proceed by shortly explaining each one.

2.1.1 Optical Tracking Systems

Optical Tracking is a technology used in a lot of industries, that has also been adopted in the footballing world as one of the best methods for tracking the positions of the ball and players during matches. It has already been used in major tournaments like the 2022 FIFA World Cup and the 2024 UEFA European Football Championship, playing an important role in both refereeing decisions (semi-automated offside technology) and performance analysis. It is regarded as one of the more precise methods, as it uses multiple calibrated high-resolution cameras installed in the stadium to detect and triangulate the positions in 3D space of the different body parts of the players. For deriving the atomic position of the player, a model is used to extract his center of mass.

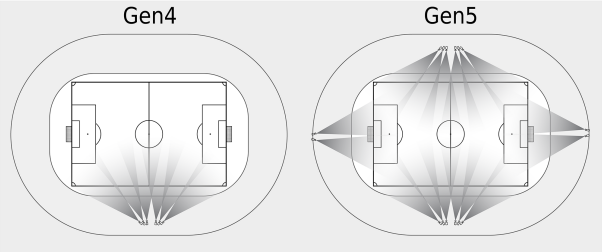


Figure 2.1: Camera setup for 2 Generations of the TRACAB OT System [4]

2.1.2 Vision-Based Models

Vision-Based Models for football tracking are relatively new, and work by approximating the positions of players using only broadcast images. This allows for a lot of flexibility, as no expensive calibrated camera equipment is required. The positions of players outside the broadcast frame will be approximated by extrapolation which is explicitly mentioned for each player in each frame, if it is the case. This method has lower precision compared to Optical Tracking, especially when extrapolating. Even small changes in positions along the trajectory of movement can lead to high fluctuations in speed approximations, thus yielding imprecise metrics.

2.1.3 Global Positioning Systems (GPS)

GPS is the most common method used for tracking objects in outside environments. It is one of the most impactful inventions of the 20th century and has been constantly improved over the last few decades. We can also see its adaption in football tracking. GPS can vary a lot in precision and it is highly impacted by the surrounding environment. For example recording positions next to high structures creates a significant error in the data,

known as GPS Drift. Large football stadiums like the Estadio Santiago de Bernabeu or Wembley Stadium would complicate the measurment of player positions using this method. Moreover, the ball cannot be tracked using GPS, as there is not a solution yet for fitting such a device within the ball without affecting its movement physics.

2.1.4 Local Positioning Systems (LPS)

Local Positioning Systems use base stations (short-range signaling beacons, with a known exact location) to extract the positions of transceivers worn by the players and the ball. Similarly to GPS, they also use triangulation, but because of the base stations, LPS systems are a lot more precise and are influenced less by the surrounding environment. LPS solutions typically exhibit an error margin ranging between 19 and 27 cm.

2.2 Inertial Movement Data

Inertial Movement data is captured by IMU(Inertial Movement Units) devices, which are basically accelerometers worn by the players to precisely measure their changes in speed. This type of data is also highly significant in performance analysis and is being used by some of the best football teams. Moreover, the same technology is also used in footballs. In combination with accurate player and ball tracking data, it facilitates the use of semi-automated offside technology.

2.3 Speed approximation

One of the most important metrics in performance analysis is the speed at which a player is moving. Its extraction cannot be instantaneous, but speed values can be approximated by using the euclidean distance and timestamp difference of recorded positions in a trajectory. We take inspiration from [18] and approximate the speed in the following way:

The speed calculation in the continuous domain at time t is

$$s(t) = \lim_{\Delta \to 0} \frac{\sqrt{(x(t+\Delta) - x(t-\Delta))^2 + (y(t+\Delta) - y(t-\Delta))^2}}{2\Delta}$$

x, y - functions that model the movement of the player on each axis of the pitch

In the discrete domain, that we also use in practice, the formula above translates to

$$\hat{s}(t) = \omega \frac{\sqrt{(x(t+1) - x(t-1))^2 + (y(t+1) - y(t-1))^2}}{2}$$

ω - Frequency of tracking data in Hz

The measurement unit of $\hat{s}$ is $[\frac{m}{s}]$

Chapter 3

Smoothing Techniques

Tracking data tends to be noisy, especially if the surrounding environment does not respect the requirements of the tracking systems acquiring it. As we have seen in Chapter 2, some tracking systems work in fundamentally different ways and differences between their outputs are inevitable. In an attempt to bring the tracking data of different tracking systems closer together, some kind of filtering must be applied. For this purpose, we resort to smoothing techniques. Positional data can be represented as two one dimensional signals, so any filter that can be applied to one dimensional discrete signals, can be extended to work for this use case.

In this chapter, we will present two different smoothing techniques which can be used to filter football tracking data. Firstly, we will present a filter from the 1960's which is still in use today and is an industry standard for removing noise. It is defined for one dimensional signals, but as we will see, we can also extend this to two dimensions. Afterwards, we will present the novel Circular Arc Smoothing technique and explain how it is applied to the tracking data.

3.1 The Savitzky-Golay Filter

A current solution used for smoothing noisy signals, is the Savitzky-Golay Filter. It is applied in the noise reduction of many different types of signals, including football tracking data [17].

In this subsection, we will explain how the Savitzky-Golay filter works and how it can be applied on football tracking data. We do this, as later on in chapter 5.5, we will put our geometric smoothing technique and the S-G filter side by side in a comparison.

Filtering of a 1D discrete signal with S-G

Given a discrete signal $x : \{0, ..., P\} \to \mathbb{R}$, the S-G filter works in the following way:

We pass through x with a window size of $2M + 1$. For each window, we fit a polynomial

$$p(n) = \sum_{k=0}^{N} a_k n^k$$

of degree N to the data, that minimizes the mean squared error

$$\mathcal{E} = \sum_{n=-M}^{M} (p(n) - x[n])^2 = \sum_{n=-M}^{M} (\sum_{k=0}^{N} a_k n^k - x[n])^2$$

Assuming the window is centered at i, the smoothened output at i is given by

$$y[i] = p(i)$$

For $i \leq M \ \ \& \ \ i \geq P - M$, we have $y[i] = x[i]$

In the original paper [11], it is shown that this smoothing is identical to the convolution of the signal x with a fixed finite impulse response h,

$$y[n] = \sum_{m=-M}^{M} h[m]x[n - m] = \sum_{m=n-M}^{n+M} h[n - m]x[m]$$

which is dependent on the window size and polynomial degree. We do not provide the proof in this thesis, however, it can be found in the original paper [11] or in the IEEE Signal Processing magazine [12].

Applying the S-G filter to football tracking data

We take inspiration from [17] and perform S-G filtering on both axes of the tracking data separately. For the implementation, we used the savgol_filter() method of the signal module from the **scipy** python library, which takes as arguments an uneven window size and the degree of the polynomials used. Compared to the circle spline smoothing technique, S-G filtering, offers us two more degrees of freedom in parameter choice for the smoothing. Finding the best combination of these parameters for football tracking data is a problem in itself and we will show multiple combinations in chapter 5.5.

3.2 Circular Arc Smoothing (CAS)

In this section, we will present how the novel smoothing technique is defined, how it changes the computation of speed approximations and also go over our implementation of it. The technique is designed so that it can be applied multiple times over a trajectory (we denote this as performing multiple iterations of the technique). Intuitively, after each iteration, the points of the trajectory are moved onto circular arcs such that the trajectory becomes a circular spline.

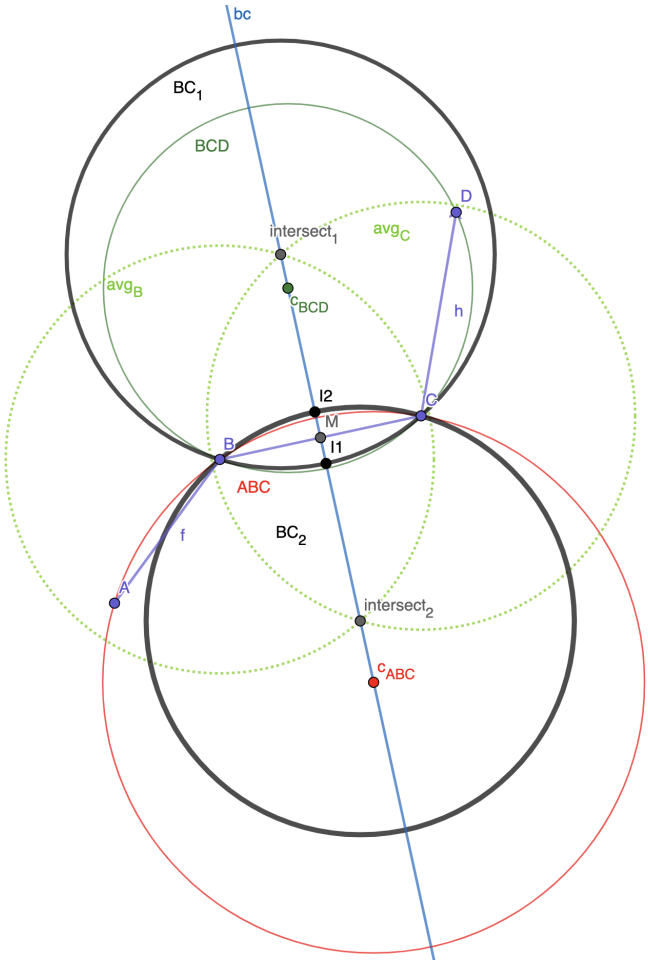


Figure 3.1: Visualisation of circle splines used for smoothing

3.2.1 Definition

Let $T \in \mathbb{R}^{n\times 4}$ be a trajectory that contains n datapoints

Let $p_i \in T$ be a datapoint which contains the following values:

- $p_i[0]$ is the coordinate of the i-th point with respect to the x-axis
- $p_i[1]$ is analog but for the y-axis
- $p_i[2]$ is the arc length (defined later; set to -1 initially and gets changed after the first iteration)
- $p_i[3]$ is the timestamp

For simplicity we analyse a contiguous sequence of points in trajectory T (see **Figure 3.1**), that we denote as $A, B, C, D \in \mathbb{R}^2$ and describe (steps 1 - 4) how a **new interpolated point** I is found between B and C. After that, in step 5, we describe how we use the newly interpolated points to move the original points of the trajectory, thus completing one iteration of the algorithm.

1. Find circles ABC and BCD which are determined by the 3 points they pass through. If both sequences A, B, C and B, C, D are collinear, choose the midpoint M. Otherwise continue
2. Compute harmonic mean r of the curvatures $1/r_1$ of ABC and $1/r_2$ of BCD
$$r = \frac{2}{\frac{1}{r_1} + \frac{1}{r_2}}$$
3. Compute $intersect_1$ and $intersect_2$ which lie on the intersections of the circles with centers B, C and radii r.
In the case that c_{ABC} and c_{BCD} are not on the same side with respect to the trajectory, we choose the *intersect* point with lowest euclidian distance to the center of the circle with largest curvature. We define the circle with center *intersect* (in this case = $intersect_1$) and radius r as BC_1
4. The new point I (in our case $I = I_1$) is found at the intersection of the arc between the points B and C of the circle BC_1 and the bisector bc
5. After the computation of all new points between all original points (just like I is for BC), we perform the same procedure again (1 - 4) on the set of newly found points, thus creating another set of new points that will replace the original points (A, B, C, D), slightly adjusting their position. Moreover, we store for each point the arc length of the circle on which it was interpolated on (for the original point B we denote the adjusted point as B' and the arc length as $arc_{B'}$). This completes a full iteration of the algorithm.

3.2.2 Changes to speed approximation

For smoothened trajectories, we avoid the speed approximations mentioned in Chapter 2.3 and decide to use another method for this. As we are moving

the points onto circular arcs, we also decide to compute the speed using the circular arc length.

The speed of the new interpolated point will be the arc length arc_I(length of arc BC of circle BC_1) divided by the time difference between frames. We enforce in the trajectory extraction that all time differences respect the mentioned sampling frequency ω, we simplify the calculations to:

$$v_I = arc_I * \omega$$

As we will use the arc length later for the calculation of the distance covered metric, we store the arc length instead of the speed when smoothing the trajectories. The speed is computed afterwards as mentioned before.

For example, Let $p \in T$ so that it corresponds to B in the diagram shown above, then $p[2]$ will be set to $arc_{B'}$ at the end of one full smoothing iteration.

3.2.3 Implementation

In this section we will go over the implementation of the smoothing filter. For the purpose of not going into too much detail, we provide the two most important functions: computeNewPoints and performSmoothing.

The function $computeNewPoints()$ is used to compute the new points described in the definition of the technique (steps 1 - 4).

"**Algorithm 2: [. . .]**" is part of the $computeNewPoints()$ method and it is inserted at line 17 in "Algorithm 1".

The function $performSmoothing()$ is used to complete the smoothing for an arbitrary number of iterations. It takes as input the datapoints that will be smoothened in the format described in chapter 4.2 and the number of iterations.

The helper functions do the following:

- **getCircle([p1, p2, p3])** - returns the circle that passes through p1, p2, p3. Defined only for non-collinear points.
- **areCollinear(p1, p2, p3)** - returns True if points are collinear, otherwise False.
- **midPoint(p1, p2)** - returns the midpoint of p1 and p2.
- **getCircleIntersections(c1, r1, c2, r2)** - returns intersection points of circles with centers c1, c2 and radii r1, r2. If there are no intersections points return -1.
- **euclideanDistance(p1, p2)**- returns euclidean distance of p1 and p2.

- **closestPoint(n1, n2, p1, p2)** - returns n1 if $euclideanDistance(n1, p1) + euclideanDistance(n1, p2) < euclideanDistance(n2, p1) + euclideanDistance(n2, p2)$ else return n2.
- **getAngle(p1, p2, p3)** - returns $\angle p1p2p3$ in radians.
- **getExtensionPoints(datapoints)** - used to extend trajectory, so that smoothing also affects the second and second to last points in a trajectory.

  It returns (pFirst, pLast, midFirst, midLast) linearly extrapolated points at both ends of the trajectory t, which is passed through the datapoints parameter. pFirst is created so that the first point in t is the midpoint of pFirst and the second point in t. midFirst is the midPoint between pFirst and the first point in t. pLast and midLast are created similarly, but in a symmetric fashion at the end of t, using the second to last and last points.

Algorithm 1: Compute New Points

```
Function computeNewPoints(datapoints):
    new_points ← [];
    trajectory ← map(lambda x: x[: 2], datapoints);
    n ← len(trajectory);
    for i ← 1 to n − 2 do
        if i == 1 then
            p1 ← trajectory[i];
            col1 ← areCollinear(trajectory[i − 1], p1, p2);
        else
            p1 ← p2;
            c1 ← c2;
            r1 ← r2;
            col1 ← col2;
        p2 ← trajectory[i + 1];
        M ← midPoint(p1, p2);
        col2 ← areCollinear(trajectory[i + 2], p1, p2);
        [ . . . ]
        c_avg1, c_avg2 ← getCircleIntersections(p1, r, p2, r);
        if c_avg1 == −1 or c_avg2 == −1 then
            new_point ← M;
            new_points.append([new_point[0], new_point[1],
             euclideanDistance(p1, p2)]);
            continue;
        min_c ← c1 if r1 < r2 else c2;
        c_avg ← c_avg1 if euclideanDistance(c_avg1, min_c) <
         euclideanDistance(c_avg2, min_c) else c_avg2;
        d ← r / euclideanDistance(M, c_avg);
        new_point_1 ← c_avg + (M − c_avg) ∗ d;
        new_point_2 ← c_avg + (M − c_avg) ∗ (−d);
        new_point ←
         closestPoint(new_point_1, new_point_2, p1, p2);
        new_point ← [new_point[0], new_point[1], r∗
         getAngle(p1, c_avg, p2)];
        new_points.append(new_point);
    return new_points;
```

Algorithm 2: [. . .]

```
if not (col1 or col2) then
    if i == 1 then
        c1, r1 ← getCircle(trajectory[i − 1 : i + 2]);
    c2, r2 ← getCircle(trajectory[i : i + 3]);
    r ← 2/(1/r1 + 1/r2);
else if col1 and col2 then
    new_point ← M;
    new_points.append([new_point[0], new_point[1],
     euclideanDistance(p1, p2)]);
    r2 ← ∞;
    continue;
else if col1 then
    c2, r2 ← getCircle(trajectory[i : i + 3]);
    r1 ← ∞;
    r ← 2/(1/r2);
else if col2 then
    if i == 1 then
        c1, r1 ← getCircle(trajectory[i − 1 : i + 2]);
    r2 ← ∞;
    r ← 2/(1/r1);
```

Algorithm 3: Perform Smoothing

```
Function performSmoothing(datapoints, iterations):
    pFirst, pLast, midFirst, midLast ←
     getExtensionPoints(datapoints);
    for 0 to iterations do
        for i ← 0 to 2 do
            if i%2 ≠ 0 then
                og ← datapoints.copy();
                updated_datapoints ← computeNewPoints([midFirst]
                 + new_points + [midLast]);
                datapoints ← [og[0]] + updated_datapoints + [og[-1]];
            else
                new_points ← computeNewPoints([pFirst] + datapoints
                 + [pLast]);
    return datapoints;
```

Chapter 4

Data and Methods

4.1 Data used

The data for our analysis has been provided by a team from the Swiss Super League (we denote it as Team A) and it was extracted from two games of the 23/24 season. One of the games took place in December 2023 and the other in February 2024. For the first game, we received data extracted by both an Optical Tracking System and two Vision-Based models, allowing for a comparison domain for the assessment of the CAS technique. Moreover, we also received lineup information for both games and inertial movement data of the home players during the game in February. The lineup files of the different providers have been crucial for our work, as without them we could not compare metrics extracted from one data source to another. This is because every provider uses different player identification numbers in their tracking data and the mappings of player names to id's can only be found in the lineup files.
The total data received sums up to around 400k frames of tracking data which corresponds to approximately 8.7M positional datapoints.

4.1.1 Format and Providers

Optical Tracking System (StatsPerform)

The data that we have been working on that uses this method, has been obtained from StatsPerform and has an image sampling frequency of $25Hz$. The X, Y and Z coordinates are provided in meters using floating point numbers from the top left of the pitch. There is also a buffer boundary (approx. 1m wide) outside the field of play

The tracking data is given in a .txt file which contains a line for each frame with the format:

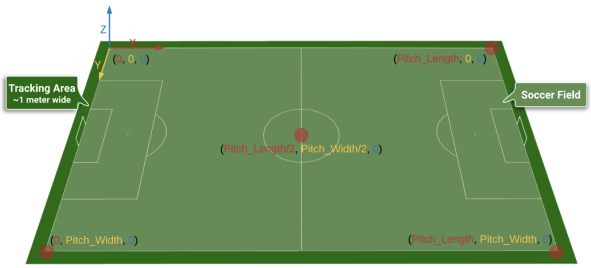


Figure 4.1: Lineup format StatsPerform

System Time in milliseconds; milliseconds of the current half, half indicator : player coordinates;: ball coordinates;

0;2600,1:3,975798,16,87.78,34.02;0,383098,24,48.22,67.69;0,606793,34,63.22,42.93;0,1051077,3,64.89,27.86;0,883903,27,55.89,8.83;0,587975,28,51.29,47.95;0,596176,19,43.69,38.43;0,587067,8,58.31,33.61;0,364737,22,36.76,41.29;0,589151,17,41.93,14.01;0,358613,10,47.41,20.48;4,383202,1,2.55,34.28;1,1058853,22,39,16.18;1,589770,4,26.35,24.59;1,377028,5,26.7,32.27;1,348973,27,30.85,46.07;1,352565,23,29.43,15.38;1,9616103,14,37.12,42.23;1,357921,20,39.04,34.36;1,379408,10,30.54,24.93;1,1061321,25,39.58,55.68;1,973049,7,46.63,34.08;:50.95,47.73,0.03;

We were also provided a .csv file which included the lineups of both teams, together with some additional data for each player. This is an example of a row in the lineup file:

30,1,John,Doe,2,17-JAN-04,1701894600000,0,1189563,1189563,2559735,2559735;

Field	Description
Jersey number	uniform
Object type	0 = Home Player 1 = Away Player 2 = Referee 3 = Home Goalie 4 = Away Goalie 5 = Ball
First name	First name
Last name	Last name
Player birth date	Player birth date
Position ID	0 = Unknown 1 = Forward 2 = Midfielder 3 = Defender 4 = Goalkeeper
Inpitch	time entered the pitch
Outpitch	time left the pitch (if 0 then he played till the end)
Player ID event	Player ID matching the player ID in event data file
Player ID tracking	Player ID matching the player ID in tracking data file
Match ID event	Match ID matching the match ID in event data file
Match ID tracking	Match ID matching the match ID in tracking data file

Figure 4.2: StatsPerform lineup line format

Vision-Based Model Tracking Systems (SkillCorner)

The data sources that we have been working with and that use this method, have been obtained from SkillCorner and ReSpo Vision, but we will focus mostly on data provided by SkillCorner.

Similar to StatsPerform, Skillcorner coordinates in the X, Y and Z axes are given in meters, but they are all set to zero at the center of the pitch [1]. The sampling frequency of this data is $10Hz$

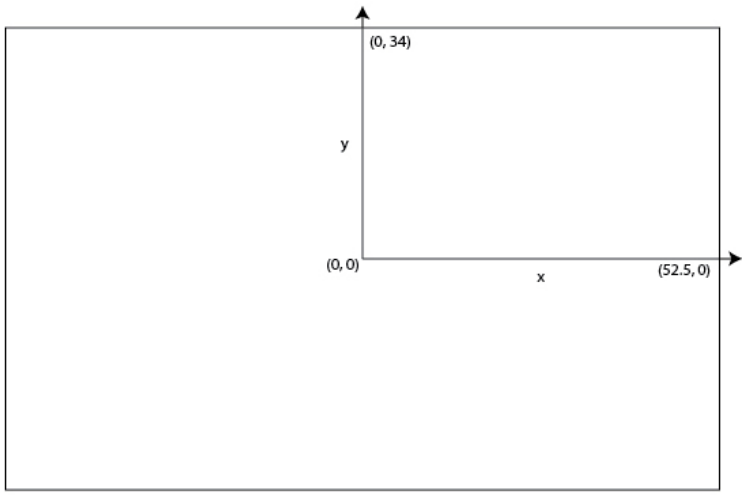


Figure 4.3: SkillCorner pitch modelisation

The original tracking and lineup files come in a .json format, but for the ease of use in our notebooks, we decided to preprocess them in the following way:

The lineup file is transformed to a .csv file with a row for each player with the columns:

```
match_id,team_name,player_id,player_first_name,player_last_name,\\
player_shirt_number,player_position,player_birthdate,start_time,\\
end_time,yellow_card,red_card,injured,goal,own_goal
```

The tracking file is also transformed into a .csv file, which contains a row for each player on the field for every frame. The format of the .csv rows is the following:

```
match_id,half,frame_id,timestamp,object_id,x,y,z,extrapolated
```

Inertial Movement Data

This kind of data was provided by Catapult and we plan to use it by comparing it to the acceleration approximation extracted from the tracking data. The sampling frequency of the devices is $100Hz$, so to be able to compare it to the tracking data, we would have to first filter it and then sub sample it to match the frequency of the tracking data. Each player has his own designated .csv file with rows that corresponded to the following header format:

```
Period, Ticks, Date/Time, Timestamp, Acceleration.forward  \\
Acceleration.side, Acceleration.up, Rotation.roll, Rotation.pitch \\
Rotation.yaw, Facing, imuOrientation.forward, imuOrientation.side \\
imuAcceleration.forward, imuAcceleration.side, imuAcceleration.up \\
Acceleration Magnitude
```

4.2 Standardising trajectory data

To be able to perform the smoothing on all these different tracking data sources, a new generalised format needed to be created for the player trajectories. This would allow the smoothing to be performed with a simple function call within our notebook. We define a player trajectory as a set of positions of the same player from the tracking data that are consecutive, respect the sampling frequency and were not extrapolated (in the case of SkillCorner and ReSpo Vision). Moreover, a player trajectory needs to have a minimum number of datapoints. For our use case, we decided to set the minimum to five points.

The extracted trajectories from a complete football match would respect the following format:

$$traj1 = [[xPos, yPos, arc\_length = -1, timestamp, frame\_id], \dots]$$

$$extracted\_Trajectories = \{player_1 : [traj1,\ traj2...],\ player_2 : [\ \dots\ ]\ ...\}$$

The smoothing is performed on data which is formatted like this. For each different tracking data provider, we first translate the data into this format and then perform the smoothing.

4.3 Methods used for the investigation

In this section, we will go through all the methods used for the investigation of the CAS technique. We will provide background for existing methods and present our own additions.

4.3.1 Analysis of Physical Metrics

Physical metrics occupy a pivotal role in football performance analysis and are a means of summarising tracking data. It is hard for a human to analyse positional data directly, so that is where these metrics come into effect. Their extraction is important, as it will allow us to compare them between non-smoothed and smoothed trajectories, which provides us with insights into the dynamics of the CAS.

Types

In the following section the focus will be set on the following physical metrics extracted from the tracking data [1]:

- Distance covered
- Maximum speed and the time it was sustained
- Sprint count
  - The number of sprints which surpass $25km/h$ and are sustained for at least 0.7s

Extraction

After calling $performSmoothing()$, we are given a format as described in subsection 4.2. As we are interested in the metrics of a specific player, we create a pandas Data Frame from the list of trajectories of this player. Each metric is extracted as follows:

Max speed

As $performSmoothing()$ does not directly return the speed of each point in the trajectory, we must create a new column in our Data Frame that computes the speed of each point according to its arc length and the sampling frequency as mentioned in 3.2.2. After that, we can easily extract the maximum value of the speed column. For the scope of the analysis, we measure how much time (ms) this speed is sustained. We consider the speed to be sustained if it is within $1km/h$ of the maximum speed for a subset of consecutive points around the position with the top speed.

Sprint count

During the same loop that we use for calculating the sprinting distance, we also count the sprints which surpass $25km/h$. This is done by keeping track of the amount of subsets of consecutive points in the trajectory that surpass this speed.

Distance covered

In the tracking data, there might be times when we would prefer not to smoothen (ex. if positions are extrapolated in a model like the one developed by SkillCorner), so we need to be able to combine the distance covered of both these separate sections.

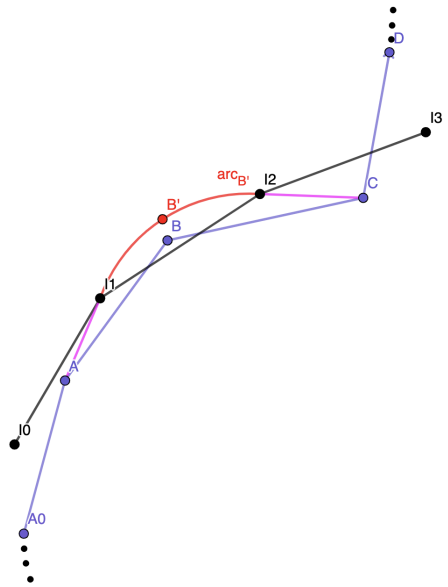


Figure 4.4: Extracting distance covered from smoothed trajectory

I_i - new points (computed in steps 1-4)

X' - new points meant to replace the original ones (ex. B', computed during step 5)

When performing the smoothing, we also return the arc length associated with each newly computed point (ex. $arc_{B'}$ for B').

- For simplicity let's assume all points before A and after C are interpolated linearly and we do not perform smoothing on them.
  - In this case, we connect the smoothed arcs (ending in I1 and I2) with linear segments to the neighbouring points in the trajectory (A and C).
- Otherwise, if A or C (or both) would also be updated to C' and A', then we simply add the lengths of the arcs together to retrieve the total distance of the smoothed segment. (6)

Usually we would add all the smoothed and linear segments together for the total distance metric, but for the scope of this investigation, we decide to completely ignore extrapolated points (in the case of Vision-Based tracking) and only focus on smoothed segments. Thus, only the distance for sequences as mentioned in (6) is added and all other linear segments are ignored.

4.3.2 Trajectory Similarity

For performing further analysis on the way trajectories are impacted by our smoothing technique, we must be able to quantitatively assess how similar two trajectories are. This will be especially useful in chapter **??**, where we try to reconstruct a trajectory after high levels of noise are applied to it. Trajectory similarity can be calculated in multiple ways. The first, most naive solution, would be to compute the Lock-Step distance between two sets of points of the same size. The Lock-Step distance is computed by simply adding up the euclidean distance of the corresponding pairs of points in each set. (in our case, we use timestamps to pair the points between sets). For example, for two trajectories of length four, with points recorded at times t1, t2, t3 and t4, the Lock-Step distance will be the sum of euclidean distances between the points at time t1, t2, t3 and t4. This approach would not work as well in our use case, because of the nature of our smoothing technique. Throughout our experimentation with the technique, we have noticed that the tracking points get changed mostly along the tangent of the trajectory. This mostly impacts the speed and acceleration values, but keeps the shape of the trajectory quite similar. The Lock-step distance would be unnecessarily large in this case, so we decide to use another approach called Dynamic Time Warping, which will reduce the importance of the point modulation along the trajectory tangent.

Background on Dynamic Time Warping

Dynamic Time Warping (DTW) is a common technique used for mapping two time-dependent sequences together (Fig. 4.5). Moreover, it can be used to extract a total cost between these sequences which is indirectly proportional to their similarity. DTW has many applications in different areas like speech [6] and handwriting [9] recognition, finance [15], human movement recognition [2], etc. Firstly we will explain how classical Dynamic Time Warping works and afterwards we will apply this idea to two dimensional tracking data, so that we are be able to compute the trajectory similarity.

Classical DTW

Let us define two sequences $X := (x_1, x_2, x_3, ..., x_N)$, $Y := (y_1, y_2, y_3, ..., y_M)$ and a feature space $\mathcal{F}$ so that for all $i \in \{1, ..., N\}$ and $j \in \{1, ..., M\}$: $x_i, y_j \in$

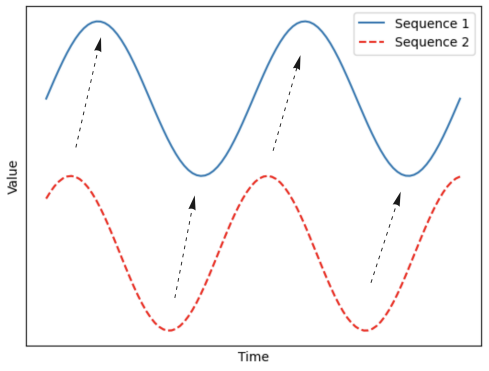


Figure 4.5: Intuition for mapping sequences with DTW

$\mathcal{F}$.

As a next step, we need to define a local cost function $c : \mathcal{F} \times \mathcal{F} \to \mathbb{R}_{\geq 0}$, that returns the cost $c(x_i, y_j)$ of mapping feature x_i to y_j. The cost $c(x_i, y_j)$ should be small if features x_i and y_i are similar and large if they are not. With the definition of this cost function, we can define a local cost matrix $C(i, j) := c(x_i, y_j)$ which we will use later to find a path with minimal accumulated cost. A commonly used local cost function for $\mathcal{F} = \mathbb{R}$, would be the absolute difference $c(x_i, y_j) := |x_i - y_j|$

After computing matrix C, we proceed with a Dynamic Programming approach to compute the (N, M)-warping path through the cost matrix C, that minimizes the accumulated cost, which we denote as the distance d between the two sequences.

Definition 4.1 *An (N, M)-warping path (or simply referred to as warping path if N and M are clear from the context) is a sequence $p = (p_1, ..., p_L)$ with $p_l = (n_l, m_l) \in \{1, ..., N\} \times \{1, ..., M\}$ for $l \in \{1, ..., L\}$ satisfying the following three conditions.*

1. *Boundary condition: $p_1 = (1, 1)$ and $p_L = (N, M)$.*
2. *Monotonicity condition: $n_1 \leq n_2 \leq ... \leq n_L$ and $m_1 \leq m_2 \leq ... \leq m_L$.*
3. *Step size condition: $p_{l+1} - p_l \in \{(1, 0), (0, 1), (1, 1)\}$ for $l \in \{1, ..., L - 1\}$.*

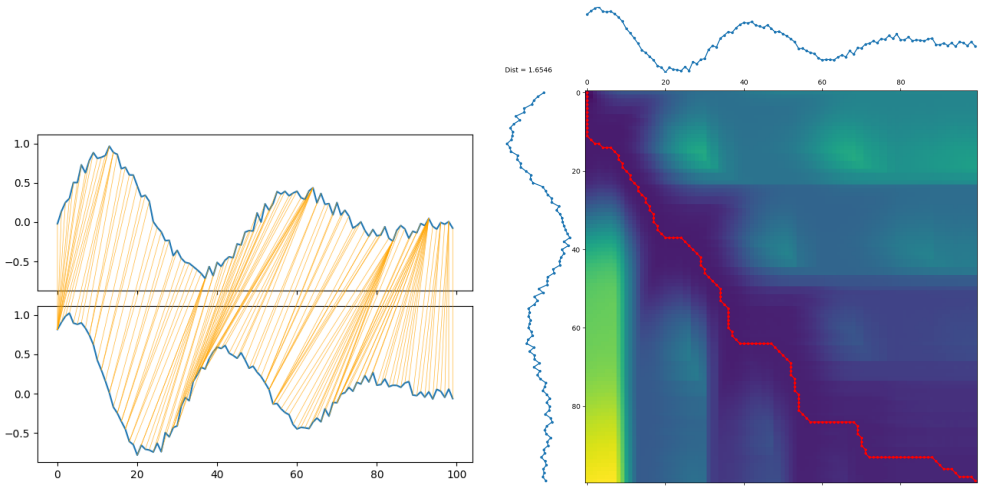
We now define the accumulated cost matrix $D \in \mathbb{R}^{N \times M}$. Respecting the three conditions of warping paths from 4.1, we can compute the matrix as follows:

For $n \in \{1, ..., N\}$ and $m \in \{1, ..., M\}$ we have: $D(n, 1) = \sum_{k=1}^{n} c(x_k, y_1)$ and $D(1, m) = \sum_{k=1}^{m} c(x_1, y_k)$ and

$$D(n, m) = min\{D(n - 1, m - 1), D(n - 1, m), D(n, m - 1)\} + c(x_n, y_m)$$

[6]

By backtracking through the accumulated matrix D, we can find the optimal warping path p and the total distance of the two sequences $d = D(N, M)$.



(a) Mapping of two signals

(b) Cost Matrix and optimal warping path (red)

Figure 4.6: DTW visualisations

DTW for 2D tracking data

As we are working with 2D tracking data we have to define a feature space $\mathcal{F} = \mathbb{R}^2$ and a cost function, which will be the squared euclidean distance $c : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$, $c((x_1, x_2), (y_1, y_2)) := (x_1 - y_1)^2 + (x_2 - y_2)^2$. For computing the DTW distance, we used the dtw_ndim module from the **dtaidistance** python library, which supports these definitions by default.

4.3.3 Jerk/Jolt extraction from Speed Time Series

Jerk also know as Jolt [3], denoted by j, is defined as the derivative of the acceleration a, so the 3rd derivative of distance with respect to time t. In performance analysis, large values of jerk are in close relation with physical exertion and fatigue [19]. One of the goals of the smoothing technique is to reduce the noise in the jerk signal and also reduce its values, with the goal of making its time series more credible. Mathematically, jerk can be expressed as:

$$j(t) = \frac{\mathrm{d}a(t)}{\mathrm{d}t}$$

We also use the rate of change of the standard deviation of the jerk with respect to the number of iterations to assess how fast a trajectory is smoothed. This is done to show the importance of the tracking frequency when choosing the number of iterations to perform on a trajectory.

4.3.4 Noise Experiments

In the development and evaluation of any smoothing technique, it is important to perform noise experiments for the assessment of the methods robustness and reliability. In the domain of signal processing, noise is defined as a general term for unwanted modifications that a signal may suffer during capture, storage, transmission, processing, or conversion [16]. As we have seen in [13], different tracking systems induce different types of noise in their capture of the data. We take this a step further, by increasing the amount of noise in the signal in an intuitive way, to see how the smoothing technique manages to hold up and reconstruct the original signal.

In this chapter, we will introduce concepts necessary to show the robustness of the smoothing technique by means of noise experiments. We will first show on a specific isolated trajectory, the visualisation of the artificially added noise and how the smoothing improves the quality of this trajectory. Afterwards we will perform aggregated experiments across all trajectories of the team players during the game in February 2024.

Applying the noise

When applying the noise to the tracking data, we decided to use a more intuitive approach, that we think would more accurately replicate real-world noise during tracking. Instead of simply adding random values from the same one dimensional distribution to the x and y coordinates of each datapoint, we decided to take into account both neighbouring datapoints by defining a multivariate distribution with axes which are given by the vector that connects these two neighbouring datapoints and the vector perpendicular to it, which is scaled according to the desired distribution. This allows us to add more noise tangent to the trajectory, which will increase the speed time series noise.

Thus, the noise we apply follows a 2D Gaussian Distribution with both means set to 0, and the standard deviations along the perpendicular and tangent axes that correspond to σ_p and σ_t. We represent this type of noise as $\mathcal{N}(0, \Sigma_{\sigma_p \sigma_t})$, where $\Sigma_{\sigma_p \sigma_t}$ is the covariance matrix which respects the axes mentioned.

Chapter 5

Results

In this chapter, we will apply the methods described in Chapter 4.3 on the data from Chapter 4.1 and present the results. We will conduct a comprehensive investigation, through which we aim to better understand how the smoothing affects the approximated speed time series and to see if it brings two different tracking data sources closer together.

5.1 Results of physical metrics analysis

5.1.1 Initial differences across data sources

The following metrics have been extracted from the tracking data of different Team A players (name mentioned before each metric table) during the game in December 2023. We only considered non-extrapolated trajectories, to be able to make an accurate comparison between SkillCorner and StatsPerform data. We did this by matching the match timestamps of both data sources.

Player 1 (Optical [StatsPerform] and Broadcast [SkillCorner])

StatsPerform (Optical tracking)

- Top speed: 7.99 m/s
- Distance covered: 2801.51 m
- Top speed sustained: 680 ms
- Sprint count: 3

SkillCorner (Vision-Based model)

- Top speed: 8.71 m/s
- Distance covered: 2970.96 m
- Top speed sustained: 900 ms
- Sprint count: 6

The difference of the distance covered is about 169m, which is quite substantial, showing a large trajectory difference between data sources. Moreover, the sprint count is completely off in the SkillCorner data, as a lot of noisy speed peaks are counted, thus yielding double the amount of sprints compared to the more accurate optical tracking system. Interestingly, the top

speed which is significantly larger, is also sustained more in the SkillCorner data. This is an outlier, as when analysing other players, we see that usually this is not the case, but it could imply some kind of filtering of the raw data before showing the final tracking positions.

Player 2:

StatsPerform

- Top speed: 9.25 m/s
- Distance covered: 6498.64 m
- Top speed sustained: 840 ms
- Sprint count: 16

SkillCorner

- Top speed: 9.10 m/s
- Distance covered: 6471.92 m
- Top speed sustained: 800 ms
- Sprint count: 16

In Player 2's metrics, it is interesting to notice that the top speed of the optical system is larger. This is quite unusual, as a higher variance in speed values (noise of a signal compared to ground truth) should cause higher peaks. The distance covered and sprint counts are closer together this time.

Player 3:

StatsPerform

- Top speed: 9.06 m/s
- Distance covered: 4544.49 m
- Top speed sustained: 520 ms
- Sprint count: 10

SkillCorner

- Top speed: 11.47 m/s
- Distance covered: 4552.72 m
- Top speed sustained: 500 ms
- Sprint count: 12

Here we can see once again, really different values in the top speed. The top speed of 11.47 m/s is not possible, as it translates to 41.3km/h which is way above what we would expect. To give some context, according to this article from Sky Sports, the fastest speed ever recorded in the Premier League, was that of Micky van de Ven during Tottenham's 3-2 win over Brentford on the 21st of January 2024, when he managed to reach 37.38km/h. What is also quite alarming, is that this speed is sustained for around 500ms in the SkillCorner Data, which will have a large impact on the smoothing later.

5.1.2 Impact of smoothing

In this subsection, we analyse the changes caused when performing multiple iterations of the smoothing algorithm.
As there is no ground truth for these metrics, we try to compare the more precise optical tracking with the model based tracking to see how the differences evolve. It is important to note that the higher granularity in the StatsPerform data (25Hz compared to 10Hz for SkillCorner) means that its

trajectories will be influenced ”slower” by the iterations compared to SkillCorner. This is described in more detail in chapter 5.2.

Player 1

StatsPerform

Its.	Max. Speed (m/s)	Sustained (ms)	Sprint Count	Distance Covered (m)
2	7.9954	680	3	2801.5148
5	7.9899	680	3	2801.4607
10	7.9778	680	3	2801.3828
50	7.8899	760	3	2800.7308
100	7.7989	840	3	2799.9610
150	7.7235	920	3	2799.2071
200	7.6598	960	3	2798.4602

SkillCorner

Its.	Max. Speed (m/s)	Sustained (ms)	Sprint Count	Distance Covered (m)
2	8.7057	900	6	2970.1050
5	8.6840	800	6	2969.1668
10	8.6404	900	5	2968.0586
50	8.3179	1000	3	2962.1710
100	7.9827	1100	3	2956.4324
150	7.7019	1100	2	2951.4185
200	7.4603	1200	2	2947.0675

Player 2

StatsPerform

Its.	Max. Speed (m/s)	Sustained (ms)	Sprint Count	Distance Covered (m)
2	9.2442	840	16	6498.7006
5	9.2380	840	16	6498.6144
10	9.2309	840	16	6498.4864
50	9.1742	840	16	6497.5016
100	9.1038	880	15	6496.1551
150	9.0347	840	15	6494.7694
200	8.9686	880	14	6493.4188

SkillCorner

Its.	Max. Speed (m/s)	Sustained (ms)	Sprint Count	Distance Covered (m)
2	9.0739	800	17	6470.9906
5	9.0353	900	17	6469.5977
10	8.9821	900	17	6467.7659
50	8.7064	1200	10	6456.8247
100	8.4780	1300	9	6445.6504
150	8.2953	1500	9	6435.5181
200	8.1919	2700	8	6426.0580

Player 3

StatsPerform

Its.	Max. Speed (m/s)	Sustained (ms)	Sprint Count	Distance Covered (m)
2	9.0555	520	10	4544.5326
5	9.0450	520	10	4544.4644
10	9.0248	520	10	4544.3618
50	8.8751	560	10	4543.4952
100	8.7766	1240	9	4542.3580
150	8.7462	1240	9	4541.2249
200	8.7151	1280	9	4540.1154

SkillCorner

Its.	Max. Speed (m/s)	Sustained (ms)	Sprint Count	Distance Covered (m)
2	11.4333	400	12	4551.7442
5	11.2768	500	12	4550.5350
10	11.0467	500	12	4548.9925
50	10.6320	500	11	4539.7564
100	10.2081	500	8	4530.9101
150	9.7938	500	8	4523.2593
200	9.4333	500	6	4516.2815

Distance covered

We can observe a trend of distance covered reduction, this is consistent across both the data sources and the players. This reduction is also noticeable when looking directly at the plotted trajectories (Figure 5.1), where the turns are starting to follow a rough circular arc, which cuts sharp turns and leads to a lesser covered distance as compared to the original trajectory. The rate of change of this metric is more accelerated in the lower frequency Skill-Corner data, which helps with the fact that in most cases it is overestimated with respect to the optical tracking data.

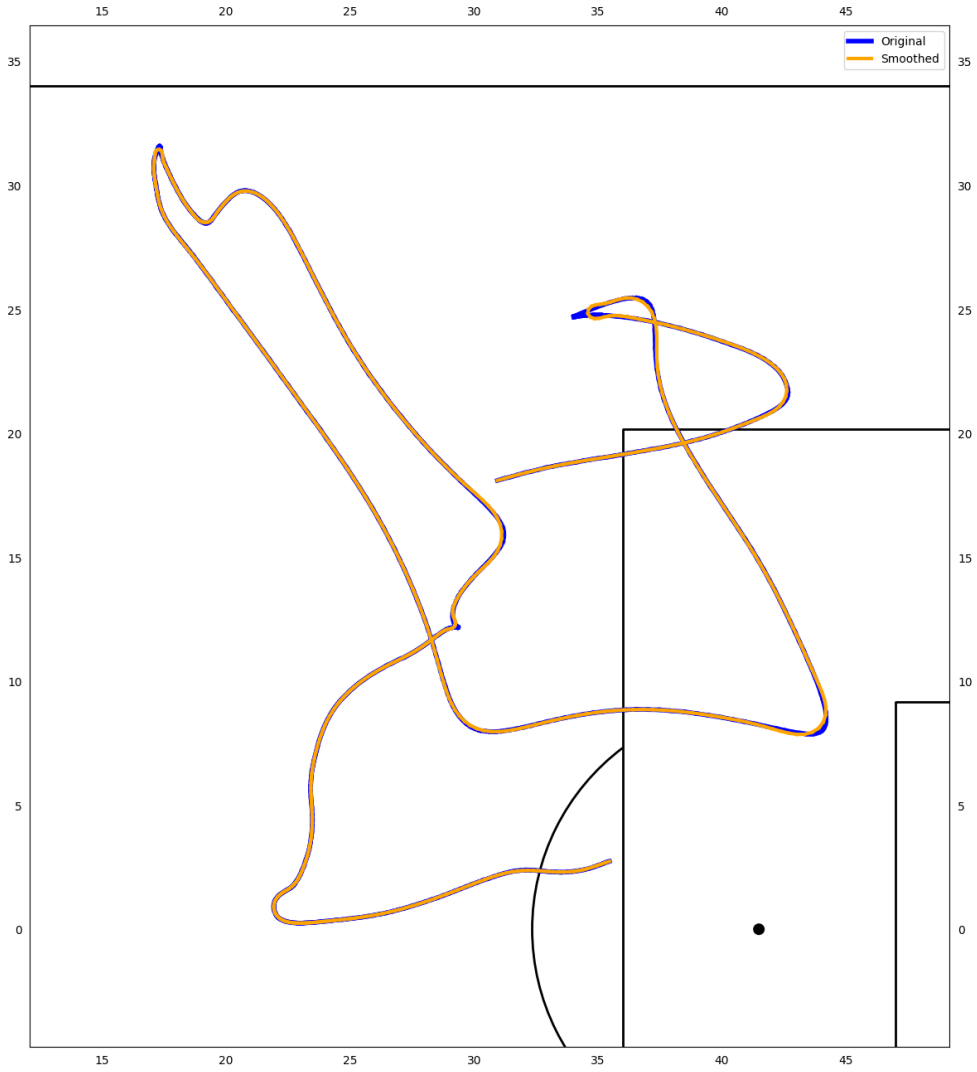


Figure 5.1: Smoothed (200 iterations, orange) vs original (blue) trajectory visualisation

Maximum speed

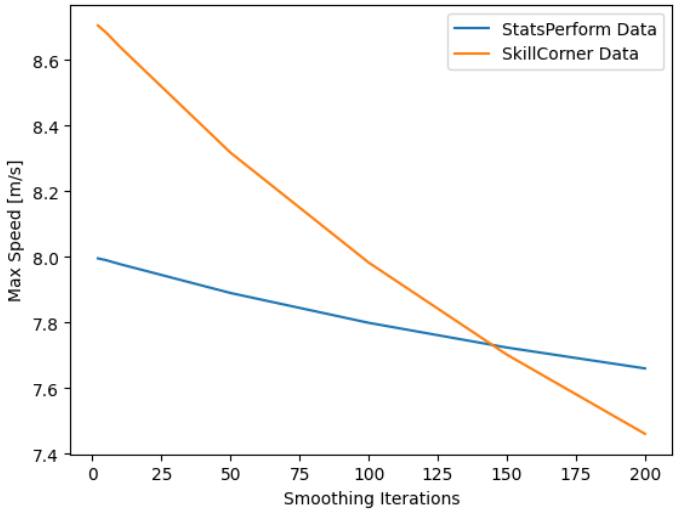


Figure 5.2: Maximum speed values of Player 1 at different smoothing iterations

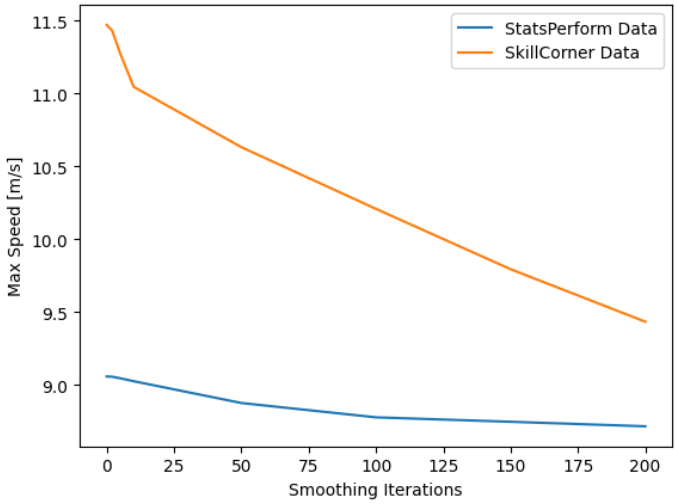


Figure 5.3: Maximum speed values of Player 3

When smoothing the tracking data, we slightly adjust all the positions of the original points in the trajectory in such a way, that the speed of the player gets indirectly smoothed as well. This has a large effect on the maximum speed as can be seen in Figure 5.2. Once again, it is interesting to see that the SkillCorner Data reacts much faster to the smoothing than the more precise

optical data. After around 100 iterations, the maximum speed recorded from the model tracking reaches the value of the non-smoothed optical one. The number of iterations for this to happen is highly dependent on how much noise and how sustained it was in the signal. For example for Player 3's trajectory which reached an impossible maximum speed of $11.43 m/s$ would take many more iterations to get smoothed out until it reaches the original optical value. By doing this, relevant information that we still need about the speed will get smoothed out. Moreover, the general trajectory would get oversimplified, which might influence other types of performance analyses in a negative way.

What is also worth mentioning, is the sudden jump in between iterations 2 and 10 (Figure 5.3). What happens here, is that a completely different position (most likely part of another sprint) becomes the source of the new maximum speed. This can happen if that speed is more sustained, thus the smoothing having a smaller impact on it, while bringing the position with the original maximum speed to a lower value.

Sprint count

Similar to the distance covered, the sprint counts are overestimated in the SkillCorner data. We see that usually the values of this metric are brought together in both data sources at around 10-50 iterations.

5.1.3 Reduction of differences in metrics as a result of smoothing

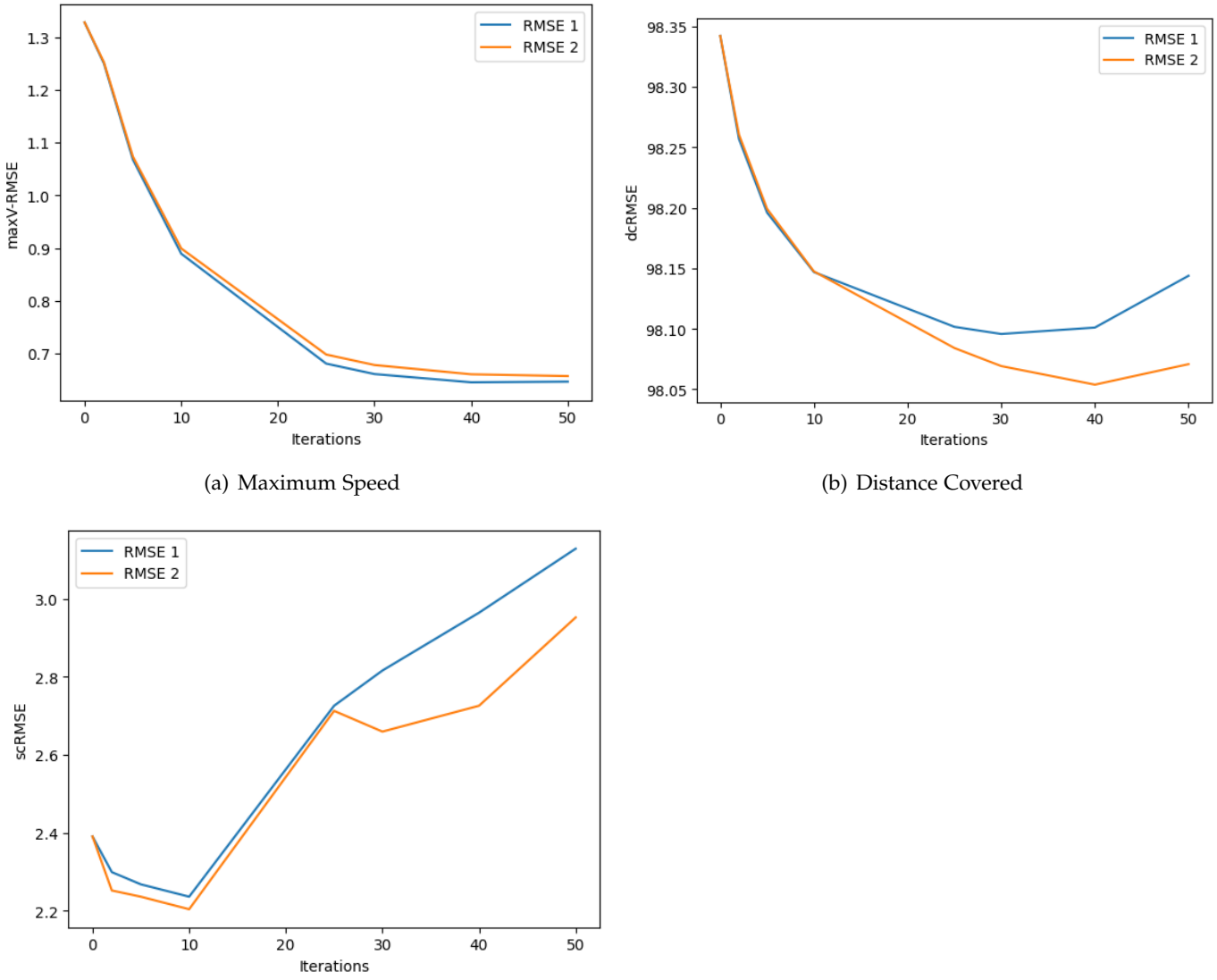
While looking at the values between iterations 0-50, we noticed that the differences in metrics between the two tracking systems seem to be slightly reduced. To quantify this, we extract the metrics for all players during that game while using multiple smoothing iterations (#iterations $\in$ {0, 2, 5, 10, 20, 25, 30, 40, 50}).

To do this, we first create for each metric and tracking system, a vector $m \in \mathbb{R}^{1\times N}$ (N: number of players analysed), which contains the value of that specific metric for each player according to data extracted with that specific tracking system. For each smoothing level (#iterations $\in$ {0, 2, 5, 10, 20, 25, 30, 40, 50}), we extract all these metric vectors.

We define "RMSE 1 at iteration i" as the root mean squared error ($RMSE(m, r)$) between metrics vector m of the SkillCorner tracking system for i iterations of smoothing and the metrics vector r of the StatsPerform tracking system for 0 iterations of smoothing. "RMSE 2 at iteration i" is defined similarly, as $RMSE(m, q)$, where q is the metrics vector of the StatsPerform tracking system for i iterations of smoothing.

Intuitively, RMSE 1 assumes StatsPerform data is a lot more accurate (does not require smoothing) than SkillCorner data and tries to show how much

closer it brings their metrics together by only smoothing the SkillCorner trajectories. RMSE 2 is used to show how much closer the metrics of both tracking systems are brought together when smoothing both data sources with the same number of iterations.



(a) Maximum Speed

(b) Distance Covered

(c) Sprint Count

Figure 5.4: Root mean squared error of metrics

In Figure 5.4, we plot RMSE 1 and RMSE 2 for the maximum speed, sprint count and distance covered metrics. As we can in Fig. 5.4(a), the maximum speed RMSE is reduced by quite a lot, showing that the smoothing brings this metric closer together for both tracking systems.

For the distance covered we see that the distance covered RMSE is reduced until a specific number of iterations, after which it starts diverging. This is caused by a combination of the cutting of the corners in the trajectory after smoothing (as described in chapter 5.1.2 and shown in Figure 5.1) and the higher values of the distance covered in the SkillCorner data. The diverging of RMSE 2 is delayed compared to RMSE 1 as the StatsPerform data is also smoothed before extracting the metrics.

The sprint count is negatively affected by more than 10 iterations of smoothing. More tracking data would need to be used to clarify this phenomenon.

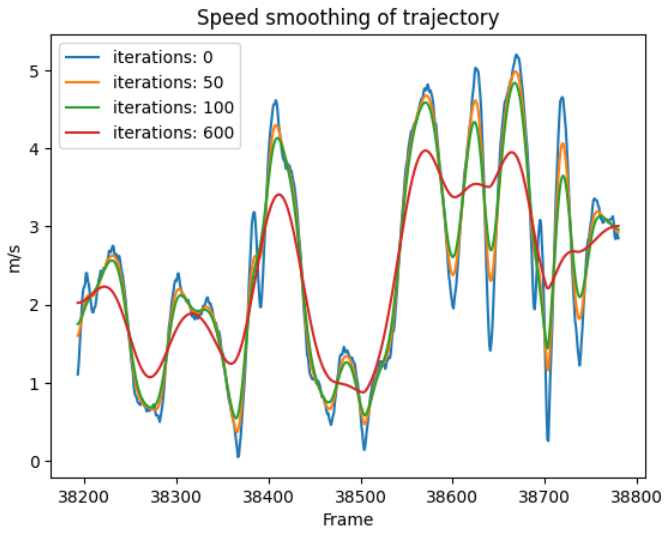
It is important to note that smoothing also impacts trajectories at different sampling frequencies in different ways. As can be seen from the tables with the metrics values of the three players, metrics extracted from StatsPerform data (25Hz sampling rate) are changed less by the same number of smoothing iterations in comparison to SkillCorner data, which has a sampling rate of only 10Hz. This is discussed more in chapter 4.3.3, subsection "Assessing the impact that the tracking frequency has on the speed of the smoothing".

Only using these two tracking data providers for a single game, it is hard to know if the reduction in differences happens consistently. What is a fact, is that for the given data, performing 10 iterations of smoothing brings the analysed metrics closer together. The sprint count and distance covered metrics are not greatly influenced in a positive way, but the maximum speed root mean squared error is reduced by 19.59% (approx. 50%, when performing 50 iterations of smoothing).

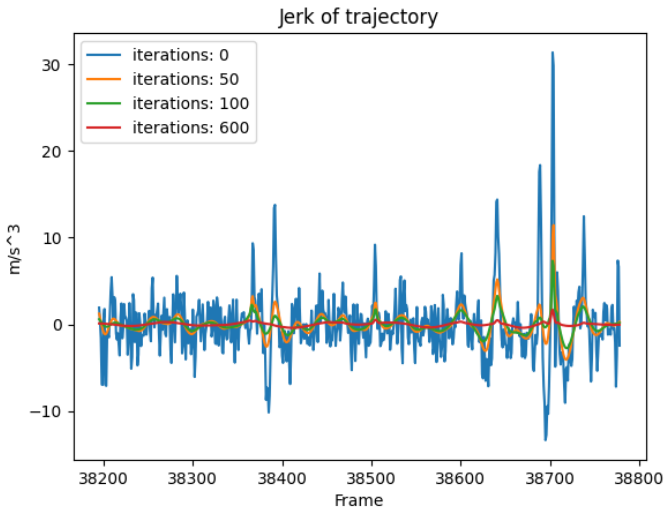
5.2 Speed Time Series Analysis

In this subsection, we will go into the analysis of the speed time series under the influence of the smoothing. For the analysis, the focus will be set on the same trajectory as in Figure 5.1, which was extracted from the movement of a player of the home team using SkillCorner during the game in February 2024. The effects of the smoothing will also be quantified by the change in standard deviation of the 3rd derivative of distance with respect to time, which is known as jerk/jolt 4.3.3.

As a result of moving the points along circle arcs of averaged curvature, the speed time series of the trajectory gets smoothed. This indirect smoothing can be clearly seen when plotting the time series in Figure 5.5(a) for multiple iterations. The jerk time series is strongly affected by CAS (Figure 5.5(b)). The technique seems to have a positive impact on this time series, as the original values are noisy and we manage to reduce their sudden fluctuations. Performing 600 iterations of smoothing on the data would not be a good idea in practice, as we have seen in 5.1.2, but still it is interesting to see how it manifests on these time series. Moreover, we tend to believe, that the smoothing of the trajectories converges after a large number of iterations. This would also imply that the speed time series would converge at some point. It would be important to confirm this in a data driven way, by using a computationally strong machine, as performing CAS thousands of times on multiple trajectories (using the current implementation) takes a lot of runtime.



(a) Speed Time Series of Player 2's trajectory

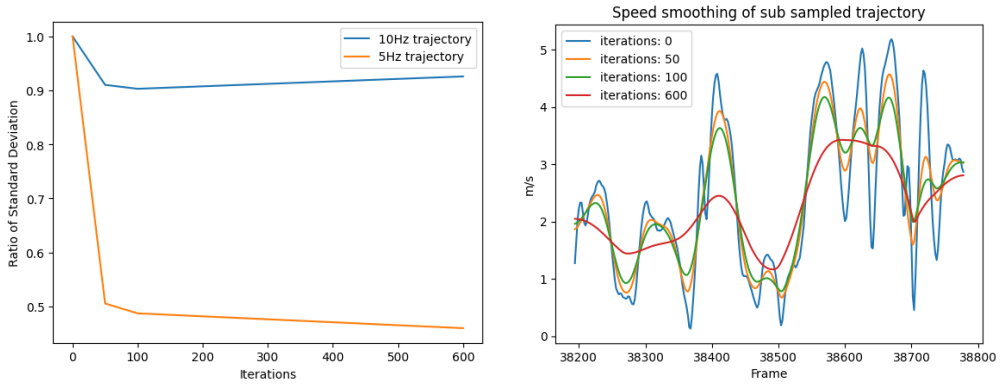


(b) Jerk Time Series smoothing

Assessing the impact that the tracking frequency has on the speed of the smoothing

We start by first sub sampling the same non-smoothed trajectory as in Figure 5.1 at a frequency of $5Hz$, which is half of the $10Hz$ frequency that the SkillCorner model outputs. This results in a similar speed time series that can be seen in Figure 5.5(b). After that, we compute the second derivative of this series to get our jolt values and calculate its standard deviation for each number of iterations.

For each number of iterations of smoothing, we divide its jolt standard deviation to the standard deviation of the non-smoothed trajectory. We define this as the "ratio of standard deviation" and we plot it for both sampling frequencies in Figure 5.5(a). This clearly shows, that lower frequency tracking data needs less iterations of smoothing to be strongly affected in the standard deviation of the jolt. In other words, CAS affects it at a faster rate compared to higher sampling frequencies.

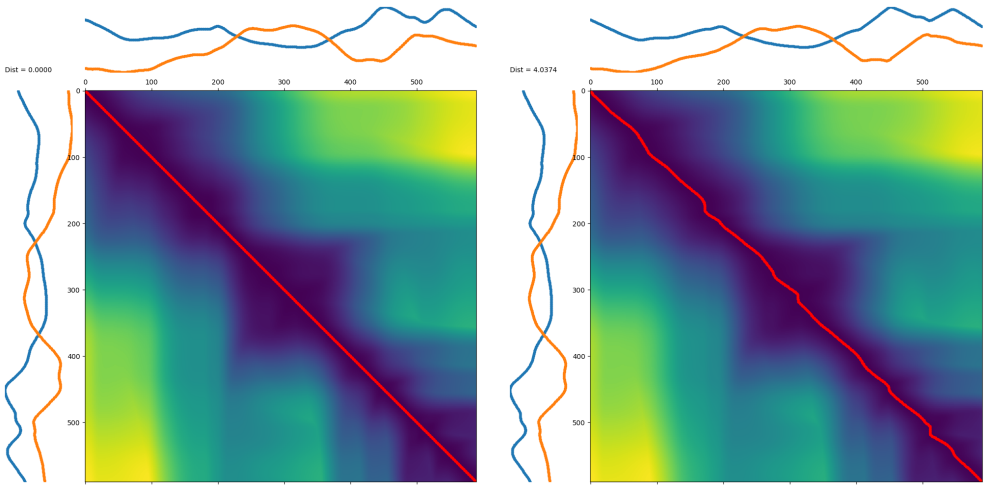


(a) Ratio of Standard Deviation

(b) Speed Time Series of sub sampled trajectory smoothing

5.3 Impact of CAS on trajectory similarity

In this section, we will show how the trajectory similarity is influenced by the CAS. For the following visualizations, we use the exact same positional data as before. In Chapter 4.1, we described how we can look at the tracking data of a player as two separate one dimensional time dependent discrete signals, using the method described in 4.3.2. This separation of the two signals can be seen in the figures below (on the top and left side of the cost matrices) where the signal for the x-Axis is blue and the one for the y-Axis is orange. The DTW distance between the compared trajectories is visible in the top left corner of each plot.



(a) Original trajectory to itself

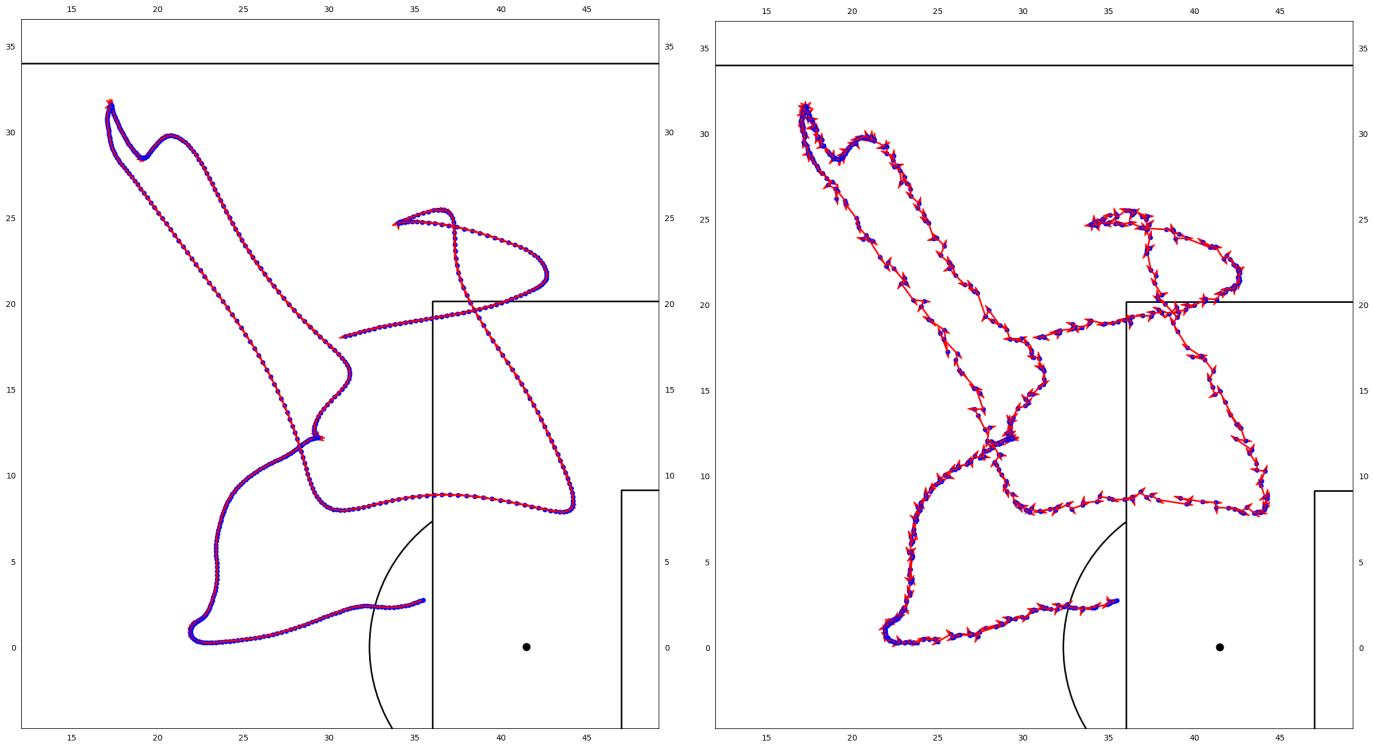
(b) Original trajectory to smoothed (600IT) trajectory

Figure 5.5: DTW cost Matrix and optimal warping path (red)

As we can observe in Figure 5.5, the trajectory has a DTW distance of zero to itself and an optimum warping path that maps each point to itself drawing a perfect line through the cost matrix. This is not the case when computing the DTW distance of the original and smoothed (using 600 iterations) trajectories, which emphasizes the warping of the positional points along the trajectory, which causes the indirect speed smoothing. As we increase the number of iterations, the optimal warping path will deviate more from the diagonal, indicating stronger warping. However, we have noticed that these deviations tend to remain constant after a lot of iterations, further suggesting that the smoothing of the trajectory converges.

5.4 Evaluating technique on noisy data

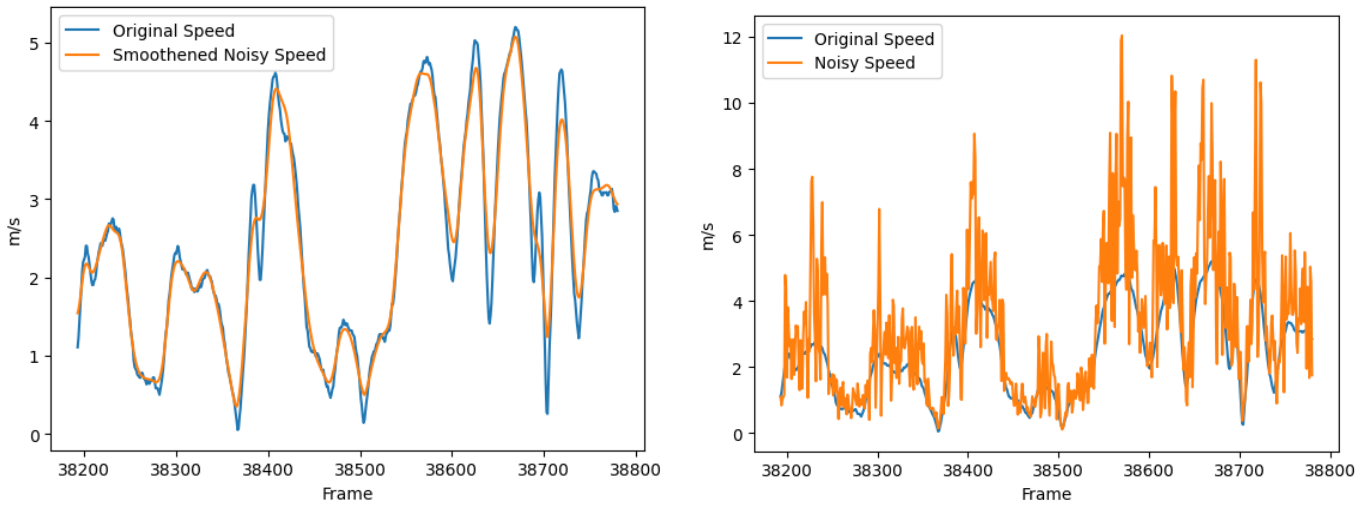
After applying noise ($\sigma_p = 0.2, \sigma_t = 0.4$) to the chosen trajectory (Fig. 5.6(b)), we perform 50 iterations of smoothing on it to see how the general shape (Fig. 5.8) and speed time series (Fig. 5.7) change. As expected, the smoothing manages to maintain the running inertia and closely approximates the original speed values. With around 25 iterations, we manage to get a better fit of the higher peaks in the original speed time series, which might be a sign that we should not smoothen more for this type of noise. The recommended number of iterations to be performed on noisy data is highly depended on the noise level.



(a) Original trajectory

(b) Noisy trajectory $\sigma_p = 0.2$ $\sigma_t = 0.4$

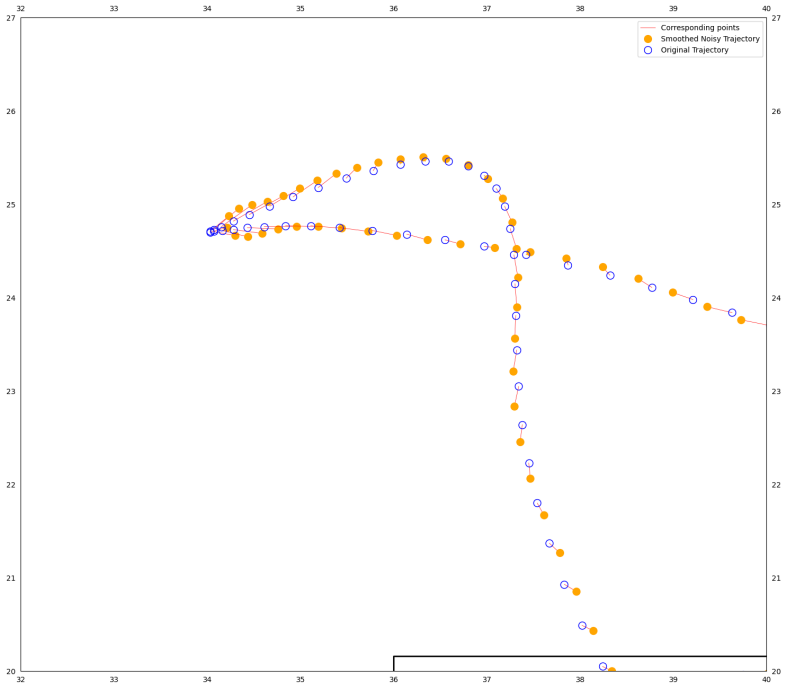
Figure 5.6: Applying noise to a trajectory



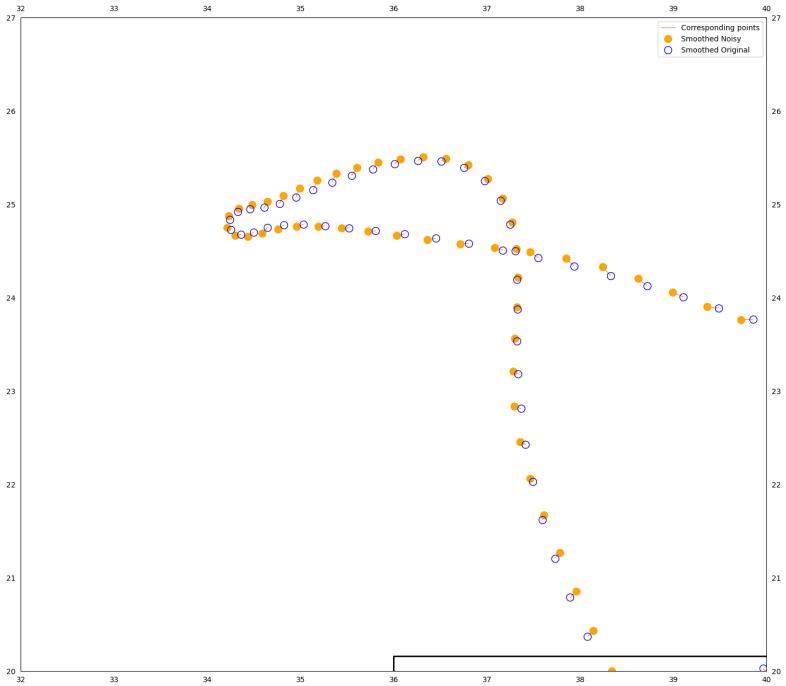
(a) Smoothed (50 iterations) noisy trajectory

(b) Noisy trajectory

Figure 5.7: Speed Time Series of



(a) Original and Smoothed Noisy trajectories



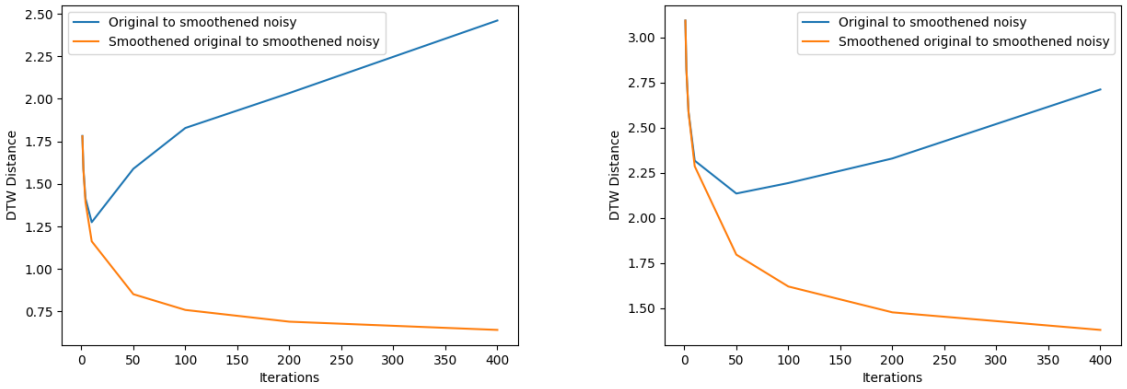
(b) Smoothed Original and Smoothed Noisy trajectories

Figure 5.8: Smoothening noisy trajectory

In Figure 5.8, we zoom into the trajectory to show in more detail how the smoothing works. In Figure 5.9(a), we display both the original trajectory and the smoothed noisy trajectory. The red lines connect the positional points that correspond to the same timestamp. We can clearly see the time warping between these two trajectories, as the red lines are quite long and the general shape of the trajectories quite similar. When we also smooth the original trajectory (Figure 5.9(b)) we can see that the trajectories come even closer together. This was expected, as the smoothing simplifies curves as mentioned in the distance covered section of chapter 5.1.2.

5.4.1 Aggregated results

It is also important to quantify how the general shape of the trajectory is changing while smoothing. For this we perform aggregated experiments over the entire game in February 2024. We take all trajectories from the team players, apply noise to them, after which we smooth them with multiple iterations and compute their DTW distance to the original, ”clean”, trajectory. From Figures 5.9 and 5.10, we can see that when both the original and noisy trajectories get smoothed, they start converging to a more similar trajectory than before (shown by the convergence of the orange function). What is also important to notice, is that the similarity of the noisy trajectory and original trajectory reaches a minimum at a unique number of iterations, after which it diverges due to the over-circularisation of the curves present in the data.



(a) $\sigma_p = 0.1, \sigma_t = 0.2$ (b) $\sigma_p = 0.2, \sigma_t = 0.4$

Figure 5.9: DTW distances of aggregated noise experiments

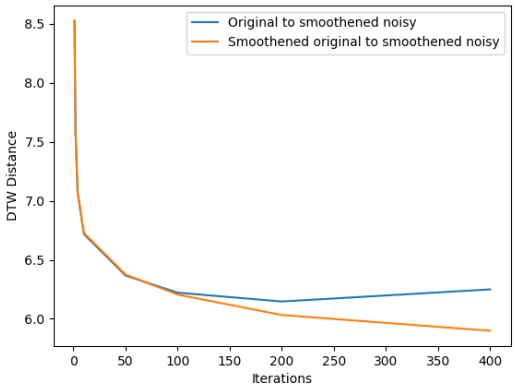


Figure 5.10: $\sigma_p = 0.6, \sigma_t = 1.2$

5.5 Comparison to Savitzky-Golay Filter

In this Chapter, we will compare the smoothing techniques presented in Chapter 3.

We will start by discussing the reduction of the standard deviation in the jerk. To reduce the dimensionality of the S-G filter parameter search, we decide to only consider the use of polynomials of a maximal degree of two. Thus we are left with choosing the window size and amounts of iterations performed. In Fig. 5.11 we plot the computed standard deviations and we can see that both the increase in window size and number of iterations highly impact the jerk values. Moreover, for larger window sizes, we can see that performing just a few iterations makes the deviation converge quickly.

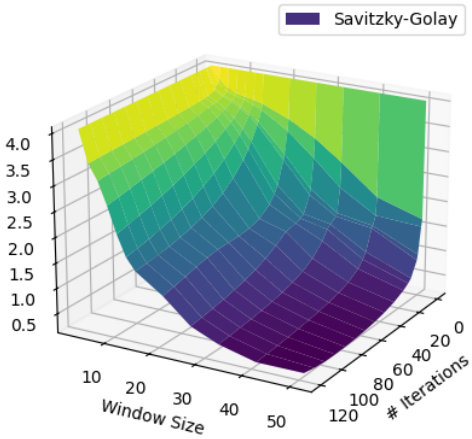
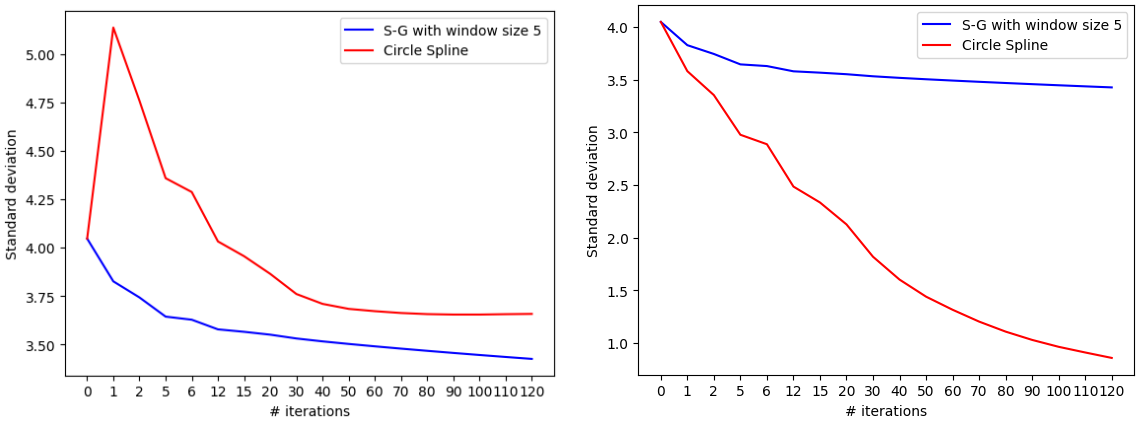


Figure 5.11: Standard deviation of the jerk after S-G filtering with polynomial degree of two

As the implementation of our smoothing technique does not work directly with a specific window size, it is hard to know exactly to which S-G filter to compare it to. What we know is that when performing the circle spline smoothing, we need at least 4 trajectory points to complete an iteration. So intuitively, we consider the S-G filter using a window size of 5.



(a) Circle arc speed approximation

(b) Euclidean distance speed approximation

Figure 5.12: Standard deviation of jerk with _ for the circle spline smoothing

As we can see in Figure 5.12(a), there is a jump in the jerk standard deviation at the first iteration of the circle spline smoothing. This is caused by the switch from the euclidean distance speed approximation for non-smoothed points [18] to the circle arc speed approximation from Chapter 3.2.2. In Fig. 5.12(b) we keep using the euclidean distance speed approximation [18] and we can clearly see a much stronger monotone reduction of the standard deviation for the circle spline smoothing.

Next, we will analyze the difference in DTW distance of the filtered trajectories to the original trajectory.

As can be seen from Fig. 5.13, the circle spline smoothing changes the trajectory much more aggresively than the Savitzky-Golay filter with a window size of 5. This has to do with the oversimplification of the changes in direction that we have seen in Chapter 5.1.2. Such oversimplifications could be avoided by not smoothing tracking points which correspond to a change of direction at low speeds (0 - $2km/h$). As a player has a lower inertia when moving at these low speeds, he will most likely not move in a circular fashion.

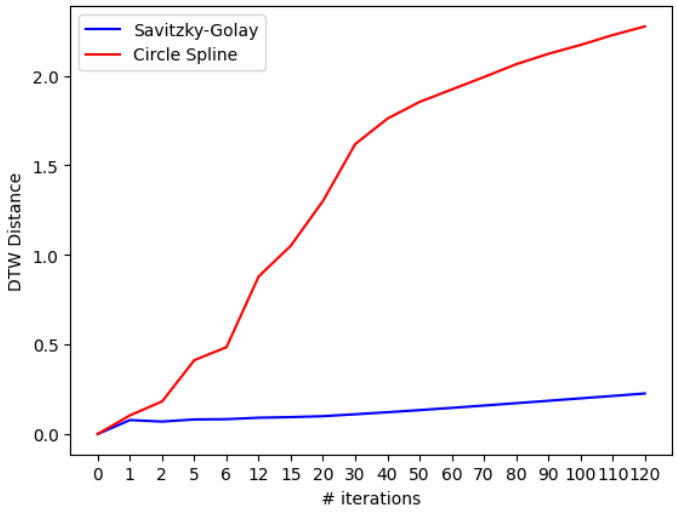


Figure 5.13: DTW distance of filtered trajectories to the original trajectory

Chapter 6

Conclusion

Having performed our investigation of the novel smoothing method, we reach the end of this thesis. We have seen how the smoothing technique affects the data of multiple tracking systems and how it tends to change the physical metrics and the speed time series of the players. Moreover, we have seen that for the positional data extracted by two different tracking systems for an overlapping soccer match, we get more consistent physical metrics after performing 10 iterations of CAS. For further backing of this finding, we would have to apply the methods presented to more matches. Specific artifacts in the SkillCorner tracking data (unrealistically high speeds sustained for a long period of time), lead us to believe that the data might have already been filtered somehow by the provider. This should be avoided in the future, as CAS would perform best on raw positional data.

CAS has shown to reduce high values and strong fluctuations of jerk in the speed approximations of players. It is also very robust against noise, allowing for accurate speed time series reconstructions.

6.1 Future Work

6.1.1 Making the CAS implementation faster

A significant bottleneck for our analysis was the large amount of runtime needed for performing the smoothing on a vast amount of trajectories. For the calculation of our aggregated experiments, we used a multiprocessing approach which was enough for the given data, but if we were to consider many more soccer matches, we would need a fundamentally different implementation of CAS. It is interesting to see how the regression problem in the Saviztky-Golay paper [11] has been reduced to a simple static filter that is convolved directly with the signal without a need for any extra computations. This kind of reduction would be significant for the implementation of

CAS.

6.1.2 CAS generalisation

A generalisation of the CAS for the three dimensional domain could be used to smoothen ball trajectory data. Some adjustments would be needed for this use case, but the principle would remain the same.
Another thing that could be parameterized, is the selection of the points in the trajectory sequence that are being used for averaging the curvature of the circular arcs. For example, we could perform smoothing with a larger ”window size” i.e. considering a sparser trajectory and mapping the positions that are not included, on the averaged circular arcs computed by the smoothing.

6.1.3 Tracking data compression

As we are moving tracking points of player trajectories onto circle splines, we might be able to compress the positional data. This could allow us to perform the needed smoothing while also reducing the total storage cost.

Bibliography

[1] Skillcorner glossaries, https://skillcorner.crunch.help/en/glossaries.

[2] L.S. Davis D.M. Gavrila. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. *IEEE International Workshop on Automatic Face- and Gesture-Recognition*, 1995.

[3] David Eager, Ann-Marie Pendrill, and Nina Reistad. Beyond velocity and acceleration: jerk, snap and higher derivatives. *European Journal of Physics*, 37(6):065008, October 2016.

[4] Daniel Linke, Daniel Link, and Martin Lames. Football-specific validity of tracab's optical video tracking systems. *PLOS ONE*, 15(3):1–17, 03 2020.

[5] Daniel Memmert and Dominik Raabe. Data analytics in football: Positional data collection, modelling and analysis. 2018.

[6] Meinard Müller. *Dynamic Time Warping*, pages 69–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[7] Banoth Thulasya Naik, Mohammad Farukh Hashmi, and Neeraj Dhanraj Bokde. A comprehensive review of computer vision in sports: Open issues, future trends and research directions. *Applied Sciences*, 12(9), 2022.

[8] Jürgen Perl and Daniel Memmert. A pilot study on offensive success in soccer based on space and ball control–key performance indicators and key to understand game dynamics. *International Journal of Computer Science in Sport*, 16(1):65–75, 2017.

[9] A. Piyush Shanker and A.N. Rajagopalan. Off-line signature verification using dtw. *Pattern Recognition Letters*, 28(12):1407–1414, 2007.

[10] DSG Pollock et al. Smoothing with cubic splines, 1993.

[11] A. Savitzky and M. J. E. Golay. Soothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36, 1964.

[12] Ronald W. Schafer. What is a savitzky-golay filter? [lecture notes]. *IEEE Signal Processing Magazine*, 28(4):111–117, 2011.

[13] Marc Schmid and Martin Lames. Correction of systematic errors in electronic performance and tracking systems. *Sports Engineering*, 26(1):30, 2023.

[14] Lorena Torres-Ronda, Emma Beanland, Sarah Whitehead, Alice Sweeting, and Jo Clubb. Tracking systems in team sports: A narrative review of applications of the data and sport specific analysis. *Sports Medicine - Open*, 8(1):15, 2022.

[15] Prodromos Tsinaslanidis, Antonis Alexandridis, Achilleas Zapranis, and E. Livanis. Dynamic time warping as a similarity measure: Applications in finance. In *Hellenic Finance and Accounting Association*, December 2014.

[16] Vyacheslav Tuzlukov. *Signal processing noise*. CRC Press, 2018.

[17] Ferran Vidal-Codina, Nicolas Evans, Bahaeddine El Fakir, and Johsan Billingham. Automatic event detection in football using tracking data. *Sports Engineering*, 25(1):18, 2022.

[18] Lucas Y Wu and Tim B Swartz. The calculation of player speed from tracking data. *International Journal of Sports Science & Coaching*, 18(2):516–522, 2023.

[19] Lichen Zhang, Mohsen Diraneyya, Juhyeong Ryu, Carl Haas, and Eihab Abdel-Rahman. Jerk as an indicator of physical exertion and fatigue. *Automation in Construction*, 104, 05 2019.