

TOWARDS AUTOMATED FOOTBALL ANALYSIS: ALGORITHMS AND DATA STRUCTURES

Joachim Gudmundsson^a, Thomas Wolle^{a,b}

^a*NICTA Sydney*

Locked Bag 9013, Alexandria NSW 1435, Australia

^b*Corresponding author: thomas.wolle@nicta.com.au*

Abstract

Analysing a football match is without doubt an important task for coaches, clubs and players; and with current technologies more and more match data is collected. For instance, many companies offer the ability to track the position of each player and the ball with high accuracy and high resolution. Analysing this position data can be very useful. Nowadays, some companies offer products that include simple analyses, such as statistics and basic queries. It is, however, a non-trivial task to perform a more advanced analysis. In our research, we assume that we are given only the position data of all players and the ball with high accuracy and high resolution. In this paper we present two tools.

Our first tool automatically extract (from the position data) a list of certain events that happened during the football match. These events include kick-offs, corner kicks, passes etc. In experiments we could observe that our method is very fast and reaches a high level of correctness. We also learned that errors in the event detection are hard to avoid completely, when looking at only the position data.

Our second tool aims at analysing a single player's trajectory (the sequence of all positions during a game). More precisely, we look for movements of a player that are repeated often (so called subtrajectory clusters). For example a left wing attacker runs from the centre-line along the left side of the field towards the opponent's goal. And this attacker might repeat this type of movement very often during a game (or perhaps multiple games). Our goal is to detect this kind of frequent movements automatically. Experiments showed that this method is computationally expensive. Nevertheless, it reliably identifies subtrajectory clusters, which then could be used for further analysis.

Keywords: team sport, position data, trajectory analysis, event detection, clustering

1. INTRODUCTION

Recent years have witnessed massive improvements in tracking technologies that allow for recording the positions of moving entities with high spatial and temporal resolution and also with high accuracy. As a consequence such systems are used more and more often in many domains, including sports, urban planning and development, defence, location based services and animal research. In the world of football (also known as soccer), there are already many companies that provide the capabilities to track the movement of all football players during a match, and they also provide tools to analyse this data to a certain degree.

In our project we focus on applications in team sports and in particular on football. We look at these applications mainly from a computer science point of view. Assuming we are given the trajectories, i.e. the data describing the movement, of all players and the ball, we aim at a more sophisticated analysis that takes the interaction and relationships of different trajectories into account, recognises formations and perhaps even more general patterns and trends. Over the past few years we have developed several algorithms and tools for analysing trajectories. Two of them are presented in this paper.

Our first tool, introduced in Section 3, is algorithm-

mically rather simple. Nonetheless, we consider it as an important tool that is necessary for a more advanced automated analysis. From the trajectories of all players and the ball, it automatically extract a list of basic events that happened during the football match; detected events include kick-offs, corner kicks, passes etc. Experimental results show that our method is very fast and reaches a high level of correctness for many types of events. However, some types of events seem to be hard to detect automatically when looking only at the position data.

Our second tool, presented in Section 4, aims at analysing a single player’s trajectory. However, it can be easily extended to take multiple trajectories as input. In the input trajectory we look for subtrajectory clusters, which are movements of a player that are repeated often. This topic is motivated by questions such as “How is the ball transported from the defense region to the attack region?” We present a prototype where a user can specify certain parameters of the cluster, and then the prototype will reliably detect the clusters according to the chosen parameters. The current version of the prototype might not be suited as an interactive tool, because the time to answer a query can be rather long.

We believe our algorithms and implementations belong to the still immature research area that aims at automated football analysis. Despite the young age of this area, it becomes more and more popular and important as can be seen by the increasing amount of related work. For example, Kang, Hwang and Li (2006) propose a method to quantitatively evaluate the performance of football players. Their approach is based on four different measures that include different regions for each player and the kicks the players perform. Another approach that is also based on regions is presented by Fujimura and Sugihara (2005). Their regions are based on the generalised Voronoi diagrams. Grunz, Memmert and Perl (2009) address the analysis of actions in a football match. Their actions are coarser than our basic events, and the used techniques for the detection are very different.

2. PRELIMINARIES

The position of a moving object can be described by the spatial coordinates x , y and possibly also z at a certain time t . Together, these values form the

sample (t, x, y, z) . A sequence of such samples, ordered with respect to time, is called a *trajectory*. A trajectory describes the movement of an object.

The data that we used in our experiments is data from a real football match. It was anonymised and kindly provided to us by ProZone. It includes the trajectories of all the players. These trajectories have a spatial resolution of a decimeter and a temporal resolution of at least ten samples per second. However, we do not know the accuracy of the data. For the analysis we aim to do we also need the trajectory of the ball. Unfortunately, it is not included. It does, however, include a list of annotations of the match, which is a list of events such as “touch” and “pass” – very similar to the events we want to detect. These annotations were created manually by people watching the match (and/or a video thereof). From these annotations, we re-construct the ball’s trajectory by using the time and spatial coordinates of each event to create samples of the ball’s trajectory. The resolution of the annotations are only one meter and one second, so the obtained ball trajectory is only a rough estimate of the ball’s movement. We believe that the results in Section 3 can be improved considerably if the ball trajectory would be given with higher accuracy.

Both our algorithms and prototypes have been implemented in Java. Our experiments were performed on an off-the-shelf PC with an Intel dual core processor running at 2.33 GHz and 2GB of main memory.

3. BASIC EVENT DETECTION

3.1. METHODS

For computing basic events of a football match, we need both the ball’s and the players’ trajectories. Here, we briefly describe how our algorithm for detecting events works.

Our event detection works on different levels of events. The bottom level is the physical event level. These are events that can be detected without any knowledge of the football rules (we do, however, require knowledge of the dimensions and lines on the football pitch). Bottom level events are for instance “ball-out”, “ball-in” and “touch” events. The first two occur when the ball moves out of (or: back into) the pitch. “Ball-in” events can be refined into “throw-in”, “corner kick” etc., depending on



Figure 1: Screenshot of the programme to animate a match. The detected events are displayed in the area below the pitch.

where the “ball-in” event takes place. A “touch” event is detected when the ball changes its speed and/or direction. For these events we only need the ball’s trajectory.

Now, if we also take the player’s trajectories into account, we can refine the events. For instance, the player that is closest to the ball when a “touch” event happens is assumed to have touched the ball (although there are exceptions). Or the player that is closest to the ball, when a “throw-in” happens is assumed to have performed the throw-in. As a result we obtained a list of more advanced events.

These more advanced events can be refined in a second round. For instance, two consecutive “touch” events that were performed by different players from the same team, constitute a “pass” event. In a similar way we can detect “possession”, “pass interception”, “shot on goal”, etc.

In yet another refinement round, we arrive at the highest event level, which includes events such as “free-kick”, “substitution”, “offside”, “foul”, “red-card”, etc.

3.2. EXPERIMENTAL RESULTS

Figure 1 shows the main screen of the event-detecting prototype. It includes a football pitch, where the match can be viewed as an animation. The area below this pitch is the area where the events are displayed synchronously linked to the animation. An example snippet of the list of events is shown in Figure 2.

```

34:51 |Intercept      |( 37.6, -23.9,  0.0)| Player04 | Player08
34:51 |Touch           |( 38.9, -24.6,  0.0)| Player08
34:53 |Ball Out        |( 59.0, -26.0,  0.0)|
35:13 |Corner Cross    |( 57.2, -29.1,  0.0)| Player23
35:13 |Touch           |( 57.2, -29.1,  0.0)| Player23
35:13 |Shot            |( 57.2, -29.1,  0.0)| Player23
35:15 |Touch           |( 54.3, -5.3,   0.0)| Player09
35:15 |Goalkeeper Catch|( 54.3, -5.3,   0.0)| Player09
35:15 |Pass            |( 54.3, -5.3,   0.0)| Player09 | 14.1 m/s
35:16 |Receive         |( 36.3, -2.0,   0.0)| Player17

```

Figure 2: An extract of the list of detected events, showing for each event: time, type, coordinates and involved players etc.

Event type	<i>falsePositive</i>	<i>falseNegative</i>	F_1 -score
touch	7	3	0.979
pass	2	2	0.955
intercept	1	1	0.960
ball-out	1	4	0.935
throw in	0	2	0.963
corner kick	0	1	0.968
goal kick	1	2	0.909
kick off	0	1	0.960
shot	1	3	0.875
goalkeeper catch	0	2	0.941
goal	0	1	0.960
foul	3	6	0.823
offside	5	3	0.556
free kick	0	3	0.953

Table 1: Summary of some statistical results for some of the detected events. The higher the F_1 -score the better.

Computing all the events for an entire match takes only a couple of seconds. Hence the running time does not seem to be an obstacle in practice, and therefore we focus more on the study of the correctness of the detected events.

As mentioned above, together with the data that we used for our experiments, we were given a list of annotations. To estimate the correctness of our event detection, we compare our list of events to these annotations, where we consider the annotations as 100% correct (even though they might not be).

The measure that we use to report the level of correctness is the F_1 -score, given as:

$$\frac{2 \cdot \text{truePositive}}{2 \cdot \text{truePositive} + \text{falseNegative} + \text{falsePositive}}$$

The values of *truePositive*, *falseNegative* and *falsePositive* were simply counted when comparing the list of detected events with the annotations.

Table 1 shows the values of *falsePositive*, *falseNegative* and the resulting F_1 -score for some of the detected events. Note that some events are omitted from this table as they did not occur in the match, such as “penalty kick” or “red card”. Also, some events are not considered because they are

not possible to detect without additional information, for example “yellow card” events.

From the numbers in Table 1, we can conclude that automated basic event detection is possible. Some events, especially the bottom level events can be detected with very high accuracy. Higher level events such as “foul” and “offside” are not reliably detected.

There are several factors that impact the correctness of our results. We could observe that the annotations themselves contain errors, which is also true for some trajectories. But more importantly, the ball’s trajectory had to be re-constructed from the annotations. This might have huge positive and/or negative effects on the detection of certain events. All in all, with the current data it is very hard, perhaps even impossible, to estimate the accuracy of our event detection methods. As a consequence, we did not put more efforts into fine tuning our methods, as this would possibly increase the accuracy, but only for the given data with the lack of a real ball’s trajectory.

4. SUBTRAJECTORY CLUSTERING

4.1. METHODS

The techniques proposed by Buchin, Buchin, Gudmundsson, Löffler and Luo (2008) are the basis of our tools. In the following, we briefly review those techniques and then, in Section 4.2, we will consider the experimental results.

The similarity between two trajectories can be defined in different ways, for example using the Longest Common Subsequence model (Vlachos, Gunopulos and Kollios, 2002), a combination of parallel distance, perpendicular distance and angle distance (Lee, Han and Whang, 2007) and the average Euclidean distances between paths (Nanni and Pedreschi, 2006). We will use the Fréchet distance, which is a distance measure for continuous shapes such as curves and surfaces, and is defined using reparameterisations of the shapes. Because it takes the continuity of the shapes into account it is generally regarded as being a more appropriate distance measure than the Hausdorff distance for curves (Alt, Knauer and Wenk, 2004).

The Fréchet distance can be intuitively explained in the following way: Imagine a person walks their

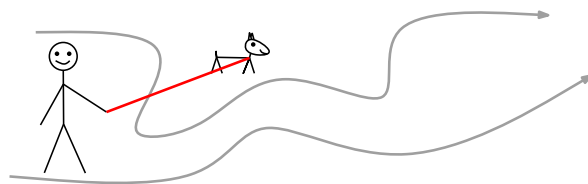


Figure 3: Illustrating the leash length between a person and their dog walking along their trajectories.

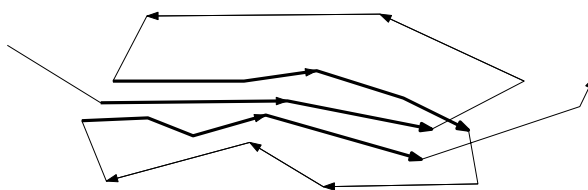


Figure 4: A subtrajectory cluster (indicated fat) of a trajectory.

dog on a leash (see Figure 3). The person will follow a certain trajectory or path T_p , while the dog follows a different path T_d . The Fréchet distance between T_p and T_d is the smallest length of a leash that allows the person and the dog to walk on their paths, where the person and the dog can change their speed or even pause, but they are not allowed to backtrack.

The Fréchet distance comes in two flavours: continuous and discrete. Intuitively, in the continuous version, the person and their dog walk on their paths in a continuous movement, while in the discrete version, they “jump” from one vertex to the next vertex of the path. Note that the continuous version can be approximated by the discrete version when using paths with many vertices, i.e. having data with high resolution. We chose the *discrete Fréchet distance*, because the corresponding algorithms are easier to implement (Buchin, Buchin, Gudmundsson, Löffler and Luo, 2008).

During a match, a player might move along certain paths multiple times. Hence, when given the trajectory T of that player, certain subtrajectories of T might form a subtrajectory cluster (see Figure 4). We follow Buchin, Buchin, Gudmundsson, Löffler and Luo (2008) who define a subtrajectory cluster depending on three parameters: m , ℓ and d . We say that a *subtrajectory cluster* consists of m non-overlapping subtrajectories T_1, \dots, T_m of T . At least one subtrajectory has length ℓ , and the distance between the subtrajectories is at most d .

Computing subtrajectory clusters exactly turns out

to be a hard problem (Buchin, Buchin, Gudmundsson, Löffler and Luo, 2008). Deciding whether a trajectory T contains a subtrajectory cluster with specified parameters m , ℓ and d is NP-hard. The problem to maximise the number of subtrajectories or to maximise the length of the subtrajectories (while the other parameters are fixed), is also NP-hard, even when computing an approximation where m or ℓ are approximated within certain factors and d is approximated within < 2 . (Intuitively, for an NP-hard problem, there is no known efficient algorithm to solve it.) That is why we look at approximation algorithms where d is approximated within a factor of ≥ 2 .

The main result by Buchin, Buchin, Gudmundsson, Löffler and Luo (2008) that we will be using is the following: Given a trajectory T , there is an algorithm to compute, under the discrete Fréchet distance, a subtrajectory cluster of maximum length, where the distance d is approximated by a factor of 2 (i.e. we allow the subtrajectories to have a distance twice as large as specified by the parameter d). This algorithm runs in $O(n^2 + nm\ell)$ time and uses $O(n\ell)$ space, where n denotes the number of vertices of T , ℓ denotes the maximum number of vertices of a subtrajectory in the subtrajectory cluster, and m denotes the number of subtrajectories in this cluster.

The implemented prototype can be configured to only report subtrajectory clusters according to the following parameters: *distance* (this is the maximum allowed Fréchet distance between subtrajectories), *minimum cluster size*, *duration* and *length* (these are thresholds to avoid reporting very small and meaningless clusters).

4.2. EXPERIMENTAL RESULTS

By running experiments on the subtrajectory clustering, we would like to find out how well the clustering works, i.e. how much useful information is found and also how fast the clustering works, i.e. do the run times allow for an interactive tool, where an analyst specifies the parameters of a cluster and the tool will “quickly” find the corresponding clusters.

4.2.1. Usefulness

One way to evaluate the usefulness of the clustering is by examining the results visually, i.e. drawing the clusters together with the trajectory into an image



Figure 5: Screenshots showing the trajectory and a subtrajectory cluster of a left-wing player moving forward.



Figure 6: Screenshots showing the trajectory and a subtrajectory cluster of a right-wing player moving backward.

of a football pitch. In Figures 5-8, we provide example screenshots for the ball’s trajectory and the trajectory of two players. Each screenshot shows the entire trajectory of one half of the match. Also, in each screenshot, one cluster is highlighted in red and yellow. (The yellow subtrajectory is the *representative subtrajectory* for this cluster (Buchin, Buchin, Gudmundsson, Löffler and Luo, 2008).) For instance, in Figure 5, the trajectory of a left wing player (Player1) and the cluster indicates that several (at least six) of his attacking runs in the first half of the game started near the middle of the left half of the centre line, describing a long arc going forward and further to the left of the field and then making a sharp turn back towards the middle of the field.

From the screenshots we can conclude that the subtrajectory clustering reliably finds the clusters ac-



Figure 7: Screenshots showing the ball’s trajectory and a sub-trajectory cluster of goal kicks.

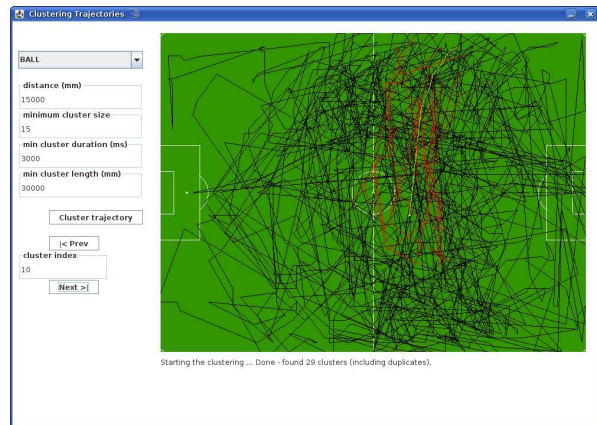


Figure 8: Screenshots showing the ball’s trajectory and a sub-trajectory cluster of ball movements from top to bottom.

ording to the set parameters. Therefore, from a computer scientist’s point of view, this confirms that the subtrajectory clustering algorithms work well. However, longer and bigger (in terms of the number of subtrajectories) clusters would probably be of more interest to domain experts.

One way to give clusters more meaning is to consider the trajectories of multiple matches, and then to look for clusters taking all these trajectories into account. In doing so, we are likely to find longer and larger clusters which can be used to identify interesting emerging patterns. However, we are also likely to face much longer running times (see Section 4.2.2). Another way to increase the meaning of the clustering is to not only look at purely geometry-based clustering, as is done in this study. Instead, one could consider a combination of geometry-based clustering and event-based clustering.

4.2.2. Running times

From the way the algorithm works and from the way it has been implemented we would expect certain parameters to have no, or only marginal, impact on the running time. These parameters are the *minimum cluster size*, *duration* and *length*, which could be confirmed in preliminary experiments. That is why we only report on the running time behaviour depending on the distance d and the number n of vertices in the trajectory, see Figure 9 and 10.

Interestingly, the chosen Fréchet distance d has an impact on the running time (see Figure 9), even

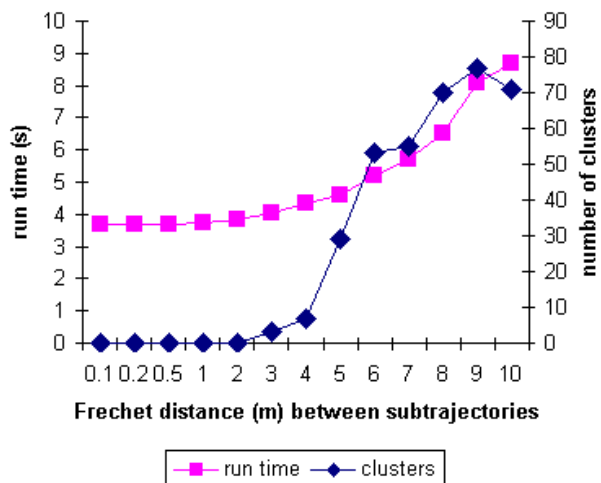


Figure 9: Chart of the running time and the number of clusters depending on the chosen Fréchet distance.

though this is not reflected by the results from the theoretical worst-case analysis (see Section 4.1). This dependency can be explained by the way the algorithm works. It iterates over the vertices of the trajectory and in each iteration it keeps track of all other vertices that have a distance of at most d . For larger values of d , this strategy inevitably results in larger running times, as more and more vertices need to be considered. A bigger impact on the running time is caused by the size of the trajectory (see Figure 10). To achieve different sizes, we simplified the trajectory to various degrees, by removing vertices that are less crucial for the overall shape of the trajectory. The theoretical analysis gives us a quadratic dependency, which we could also observe in our experiments.

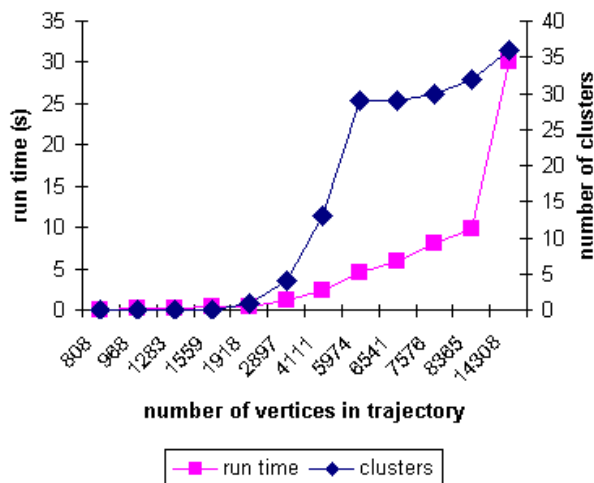


Figure 10: Chart of the running time and the number of clusters depending on the size of the trajectory.

We also see a perhaps surprising dependency of the number of clusters that are found. Clearly, a small Fréchet distance makes it harder for clusters to exist. A large Fréchet distance, on the other hand, can give rise to long clusters. As clusters may not overlap, such long clusters might hence prevent subsequent clusters to exist. Furthermore, the fact that, for small trajectory sizes, many clusters will not be found, is caused by our choice to use the discrete Fréchet distance. In our version, all subtrajectories of a cluster must start and end with vertices that are close to each other. Because of the simplification, we cut out vertices making it less likely to identify clusters for small size of the trajectory.

From the graphs in Figure 9 and 10, we can conclude that finding clusters could potentially be done in an interactive tool for certain parameters and small input sizes. However, the current implementation does not seem to be suitable for interactive use for more interesting settings, e.g. where we have large input or where we allow a large Fréchet distance.

One should note, however, that the current implementation was not optimised for speed. It should be mainly seen as a prototype for a feasibility study that reflects at least certain asymptotic running time behaviour. Also, choosing different or more sophisticated algorithms might result in very different running times, and has the potential to offer drastic speedups.

5. CONCLUSIONS

Our presented algorithms and prototypes are first steps towards completely automated football analysis. Of course, it is not clear that a completely automated analysis will ever be reached, but the more tools we can provide to coaches, the better their team might perform.

We have seen that basic event detection works well and efficiently, and that the clustering also works well, but can be expensive and might have only limited use for coaches. We will continue to work on similar problems and to make our current tools more accurate. Also increasing the speed of the clustering algorithm is a topic for further study.

The area of football analysis is a mix of many different domains. If we want to fully benefit from this mix and contribute to this area, we need a stronger and more effective dialog between experts in the different domains.

Acknowledgements

We wish to thank our students Susan Howlett, John Jiang, Sidong Liu, Cahya Ong, David Rizutto and Merlin Zhang who worked with us on this project during the past few years.

References

- Alt, H., Knauer, C., & Wenk, C. (2004) Comparison of distance measures for planar curves. *Algorithmica*, 38(2):45–58.
- Buchin, K., Buchin, M., Gudmundsson, J., Löffler, M., & Luo, J. (2008). Detecting commuting patterns by clustering subtrajectories. *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC 2008)*, pages 644–655, Berlin, Heidelberg. Springer-Verlag. To appear in *International Journal on Computational Geometry and Applications*.
- Fujimura, A., & Sugihara, K. (2005). Geometric analysis and quantitative evaluation of sport teamwork. *Systems and Computers in Japan*, 36(6):49–58.
- Grunz, A., Memmert, D., & Perl, J. (2009). Analysis and simulation of actions in games by means of special self-organizing maps. *International Journal of Computer Science in Sport*, 8(1):22–36.
- Kang, C.-H., Hwang J.-R., & Li, K.-J. (2006). Trajectory analysis for soccer players. *ICDMW '06: Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 377–381, Washington, DC, USA. IEEE Computer Society.

- Lee, J.-G., Han, J., & Whang, K.Y. (2007). Trajectory clustering: a partition-and-group framework. *Proc. ACM SIGMOD international Conference on Management of Data*, pages 593–604.
- Nanni, M., & Pedreschi, D. (2006). Time-focused density-based clustering of trajectories of moving objects. *Journal of Intelligent Information Systems*, 27(3):267–289.
- Vlachos, M., Gunopulos, D., & Kollios, G. (2002). Discovering similar multidimensional trajectories. *Proc. 18th International Conference on Data Engineering (ICDE)*, pages 673–684.