Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Final Report

2IMV10 Visual Computing Project

Jeroen Brouns - 0856180
Yuntao Li - 0910663

Eindhoven, Januari 2018

Abstract

This paper describes the findings and development process of the TU/e Visual Computation course. This paper presents the implementation of a visualization-system that is able to use various datasets of XYZ-positional body-sensors and corresponding videos from several professional soccer games[1]. The dataset includes anonymized player-data among which are time-stamped field positions, speed, acceleration and directional vectors. The datasets can be used to provide insights in player's performance and spectator statistics. To accomplish this, computer-vision algorithms that implement techniques such as feature extraction, object-tracking, background subtraction a.s. can be utilized alongside the video-feed which enables the visualizations to be tested against a ground truth. Furthermore an evaluation, discussion of results and future expectations will be given in this paper.

Contents

Chapter 1

Introduction

With football being the most popular type of sport in the world (an estimated 3.5 billion fans worldwide of which one out of five currently practices or used to practice football [2]), many industry-related companies revolve around football with their core business. The FIFA World Cup in 2006 measured an astonishing 30 billion (accumulated) views. The top earning clubs (Manchester United, Real Madrid) have revenues that approximate 650 million euro. Clubs like these have a demand for having more insight in player's performance (so they can beat the competition) and providing real-time player statistics to the match-spectators. For the football industry, video capturing and analytic systems are becoming must-have tools. Currently, many football clubs use video-enabled object-tracking to gain statistical insight, however these technology are just not quite there yet. The complexity of these video-systems makes it difficult to gain high accuracy results, since they are prone to environmental influences such as lighting and weather conditions, which are hard to control.

However, a state-of-the-art 3D-positional sensor system could provide results that are much more accurate to their ground truth. This paper uses various datasets of body-sensor traces and corresponding videos from recorded football match using a Real-Time Locating System deployed on a football field of the Nuremberg Stadium in Germany. This paper introduces a visualization application to automate the process of benchmarking football-matches and football-players by analyzing this sensor data[1]. Furthermore, the application is able to visualize generic logged sensor-data onto a three-dimensional rendered playing field, providing the analyzers with visual aid to their benchmarks. The goal of the system is to reduce analysis time and resources: current solutions require the involvement of human beings (i.e. for ball possession statistics, an operator manually presses a switch upon ball-possession change). We will discuss the development, implementation and mathematical challenges that were subjected to us in the development phase of a visualization-system for this high-end dataset.

Chapter 2

Requirements and Project-scope

By means of the MoSCoW requirements priorization technique, we provide a list of requirements that define the project-scope.

Table 2.1: **List of MoSCoW Project Requirements**

(M) Must have	3-dimensional plotting of the positional data in the data-sets
(M) Must have	Individual player statistics for acceleration and speed
(M) Must have	Menu-interface to interact with the visualizations
(M) Must have	Play-button and pause-button to start/stop the visualization
(S) Should have	Video-feed playback for a certain time-stamp alongside the visualizations
(S) Should have	Noise-reduction algorithm that detects anomalies in input-sensor data
(S) Should have	Preprocessing pipe-line that converts measurement rate in a more usable format
(S) Should have	Heatmap of all player's positions within a specific time-interval
(C) Could have	Heatmap of an individual player's position within a specific time-interval
(C) Could have	Additional statistics: ball possession
(C) Could have	Additional statistics: Fastest shot
(C) Could have	Additional statistics: Player moral and mental state
(C) Could have	Additional statistics: (Dangerous) attacks
(C) Could have	Additional statistics: Corner kicks
(C) Could have	Additional statistics: Shots on goal
(C) Could have	Support for multiple data-source formats
(C) Could have	Additional statistics: Shots on goal
(W) Won't have	Integrated preprocessing pipeline

Chapter 3

Dataset

We use a data-set that was shared by DEBS in 2013. DEBS annually provides a series of challenges which seek to provide a common ground and evaluation criteria for a competition aimed at both research and industrial event-based systems. The goal of the Grand Challenge competition is to implement a solution to a problem provided by the Grand Challenge organizers The DEBS Grand Challenge series provides problems which are relevant to the industry at large. DEBS Grand Challenge problems allow for evaluation of event based systems using real-life data and queries.

DEBS 2013 is about the applicability of event-based systems to provide real-time complex analytics over high velocity sensor data along the example of analyzing a soccer game. However, we did find similar data-sets that were recorded using the same kind of XYZ-sensoring system. Preferably we would design the visualization-application in such a way that it can read various datafiles from different football matches, not only the dataset provided by the DEBS 2013 challenge.

As already mentioned, the data originates from sensors in a positional tracking system. Per football player there is a sensor attached to each foot. This sensor measures at a rate of 200Hz. A sensor was also placed in the ball, which measures at a rate of 2000Hz. The goal keeper is equipped with two additional sensors, one at each hand, which just like the feet-sensors, measure at a rate of 200Hz.

The total data rate reaches roughly 15.000 position events per second. Every position event describes position of a given sensor in a three-dimensional coordinate system.

The sensor-event logging data file is encoded on a linebase. Every new line defines a unique and separate logging event. An event encoding adheres to the following format:

$$sid, ts, x, y, z, |v|, |a|, vx, vy, vz, ax, ay, az$$

Table 3.1: Variable elaboration

Variable	**Description**	**Unit/type**
sid	Sensor id	Integer
ts	Timestamp	Picosecond
x,y,z	3-dimensional position of event	mm
|V|	Absolute speed	m/s
vx, vy, vz	Direction of the event	Integer

The grid-system that is used by the dataset is depicted in figure 3.1. The center of the playing field is at coordinate (0, 0, 0). The unit of measurement is in mm; a very precise unit. Observe that the output of the sensor-positions are not according to a rectangular-shaped 2-dimensional plane (the domain of the function f(x) is of equal size for all f(x) values). On the outer corners of the grid, there are inconsistencies in the measurements: one would expect the x value to be zero at the bottom-left corner, yet it is 50mm. Either the sensor is wrongly calibrated or the football field's middle-cut is not exactly perpendicular (90 degrees) to the field's length. The grid-corners thus have inconsistent values. We designed a grid-system that uses a scaling on this large coordinate system that the sensor uses (in mm). This makes them easier to use in the visualization-system. The scaling is done through a simple range change function, which uses the outer values of the domain's range(max and min values) and translates them onto a new domain-range:

$$NewValue = \frac{(((OldValue - OldMin) * (NewMax - NewMin))}{(OldMax - OldMin)) + NewMin}$$

These large coordinates will hence automatically fit our grid-system,since we convert each coordinate into a more usable domain and range in an exact coordinate system in cm. A schematic-view of the playing-field's old and new coordinate-system is depicted in figure 3.1 and figure 3.2

It is to be noted that the timestamp that marks the start of the football-match is 10753295594424116 and the end timestamp is 14879639146403495. We left these in the dataset 'as is'. We could have chosen to substract the start amount (10753295594424116) from every timestamp and cutting the edges of the data where $10753295594424116 > x > 14879639146403495$. Instead, we coded parameter-variables that can be set to define the start and stop timestamp. This enables for use of other datasources without having to preprocess the data.

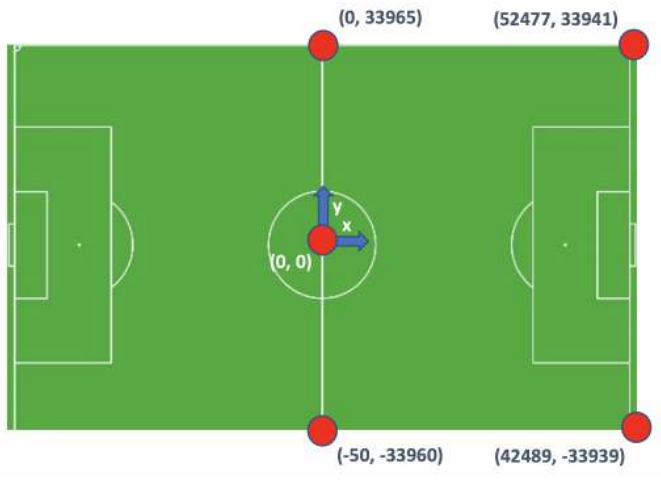


Figure 3.1: Gridsystem before scaling (original grid-coordinates)

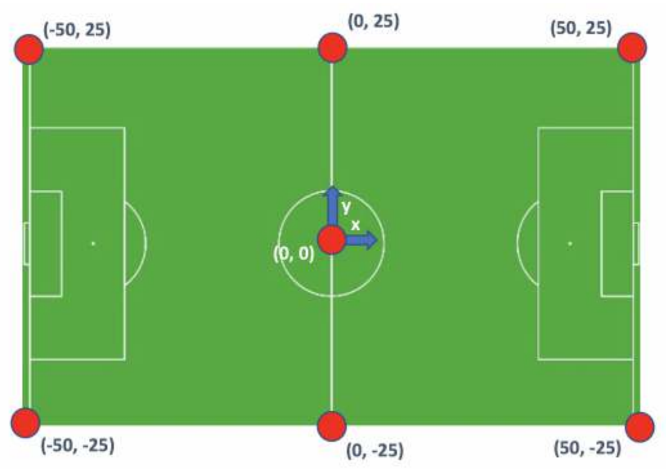


Figure 3.2: Gridsystem after scaling

Chapter 4

Design Decisions

We take this project from the DEBS 2013 challenge, the problem is described as the problem owner seeking to have a way of demonstrating the applicability of event-based systems to provide real-time complex analytics over high-velocity sensor data along the example of analyzing a soccer game, as well as the real-time analytics includes the continuous computation of statistics of relevance to spectators (ball possession), trainers and team managers (running analysis of team members). Indistinctly, we linked our concept with the popular video game FIFA developed by Electronic Arts. In principle, FIFA is a football simulation game that comprises of a match simulation with coherent match graphics, real-time speed meter, ball possession, heatmaps etc. So we decided to make a similar system that takes the sensor data as input to achieve the same goal.

From the start it was clear that we wanted to visualize the three-dimensional data depicted in the datasets as is. This would be the 'base'-functionality of the system and would, among others, have as purpose providing a ground-truth (in combination with a parallel video-feed) and be a base of additional functionalities such as ball-prediciton visualizations. Moreover, we wanted to perform player performance evaluations, specifically to get to know wheter a certain player understands his position in the field with respect to other players in his team. We decided that a heatmap would provide for these results. A heatmap can accurately give insight on where a player is the most active, which most naturally gives insight in 'what a player is actually doing' and 'whether he understands his position' on the field. Furthermore we decided to present team-dominance statistics in a slider-bar. This form of visualization instantly gives insight in how team-dominance is divided, unlike textual or numerical visualizations. Unless we would include historic analysis for ball-possession, bar/line charts would have been overkill for this purpose, since there are only two teams to visualize team-dominance on.

Note that the design decisions were also inspired and motivated by DEBS, the provider of the dataset. Their challenges involved, among others, real-time visualisations, creating heatmaps and ballpossession statistics.

Chapter 5

Implementations

5.1 Main Engine

As main engine and programming framework, Unity was used. Unity is a multipurpose game engine that supports 2D and 3D graphics, drag-and-drop functionality and scripting using C#. The engine targets among others Direct3D on Windows and OpenGL on Linux, macOS, and Windows. For 2-dimensional renderings, Unity supports a sprite-based system. For 3-dimensional renderings, Unity enables the developer to use techniques such as bump-mapping, reflection mapping, parallax mapping, screen space ambient occlusion, dynamic shadows using shadow maps, render-to-texture, full-screen post-processing effects, raycasters. Manny fully-fledged techniques an be used and Unity has by now proven to be among the more mature game engines.

5.2 Components

The main software components that are of importance for the application are:

- **Preprocessor** - Responsible for minifying datasets
- **StreamContoller** - Responsible for reading the sensor-data.
- **FileController** - Responsible for import and export of datafiles (heatmaps and data-exports)
- **HeatmapController** - Responsible for creating and reading heatmapdata in the Unity application.
- **StatsController** - Responsible for generating statistical information on the the datasets
- **Python Pipeline** - Responsible for the communciation between Unity and the Heatmap Generator
- **Heatmap Generator** - Responsible for the generation of the heatmap image.

A schematic overview of how these software components are placed is depicted in figure 5.1.

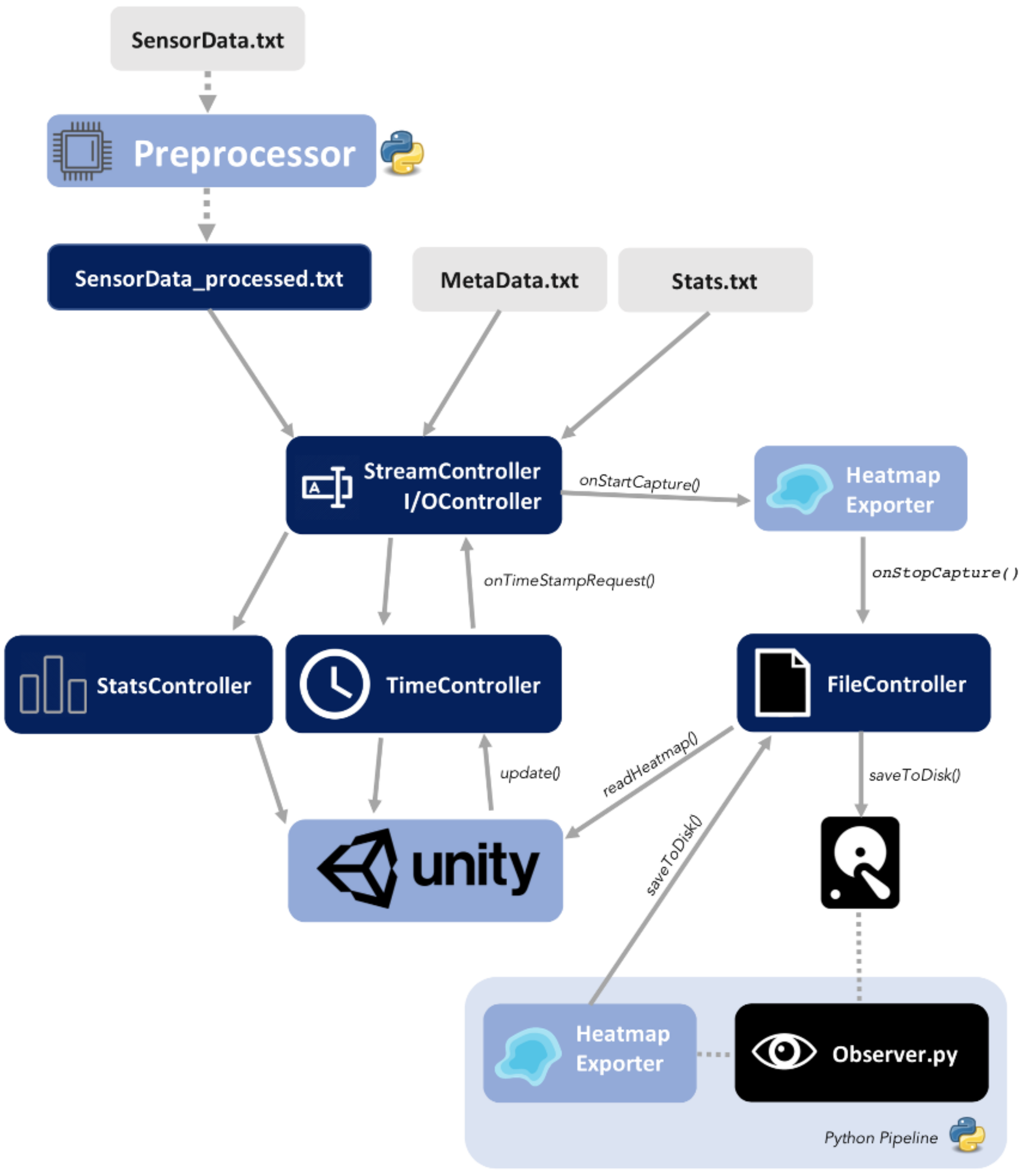


Figure 5.1: Schematic overview of the software components

Below, a textual elaboration is given for the more complex or important components.

5.2.1 Preprocessor

Since the dataset features an approximate 15.000 position evens per second, we initally decided to preprocess the dataset to improve usability. The preprocessor filters out similar measurements on short intervals; lowering the Hertz-rates to a rate that still allows for continuous visualization, yet is not so resource-demanding. Data preprocessing is done via a different pipeline; it is not coupled with the Unity visualization application and needs to be executed manually. However, one of the goals that we set ourselves initially, was to be able to read the full extend of the dataset in realtime, meaning not to use any preprocessing techniques. We channelled our focus into improving the streaming-algorithm, after which we managed to obtain good results. Subsequently, we decided to drop the preprocessing timeline, since now we could read significantly quicker than real-time playback.

Note that, the data files for meta-data and statistics do not need any preprocessing and can be read directly by the application.

5.2.2 StreamController

All logging events that were captured in the dataset, need to be read from its respective text file, in order to analyze or visualize them. We implemented a streaming algorithm that runs in an unmonitored and unsyncronized thread, this since unity does not support multi-threading. As like any kind of application that is only able to utilize one thread, issues are introduces when attempting resource-demanding operations, i.e: complex mathematical calculations, NP-hard algorithms or continuous I/O-operations. Since our dataset contains an approximate 15.000 logging events per second, continuous I/O operations are required. When performing these I/O-operations in the main thread (which is also responsible for the rendering of the objects and GUI), it was not possible to reach consistent frames per second. The application stuttered, became irresponsive and finally imminently crashed. As a result of executing the I/O operations in an unsyncronized threads, possible deadlock-scenarios arise. A deadlock occurs when a process or thread enters an awaiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. Possible solutions to this problem are software and hardware locks which handle shared resources and implement process synchronization. We attempted implementing C# Mutual exclusion locks, but results were minimal. In diagram 5.2, it is visible what component belongs to what thread and where potential synchronization issues (can) arise.

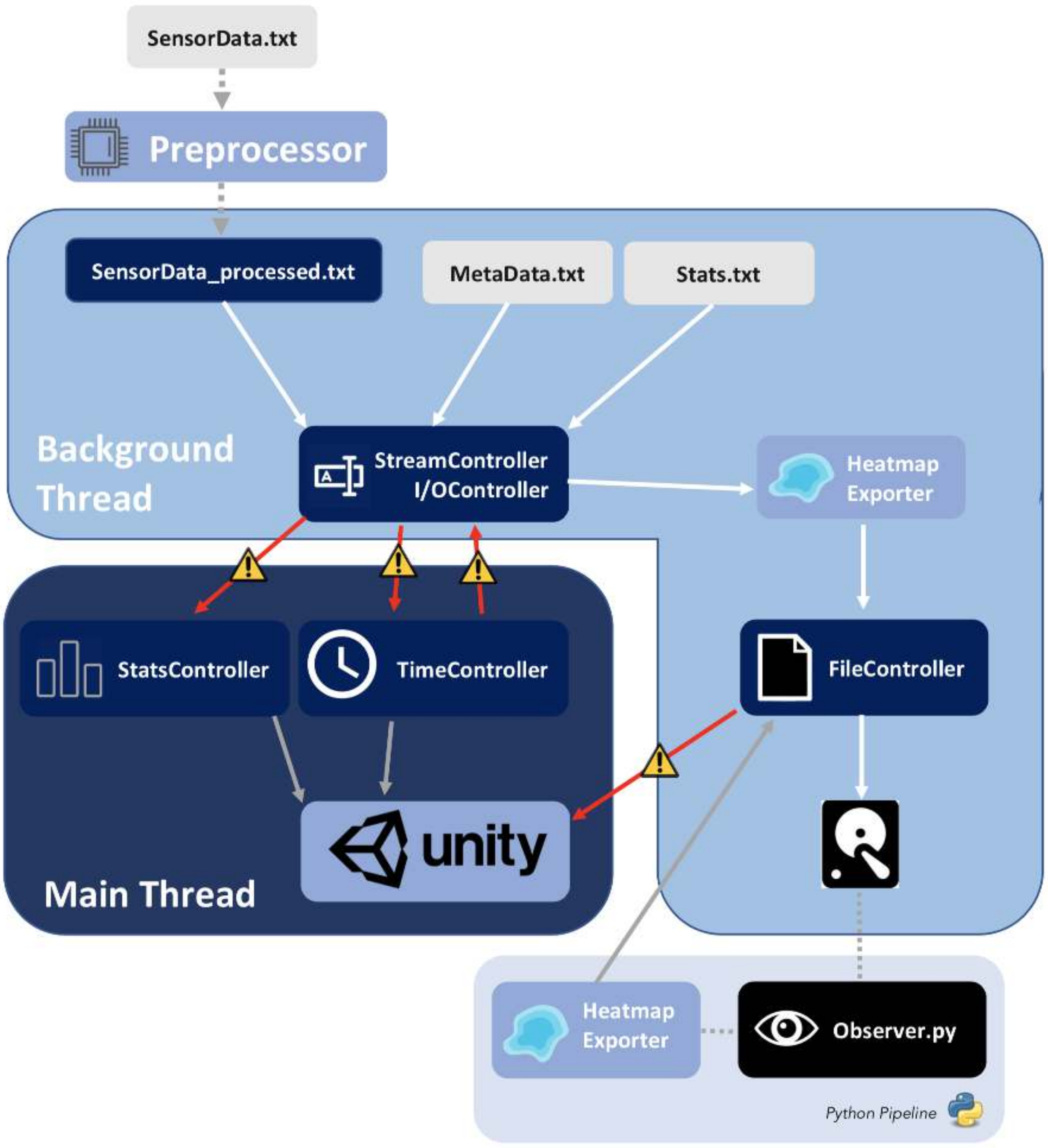
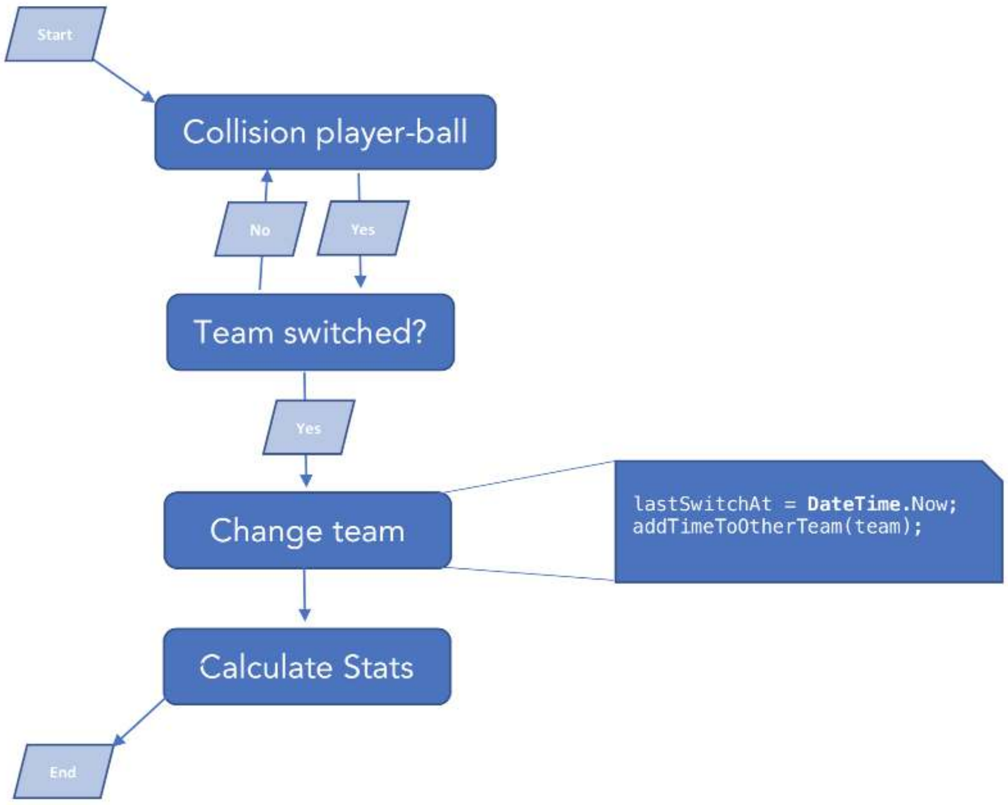


Figure 5.2: Schematic overview of the software components and their respective thread

5.2.3 StatsController

The Statscontroller has access to various static collections that contain among others, time-stamp mappings, location-data, object-acceleration and object-velocity data. These variables are static and decoupled from the main thread and get updated periodically by the StreamController. It uses this information to calculate, among other, the ballpossession. Current solutions consist of a human operator flipping a switch when a player of the other team touches the ball. We decided to adhere to this method of calculatingballpossession; every time a player touches the ball, we update our statistics. We decided to use Unity's build-in raycaster and collide-engine for collision detection instead of the by DEBS 2013 proposed method: comparing position of objects and detecting overlap if objects are within a certain range of each other. A schematic view of the ballpossesion-calcution process is depicted in 5.3



```
lastSwitchAt = DateTime.Now;
addTimeToOtherTeam(team);
```

Figure 5.3: Schematic overview of the ballpossession calculation process

5.2.4 Python Pipeline

Attempts were made to implement a gradient-based heatmap in Unity itself by utilizing shaders. However this brought along some issues; performing (any kind of) gradient-calculations,which are necessary for shaders, are very resource-demanding, which in turn resulted in a frame-rates of under 2 frames per second. A second approach was attempted which included the generation of a 3-dimensional cube-grid; 5000 respective cubes (100x50) were placed above the playing-field, which were programmed to change its color, reflecting a grid-position's 'hotness'. This also resulted in performance-problems.

After reflecting on these intermediary results, the decision was made to discard these methods and use another technology to generate the heatmaps with; doing heavy calculations in the Unity

Main Thread or multi-threading solutions are just not possible. Python is known for its quick data processing, and thus we chose to utilize Python for this. A complete Python pipeline was implemented to take care of the heatmap-generation. The complete pipeline is depicted in figure 5.1. Since there is no programming interface to let the Unity engine communicate with any external Python services or processes, we decided to use the Observer architectural style. The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, by calling one of their methods. Unity outputs a data-export for the set parameters (playerId, timeframe). Upon write-completion of this file to disk, the Python Pipeline's Observer observes this newly created file and queues it for processing.

5.2.5 StatsController

Instant speed Calculator

In the dataset, the position data of each sensor is offered in vector form, therefore, we could calculate the instant velocity $\vec{v}$ as:

$$\vec{v} = \lim_{\Delta t \to 0} \frac{\Delta \vec{D}}{\Delta t} = \frac{d\vec{D}}{d\vec{t}}$$

Where $\vec{D}$ is the vector change in position, Δt is change in time, $\frac{d\vec{D}}{d\vec{t}}$ is the derivative of vector position with respect to time. This calculation will be updated along the match. This calculation will be updated along the match, thus the result show in the system is the instant velocity in real-time.

Ball Possession Calculator

Ball possession is the amount of time a team possesses the ball during a game of football, and it is usually expressed as a percentage. In the modern football match, the possession is always computed as the total amount of time a team possesses a ball while on offense in percentage. The calculation of this figure relies on the level of match and statistics recorded in real time.

In our system, as the timestamp is provided in the dataset, we calculated it by dividing the time for which a side controlled the possession by the total game time.

$$P = \frac{endtime - starttime}{matchtime}$$

This method is being widely used by many football simulation game such as FIFA by Electronic Arts. The possession calculation will be updated along the match, thus the result show in the system is the real-time possession.

5.2.6 Heatmap Generator

Via the Python pipeline, the Heatmap Generator script is invoked upon receiving a newly queued data-export file.

The script reads the coordinates of the player from the sensors embedded in the data-export file. Subsequently, these position-points get scaled and plotted on a scaled pitch grid with size $x \in [-50, 50], y \in [-25, 25]$. In this pitch grid, each dot represents one single recording of the position of the player. We first use the 2D-histogram function from textitmatplotlib to plot the blocky results.

However, a major problem with the heatmap made by histograms is that the choice of binning can have a disproportionate effect on the resulting visualization. Therefore we decided to apply the Kernel-Density Estimate (KDE) and contour the results.

Kernel-Density Estimation (KDE) is a method to estimate the probability density function of a random variable in a non-parametric way. It works for both uni-variate and multi-variate data. In fact, if $(x_1, x_2, , x_n)$ is a d-variate independent and identically distributed sample vectors drawn from some distribution with an unknown density f. KDE targets in estimating the shape of this function f. The KDE $\hat{f}_h(x)$ can be formalized as:

$$\hat{f}_h(x) = \frac{1}{nh} \cdot \sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

where K is the kernel function, and h is the parameter in N^+ that determines the smoothness.

In our system, the KDE is essentially a smoothed histogram (density function). Instead of a point falling into a particular bin, it adds a weight to surrounding bins.

Chapter 6

Results and Evaluation

The system targets to help the football coach to have a better understanding of the match status and player's performance. As the consequence, the system analyses results visually and in parallel also streams some aggregated statistics in real-time. In this chapter, we will show some use cases and justify the results.

6.1 Basic Mandatory Features

From the requirements of the project, we are asked to implement some mandatory components. They are:

1. Plot the player positioning in a 3D manner

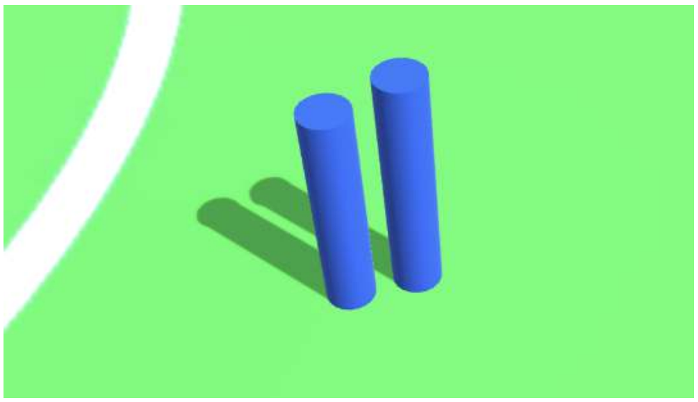


Figure 6.1: A player plot. A cylinder represents a single foot-sensor

2. Speed meter for each player

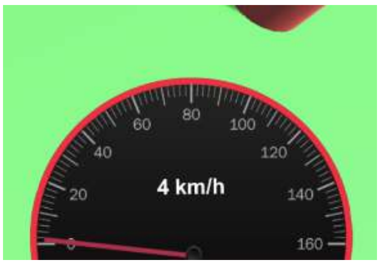


Figure 6.2: Speed Meter

3. Menu-interface to interact with visualizations and statistics

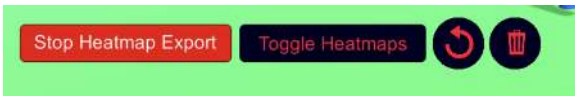


Figure 6.3: Speed Meter

4. Play/stop button to start/stop the simulation

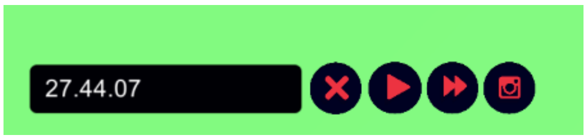


Figure 6.4: Speed Meter

Components 1, 3, 4 are easy to validate as we showed during the demo session where we showed the functionality of them respectively. For speed meter functionality, we will justify them below accompanying with other features of the system.

6.2 Possession Statistics Functionality Evaluation

We start monitoring the ball possession from 02:02:10 to 02:16:58 during the match. In this time period, team blue is attacking (starting from a defensive position), and team red is defending. We can see that team blue, possess the ball all the time until 2:16:58.

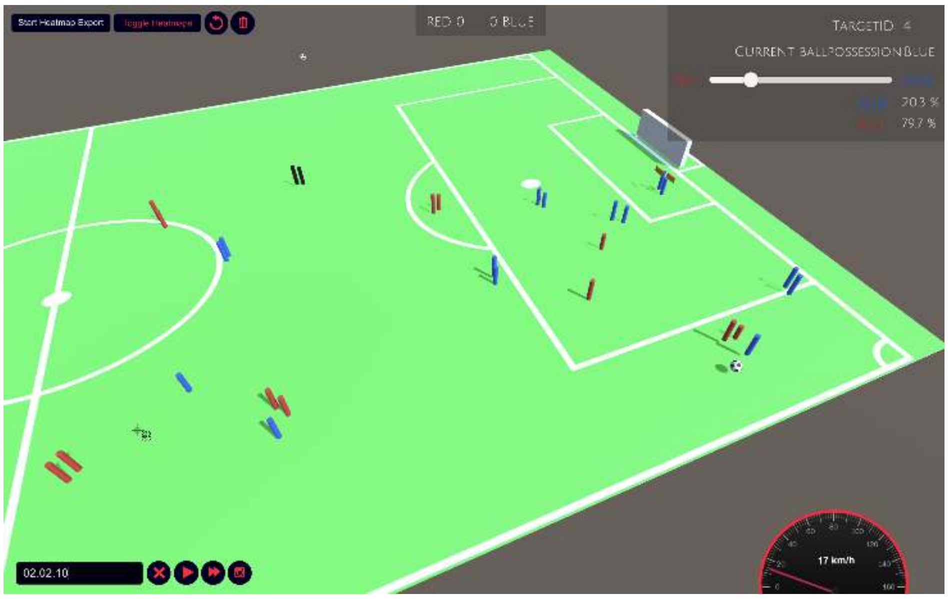


Figure 6.5: Ball Possession at 02:02:10

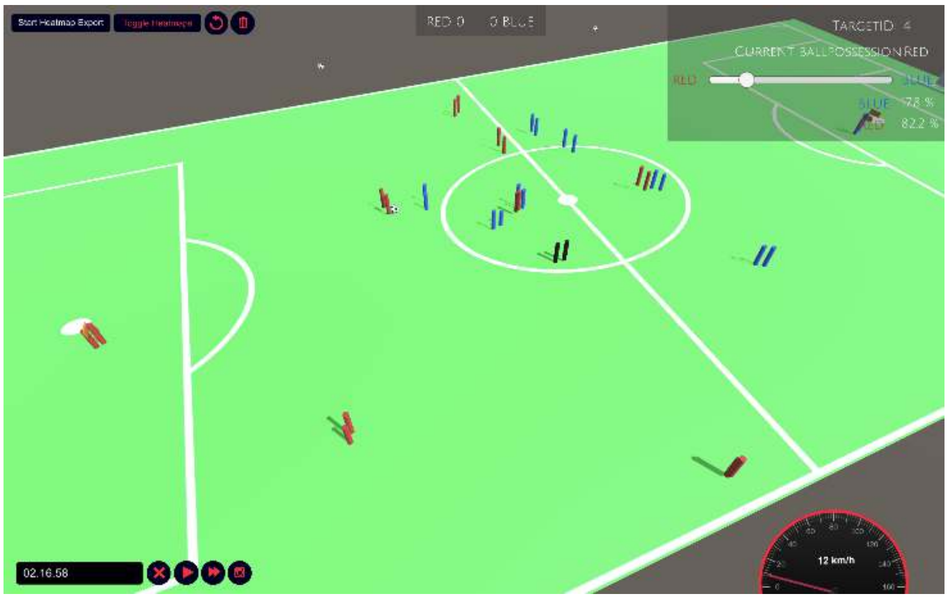


Figure 6.6: Ball Possession at 02:16:58

6.3 Instant Speed Monitor Functionality Evaluation

We tracked player with id 14. From the simulation and real-time video, we know the player is one of the goalkeepers. When the goalkeeper's respective team is performing an offense, the goalkeeper is left alone and barely has a displacing velocity. He moves about around in the box, but not much. However when the opponent team gets close to the box, his speed increases abruptly, which means that he reacted to the potentially dangerous attack.

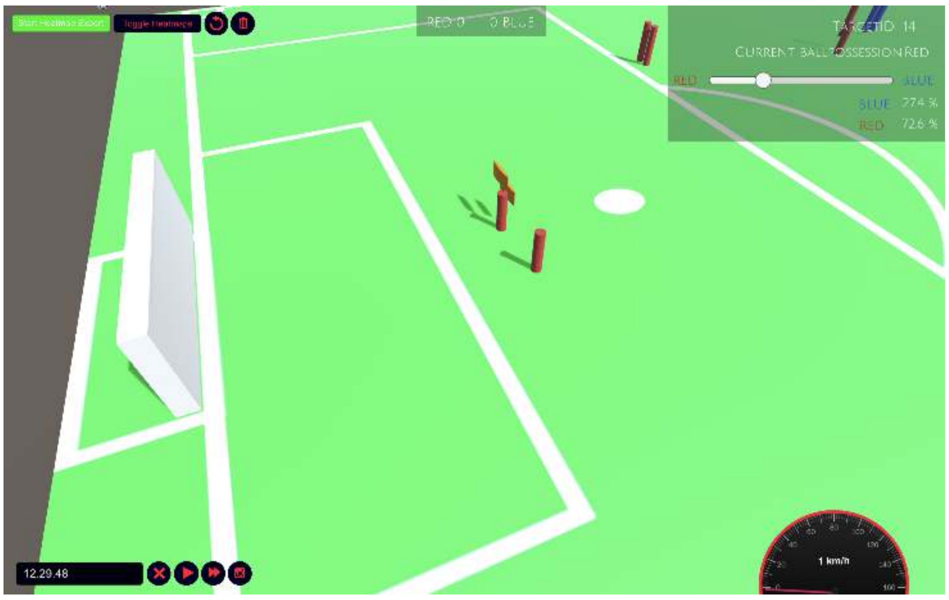


Figure 6.7: Goalkeeper instant speed at 12:29:48

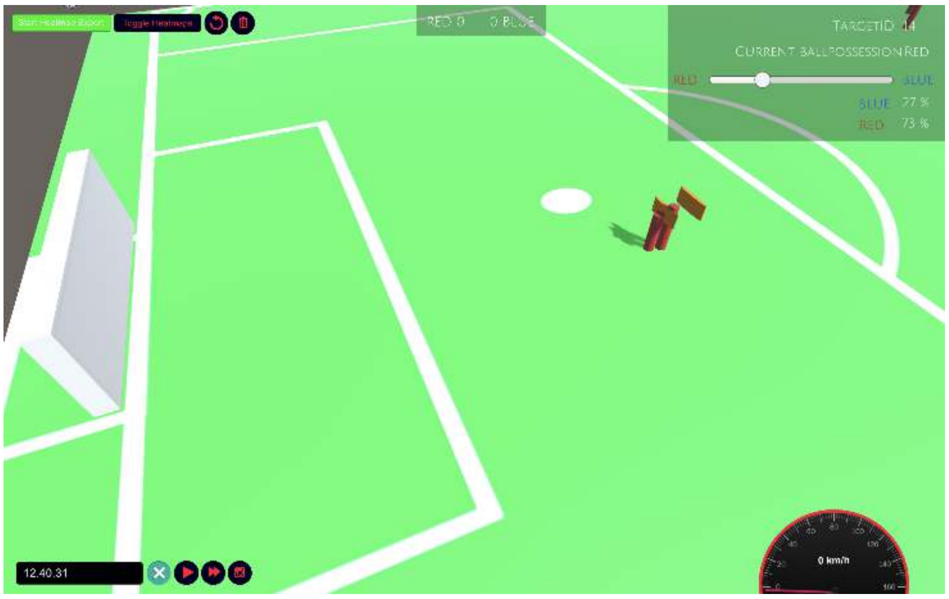


Figure 6.8: Goalkeeper instant speed at 12:40:31

6.4 Heatmap Functionality Evaluation

6.4.1 Goalkeeper Heatmap Example

Heat Maps in football or any other Sport are used to identify the frequency of events spread in a given particular area. Specifically for Football, heatmaps are an indicator of the effectiveness of a player in different parts of the pitch. The map gets heated up in areas where the player has had more control of the ball and does most of his work, i.e the color turns deeper as the player's presence in a particular area increases.

We record the player with id 61 during the time-range of 01:20:10 to 08:40:90 to be able to evaluate his performance.

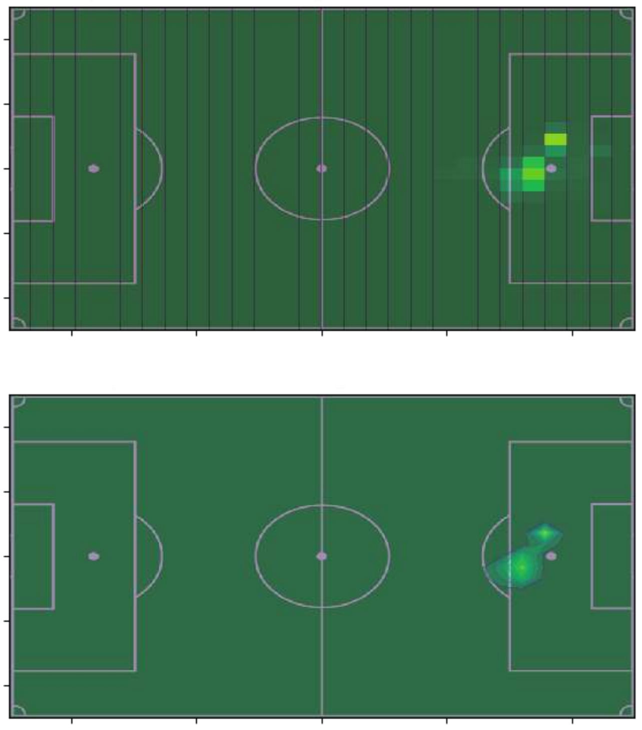


Figure 6.9: Goalkeeper Heatmap

Sensor id 61 belongs to team red's goalkeeper. We observe in the heatmap that he mainly stays in the goal's box area. We notice that he moved forward several time, because he attempted stopping the ball before the rival player dribbled it to the zone. These cases are slightly observable in the heatmap as well. We can conclude that this goalkeeper understands his position and does not deviate much from his 'job-description'. Subsequently, we recorded a player with id 49 during 00:50:01 to 45:18:73 to be able to evaluate his performance.

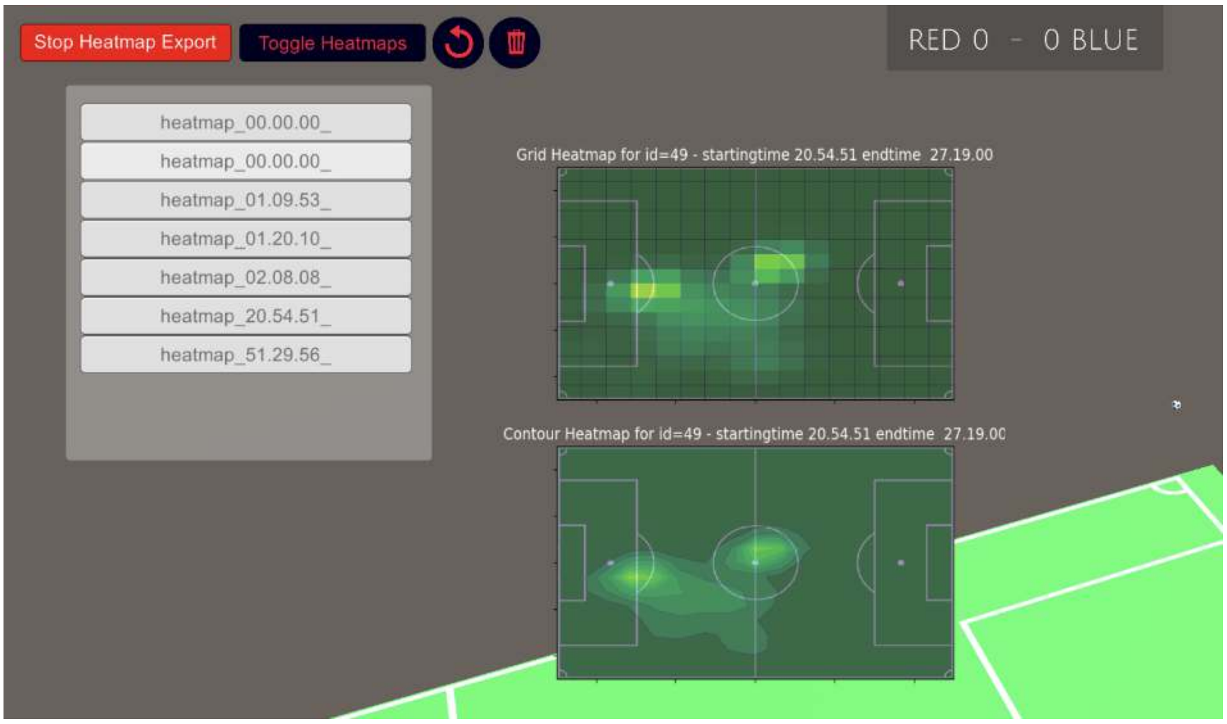


Figure 6.10: Rightback Heatmap

This player's position is right-back. In (professional) football, the right-or-leftback's responsibility is not only to defend the flanks during the match, but also to help/cover the left/right-wingers during an offense. In other words, the left/rightback should remain in his own zone during any defensive manoeuvres or when the opponent team is performing an offensive which reaches the goal. On the contrary, when his own team is attacking or possesses the ball, the left/rightback should cover his side as much as possible to contribute in the offense.

From the heatmap, it can be observed that this player fulfilled his defending task as he mainly roamed around the box to join the defense. However, the player also appears to not flank the rivals accompanying the rightwings during an attack (20:10:45) of his own team. This information can subsequently be used to instruct the player to join the attack with his team more instead of purely defending.

6.4.2 Ball-position Heatmap Example

Another example evaluation can be based on the position of the ball. In this example, the position is recorded from 00:00:47 to 07:18:01. From the ball-possesssion statistics, we can deduce that team red (left-side) is dominating with a ball-possession percentage of 62.1% (team blue: 37.9%, right-side). Interesting to observe in the ball's heatmap is that the hottest area is on the right (mid) field, which belongs to team blue. Both evaluation-methods hence validate each other. This heatmap can be used to improve tactics: since much ball activity is recorded in the mid-field of the team that is taking defensive positions, the offending (and dominating) team could attempt to divert more attacks via the left or right flanks. Note that this information alone cannot give insight on whether performing flank-attack is actually possible. It does however tell us that almost all attack are performed via the center, without any results (there are no goals).

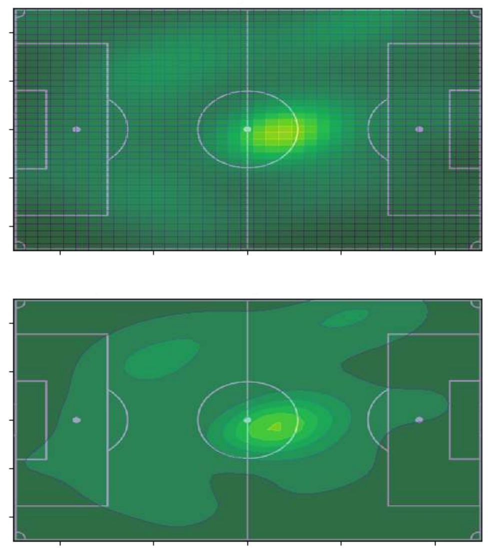


Figure 6.11: Heatmap of the ball's position during 00:00:47 - 07:18:01

Chapter 7

Conclusion and Future Work

7.1 Conclusion and Future Work

Current implemented solutions by the biggest football club for monitoring and benchmarking player's performance use image recognition technologies on recorded footage of the matches. However,the quality of image recognition is limited by many factors, e.g the quality of the video, the weather-conditions of the match day. The image recognition algorithm will perform poorly during rain particularly. However,since we use data gathered from sensors attached to the player, all of the constraints mentioned above will have no impact on our analysis result. It is because of this that we are able to present superior results. On the other hand, we did not know much about the sensor they are using in the football match, hence we do not know the weaknesses of these sensors (think about sensor measurement bias, accuracy a.s.). Football is a quite an aggressive sport involving much body contact. Therefore, we predict that during the physical contact, there might be an impact on the sensor. This could cause some biases during the recording. We would like to verify this, but unfortunately the match data we had is from a test training match, there is no aggressive physical contact like tackling. So this issue still remains unknown to us.

Current implementation also still has some present issues. As we used Unity to construct the system, we were not able to synchronize multiple threads leading to probable synchronization problems. In terms of visualization, the rendered result sometimes shows stuttering during the streaming. We tried the implement a supply-consume model (mutex) and semaphore, neither of them however seemed to solve the issues. We acknowledged the fact that it is not possible to solve the issue as this feature is only supported in the paid version of Unity or by utilizing third-party premium plugin frameworks.

Also, we did not process the video data. We simply play the video in parallel of our streaming model. If we fast-forward/playback our simulation model, the video will not keep the same pace as we did not index the video for timestamps. (in fact it is just the raw match video). It would be a great improvement for our system to have the video synchronized with our streaming model, which helps to provide the real-time justification of the visualizations. We propose a solution using a either a different technology which is able to natively manage thread synchronization or implement robust threading functionality using a third party plugin. Moreover, to improve robustness and generalization, we also suggest supporting multiple data sets from different sources (with possible different formats) in the next iteration of the application.

Bibliography

[1] DEBS 2013 Grand Challenge Soccer Monitoring
Taken from: *http://debs.org/debs-2013-grand-challenge-soccer-monitoring/*
At: $09 - 03 - 2018$ ii, 1

DEBS.org. (n.d.). Retrieved April 09, 2018, from http://debs.org/debs-2013-grand-challenge-soccer-monitoring/

[2] Top 10 List of the World's Most Popular Sports
Taken from: *http://www.topendsports.com/world/lists/popular-sport/fans.htm*
At: $09 - 03 - 2018$ 1