Evaluation of Models for Estimation of Handball Game Flow

William Rosengren

Master's thesis
2016:E16

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Foreword

This project has been carried out at Spiideo in Malmö in the spring of 2016. I would like to thank my two supervisors Klas Josephson from Spiideo and Håkan Ardö from the department of mathematics at Lunds Tekniska Högskola for their guidance thorughout this project. I would also like to thank Spiideo for giving me an opportunity to work with one of my biggest passions, sports, as well as including me as one of their own during my time at the office.

Abstract

This thesis presents and evaluates different models aimed at tracking the play in handball matches. The tracking is designed to be used with automatic broadcasting of handball matches without any camera operator present. To solve this problem the Kanade-Lucas-Tomasi tracker is used to generate data of player movements on the handball court. From the tracker data, features are extracted and used as input to the models being evaluated. Three different types of models are evaluated. One of the models are an artificial neural network (ANN) model, created from machine learning algorithms. The result shows that using an ANN model is the best approach of the models tested. They also show that the features chosen to describe the game flow used for making estimations are more important than the structure of the ANN. When using ANN, the model estimates the game play in almost all situations but some key events may still be missed.

Contents

1 Introduction

The demand for more complex and in depth analysis of sports have increased in recent years. In individual sports like *track and field* as well as *swimming* and team sports like *ice hockey*, *football* and *handball*, video analysis could improve the performance of individual athletes as well as entire teams. Handball is a popular sport in northern Europe and is played at many different levels. In Sweden, there are two elite leagues, one for women and one for men. The matches from these two leagues are usually televised.

Even though matches are televised, the content available for coaches to use for analysis is limited. The focus of the production is not necessarily to help coaches to analyse the game, but rather to make the production as good as possible for the viewers. The production of handball games includes filming the audience, filming the goalkeeper after a save and other content unrelated to the game itself. While this might be enjoyable for the TV audience, it is not the best possible data to use for evaluating team performance.

A big part of a club's activity is not playing acctual matches, but rather to practice. By installing a camera system in the sports arena, which records matches automatically, it would be possible for the coaches to review practice sessions as well. This system could also be used to broadcast matches that otherwise would not have been shown. Broadcasting via the internet has become a popular way to distribute sports. This medium makes it simpler to watch sports and it makes sports more available to the public. Junior team matches, in for example tournaments, could be broadcast and shown to relatives of the competing teams.

One other advantage with an automatic system is that cameras can be installed in places where it is not possible to have a camera operator filming with the cameras. The cameras can for example be installed in the ceiling of a sports arena to provide a view of the court not used normally. An automatic system would also reduce the human factor when filming. If the camera operator is not experienced or if long sessions are filmed, the focus level of the person filming will decrease. This could potentially affect the quality of the recorded content. An automatic system will not be affected by the duration of the content being filmed.

In this thesis, different models for performing automatic tracking of handball games are evaluated. The models will be developed offline. This means that during the development of the models, all possible data are available at all times and the suggested model could be *non-causal*. This is generally not a problem since broadcasted content today has a delay even when they are manually recorded. Using data from future frames would in all likeliness not affect the viewer experience. To describe how a handball match is played and how the game changes over time, the term *game flow* will be used. Different methods for estimating the game flow will be tested and evaluated. Video content of handball matches is used as input data in the development of the automatic system. To get content, cameras were installed in a arena where matches were recorded. This was performed before this project started. To generate input data to the models, information about the player positions on the court must be acquired. To get information about players positions and movements on the court, motion detectors are used.

1.1 Computer Vision in Sports

Computer vision is a field within image analysis which aims at analysing images and higher dimensional data [38]. Computer vision has previously been used in a variety of applications related to sports. Some systems are intended for decision making.

One of the most famous computer vision application in sports is the Hawk-Eye system [16]. It is used in tennis as well as a few other sports. For the tennis application, the Hawk-Eye system is used to compute a trajectory of the ball. It is used by the umpire as a tool to help making better calls. This is an example of a system making computations of its own.

There have been several attempts to make systems to track individual players in team sports. Particle filters, have for example, been suggested to find players in both football and ice hockey [4]. GPS systems have been tested in football to track how many kilometres a players has moved during a game. There are a number of companies in the world who provide GPS tracking services. Seo et al [32] used a colour based tracking method to find players on a football field. In 2000 Pers and Kovacic [27] proposed a system to track individual players in handball matches . All of these methods have been able to provide data of multiple individual athletes moving on a playing field. Trackers that

follow team sports game flow has however not been researched to the same extent.

Spiideo [36] has implemented methods to follow *track and field* athletes running in indoor arenas. A similar method has implemented to follow skiers. These systems makes real time estimations of athlete position and movements. The real time estimation is needed because the cameras, used for tracking, are moving.

1.2 Problem formulation

There are suggestions of how to track multiple individual players in different team sports. This information might not sufficient to make estimations of how the game is being played. Team sports are complex and it is non-trivial to obtain information about individual players and transform it to game flow. The available information is the positions of objects on the court obtained by a tracking algorithm. From this information, important game properties, known as *features*, can be extracted to create a model. The task in this master's thesis is to find the best set of features and the best model to describe the game flow. This will be done by evaluating a number of different models. The implementation of the models will be made offline which makes a *non-causal* solution possible.

1.3 Aim of Master's Thesis

The aim of this master's thesis will be to investigate if and how automatic tracking models for handball game flow can be developed. The evaluation will take into account how well the models are suited for broadcasting. Aspects of the model creation process and data needed for creating the models, will be analysed. The aim is also to find where the models fail to make good estimations.

2 Theory

In this sectionthe theory behind estimation model construction will be presented. To build a model, data from handball games needs to be generated. This is discussed in section 2.1. After this theory regarding statistical models and machine learning will be presented.

2.1 Motion analysis in images

Motion analysis, also known as tracking, is a part of computer vision and image analysis, which aims at finding moving objects in images. Objects in this context is defined as a patch of neighbouring pixels which have the same pixel intensity. The applications of tracking are many and it can be used in a wide variety of areas. To describe how objects moves in images, optical flow can be used.

2.1.1 Optical flow

Optical flow describes how pixel intensities in images change over time. The assumption made by in optical flow is that an object has the same intensity regardless of time, but that the position of the object changes in the image. In figure 2.1, a black circle can be seen on a grey background. It is possible to find the black circle by comparing the pixel intensities of the black area and the grey area. In figure 2.2, the circle has moved to a new position. The movement between the two figures can be expressed by optical flow.

The intensity at pixel position (x_0, y_0) at time t_0 in image I will change position to $(x_0 + d_x, y_0 + d_y)$ at time $t_0 + d_t$. The optical flow model for the intensity can be written as:

$$I(x_0, y_0, t_0) = I(x_0 + d_x, y_0 + d_y, t_0 + d_t). \tag{2.1}$$

By performing a Taylor expansion of the expression in equation (2.1) a new equation is obtained. The taylor expansion becomes:

$$I(x_0 + d_x, y_0 + d_y, t_0 + d_t) = I(x_0, y_0, t_0) + \frac{\partial I}{\partial x} d_x + \frac{\partial I}{\partial y} d_y + \frac{\partial I}{\partial t} d_t \tag{2.2}$$

The left side of equation (2.2) is equal to the right side in equation (2.1). By subtracting the term $I(x_0, y_0, t_0)$ from both the left and the right side, the equation for optical flow becomes:

$$\frac{\partial I}{\partial x} d_x + \frac{\partial I}{\partial y} d_y + \frac{\partial I}{\partial t} d_t = 0. \tag{2.3}$$

When the optical flow equation in (2.3) is solved, the change in the x- and y-positions are obtained. It can be used for tracking [9].

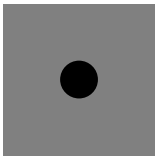


Figure 2.1: The black circle has a different pixel intensity compared to its neighbours. It is therefore possible to identify it from its neighbours and it makes the circle possible to track.

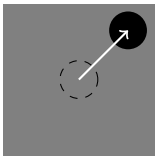


Figure 2.2: The filled circle is the same as in figure 2.1. The dashed circle is the original position of the circle. The white arrow represents the movement of the circle. Optical flow describes this movement.

2.1.2 Kanade-Lucas-Tomasi Tracker

In 1981 Lucas and Kanade introduced the first version of the Kanade-Lucas-Tomasi Tracker (KLT-Tracker) [21]. Properties or features have been added to the tracker and in 1994 Shi and Tomasi [34] wrote a paper on how to find interesting patterns in images for tracking .

By finding objects, which contains large intensity gradients, the tracker can find the same object in successive frames. This is illustrated in figures 2.1 and 2.2. There is an intensity gradient between the black circle and grey background. Once an object has been found, the tracker aims at finding the best match between the position of the object in the current frame compared to the position of the object in a previous frame. The difference in position is known as the disparity vector **h**. The

disparity is represented by the white arrow in figure 2.2 and it is defined in (2.4), where d_x and d_y are changes in x- and y-positions respectively of the object,

$$\mathbf{h} = (d_x, d_y). \tag{2.4}$$

If the disparity vector can be found, it would be possible to say how an object has moved. The KLT algorithm uses intensity gradients to iteratively find a disparity vector that best matches the displacement of an object. One way of doing this is to use a gradient descent algorithm (see section 2.2.2).

The intensity gradients are used to find *corners* in the image. A corner is defined as a pixel, which has large intensity gradients in more than one direction. If the pixel has large intensity gradient in exactly one direction, it is called an *edge*. An example of how the KLT-tracker could work is found in figures 2.3 and 2.4. The figures represents a handball court with one player. The black and red rectangles represents a player's shirt and shorts respectively, and the white rectangle represents the playing court. The point on the court where these three rectangles meets forms a corner, which is called p_1. By first identifying the corner in figure 2.3 and then finding the same corner in figure 2.4, one can tell how the player has moved.

This fairly intuitive concept is the basis of how the KLT tracker operates. The tracker finds corners and then tracks them through successive frames. The position history of a corner is called a *track*. The assumption the algorithm makes, is that a corner found in image I at time t_0 is going to have a similar position at time $t_0 + d_t$.

One important concept of the KLT tracker is that it does not follow single pixels. One reason for this is that it is in general quite hard to follow a single pixel. The value of the pixel can change due to measurement noise or be mistaken for an adjacent pixel. The algorithm uses windows of pixels instead to improve the tracking. [34]

To simplify the tracking, the input images are transformed to grey scale images. Colour images typically have three colour channels, one for red colour, one for green colour and one for blue colour. These images are often referred to as RGB-images. All colours are build up by combining different levels of the three channels. If the KLT-tracker would use RGB images it would need to compute intensity gradients for all three channels for all pixels to find intensity gradients. To reduce the number of computations needed for doing gradient computations, the images could instead be transformed to a grey-scale image. A grey scale image is one dimensional in its colour channels. Each pixel is represented by a value between 0 and 255 where 0 is black and 255 is white.

2.2 Machine Learning

Machine learning is a technique to build models to cluster unknown objects (*Unsupervised Learning*) or to train a model to make estimations based on previous observations (*Supervised Learning*) [11]. There is one more type of machine learning, which is called *Reinforcement Learning*. Reinforcement learning is implemented for decision making problems. Machine learning has proved itself to be quite powerful when classifying and estimating data. The focus of this degree project will be to use supervised learning, which is presented in section 2.2.1.

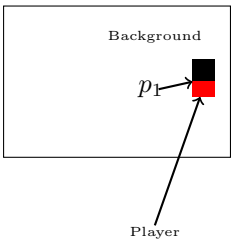


Figure 2.3: The red and black rectangles represents a player on a handball court. Between the red and black rectangles and the background an edges is created as point p_1.

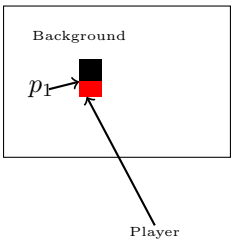


Figure 2.4: The same scenario as in figure 2.3 taken at a different time point. The point p_1 has changed positions. By finding p_1 in this figure, is it possible to say how the player has moved.

2.2.1 Supervised Learning

In supervised learning, a model is trained by feeding data to the model as well as a desired output, which here is referred to as *Ground Truth*. The model generates an output, which is compared to the ground truth. The model variables are updated so that the difference between the output and the ground truth becomes smaller [19]. The *Learning Rate* introduced in section 2.2.5, the *Optimisation Algorithm* introduced in section 2.2.2 and the *Loss Function* introduced in section 2.2.6 are used for updating the model variables.

A simple version of supervised learning can be found in figure 2.5. The model variables, $\mathbf{w}$, are initialised as random numbers. This can for example be a set of samples from some distribution. An input, $\mathbf{x}$, generates a model estimation $f(\mathbf{x})$. This is compared to the desired value y to create the error, $e = y - f(\mathbf{x})$. The error is to be minimised by updating the model variables $\mathbf{w}$. The optimisation is described in section 2.2.2. If there is no correlation in the error, it will not be possible to find a better set of model variables given the current setting.

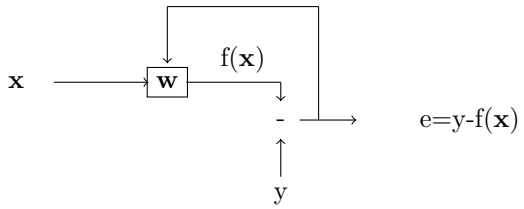


Figure 2.5: Chart of a simple a supervised learning algorithm. The data vector, $\mathbf{x}$, is fed into the system. The error, e, is the difference between the desired value, y, and the estimated value, f($\mathbf{x}$). By minimising e, the model variables, w, are updated.

2.2.2 Gradient Descent

To find a model suited for estimation, some optimisation of the model variables, $\boldsymbol{w}$, must be performed. One quite common method to find a good model is to use *Gradient Descent* (GD). Like the name suggests, gradient descent uses the gradient of a function to update model variables. The optimisation aims at minimising the sum of errors (see section 2.2.6). If w_{t-1} is the old set of weights, γ is learning rate (see section 2.2.5), $\nabla_{w_{t-1}}$ is the gradient with respect to the previous weights and $L(w_{t-1})$ is the error given the previous weights, the new set of weights, w_t, can be computed as,

$$w_t = w_{t-1} - \gamma \nabla_{w_{t-1}} L(w_{t-1}). \tag{2.5}$$

When computing the updated model variables, their new value depends on the learning rate, γ. The higher the learning rate, the more the variables are allowed to change between iterations[15].

It is important that the loss function is differentiable for all weights. If it is not differentiable for some weights, it will not be possible to compute $\nabla_{w_{t-1}}$.

2.2.3 Stochastic Gradient Descent

There are some problems with using gradient descent. If the data set is large it will be computationally costly to use the entire data set before updating the weights. This leads to slow convergence. To speed up computations, a *Stochastic Gradient Descent Method* (SGDM), could be used. The SGDM divides the entire data set into equally large batches of data. This is done by doing a random permutation of the entire data set.

Once the data has been sufficiently shuffled, it is partitioned into batches. After assigning data into batches, the training s commenced. In each new iteration of the training loop, a new batch is used as input data. Once all available batches have been used in the training, new batches are generated by repeating the shuffeling. These new batches are then used in the next iterations until all batches have been used in the training. This continues until the model has converged.[3]

Given that the batch size is large enough to represent the entire data set, the SGDM provides faster convergence than the GD. This is because the SGDM update the model variables more often than what the GD method does. Problems can occur if the batch size is chosen to be so small that it no longer represents the entire data set. If this occurs, the updating of the model variables might

change drastically between each iteration. This is not good for the convergence of the system When choosing batch size, it is important to choose it large enough to represent the entire data set, but small enough to assure fast convergence of the system variables. [3]

2.2.4 Momentum Optimiser

An extension of the gradient descent algorithm, is the *Momentum Optimiser* (MO). The idea behind MO is to capture the general trend of how the model variables have been updated. In the standard case of using gradient descent optimisation, the model variables are updated such that the total loss is reduced the most. This might however not be the best strategy for finding the best model variables. Instead, a *moving exponentially weighted average* of previous weights is used [24]. This leads to a smoother change of model weights and sometimes faster convergence. [22]

2.2.5 Learning Rate

The learning rate is very important for the convergence of a machine learning algorithm. The learning rate is used to say how much the system variables are allowed to change in each iteration, when updating the system variables. If the learning rate is too small, it will take long time for the variables to converge. If it is set too high, the system might become unstable and the model estimations will diverge. The learning rate is adjusted for each new model training situation and there is no consensus on how high it should be chosen.

The system variables are usually initiated randomly. It is therefore preferable to have a large learning rate at the beginning to quickly find better model variables. After having trained a model for several iterations, the model variables should be closer to a good solution. When this happens, it is desired to fine tune the model variables. This can be done by lowering the learning rate.

One way to implement a smaller and smaller learning rate is to use a exponentially decaying learning rate [33] expressed in equation (2.6). The variable on the left hand side, $\gamma(t)$, is the current learning rate, $\gamma(0)$ is the initial learning rate, r is the decay rate and i is the current number of training iterations. The current learning rate can be computed as:

$$\gamma(t) = \gamma(0)10^{\frac{-i}{r}}. \tag{2.6}$$

When using a decaying learning rate, it can implemented as a staircase function, see figure 2.6.

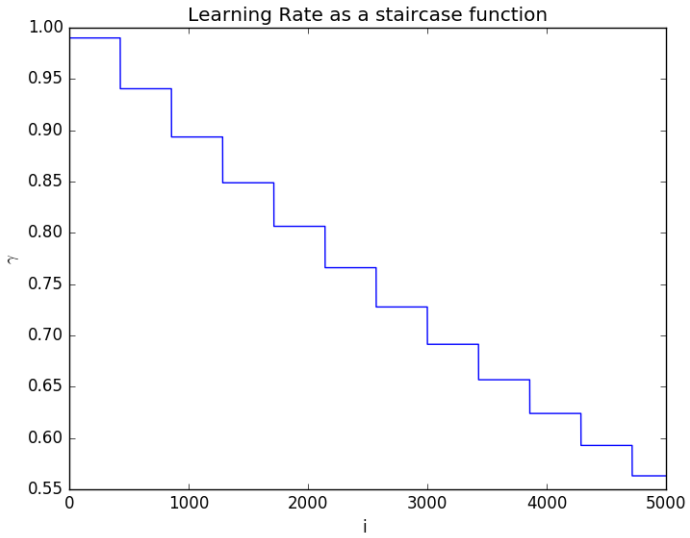


Figure 2.6: Learning rate as a straircase function from equation (2.6). The initial learning rate as 9.910^{-1} and $r = 0.95$.

2.2.6 Loss Function

The loss function is used to describe the difference between the ground truth and the model output. This is known as the *error* or the *loss*. The error depends on the type of function used and the

model variables. When optimising the model variables using gradient descent, the derivative of the loss function decides how the variables should be updated. In one training iteration, all data in one batch is run through the model and the error for each data point is computed. Due to stochasticity in the data, the error will be different for each data point in the data set. Since the goal for updating the weights, is to make the error smaller, it is needed to use a method to take all errors and transform it to one value. There are mainly two methods to see if the error becomes smaller. Both the sum or all errors and the mean of all errors can be used. The latter is often implemented to make scaling easier.

Once the training algorithm has been run for some iterations, a series of errors has been acquired. This series says something about how much information that is left in the model. If there is some correlation in the series, there is still information to be gained by running more training iterations. If the error behaves like a white noise process, there is no more model structure to be gained from further training with the current settings.

2.2.7 Squared Loss Function

There are several well documented loss functions, each with their benefits and disadvantages. The *squared error function* is one of the more commonly used functions. It is defined in equation 2.7. If the variables to be optimised are $\boldsymbol{w}$, y is the ground truth value and $f(\boldsymbol{w})$ is the model output, the squared loss function, $L(\boldsymbol{w})$, can be computed as

$$\boldsymbol{L(w) = (y - f(w))^2.} \tag{2.7}$$

2.2.8 Tukey's Biweight Function

Data points which differs too much from all other data points are called *outliers*. These measurements are considered to be corrupted in some way. One potential problem with the squared loss function is that it adjusts too heavily for outliers when updating the model variables. One way to handle outliers is to use another loss function. One alternative is *Tukey's Biweight function*. If the difference between the ground truth and the model is defined as $e = y - f(\boldsymbol{w})$ and c is a constant, the loss function dependent on the difference, $L(e)$, can be computed as:

$$L(e) = \begin{cases} \frac{c^2}{6}(1 - [1 - (\frac{e}{c})^2]^3), & \text{for } |e| < c \quad (2.8) \\ \frac{c^2}{6}, & \text{otherwise} \quad (2.9) \end{cases}$$

The function is shown in figure 2.7.

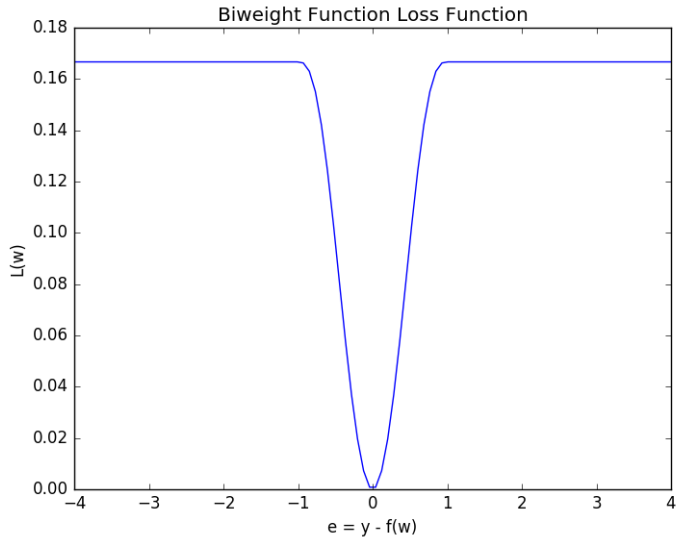


Figure 2.7: Tukey's Biweight loss function with c = 0.25

As one can see in figure 2.7, Tukey's Biweight function is similar to a squared function. The difference being that for values larger than some constant, c, the function output becomes constant. There is a maximum output value, $c^2/6$, which the function can generate.

2.2.9 Huber Loss Function

The constant output of the function could cause a problem. In the constant "zone" of the function, the gradient with respect to model variables, is zero. This will make it hard for the optimising algorithm to find a better set of weights. To handle this potential problem, a loss function which is linear instead of constant, could be used. If $L(e)$ is the loss function with respect to the difference $e = y - f(\boldsymbol{w})$, where $\boldsymbol{w}$ is the model variables, y is the ground truth and $f(\boldsymbol{w})$ is the model output and δ is a constant, the *Huber Loss Function* can be written as,

$$L(e) = \begin{cases} \frac{1}{2}(e)^2, & \text{if } |e| < \delta \quad (2.10) \\ \delta|e| - \frac{1}{2}\delta^2, & \text{otherwise.} \quad (2.11) \end{cases}$$

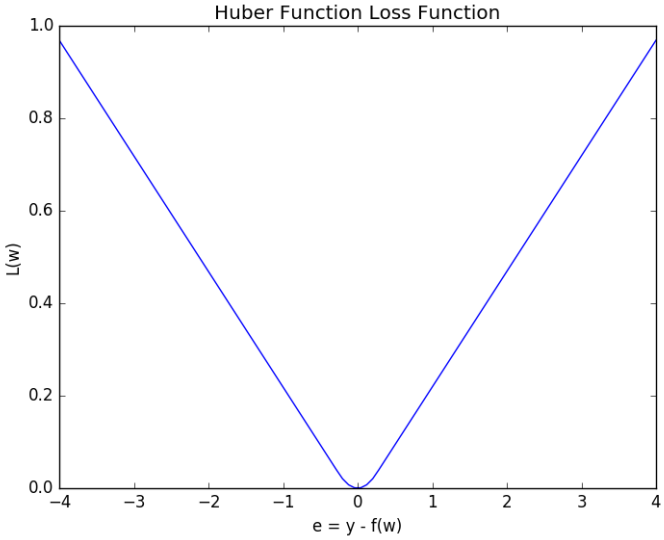


Figure 2.8: Huber loss function with $\delta = 1$.

This is also shown in figure 2.8. The function is a quadratic function until it reaches some threshold. When the threshold is reached, the function instead becomes linear. Unlike the Tukey's Biweight function, the Huber loss function will increase its output with increasing difference between the desired value and the model output. It will however not penalise outliers as much as a strictly squared function.

2.3 Linear regression

A common statistical model is the linear regression model. Linear regression is used to model and estimate linear relationships. The model variables, $\boldsymbol{w}$, are multiplied with the data input vector $\boldsymbol{x}$. To the multiplication a bias, $\boldsymbol{b}$, is added to generate an output $f(\boldsymbol{x})$. The equation for a linear regression model is defined by,

$$f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + \boldsymbol{b}. \quad (2.12)$$

2.4 Artificial Neural Networks

Another type of model, which can be used for estimation, is an *artificial neural network* (ANN). The main difference between a linear regression model and an artificial neural networks is that the artificial neural network uses non-linear functions for estimation. This makes ANN models able to

model more complex problems than what the linear regression model is able to do.

2.4.1 Inspiration for Artificial Neural Networks

The construction of artificial neural networks are inspired by the neurones found in the human brain. The function of the neurone is to lead electro-physical signals, called *action potentials*, generated somewhere in the body and transmit the signals to or from the brain. The way they work is that each neurone has one end, which receives signals produced by other neurones. The signal-receiving part of a neurone is called *dendrite*. If enough input is generated to the dendrites of a neurone, the neurone will transmit a signal of its own. Each neurone has a threshold, for whether a signal will be transmitted or not. The signal is transmitted through a cell "wire", called *axon*, to the transmitting part of the cell, called the *axon terminal*. The axon terminal is in turn connected to the dendrites of one or more neurones. Once the signal reaches the axon terminal, it will transmit signals to the connected dendrites. In this fashion, a signal is transmitted throughout the body. One very important feature of the neurone is the refractory period. Once an action potential has been fired from the neurone it is impossible for the neurone to fire again until the ion balance in the cell has been partially or fully restored. This makes it impossible for the signal to go in two directions in the neurone and all of the energy will be used in one direction. [15]

2.4.2 Structure of Neural Networks

Neural networks are built with a similar structure as the human neurone network. The network consists of layers of *nodes* which are connected to each other (see figure 2.9). A node can be seen as a mathematical function, which generates an output dependant on the input it has gotten. The first layer, also known as the input layer, is where the data enters the node network. Once data has entered the input layer, an output is generated from each node in the layer. Let's call the output vector from all nodes in the first layer $\boldsymbol{O_1}$ which becomes the input to the second layer which then generates an output $\boldsymbol{O_2}$. The output from a node is computed in the following way. The $k:th$ node in the input layer, n_k^1, receives an input, $\boldsymbol{I_k}$. The node has a set of weight, $\boldsymbol{w_k^1}$ and a bias, b_k^1, which are multiplied and added to the input, creating an intermediate input, s_k^1. This is shown in equation (2.13).

$$s_k^1 = \boldsymbol{w_k^1 I_k} + b_k^1 \tag{2.13}$$

Once s_k^1 has been computed, an *activation function* is used (see section 2.4.4). It is the activation function that decides the output from the node. If the activation function is called A_k^1, the output, O_k^1, from node n_k^1 becomes:

$$O_k^1 = A_n^1(s_k^1). \tag{2.14}$$

The procedure of layers generating output to the next layer in the network repeats itself until the last layer is reached. This layer is called the output layer. All layers, which are not the input or the output layer, are called *hidden layers*. A figure showing a simple neural network with one hidden layer in shown in figure 2.9. If all nodes in one layer are connected to all nodes in the next layer, the network is said to be fully connected. [15]

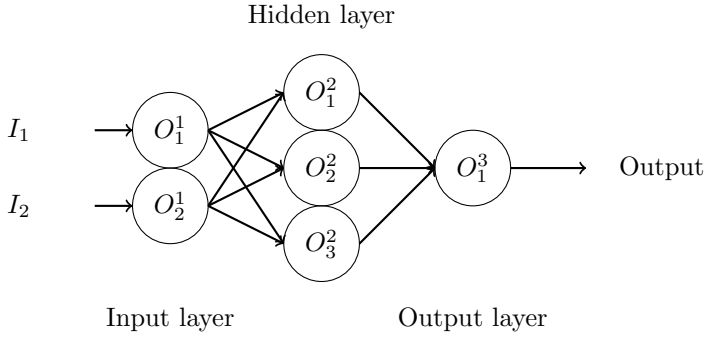


Figure 2.9: Chart over a simple neural network. Two different inputs enters one input node each. The network generates one output. The network is fully connected.

The ANN could be used for estimation in a similar manners as a regression model would be used for estimation. The big difference is that the output from the ANN is a non-linear transformation of

the input data whereas the output from a linear regression model is a linear transformation of the input data.

2.4.3 Types of Neural Networks

Feed Forward Neural Networks The simpelest type of neural networks are called *Feed Forward Neural Networks*. In this type of structure signals are only allowed to travel from the input layer towards the output layer. When training feed forward neural networks, the weights are updated after one entire data set has been fed in the model. [12]

Recurrent Neural Networks One other type of neural network that is fairly common is the *Recurrent Neural Network* (RNN). In this type of structure, signals are allowed to propagate backwards in the structure towards the input layer. To avoid signals looping through the same node several times before updating the network variables, a refractory period is introduced. The refractory period works in a similar way as in the human neurone. After the node has produced an output, it will become inactive for some time before it can produce a new output. The structure of the recurrent neural network makes it possible for the network to have a "memory" and to use previous inputs. RNN has applications in for example handwriting recognition and speech recognition. [30] [13]

2.4.4 Activation Function

Just as the threshold of the biological neurone, the output of a node in the neural network depends on the activation function. There are several types of activation functions and some functions are discussed by Hagan et al [15]. Three popular activation functions are the sigmoid function (figure 2.10), the hyperbolic tangent sigmoid (figure 2.11) and the rectified linear function (figure 2.12). The equation for each of these activation functions are defined below in equations (2.15), (2.16) and (2.17) respectively.

$$F(x) = \frac{1}{1 + e^{-x}} \tag{2.15}$$

$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.16}$$

$$F(x) = max(0, x) \tag{2.17}$$

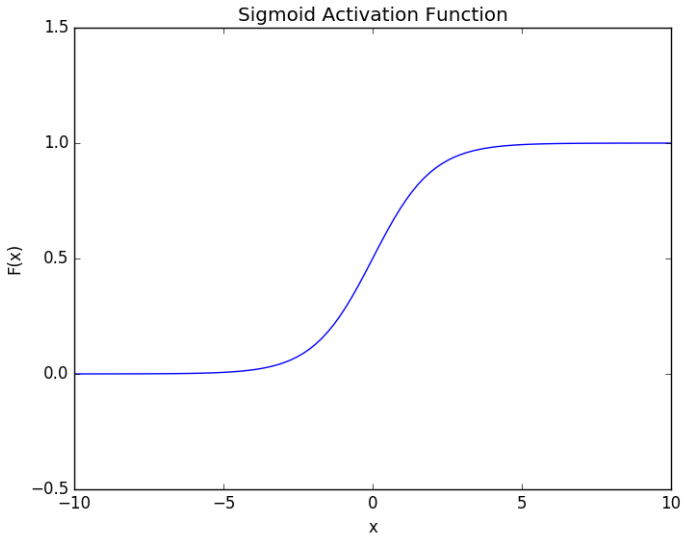


Figure 2.10: The sigmoid function.

2.4.5 Backpropagation

Backpropagation is a way of determining the weights, $\boldsymbol{w}$, in the neural network during training and it is central to the performance of the neural network. The problem with updating weights in neural networks is that there is not a linear relationship between the output and all the weights in the

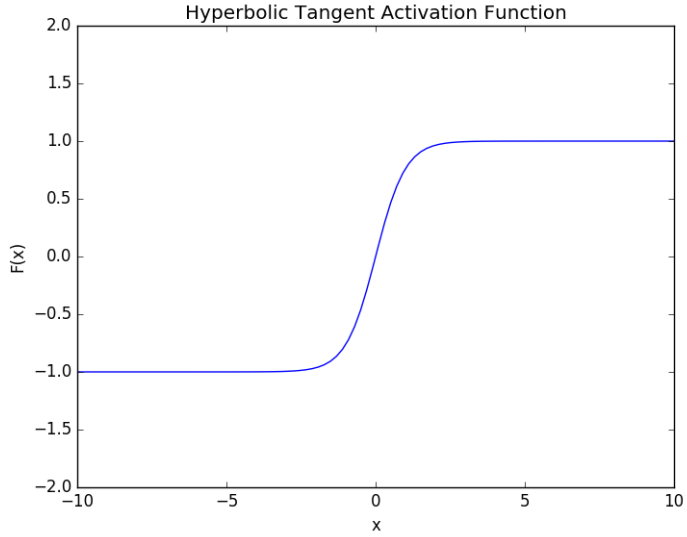


Figure 2.11: The hyperbolic tangent function.

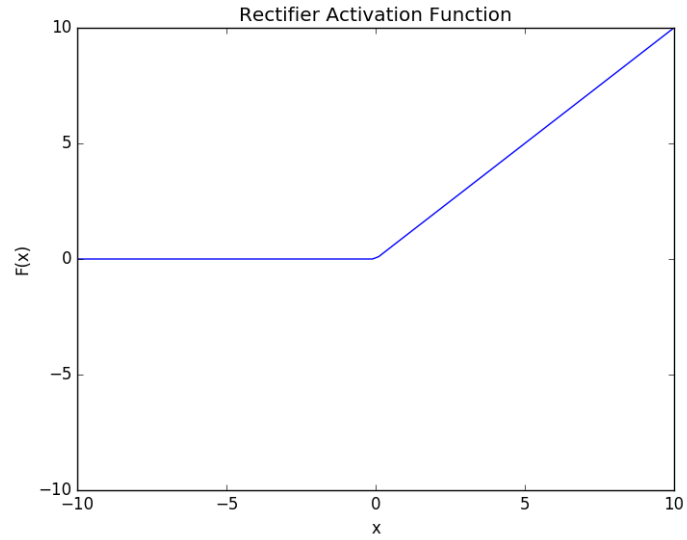


Figure 2.12: The rectifier function.

network. A neural network can be viewed as a series of functions, where the function output from one layer becomes the input to the next function in the series. A neural network with 4 layers can be described by equation (2.18). O is the output of the network, $F^l(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})$, $\boldsymbol{W^l}$ and $\boldsymbol{b^l}$ are the activation function, the weight matrix and the bias vector at layer l respectively and $\mathbf{x}$ is the input vector containing data. The series of functions can be written as:

$$O = F^3(\boldsymbol{W^3}F^2(\boldsymbol{W^2}F^1(\boldsymbol{W^1}\boldsymbol{x} + \boldsymbol{b^1}) + \boldsymbol{b^2}) + \boldsymbol{b^3}) + \boldsymbol{b^4}. \tag{2.18}$$

To optimise the weights for each function, an optimisation algorithm like the gradient descent could be used. Since each function is dependent on previous functions, the system is not straight forward to optimise. It is possible however to use the *chain rule* for functions, found in calculus. If a function, $F(w)$, is dependant on a function $G(w)$ and where w is an independent variable, the chain rule can be written as:

$$\frac{\partial F(G(w))}{\partial w} = \frac{\partial F(G(w))}{\partial G(w)} \frac{\partial G(w)}{\partial w}. \tag{2.19}$$

This relationship can be used to compute the derivatives of the loss function with respect to different weights in the network. The reason why the algorithm is called backpropagation is because the derivates are computed starting with the output layer and they are then computed all the way back to the input layer.

2.4.6 Design of Artifical Neural Networks

Figure 2.9 shows a simple form of a neural network. Neural networks can in practice have a large number of layers with a large number of nodes in each layer. The more layers and nodes a network has, the more complex the model becomes and the more complex problems it can solve. [15]

When designing neural networks, one must take into consideration the complexity of the problem at hand. Like with most other problems, it is preferable to keep the model complexity as low as possible. Three issues related to overly complex networks are: *Overfitting*, adressed in section 2.5, the training time of the network and the data needed to train the network.

The initiation of variables and the architecture of the network has an impact on the performance of a neural network. According to Nugyen and Widrow [23], the variables in a 2-layer neural network should be initialised with random values in range [-1,1]. The authors used a uniform distribution. The architecture of the network is harder to generalise. Hagan et al [15] suggest that the number of nodes should increase with the square root of the data. To find the best set of nodes in different layers is dependent on the problem and must be evaluated after each training.

In reality the design of neural networks becomes a trial and error mission. It is quite hard to know beforehand how the system should be initialised to give the best results. The design of a neural network with multiple layers can be performed in the following way. Initiate the first layer with the same number of nodes as the size of the input data. Initiate all other layers with a large number of nodes. Initiate the model variables randomly by sampling from some distribution. Evaluate the model after some number of iterations. If the results are not satisfactory, increase or decrease the number of nodes in the second layer. Repeat. If repetition does not yield satisfactory results, consider using one more layer of nodes. [15]

Other considerations, which are decided before the network can be used are: the loss function, the learning rate, optimiser, the activation functions of each layer, the type of network (feed forward or recurrent).

2.5 Overfitting

One common situation, which occurs when using machine learning is overfitting. Overfitting is a phenomenon where the error for the training set decreases but error for the validation and test sets increases (see section 2.5.1 for reference). The phenomenon occurs because the model variables have been too specialised on the training data. The model output for the training data will have a small error but the error for the validation data will increase. This is a critical problem to solve, since estimations are made on unknown data sets in supervised machine learning. In this section, some methods that are used to avoid overfitting are presented. Methods to prevent overfitting are generally referred to as *regularisation methods*.

2.5.1 Cross-validation

When training models, data is usually limited. To get maximum training out of the data available, *cross-validation* can be used. In cross validation the entire data set is divided into subsets. One subset is used for testing, one subset is used for validation and the remaining subsets are used to train the model. These are referred to as *validation set*, *test set* and *training set*. The training set contains most of the data and it is used to build a model. After training a model for some time, the test set is used to evaluate the model. The error of the current test estimation is compared to the error of the previous test estimation. If the error has decreased compared to last time, the training continues. If the error of the test set has increased, the training is terminated. This is known as *early stopping* [28]. Once the training has been stopped, the validation set is used to make a final model estimation. It is the estimation made by the validation set which is considered to be the result.

By doing several permutations of validation and testing, multiple data configurations are used to build and test models. The final result of the training is the average result from all validations. The results from cross validation will be more general than what a result from a single model training would have been.

2.5.2 Dropout

One way to address overfitting is to use dropout [37] . Dropout is implemented so that there is a certain probability that the output from each node in the dropout layer becomes zero. Dropout is implemented for one entire layer and it is usually implemented in hidden layers close to the output layer. What this does is that it forces the neural network to be more general, since the neural network cannot rely on the output from all nodes at all times. When using the dropout layers, one can see it as creating a new model in each new training iteration.

In a neural network which uses the dropout method, each node is assign a probability p to be used in the training, and subsequently a probability $1 - p$ to not be a part of the training. One does not know beforehand which nodes that will or will not be used in the training. Dropout is illustrated in figure 2.13.

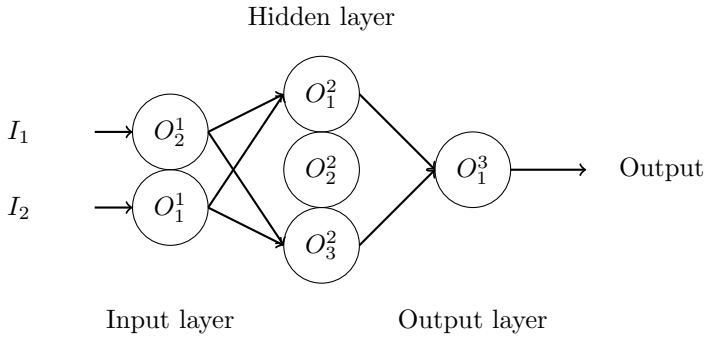


Figure 2.13: The network illustrates how dropout works. This is the same neural network structure as in figure 2.9. The difference is that the middle node in the hidden layer has been excluded. The output from this node will not be used and its weight will not be updated during this specific iteration.

After training, the entire model is used, without using dropout. The expected value of each node is affected by the dropout probability, so adjustments to account for the dropout are needed. Let us assume that the expected value of node n_k^l in layer l is O_k^l. If the node has the probability p to be kept, the new expected value of the output from node n_k^l becomes pO_k^l. When doing estimation, one can scale the output from each node by multiplying it with the probability, p, that the output from the node will be zero. If the model makes estimations equally well for the validation data set as for the training data set, the model is said to be robust. [37]

2.5.3 Weight Decay

Weight decay is a technique, which can be helpful when generalising a neural network [20]. If $\boldsymbol{w}$ is a vector of weights for a neural network, $L_0(\boldsymbol{w})$ is the loss from the loss function and λ is a constant deciding how much the weights should be penalised, a new loss, $L(\boldsymbol{w})$, can be written as:

$$L(\boldsymbol{w}) = L_0(\boldsymbol{w}) + \frac{1}{2}\lambda \sum_i \boldsymbol{w}_i^2. \tag{2.20}$$

Larger weights will results in greater loss. The optimising algorithm has an incentive to keep the weights as small as possible, which helps the generelisation.

3 Data

In this section the data used in the models is discussed. There are several important things to keep in mind when handling the data. If the data is of poor quality, it will be quite hard to achieve good results. This is true for both the ground truth as well as the data generated by the motion detector.

To make assumptions easier, the court was parametrised using an x-axis and a y-axis (see figure 3.1). The long side of the handball court corresponds the x-axis and the short side of the handball court corresponds to the y-axis. When x- and y-values are referred, they corresponds to a position on the long and short side of the court respectively. The penalty lines are marked out as well. They are 7 meters from the goal on their respective sides.

3.1 Data Collection Scene

The scene in which the analysis took place is quite important for the result of the tracking algorithm. External conditions like lighting are important to keep in mind when working with image analysis. Since the scene was indoors, the lighting was assumed to be constant.

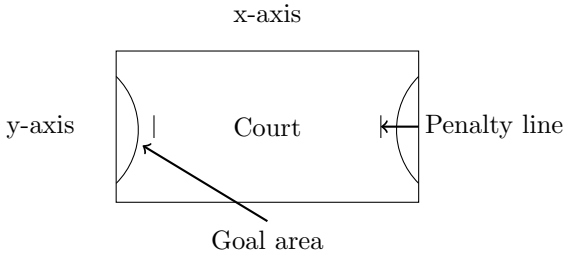


Figure 3.1: An image of the handball court. The x-axis corresponds to the long side and the y-axis corresponds to the short side. The two arcs on the short sides represents the goal area. The penalty line is marked out as well as the goal area. The penalty lines are 7 meters from the goal on each side.

There were in total two camera views as well as an artificially stitched view. The three views are referred to as *left, right* and *wide*. The *wide* view was artificially made by stitching the *left* and *right* views together. These are shown in figures 3.2a, 3.2b and 3.2c respectively.

3.2 Ground Truth

As discussed in section 2.2, it is of utter importance that the ground truth has high quality when using a supervised learning approach. In this case, the ground truth should represent how a camera operator would film a handball game. Or even better; how coaches viewing the content would like to watch it. The collection of ground truth data is presented in section 5.2.1.

3.3 Assumptions for the Data

When collecting data some assumptions were made. The handball court was not equally distributed in the image. As seen in figure 3.2c, the long side of the court, which is closest to the cameras appears longer compared to the long side on the other side of the court. It was however assumed that the long court side had the same length in pixels, regardless of position. The court was assumed to have come from a view like in figure 3.1.

3.4 Features

To build a model, data of how players move and interact with each other was used. This data will be referred to as *features*. The tracking of a handball might require a wide variety of features. The features could either be used directly as estimations themselves or used as input to a mathematical model. Since the model was developed offline, the could be non-causal. Given the same resulting estimation, the fewer number of features, the better. Each feature should contribute something unique to the estimation for it to be used in the model.

What is important to keep in mind when reading about the features, is that the KLT-tracker can generate more than one track per player. As stated in section 2.1.2, the KLT-tracker finds corners

(a) Left court side

(b) Right court side

(c) The wide view. The view is a stiched version of the left and the right sides. The closest x-side appears longer than the x-side on the other side of the court. This is an illusion due to the camera position and possibly the stiching.

Figure 3.2: Both court sides which were used to stitch together the wide view.

in the images and one player can create more than one track.

The features presented in this section are not the same as the features which the KLT-tracker used to create and follow tracks. The KLT features were based on intensities in the image. The game features were based on data generated by the KLT-tracker.

3.4.1 Observations of handball game flow

Before selecting features one needs to think about how a handball game is played. There is usually not much play in the middle of the court. The game flow in a handball game usually goes from one side to the other and then back again. One team is on the offensive and the other team is playing defence. When a goal is scored, if there is a foul, or the offensive team loses the ball in some way, the game will change over to the other side of the court.

To model this it is necessary to use information on player positions. One also needs to know how the players moves during the course of a game.

3.4.2 Frame Shift

The implementation of the models is non-causal. Features from previous and future frames are used in the position estimation. The method of using features from frames in the past and in the future will be referred to as *frame shift*. An example of frame shift is presented below.

Let's assume that the features at frame I_{t_0}, frame I_{t_0-25} and frame I_{t_0+25} were V_{t_0}, V_{t_0-25} and V_{t_0+25} respectively. A new feature vector W_{t_0} was created by combining the features from the three images. The new resulting feature vector was written as $W_{t_0} = [V_{t_0-25}, V_{t_0}, V_{t_0+25}]$. The features were sampled every 25 frames (one second apart).

The reason why features from multiple frames were used, was to see how the features changed over time. This in itself could be an interesting property to use for the estimation of game flow.

3.4.3 Making Non-Causal Broadcasting

In this project non-causal models, which could potentially be used for estimation of game flow in a live broadcast, were created. The reason why a non-causal model from broadcast could be used, was because of the camera set up. The camera system used for capturing images in this project were configured such that the cameras, between themselves, covered the entire playing court. Both cameras were used to record content at all time. The goal of the models was to find which content that was relevant for the game flow.

The following example will show how it would be possible to use a non-causal model for broadcasting. Given that all possible match content had been recorded at time t_{rec} from both cameras, the non-causal models could be implemented by using a delay in the broadcast. The broadcast time, t_{broad}, was delayed by some time t_{delay} of t_{rec}. The broadcasting time can be written as: $t_{broad} = t_{rec} - t_{delay}$. When broadcasting content at time t_{broad}, information at time t_{rec} is available. Basically, by broadcasting past events, data from the present can be used in the game flow estimation. The concept of how this works is shown in figure 3.3.

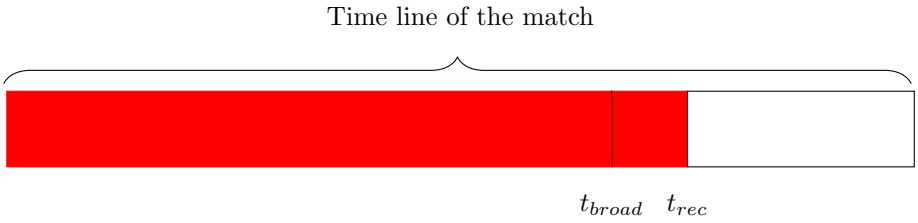


Figure 3.3: The red rectangle shows how much data that has been recorded compared to the entire length of the match. t_{rec} shows how much of a match that has been recorded and t_{broad} shows how much that has been broadcasted. By broadcasting content that has happened some time in the past, data from the present image can be used for estimation.

3.4.4 Types of Features

There are some requirements for the features that are important to consider before choosing the features. In this project two different classes of features were defined: positional and changing features. The positional features said something about where the players were on the court. The changing features were related to how the player positions changed between frames.

The features should preferably be fast to compute and be general so that it is possible to use the same features in matches in other similar arenas. This is achieved by normalisation of the features. It is also important that the number of features in each frame is constant. This is to ensure that the same model can be used for all different situations. Some features uses distributions. Let us assume that the number of players on each court side is a feature. The feature from this could be the number of player on each side divided by the number of players in total on the court. This ensures that the features all are decimal numbers between 0 and 1. Averaging could also be a way to map a lot of observations to one value.

Below follows suggestions for possible features to be used in the estimation of the game. These features are based on the data obtained by the KLT-tracker and it might not be possible to use the exact same features for other motion detectors.

3.4.5 Center of Mass

The KLT algorithm generates quite a lot of tracks. One way to create a feature of the positions of all tracks, is to compute an average, here referred to as the *center of mass*. The center of mass is computed by taking the average of coordinates for all tracks in each frame. Tracks considered to be part of the background are removed before computing the center of mass. If $\boldsymbol{r}$ is a vector with all tracks in image I_τ, x_i is the x-position of track i in $\boldsymbol{r}$ and there are n elements in $\boldsymbol{r}$, the center of mass, C, can be written as:

$$C = \frac{1}{n} \sum_i x_i. \tag{3.1}$$

3.4.6 Distribution of Tracks

One interesting feature is how the tracks are distributed on the court. This can be computed using a histogram. By dividing the court into equally large intervals, also known as *bins*, one can compute how many tracks there are in each of the bins. The distribution of tracks could potentially be informative for two reasons, the first being to find where most players are at the moment and the second is to say what type of situation the game currently is in. If most players are located at one court side, it is most likely because the play is at that side. If the players are spread out on the court, it is most likely due to changes in the game.

The number of bins chosen carries different information. If a smaller number of bins is used, there will be more tracks inside each bin and it will take longer time for tracks to switch between bins. If a larger number of bins is used, there will be less tracks in each bin and changes will happen faster. Both of these properties could be interesting for estimating the gameplay and both can be used at the same time.

3.4.7 Velocity of Tracks

Instead of using the positions of the tracks, the change in position for each track can be computed. The change of position is also known as velocity. The velocity carries information about how the player positions changes between frames. There are different alternatives for computing the change of position. The first alternative is to take the difference between the center of mass of the current frame and the center of mass for a frame in the past. If using this as a feature, it is important that the time between two frames is not too long nor too short. If the time is too long, the game situation might have changed too much for the feature to be of any use. If the time is too short, the center of mass might not have changed enough for the feature to be of any use.

Another alternative is to compute the velocity of each track. If this is computed, there are a few considerations which have to be addressed. Fundamentally the computation is the same as it is for the change of center of mass. The big difference is that some tracks are being generated and some vanishes in each new frame. If there is no previous track position, it is not possible to compute the velocity.

Much like the problem with using a track on its own as a feature, it is not preferable to use a single velocity of a track as a feature. Instead the distribution of velocities can be used. If there are a lot of tracks, which all have high velocity in the same direction, it is most likely because the play is switching court sides. Just like with the distribution of positions, a histogram for the velocities can be used to get a fixed length feature.

4 Software and Hardware

4.1 Software

4.1.1 Python

All of the code generated was implemented in the *python* programming language[29]. The version of python, which was used for the most part was python 2.7. The open source library *numpy* [25] and *scipy* [31] has been used to make numerical computations.

Open CV There is an open source library for image analysis implemented in python, as well as in C++, called *Open CV* [26]. It contains several of the tracking algorithms and imaging processing tools used and discussed in this project. The library uses a function called goodFeaturesToTrack [34]. The goodFeatureToTrack function generates tracks. Once the tracks have been generated, there is a function called calcOpticalFlowPyrLK which follows the tracks. These two functions have been used to generate data for this project.

Tensor Flow In November 2015, Google released a framework for machine learning called *Tensor Flow* [39]. Tensor Flow has been used to do the machine learning and artificial neural networks in this project . The theory for how Tensor Flow works was published by google [40].

4.2 Hardware

4.2.1 Cameras for Data Collection

The cameras used for collecting the data collection were Axis cameras model P1428. The cameras collected images at a rate of 25 Hz and with a resolution of 3840 x 1080 pixels. All images were stored in a private film library owned by Spiideo.

5 Method

5.1 Pre-Processing

5.1.1 Scaling Down Images

The images from the cameras had a resolution of 3840 x 1080 pixels. This resolution was too large to be used when making fast generation of KLT-tracks. To solve this problem, images were scaled down to a resolution of 640 x 180 pixels instead.

5.1.2 Converting Images to Gray Scale

The original camera images were RGB images (see figure 5.1a). The images were converted to grey -scale, using the open CV function cv2.cvtColor(), images before the optical flow was performed, (see figure 5.1b).

(a) Original color image with three colour channels obtained from one of the cameras.

(b) A grey scale version of figure 5.1a

Figure 5.1: The two figures shows the same situation as an RGB-image and as a grey-scale image.

5.1.3 Introducing a Mask

The KLT-tracking algorithm generated unwanted data. The unwanted data was generated mainly by two sources. The first source was the viewing audience. To remove the influence of the audience a mathematical *mask* was used. The role of the mask in the algorithm was to reduce the number of tracks generated. The mask was an image, which had black or white areas within itself, see figure 5.2a. The black areas of the mask acted as a blocker for the generation of new tracks. The only area where new tracks could be generated was in the white area. The mask applied on the court is found in figure 5.2b. It was still possible for tracks to move into the black areas.

(a) The black and white mask.

(b) Shows the mask applied on the court.

Figure 5.2: The mask implemented to reduce the number of tracks generated

As seen in figure 5.2, the mask is quite small compared to the entire court size. A few different mask sizes were tried. The reasons for using a small mask was to reduce the influence of players on the

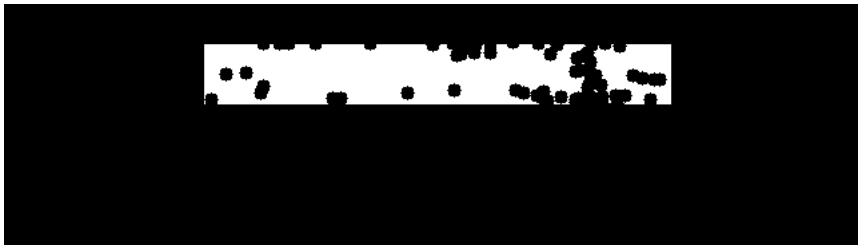


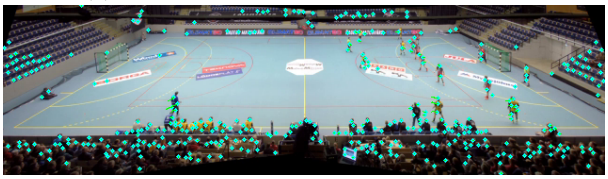
Figure 5.3: Black circles representing the tracks already in use were added to the mask in figure 5.2 to reduce the generation of new tracks.

court switching with players on the bench as well as not generating any tracks from the coaches on the bench.

The second problem which needed to be addressed was to not generate too many tracks inside the allowed white area (the playing court). The KLT tracker generated new tracks where it found corner points in the image and it caused a large number of tracks to be generated, most of which where unrelated to the players. This information was not desired to use for building the models, since it did not represent the players. To handle this problem, a minimum distance of how close two tracks were allowed to be generated from each other was implemented. This was implemented by drawing black circles inside the first mask at the positions of the tracks currently in use. The circles had a radius of 10 pixels. This is shown in figure 5.3.

In figure 5.4 the difference between using a mask and not using a mask is shown. In the top image (figure 5.4a) a mask has been applied. In the bottom image (figure 5.4b), the mask has been disabled. The tracks are represented by the coloured circles drawn in both figrues.

(a) Generated tracks when applying a mask.



(b) Generated tracks without the use of a mask.

Figure 5.4: The two images illustrates the difference when defining a mask and when a mask is not defined for track generation. The coloured circles represents the tracks currently in use.

5.2 Collection of Data

There were in total eight different data sets used in this project. For all data sets, the ground truth and tracks data were collected. Each data set was assigned a three letter code, for example ”DAH”. These types of data set codes will be referred to later in the text.

5.2.1 Collection of Ground Truth

The ground truth was collected by moving a mouse cursor in a window showing the matches. Data was collected once per frame and both the x- and y-positions were saved. To make the model more general, the ground truth values were normalised with the length (in pixels) of the the entire frame. The court length was 640 pixels, meaning that all ground truth values were divided by 640. A recording of the x-coordinate for a ground truth is found in figure 5.5a. The ground truth represented the game flow.

The recording of the ground truth had noise-like behaviour. This could for example be a result of quick involuntary movements of the mouse cursor. To adjust for this, the ground truth was low-pass

filtered. The lowpass filter used was a sixth order *Butterworth Filter*, with cutoff frequency 0.15 [5]. To keep the phase the same as the original ground truth signal had, the lowpass data was forward-backward filtered with a padding length of 150 [14]. The resulting filtered ground truth is shown in figure 5.5b.

The low passed ground truth was finally thresholded. The thresholding made all ground truth values to lie between 7 meter penalty lines (see figure 3.1). A sample of how the ground truth looks is found in figure 5.5.

5.2.2 Collection of Track Data

The KLT-tracker was used to generate tracks from 8 different handball halves. Both the x- and y-positions of all tracks in all frames were saved and stored to be used later. The track data was used to extract different features. The mask mentioned in section 5.1.3 was applied for the data collection.

5.3 Handling Disturbances and Unwanted Data from the Tracker

5.3.1 Handling Static Background Points

The mask helped to reduce unwanted data. But there was one disturbance, which could not be removed by introducing a mask. As mentioned in theory section 2.1.2, the KLT-tracker found corners to track. This included finding corners in the background. To not let the background corners affect the center of mass, they had to be remove from the track data. This was done by using the velocity feature. If the track point did not move in either the x- or the y-direction after some frames had passed, it was considered to be background and it was not used for the center of mass computation.

5.4 Untrained Estimation

5.4.1 Center of Mass as Estimator

The first estimator tried was the center of mass. This estimator did not use any machine learning to find model variables and it could be used directly for estimation. The center of mass was computed for all data sets and the difference between the center of mass and the ground truth was computed and evaluated. The center of mass was computed with a *moving weighted average* [24] when used as an estimator. If t is the current frame, α is a constant, $x_c(t)$ is the center of mass and $x_{wc}(t-1)$is the weighted center of mass at frame $t-1$, the center of mass was computed accordingly to

$$x_{wc}(t) = x_c(t)\alpha + (1-\alpha)x_{wc}(t-1). \tag{5.1}$$

The α variable was chosen to be 0.05.

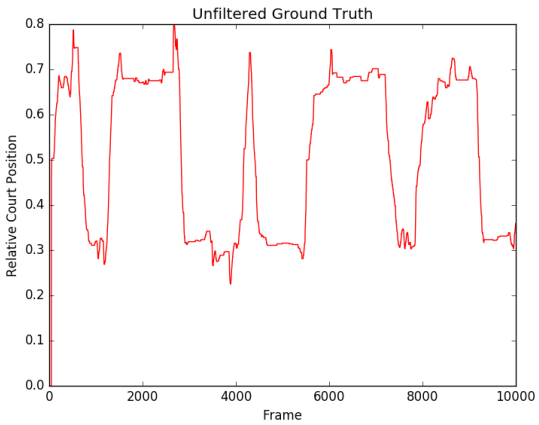
5.5 Trained Models for Estimation

All of the methods in this section used some form of training to find the best possible model variables given the input data. They used the stochastic gradient descent as optimising method and the Huber loss function as loss function. The momentum optimiser was also applied. The methods were presented in sections 2.2.3, 2.2.9 and 2.2.4 respectively.

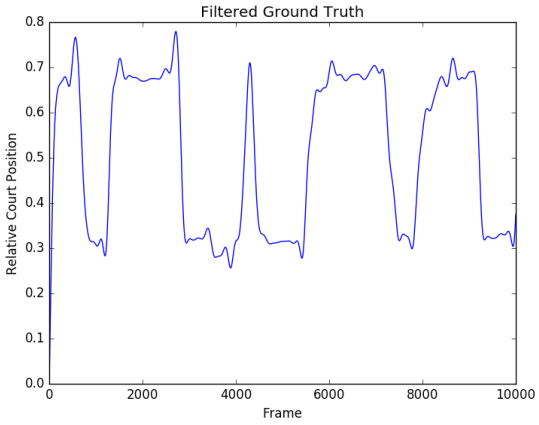
5.5.1 Dividing Data for Training, Testing and Validation

Cross validation was used to divide feature data into training, testing and validation sets. To make things easier, each of the recorded game halves, for example DFH, became one subset. This lead to the sizes of subsets to vary. This was because the game halves had different lengths. The feature data in the training set was divided into batches and a stochastic gradient descent method was implemented. The batch size for the stochastic gradient descent was 1000 samples large.

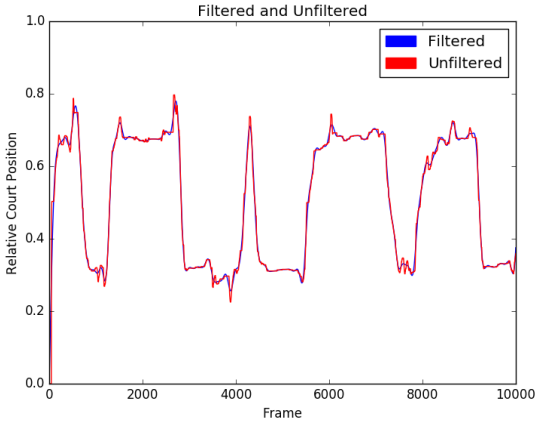
When using cross validation, there are many permutations of how the data sets can be divided. Let us assume that there are four data sets in total, A, B, C and D. Two are used for training, one for validation and one for testing. When A is the validation set, there are three different ways the model can be trained. All three remaining data sets can be used as test sets. This is shown in table 1. The same can be said for B and so on, creating in total 12 different possible ways to train a model.
In this project, there were 8 data sets in total for the training, which would generate 56 different combinations of testing and validation. To simplify the model evaluations, one model per test set



(a) Raw ground truth from data collection.



(b) Low pass filtered version of figure 5.5a. Quick movements were removed in this signal.



(c) The two signals on top of each other. The phase of the filtered signal is the same as the original signal but the sharp edges and fast changes have been changed to be more smooth signal after filtering.

Figure 5.5: The images shows an unfiltered and a filtered version a ground truth recording representing the game flow. The two signals over put on top of each other in figure 5.5c.

Table 1: Possible permutations for 4 data sets used for validation, testing and validation.

Set	Permutation 1	Permutation 2	Permutation 3
Validation	A	A	A
Testing	B	C	D
Training	C,D	B, D	B,C

was trained. Each data set was used as test set and validation set once each. This is shown in 2. This resulted in creating 8 models in total. The results from all models were averaged and became the final result of the model.

Table 2: Permutations for validation, testing and training used for the eights data sets

Set	Subset in model 1	Subsets in model 2	...	Subsets in model 8
Validation	A	B	...	H
Testing	B	C	...	A
Training	C-H	A + D-H	...	B-G

5.5.2 Initiation of Weights

The variables of the different models were initiated by sampling from a *Gaussian Distribution*. The mean of the distribution was set to be $\mu = 0$. The number of features, n, and the length of the court in pixels, $l_x = 640$, on the long side were used to set the standard deviation. The standard deviation was set to be $\sigma = \sqrt{\frac{2}{n * l_x}}$. This was used for all models. Different scalings of σ were tried and the one chosen was thought to give the best results.

5.5.3 Linear Regression Model

The first model created with training was a linear regression model. The model had one feature, the center of mass. If w_0 was the variable to be optimised and $x_c(t_0)$ was the center of mass at frame t_0, the model was

$$f(x_c, t_0) = w_0 x_c(t_0). \tag{5.2}$$

The second model created with linear regression was a model using the center of mass from three different time points. The input data to the model was the current center of mass as well as the center of mass 25 frames in the past and 25 frames in the future. w_0, w_1 and w_2 were the variables to be optimised. The model was

$$f(x_c, t_0) = w_0 x_c(t_0) + w_1 x_c(t_0 - 25) + w_2 x_c(t_0 + 25). \tag{5.3}$$

Bias was first tried in the models like described in equation (2.12), but it was not used due to results becoming less satisfactory compared to the models without having bias. The settings for training the linear regression model is found in table 3.

Table 3: Settings for the linear regression model

Setting	Value
Batch size	1000
Learning Rate	10^{-6}
Momentum	0.9
Decay Rate	0.95
Loss Function	Huber Loss Function
Huber Constant	0.25

To compute the loss of the model, the difference, e, between the ground truth, y, and the models in equations (5.2) and (5.3) was computed accordingly to:

$$e = y - f(\boldsymbol{w}). \tag{5.4}$$

The loss with respect to the model weights, $L(\boldsymbol{w})$, were computed for both linear regression models using the error found in (5.3) and *huber loss function* (see section 2.2.6). This is found in equation (5.5)

$$L(\boldsymbol{w}) = \begin{cases} \frac{1}{2}(e)^2, & \text{if } |e| < 0.25 \quad (5.5) \\ 0.25|e| - \frac{1}{2}0.25^2, & \text{otherwise.} \quad (5.6) \\ & (5.7) \end{cases}$$

5.6 Neural Networks

The neural networks were implemented with four different frame shifts, creating four models per network structure. The frame shifts implemented were 0 frames, 25 frames, 50 frames and 125 frames. The features in each frame were the same. None of the models had any bias, just weights.

The features used for modelling were an exponentially moving weighted average [24] of the center of mass, the velocity distribution and the track distribution. The velocity of a track was computed by taking the difference in x-positions for a track 10 frames apart. The intervals for the velocity used were the following (-25 to -3, -3 to -0.25, -0.25 to 0.25, 0.25 to 3, 3 to 25). The intervals were expressed in pixels. The intervals were chosen such that they should represent large negative, small negative, small positive and large positive velocities. The interval between -0.25 and 0.25 represented velocities close to zero. The intervals for the track distribution were divided into four equally large sizes of the court. Different features were tested and the set of features used were thought to best represent the gameplay. The change in center of mass was indirectly introduced for models using frame shifts larger than 0.

If the output from a neural network was $O(\boldsymbol{w})$ and the corresponding ground truth was y, an error was computed for each data point. The error, e, is found in (5.8)

$$e = y - O(\boldsymbol{w}) \quad (5.8)$$

The loss with respect to the network weights, $L(\boldsymbol{w})$, was computed using the error in equation (5.8) and the *huber loss function* accordingly to:

$$L(\boldsymbol{w}) = \begin{cases} \frac{1}{2}(e)^2, & \text{if } |e| < 0.25 \quad (5.9) \\ 0.25|e| - \frac{1}{2}0.25^2, & \text{otherwise.} \quad (5.10) \\ & (5.11) \end{cases}$$

The loss computation in (5.9) was used for all neural networks together with a stochastic gradient descent and a momentum optimiser.

5.6.1 Neural Network with Two Layers

A neural network with one input layer and one output layer was implemented. This model did not have any hidden layers. The properties of the network is shown in table 4. The network used the rectified linear as activation function.

Table 4: Settings for the two layer neural network

Setting	Value
Number of Nodes L^1	2000
Batch size	1000
Learning Rate	0.99
Momentum	0.9
Decay Rate	0.95
Loss Function	Huber Loss Function
Huber Constant	0.25

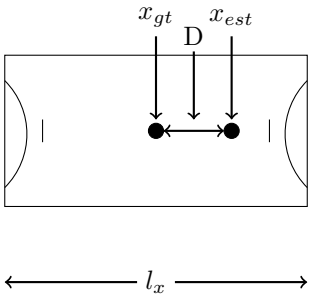


Figure 5.6: The norm distance, D, between the model estimation, x_{est}, and the ground truth, x_{gt}, was computed for all estimations. This was then divided by the court length, l_x, to create a normalised result D_n.

5.6.2 Neural Network with three layers

A neural network with one input layer, one hidden layer and one output layer was implemented. The properties of the neural network is found in table 5. The network used the rectified linear as activation function.

Table 5: Settings for the three layer neural network

Variable	Value
Number of Nodes L^1	2000
Numer of Nodes L^2	1000
Batch size	1000
Learning Rate	0.99
Momentum	0.9
Decay Rate	0.95
Loss Function	Huber Loss Function
Huber Constant	0.25

5.7 Post Processing of Result

The estimations, especially from the the neural networks, were noisy. To make the estimations more smooth, they were low-pass filtered using a sixth order Butterworth filter with cutoff frequency 0.15 [5]. To avoid altering the phase, the signals were forward-backward filtered with a padding length of 150 samples [14]. The estimations were finally thresholded to be between the two 7 meter penalty lines (see figure 3.1).

5.8 Validation Methods

5.8.1 Evaluation of Position estimation

The different models estimated a x-position on the handball court. This was normalised to be between 0 and 1 where the estimations corresponded to the left and right goals respectively. The estimated position reflected where an automatic recording system should film. The estimations were compared to the corresponding ground truth values. By computing the norm between the estimation and the ground truth, a measure of how close the estimation was to the ground truth was obtained. If x_{est} was the estimated value and x_{gt} was the ground truth and l_x was the long court side length, the normalised norm, D_n, was computed as:

$$D_n = \frac{||x_{est} - x_{gt}||}{l_x}. \tag{5.12}$$

The interpretation of difference can be seen in figure 5.6. If the difference from a computation in equation (5.12) was 0.2, it can be interpreted as the estimation being 20 % of the court length away from the ground truth value.

5.9 Evaluation of Training

To study the effect of the loss of the validation set dependant on the number of data subsets in the training set, a model was trained with between 1 and 6 data sets in the training set. The model used was the two layer neural network with a frame shift of 50 frames. The loss of the validation set and the loss of the training set are plotted vs the number of data sets in the training set. This done for the two data sets AFH and DAH and it is shown in figures 6.10 and 6.9.

6 Results

The results are presented as the distribution of the normalised norm difference, D_n between the ground truth value and the model estimation (see section 5.8.1). The results in the columns shows the normalised frequency of the differences that are within a certain percentage of the entire court length. For example, the second column shows a percentage of how many of the differences that are smaller than or up to 20 % of the entire court length. In total, 8 game halves were evaluated.

6.1 Results for the center of mass

The results for the center of mass can be found in table 6. The results are plotted in figure 6.1.

Table 6: Average results from cross validation of 8 center of mass estimations.

Estimator	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90%	100%
COM	0.3691	0.6449	0.8005	0.8899	0.9429	0.9744	0.9896	0.9964	0.9988	1.0

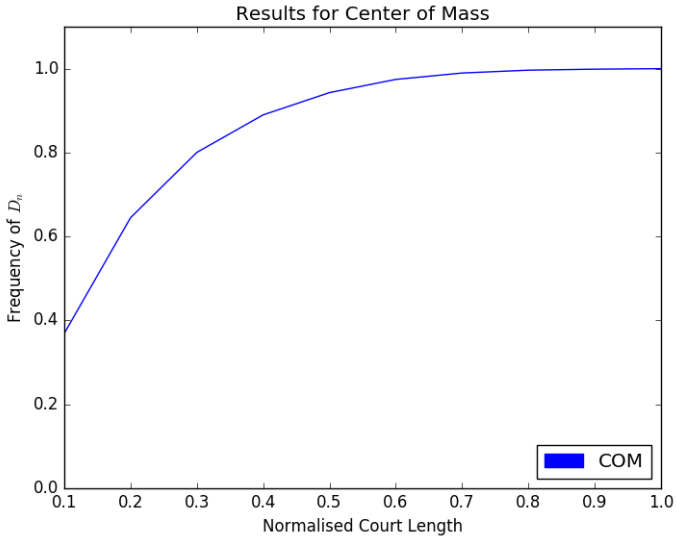


Figure 6.1: The results for the center of mass.

6.2 Results for Linear Regression

The results for the linear regression model using the center of mass from the current frame (Linreg 1) and one model using the center of mass with a frame shift of 25 frames (Linreg 3) can be found in table 7. The results are the average of the 8 cross validation models generated. The model with a frame shift of 25 frames had 3 covariates. The results are plotted in figure 6.2.

Table 7: Average results from cross validation of 8 center linear regression models.

Estimator	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90%	100%
Linreg 1	0.3235	0.5796	0.7614	0.8716	0.9329	0.9694	0.9892	0.9971	0.9993	1.0
Linreg 3	0.3209	0.5732	0.7607	0.8746	0.9344	0.9714	0.9901	0.9975	0.9996	1.0

6.3 Results for the Two Layer Neural Network

In figure 6.3, the results for the two layer neural network is shown. The average from the cross validation for the four frame shifts 0, 25, 50 and 125 are shown. For example, the frame shift 0 model has about 40 % of its estimations within 10% of the entire length of the court. The the data is also shown in table 8

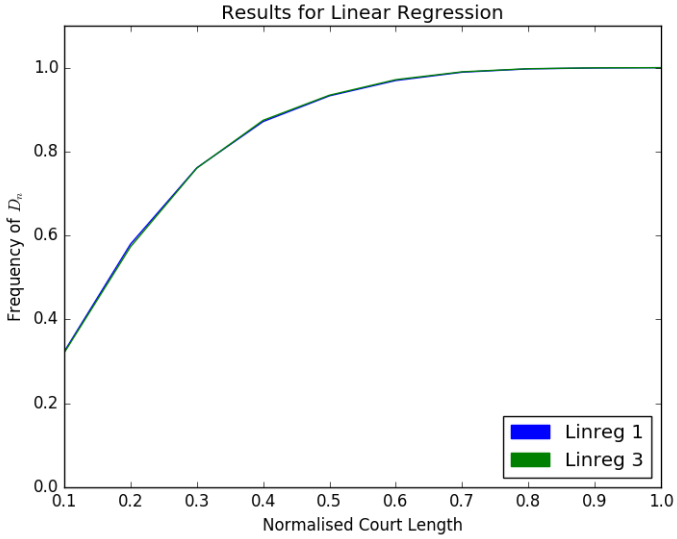


Figure 6.2: The results for the linear regression model using the center of mass from the current frame (Linreg 1) and the center of mass using a time shift of 25 frames (Linreg 3) as covariates.

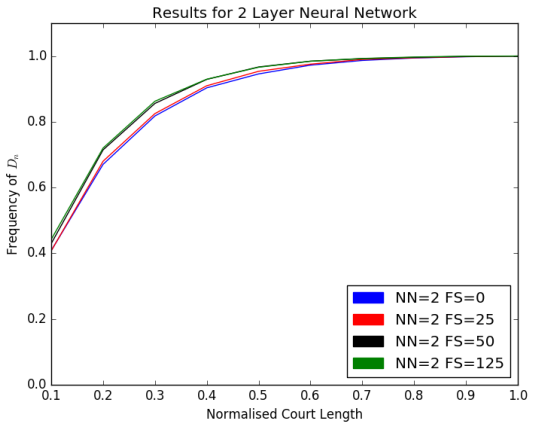


Figure 6.3: The results for two layer neural network with frame shifts 0, 25, 50 and 125 frames are plotted.

Table 8: Average results from cross validation of 8 two layer neural network models using frame shift 0, 25, 50 and 125 frames.

Frame Shift	10 %	20 %	30 %	40%	50 %	60 %	70 %	80 %	90 %	100 %
0	0.4085	0.6706	0.8178	0.9034	0.9457	0.9724	0.9867	0.9942	0.9981	1.0
25	0.4078	0.6801	0.825	0.9097	0.9536	0.9754	0.9897	0.9949	0.9983	1.0
50	0.4298	0.7147	0.8559	0.9291	0.9669	0.9844	0.9923	0.9961	0.9989	1.0
125	0.4424	0.7203	0.8626	0.9297	0.9661	0.9844	0.9927	0.9971	0.9997	1.0

6.4 Results for the Three Layer Neural Network

In figure 6.4, the results for the three layer neural network is shown. The average from the cross validation for the four frame shifts 0, 25, 50 and 125 are shown. For example, the frame shift 0 model has 37.9 % of its estimations within 10% of the entire length of the court. The data is also shown in table 9

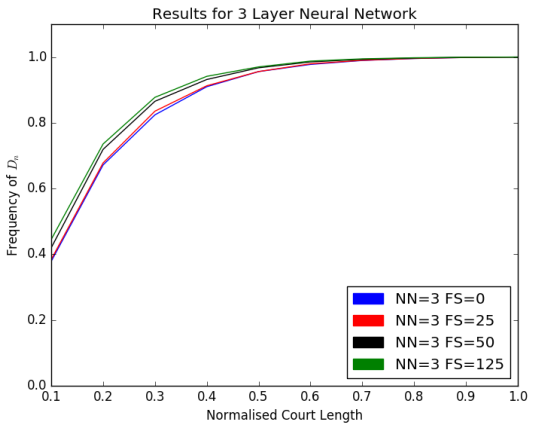


Figure 6.4: The results for three layer neural network with frame shifts 0, 25, 50 and 125 frames are plotted.

Table 9: Average results from cross validation of 8 three layer neural network models using frame shifts 0, 25, 50 and 125 frames.

Frame Shift	10 %	20 %	30 %	40%	50 %	60 %	70 %	80 %	90%	100%
0	0.379	0.6718	0.8242	0.9099	0.9559	0.9777	0.9897	0.9957	0.9989	1.0
25	0.3845	0.678	0.8354	0.9127	0.956	0.9795	0.9902	0.9958	0.9988	1.0
50	0.4201	0.7193	0.8654	0.9318	0.9674	0.9853	0.9933	0.9969	0.9993	1.0
125	0.4454	0.7356	0.8776	0.9414	0.9701	0.9878	0.9945	0.9979	0.9996	1.0

6.5 Post Processing

In figure 6.5, three estimations of the same match sequence is shown. In the top figure, a neural network estimation is shown. In the middle a thresholded estimation from the same neural network is shown. In the bottom figure a thresholded and low-pass filtered estimation from the same neural network is shown.

6.6 Plotting Results for Different Models

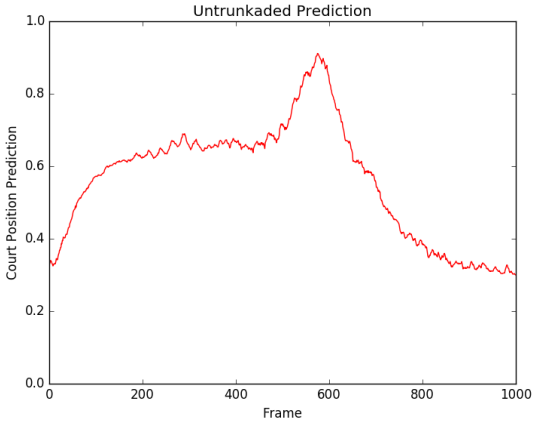
To analyse how well the neural network models performs with different features, estimations of a situation where the game is changing sides has been plotted. In figure 6.6, the estimations for frames shifts of 0, 25, 50 and 125 frames for the two layer neural network is plotted. In figure 6.7, the same situation for the three layer neural network is plotted.

In figure 6.8 a time out situation modelled by a three layer neural network with a frame shift of 50 frames is shown.

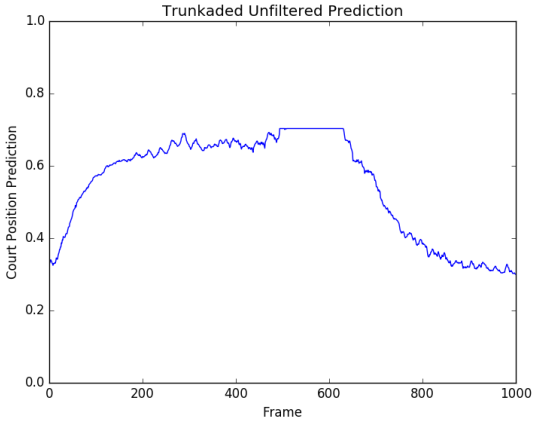
6.7 Evaluation of Training

The results of how the validation loss changes with the size of the training set is shown in figure 6.9. The data set DAH was used as validation data set and the data set DFH was used as testing set.

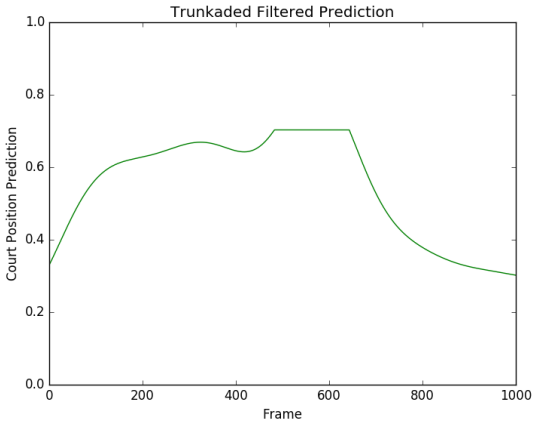
The same figure was plotted for another evaluation set, AFH, as well. This is shown in figure 6.10. The loss of the validation set decreases and the loss of the training set increases with increasing data in the training. The loss of the validation set is larger than the loss of the training set.



(a) Estimation without any post processing



(b) Thresholded estimation of the same signal as in 6.5a.



(c) Thresholded estimation and filtered version of 6.5a.

Figure 6.5: The three signals shows the same estimation with different post-rocessing. The first signal has no post-processing, the middle signal has been truncated and the bottom signal has been truncated and filtered.

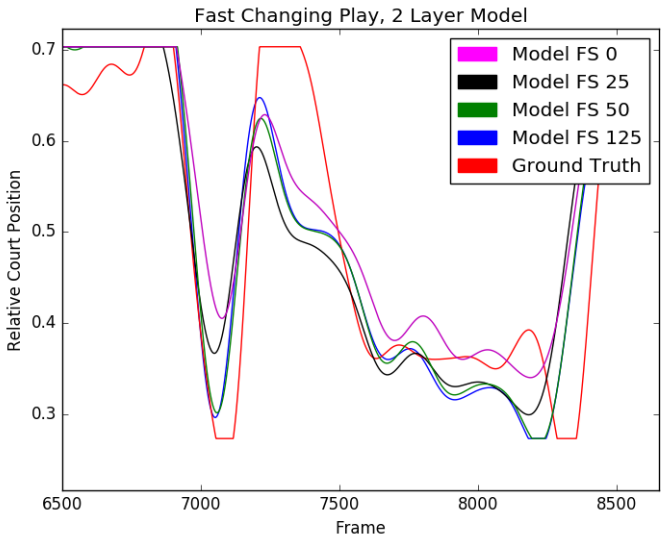


Figure 6.6: The plot shows a situation where the game switches sides back and forth. The results are plotted for the two layer neural network model and frame shift of 0 25, 50 and 125 frames.

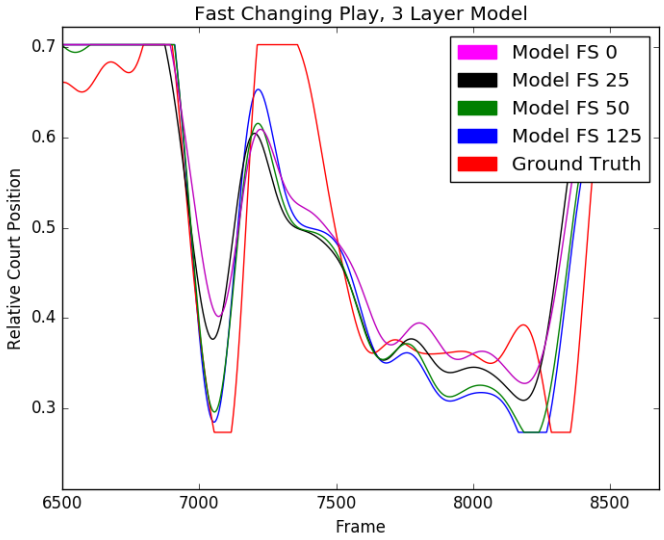


Figure 6.7: The plot shows the same situation as in figure 6.6 . The results are plotted for the three layer neural network model and frame shift of 0 25, 50 and 125 frames.

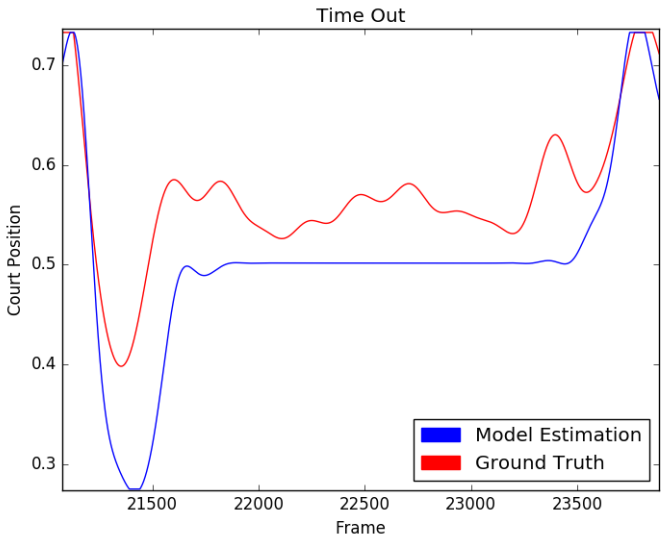


Figure 6.8: A time out situation. The model used for estimation was a three layer neural network model with a frame shift of 50 frames.

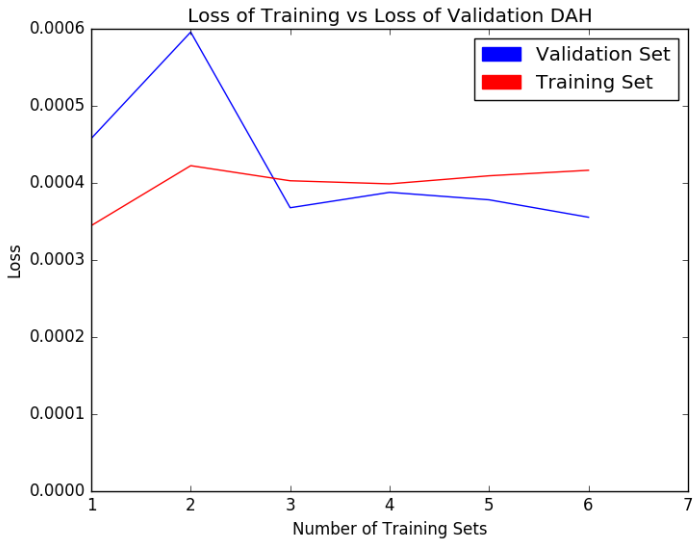


Figure 6.9: The loss for the training set and the loss for the validation set, DAH, are plotted. The loss of the validation set is smaller than the loss of the training set if more data is used in the training.

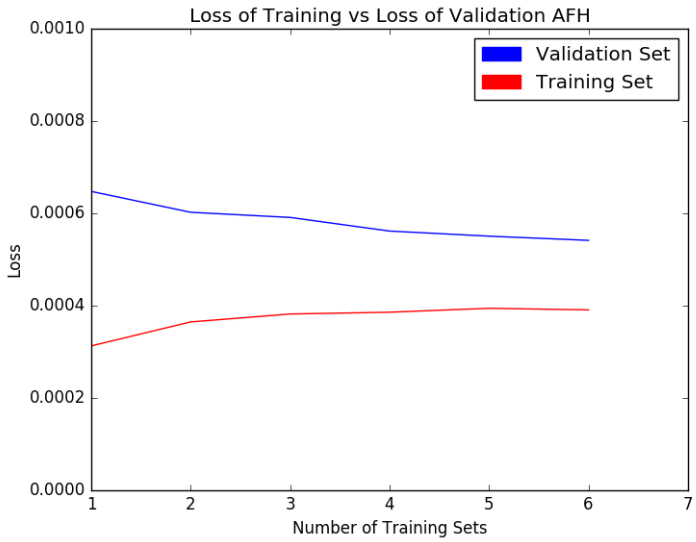


Figure 6.10: The loss for the training set and the loss for the validation set, AFH, are plotted. The loss for the validation set decreases over time and the loss of the training set increases with increasing training data. The training loss is still however smaller than what the validation loss is.

7 Discussion

7.1 Discussing the Methods

The center of mass was easy to implement. The linear regression took more time, but it was still quite fast to implement. The neural network models demanded more training before they could be used. From an implementation point of view, the center of mass estimator would be preferable to use.

The stochastic gradient descent method improved computational time significantly compared to the gradient descent method and, it was quite helpful when training the models. Given large enough batch sizes, each batch should theoretically represent the entire data set. Different batch sizes were tested and a batch size of 1000 samples was found to be large enough to give good results. One potential problem however, was that the different data sets differed from each other. This could have made the stochastic gradient descent approach a worse option than the gradient descent method.

The difference in the data sets becomes apparent when studying how the loss of a validation set changes with increasing number of data in the training set. Figure 6.9 shows that for one validation set, the validation loss is smaller compared to the loss of the training data set given enough training data. However in figure 6.10, the loss of the validation data set is larger than the loss of the training set. It appears that the loss of the validation set is declining and the loss of the training set is increasing, with increasing size of the training data. If more data would be added to the training, the loss of the validation set would most likely decrease to the level of the training data set loss.

This shows that there are large differences between data sets, which in turn would lead to differences in data batches used in stochastic gradient descent. Even if the data batches used in the stochastic gradient descent were not equal, the method is still preferred for this project compared to a normal gradient descent. This is because of the decrease in training time.

The KLT-tracker was able to produce data of how players were moving in the video content, but it did also produce unwanted track data. The filters used for handling this problem caused some information loss, though its hard to say how much. A more efficient way of removing these points are needed to make their negative influence on the methods as small as possible. One alternative to using the KLT-tracker, could be to model the background instead. This may make it easier to remove static background on the playing court. The methods could potentially be used complementary. It would probably have been better to use a more customised mask, which would have allowed tracks to be generated in the corners closest to the camera. Some of the wing-players data was lost with the mask used.

The post-processing improved the modelling results, especially the thresholding of the neural network estimations. These estimations had a tendency to "overestimate" the position, sometimes making estimations outside of the playing court. By defining a lower and an upper bound for the estimations, it was possible to get results closer to the ground truth. The forward/backward filtering produced a more smooth estimation. Some similar method would be preferable to implement if the models were to be used for broadcasting purposes.

Some measures for generalising the neural networks were studied. Both weight decay and dropout were implemented. The dropout did not improve the model, it did rather the opposite. Figures 6.9 and 6.10 shows that dropout would most likely have improved the estimations for the AFH data set but not for the DAH data set. Since dropout is only used in the hidden layers, it was not possible to implement in the two layer model. When implemented in the three layer model, the estimations became considerably worse for the validation data set. Dropout is generally implemented in layers towards the end of the network. If implemented when using only one hidden layer, too much information might be lost. The weight decay did not significantly change the estimations. A few different learning rates were tried, but it did not improve the results. The results of the network implementing weight decay were similar to the same network without weight decay. The only real difference was that it took longer for the model using weight decay to converge to a good solution. To implement weight decay more efficiently, a more detailed study of how learning rate affected the results would be needed.

7.2 Discussing the Results

The results for the center of mass shows that close to 36 % of the estimations are within 10 % of the entire playing court and 95 % are within 50 % of the playing court compared to the ground truth.

The center of mass produces reasonably good results, but the estimator would probably be hard to use for an broadcasting application. The main issue with using the center of mass as a camera estimator is, that it takes too long time for the estimator to react on quick game changes like fast breaks. This is a limitation of the estimator.

The linear regression model produced poor predictions when not using the center of mass as feature. That is the reason why only center of mass was used as covariate in the model. It did not make any difference to add two center of mass features to a linear regression model for the results. It is possible that the model could have been implemented to work better than what it did, but the model was thought to be too simple to track the game. One explanation why the linear regression model did not work as well as the center of mass could be that not enough measures for regularisation were used.

The neural network approach did improve the estimations quite considerably compared to the three previous models. The number of observations close to the ground truth increased. The model using just one input and one output layer improved the estimations for all feature sets used compared to both previous models. Using features from multiple frames increased estimation precision compared to the model using only features from one frame. There was not a lot of difference between the 0 and 25 frame shift results when studying the position evaluation. The big improvement came when introducing the 50 and 125 frame shifts. The main difference was that it was possible for the 50 and 125 frame shift models to capture game changes better. The 125 frame shift model seems to have improved the estimations slightly compared to the 50 frame shift models when studying the distance to the ground truth. This is reflected in figure 6.6. It is an important property of a model to capture fast breaks. In most other situations the different models made similar estimations.

The last model used was the three layer model. There is a bigger difference between the different frame shifts for this model compared to the two layer model. This could also have been caused by to little regularisation. The results for the 10 % court length evaluation are worse for the 0 and 25 frame shift compared to the same results of the two layer model. The three layer model is slightly better when studying the results for 125 frame shift.

The results for the training suggest that a model with one input layer and one output layer is sufficient to make estimation within 30% of the court length in around 85 % of the time. A model with one hidden layer did not improve the estimations significantly. The results also shows that the number of features is more important to the performance of the model than the depth of the neural network. If a model would be used for performing estimations, the two layer model would be prefered, since the model is faster to train and to make estimations compared to the three layer model. This could change if more data was added to the training.

A model using two hidden layers was also studied. It was however not possible to get this network to make good estimations. There are a number of potential reasons why it was not possible to do this. Different learning rates and different number of nodes of the network were studied. Due to lack of time, it was not possible to make a thorough investigation of the model architecture. The model was therefore disregarded.

7.3 Analysis of Misestimations

With the results of the estimations available, an interesting question becomes: " *What can be changed to improve the model?*". To answer this questions, it is needed to know which types of situations that are misestimated. After analysing the situations where the estimations differed the most from the ground truth, some types of situations were indentified.

The first situation, which the estimators had a hard time capturing, is when the goalkeeper has made a save and waits for a teammate to pass to. The ground truth shows the goalkeeper, but the estimator shows the players running towards the other goal. This situation is similar to a fast break. In fast breaks, players switches court sides after having been on one side for some time. It might be hard to find an estimator, which can discriminate between these two situations. And in reality, this might not be a big problem after all. In this situation the game can be seen as suspended.

Another common situation, where the estimator is often far from the ground truth, is at time outs. At time outs, the ground truth was positioned at the center of the court. This is shown in figure 6.8. The estimator however often showed one of the court sides. The side the estimator showed at time out could be dependant on how the players moved on their respective court sides. This is also a situation where it does not really matter where the camera is. There is not any match content

being missed by the estimator.

A situation on the same theme is when there has been a goal scored and the team who scored the goal is running towards its court side (Lets call it court side A). In this situation the game is suspended until the team with the ball makes a pass from the middle of the court. The ground truth will show the middle of the court but the estimator will show court side A. This is once again not a big issue. Once the game becomes active again the team with the ball will go on the offence where the estimator already is. There is not any interesting match content being missed.

It is common for some players to switch with a player on the bench between defence and offence. This is sometimes done when a team goes from offence to defence or vice versa, but it is also common for defensive players to be part of the offence for 15 - 20 seconds and then switch with a player on the bench. If a late switch happens and there has been a pause in the game, the switching of players sometimes makes the estimator believe that there is a change in the game. The estimator usually finds the play again once the game has been unpaused.

The last categories of misestimations are however more of a concern for the modelling. The first situation, which has been misestimated is when one team has been awarded with a penalty throw. Usually in this situation, one or two players of the team who recieved the penalty throw as well as the defending team, will be located at the defending team's court side (court A). The rest of the team who received the penalty throw will be on their own court side (court B) waiting to defend after the penalty has been thrown. The players on court B might move around a lot more than what the players on court A will. This could lead the system to think that the action is at court B when it really is at court A. To capture this situation better, it would be needed for the system to recognise that there has been a penalty throw awarded, and point the camera to the goal where the penalty is.

The last type of situation misestimated is fast breaks. It is more likely that the estimator does not capture fast breaks if the game has been switching sides rapidly prior to the fast break. In some situations the estimators were not able to track fast breaks due to certain players "staying at home" when their team was making a fast break play.

7.4 Improvements for Modelling

There are a number of improvements, that can be used for the models. The most critical improvement would most likely be to gather more data, to make the training more general. It would also be preferable to use more permutations for testing and validating the models. With more data in general, it would be easier to create test sets and validation sets which represents the entire data set better than what the validation sets and testing sets did in this project. Removing the situations not related to the game would most likely improve the results. As seen in figure 6.8, a time out takes almost two thousand frames. Because the network is trying to reduce the difference as much as possible, it might adjust to heavily for the time out. By removing the time outs and the other game unrelated situations from the training, these types of situations would no longer affect the training.

The network architecture such as number of nodes in each layer, activation function, loss function, learning rate and so forth were tested to a certain degree. The settings used in the modelling were the ones considered to be the best. But there was not too much effort put in to find the best settings. By finding more "optimal" settings for this, it might be possible to improve the training even further.

When performing analysis on the results from the different models, the number of features were the most important characteristic for how well the estimations went. By making a deeper analysis of how well the current features are working, the model could potentially be further improved. The analysis could consist of which of the current features than are contributing to the estimations, how the features should be sampled in past and future frames and what features that should be used from what frame.

The last analysis made was to study the situations where there was a big difference between the model estimation and the ground truth. When making a new model, these situations should be carefully studied and used to find new features to add to the existing features.

One other idea would be to change the optimisation. The optimisation implemented in this project has been focused on minimising the distance between the ground truth and the estimations. If the estimators would be used for broadcasting, it might be more important to capture all important

events in the broadcasted data, rather than having all events in the center of the image, which the current optimisation is aiming at. One idea is to have a threshold, where all estimations within this threshold would have an error of zero, and then have an increasing error for estimations further away from the ground truth. The threshold would be related to how much zoom that would be prefered. If the broadcasted content would have a zoom of 40 % of the court length, all estimations within 20 % of the court would have an error of 0, and all other errors would increase the further away the estimations would be from the ground truth.

7.5 Challenges in the Project

There have been different challenges and issues in this project. Some of these have been solved and some of them are possible to improve in future projects.

The collection of the ground truth was potentially an obstacle for the performance of the models. There is a potential problem with the precision of the mouse cursor. The consistency of the person who collected of ground truth is hard determine. Similar situations might have different ground truth. An explanation for this could be the focus level of the collector at the time. Preferably the ground truth would have been discussed with a handball coach before collection. Ground truth data is normally not filtered, but because of the circumstances of the collection of ground truth data, it was considered a good option to smooth the ground truth. The idea behind filtering the ground truth data was that it would represent camera movement better.

One problem with the ground truth, which was discovered after having trained a few models, was that there had been some error when collecting the ground truth for some matches. When the ground truth was reviewed it turned out that the ground truth did not represent the game at all in some cases. This data set was then discarded from the training. There might still have been ground truth which did not represent the actual gameplay. If this was the case, this would affect the training of the models.

The optimisation of the models has been hard to evaluate. The loss has been used to evaluate model performance. While this gives and indication, the viewer experience would be a better measure to find the best model. There are however two concerns, that makes the viewer experience hard use as evaluation method for the models. First of all, the viewer experience is impractical to use. It would require one or more persons to review each estimation and when that had been done, see if there was any consensus on what the estimations were missing and update the model accordingly. The second concern is that the viewer experience is quite subjective. It might not be possible to reach a consensus at all, which would make the viewer input impossible to use as evaluation method.

7.6 Future Development

Most of the models evaluated have had non-causal properties. Since the idea of a system, using one of the models for estimation, is to use it to record match content, the system must be able to operate at near real time speed. The next step in developing this system would be to decide which model and which features that should be used for estimation in "real time". The specification of how fast the system should be working would decide how many features that would be possible to use.

This algorithm proved itself to work fairly well in handball. The center of mass estimator was also tried on a football match and it showed great potential to be able to follow football matches as well. There are some differences between football and handball which could make it harder to use the same estimators for football. But given the promising results, it should be possible to develop a model which could follow and record football as well. If a system for football was to be developed, the market for the developed technology would increase.

8 Conclusion

The artificial neural networks proved to estimate the game flow well overall. There are some situations which are missed by the estimators. These are generally situations where the game changes fast from one side to the other. To produce an estimator better suited for the task, the optimisation process could be altered. Adding more data to the model training would most likely improve the models. The features are more important for the results of the neural networks model then what the number of layers in the network are. The neural network models could most likely be used for broadcasting purposes.

9 Sustainability

There are several aspects to consider when developing technology like the technology developed in this thesis. Like most technologies, the intension is to do good and to find a solution for a problem or a need, which is currently not satisfied. In this specific case it is about developing a tool which can help sports teams to improve their performance. It is also about broadcasting games, which otherwise would not be broadcasted. The automatic tracking system enables both of these possibilities. These features could be quite inspirational for both professionals and amateurs alike. But with all technological advancements, there is always a possibility that the intended use is not followed.

Lets start to focus on the positive effects of developing this system. The first thing to consider is how this technology could generate work opportunities and economic growth. Since there has been little literature on the subject, it is reasonable to think that the technology could generate work opportunities. If the demand for the technology would increase from what it is today, this research would improve the economic growth even more. The tracking itself could be used in other sports as well, so the potential market is quite big. Once the market grows, there will be work opportunities available which is beneficial to both society as well as for individuals.

There are many things which can be discussed when talking of video surveillance and tracking, regardless of the setting. There have been massive discussions in recent year of personal integrity and how much authority a government, for example, should have over its citizens. This is hopefully not a very big issue in the developed world, but there have been cases where governments have gathered information about its citizens without the knowledge of the people being surveilled. There are most likely more cases which have not been discovered yet either.

The system being develop in this project has a quite specified target use and it would probably not be very effective in other settings. But all development in an area drives the development further and it is hard to know what technologies might become one day. But even disregarding what future technologies might or might not do, this system is still used to film at locations where there is a possibility for the system to surveillance a large variation of people. Both potential spectators and athletes might not expect to be filmed, especially if cameras are installed in the celling and are not visible. That is important to keep in mind when using a system like this, especially at junior level matches.

One other aspect of this, which has been recently discussed in Sweden, is how young athletes are influenced by competitive sports. The Swedish Football Association (SvFF) has recently adopted a policy to not declare league winners for teams with players younger than thirteen years old[10] [7]. If there is an increase of information of the performance of junior teams there is a risk that both coaches and parents of young athletes might try to push the athletes harder to make them perform better. This could lead to increased pressure with severe consequences for the young athletes. The athletes themselves might also become to much involved in their sports.

To summarise the sustainability of an automatic tracking system: There is a demand for this type of technology. This can generate economic growth. There are however potential drawbacks both in personal integrity as well as social and psychological aspects for children and young adults.

References

[1] Baker S, Matthews I, *Lucas-Kanade 20 Years On: A Unifying Framework*, International Journal of Computer Vision 56(3), pp.221-255, 2004

[2] Beachemin S.S, Barron, J.L, *The computation of Optical Flow*, Dept of Computer Science, University of Western Ontario, Canada, 1995

[3] Bottou L, *Large-Scale Machine Learning with Stochastic Gradient Descent*, NEC Labs America, Princeton NJ 08542, USA

[4] Breitenstein M D, Reichlin F, Leibe B, Koller-Meier E, Van Gool L, *Robust Tracking-by-Detection using a Detector Confidence Particle Filter* ,2009 IEEE 12th International Conference on Computer Vision (ICCV)

[5] Butterworth S, *Theory of filter amplifiers*, Experimental wireless & the wireless engineer, 1930, pp. 536-541

[6] Cheung S-C S, Kamath C, *Robust techniques for background subtraction in urban traffic video*, Lawrence Livermore National Labratory 2007

[7] Dagens Nyheter, *Beslut: Inga segrare ska utses i barnfotboll*, url:http://www.dn.se/sport/fotboll/beslut-inga-segrare-ska-utses-i-barnfotboll/, visited 24-05-2016

[8] Dehghan A, Idrees H, Rishan Zamir A, Shah Mubarak, *Automatic Detection and Tracking of Pedestrians in Videos with Various Crowd Densities*, Pedestrian and Evacuation Dynamics 2012, Springer International Publishing Switzerland, 2014

[9] Fleet D J, Weiss Y, *Mathematical Models in Computer Vision: The Handbook*, Chapter 15, Springer, 2005, pp. 239-258

[10] Fotbolls Kanalen, *SvFF:s beslut - Inga segrare i ungdomsserierna* ,url:http://www.fotbollskanalen.se/allsvenskan/svffs-beslut—inga-segrare-i-ungdomsserierna/, visited 24-05-2016

[11] Ghahramani Z, *Unsupervised Learning*, Gatsby Computational Neuroscience Unit University College London, UK, 2004

[12] Goodfellow I, Bengio Y, Courville A, *Deep Learning*, 2016, url:http://www.deeplearningbook.org, visited: 26-05-2016

[13] Graves A, Liwicki M, Fernandez S, Bertolami R, Bunke H, Schmidhuber J, *A Novel Connectionist System for Unconstrained Handwriting Recongnition*

[14] Gustaffson F, *Determining the initial states in forward-backward filtering*, Transactions on Signal Processing, Vol. 46, pp. 988-992, 1996.

[15] Hagan M T, Demuth H B, Beale M H, De Jesus O , *Neural Network Design*, url:http://hagan.okstate.edu/NNDesign.pdf, visited: 26-05-2016

[16] Hawk Eye Home Page, url: http://www.hawkeyeinnovations.co.uk/sports/tennis, visited 07-06-2016

[17] Hinton G E, Osindera, Simon, Teh Y.W, *A fast learning algorithm for deep belief nets*, Neural Computation, Neural Computation 18, 1527?1554 (2006)

[18] Kim K, Chalidabhongse T H, Harwood D, Davis L, *Real-time foreground/background segmentation using codebook model*, 05

[19] Kotsiantis S.B, *Supervised Machine Learning: A Review of Classification Techniques*, Emerging Artificial Intelligence Applications in Computer Engineering, IOS Press, 2007

[20] Krogh A, Hertz J A, *A Simple Weight Decay Can Improve Generalization*, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 4, 1992

[21] Lucas B, Kanade T. 1981. *An iterative image registration technique with an application to stereo vision*, In Proceedings of the International Joint Conference on Artificial Intelligence, pp. pp.674-679.

[22] Läthen G, Andersson T, Lenz R, Borga M, *Momentum Based Optimization Methods for Level Set Segmentation*, 2009, Lecture Notes in Computer Science 5567: Scale Space and Variational Methods in Computer Vision, pp.124-136

[23] Nuygen D, Widrow B, *Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights*, 1990 IJCNN International Joint Conference on Neural Networks, 1990

[24] NIST/SEMATECH, *e-Handbook of Statistical Methods*, http://www.itl.nist.gov/div898/handbook/, 25-05-2016

[25] NumPy, url: http://www.numpy.org, visited 26-05-2016

[26] Open CV,http://opencv.org, accessed 25-05-2016

[27] Pers J, Kovacic S, *Computer Vision System for Tracking Players in Sports Games*, First Int Workshop on Image and Signal Processing and Analysis, June 14-15, 2000, Pula, Croatia

[28] Prechelt L, *Earlt Stopping – But When?*, Volume 7700, Lecture Notes in Computer Science pp. 53-67, 2012

[29] Python Homepage, url: https://www.python.org, visited June 7, 2016
[30] Sak H, Senior A, Beaufays F, *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocbavulary Speech Recognition*
[31] SciPy, url:http://docs.scipy.org/doc/scipy/reference/index.html, visited: 26-05-2016
[32] Seo Y, Choi S, Kim H, Hong K-S, *Where are the ball and players? Soccer game analysis with color-based tracking and image mosaick*, Image Analysis and Processing, Volume 1311 pp. 196-203
[33] Senior A, Heigold G, Aurelio Ranzato M, Yang K, *An Empirical Study of Learning Rates in Deep Neural Networks for Speech Recognition*, Google
[34] Shi J, Tomasi C, *Good Features To Track*, IEEE Conference on Computer Vision and Pattern Recognition, Seattle June 1994
[35] Shirazi H M, Vasconcelos N, *On the Design of Loss Functions for Classification: theory, robustness to outliers, and SavageBoost*,
[36] Spiideo Home Page, url: http://spiideo.com/index.html, visited 07-06-2016
[37] Srivastava N, Hinton G, Krizhevsky A, Sustskever I, Salakhutdinov R, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15 pp. 1929-1958, 2014
[38] Szeliski R, *Computer Vision: Algorithms and Applications*, Springer 2010, url:http://szeliski.org/Book/, visited: 26-05-2016
[39] *Tensor Flow*http://tensorflow.org, accessed June 7, 2016
[40] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, url:http://tensorflow.org/, 2015, visited: 26-05-2016
[41] Witten I H, Frank E, Hall M A, *Data Mining: Practical Machine Learning Tools and Techniques*, Third edition, Morgan Kaufmann Publishers - Elsevier 2011
[42] Zeiler M D, *ADADELTA: An Adaptive Learning Rate Method*
[43] Zeiler M.D, Ranzato M, Monga R, Mao M, Yang K, Le Q V, Nguyen P, Senior A, Vanhoucke V, Dean J, Hinton G E, *On Rectufued Linear Units for Speech Processing*

Master's Theses in Mathematical Sciences 2016:E16
ISSN 1404-6342

LUTFMA-3292-2016

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/