Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Soccer Analysis, 3D Pose Fusion using Extended Kalman Filtering

Bachelor Thesis

Tobias Buner

August 22, 2019

Advisors: Prof. Dr. M. Pollefeys, Dr. M. Oswald

Department of Computer Science, ETH Zürich

Abstract

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays sport is more than what it seems to be. Competitive physical activities at the highest level like Soccer, American Football, Tennis or the Olympic Games count to the most-watched television broadcasts. FIFA announced that more than half the world joined an official broadcast of the World Cup 2018 in Russia [1]. Due to the evolving technologies as high resolution cameras and super computers, it's not surprising that an interesting field of computer visualization has evolved from them.
Tools have been developed for the large number of viewers to bring them closer to the action. Intel True View end-to-end technology solution [2] makes it possible to render 360-degree replays to give the spectator the feeling of standing on the pitch himself. Intel uses for this 38 small 5K cameras in a ring around the venue which generates massive amounts of data. Other tools like the Vizrt's sports graphics and analysis tools [8] also provides data driven augmented reality but also enables the application for advanced analysis.

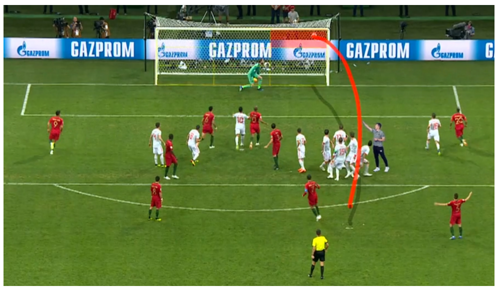

Figure 1.1: Freekick Analysis. **Source:** [8]

This leads to the second large field of application of these technologies. Advanced analysis through Visual Computing in sports is important for teams but also for the referee and the audience. The teams try to improve their tactics based on statistics or to learn from past mistakes. Interested spectators are also fascinated by facts from the statistics such as the number of shots at the goal of both teams in soccer. But it also has an impact on the game. The simplest example is the video assistant referee, he draws the referee's attention to a questionable situation and gives him the opportunity to look at the scene again. Another example with computer vision based technology is the Hawk-Eye ball tracking in tennis [6]. This allows the trajectory of a ball to be tracked purely from video in order to decide if the ball was in or out.

1.1 Focus of this Work

The swiss national soccer team approached the computer science department in order to explore state-of-the-art computer vision and visualization technology to analyze soccer games to generate and visualize player statistics and new performance measures of players or teams. The goal of this thesis was to extend the existing two-dimensional tracking system to three dimensions, thus opening up new possibilities. If, for example, not only the position on the pitch is known for each player, but also a three-dimensional skeleton model, this makes it possible to make a statement about the angle of view of the respective player.

1.1.1 Outlook

Available was the video material from multiple TV cameras which pan and zoom during the match and the two-dimensional tracking data. In chapter 3 I will explain each step in more detail, but to give you a rough overview:

- To fit skeleton models to all players in 3D space, I first had to estimate stable skeleton joint positions in 2D images using OpenPose [7].
- The camera calibrations were additionally necessary in order to be able to fuse the 2D positions.
- For the multi-view aggregation the extended Kalman Filter was used.

1.2 Thesis Organization

Chapter 2

Related Work

2.1 Body Tracking

2.2 Depth Estimation

2.3 Soccer on Your Tabletop

2.3.1 Calibration

2.3.2 Detectron

2.3.3 OpenPose

2.3.4 Depth map estimation

2.4 Filterpy

FilterPy is a Python library that implements a number of Bayesian filter like for example the Extended Kalman Filter. The author of the library also wrote a book named *Kalman and Bayesian Filter in Python* [3]. Because the free book was written using Ipython Notebook, it offers the perfect interactive guide to deal with bayesian filter. It starts with some simple filters up to the Kalman filter. It describes the implementation of the library with many examples and explains the mathematical background to understand why it works. For more information visit the the github page [5].

Chapter 3

Method

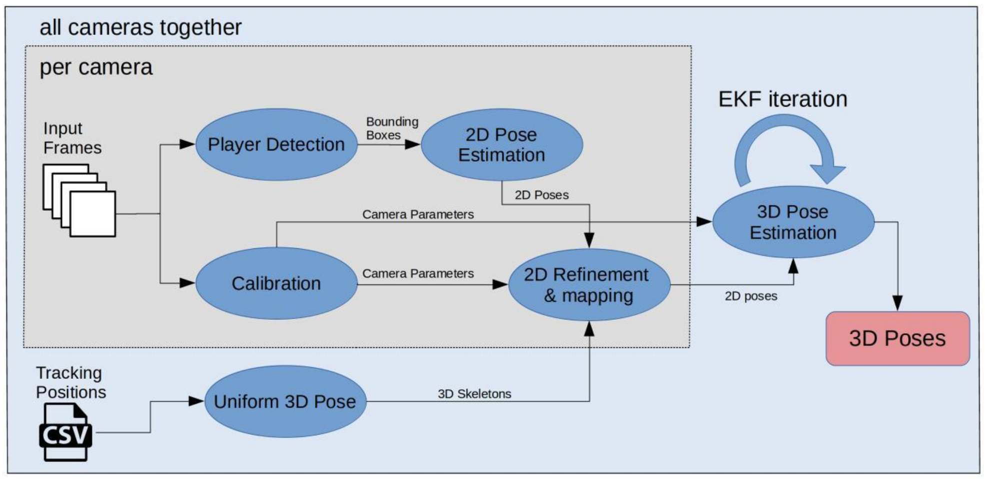

Figure 3.1: Overview of the workflow. Each step gets described in detail during this chapter. The gray area is the process for each camera separately while in the blue area the different cameras are merged.

3.1 Calibration

In order to work in three-dimensional space it is required to know how the pixels in a two-dimensional image relate to the three-dimensional world. For the camera model we consider the pinhole camera model for simplicity despite the fact that some cameras suffer from distortion. As mentioned in section 2.3.1 we were able to calibrate the cameras and got the corresponding camera matrix M which transforms homogeneous 3D world coordinates to homogeneous 2D image coordinates.

3.1.1 Camera Matrix

3.1.2 Line Detection

3.2 Player Detection

3.3 2D Pose Estimation

3.4 2D Refinement & mapping

3.5 3D Pose Estimation

Now I have to determine the points in 3D space given its projections onto the multiple images from the different camera perspectives. By knowing the camera matrix of the camera projection from 3D to 2D we know that each point in the image corresponds to a line in 3D space and every point on this line get projected on the same point in the 2D image as mentioned in section 3.1.

3.5.1 Triangulation

The method to solve this problem is called triangulation. In theory with perfect measurements this method is trivial. So let's assume we got a pair y_1 and y_2 of corresponding points in two different images. Remember that each point in the image belongs to a line in 3D space which are the green lines in the figure 3.2 below. The lines intersect at x which is the corresponding point in 3D space.

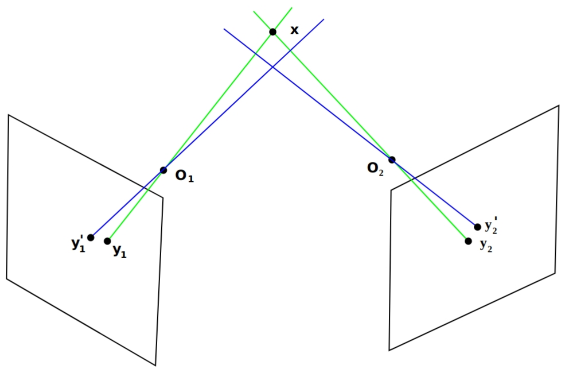

Figure 3.2: Triangulation visualization of two images with the camera's focal length O_1 and O_2. The ideal case with perfect measurements in green and the case of measurements with arbitrary accuracy in blue. **Source:** [10]

But in practice the the image points are measured with arbitrary accuracy.

This leads to the two points y'_1 and y'_2 in the figure 3.2. If you consider the corresponding lines of these two points in 3D space (blue), they must not even intersect and if they do it's not correctly the expected point x. Reasons for the deviations may be noisy measurements or an inaccurate camera calibration which lead to wrong focal lengths O_1 and O_2 and therefore to a wrong projection.

To find out which 3D point x' is the best estimate for the noisy measurements y'_1 and y'_2 you usually define an error measurement depending on x' and then minimize this error. Due to the fact that I don't always get all poses in 2D for the triangulation and the camera calibration is not error-free, I tried to use the Kalman Filter to do the triangulation. By constantly getting new measurements over time and the possibility of predicting the state further in time it seems to be a suitable application.

3.5.2 Extended Kalman Filter

The Kalman Filter (KF) is an iterative algorithm, based on the Bayesian Filter, that uses a series of measurements observed over time to find a good state estimation [3]. The filter estimates the state using a form of feedback control where the measured values come into play.

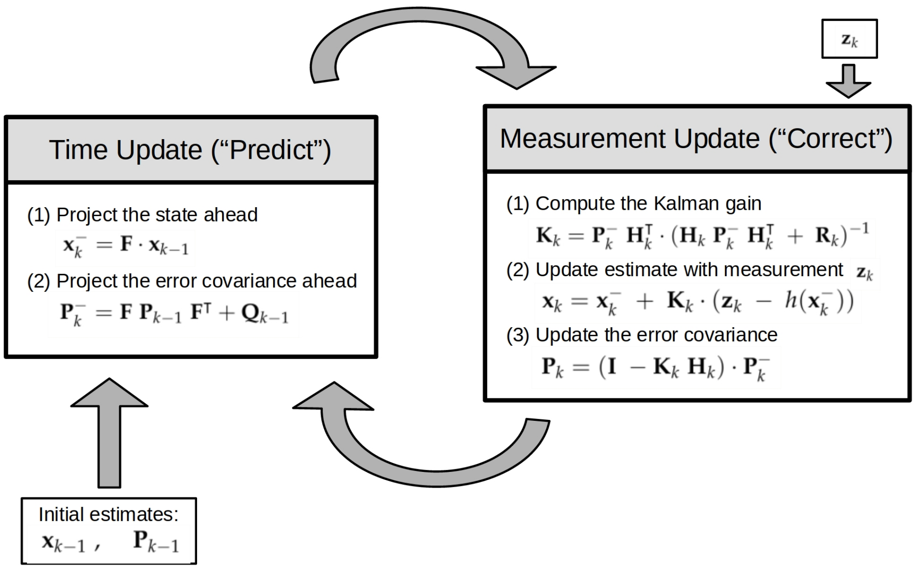
$\mathbf{z}_k$
Time Update ("Predict")
(1) Project the state ahead
$\mathbf{x}_k^- = \mathbf{F} \cdot \mathbf{x}_{k-1}$
(2) Project the error covariance ahead
$\mathbf{P}_k^- = \mathbf{F}\,\mathbf{P}_{k-1}\,\mathbf{F}^\mathsf{T} + \mathbf{Q}_{k-1}$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$\mathbf{K}_k = \mathbf{P}_k^-\,\mathbf{H}_k^\mathsf{T} \cdot (\mathbf{H}_k\,\mathbf{P}_k^-\,\mathbf{H}_k^\mathsf{T} + \mathbf{R}_k)^{-1}$
(2) Update estimate with measurement $\mathbf{z}_k$
$\mathbf{x}_k = \mathbf{x}_k^- + \mathbf{K}_k \cdot (\mathbf{z}_k - h(\mathbf{x}_k^-))$
(3) Update the error covariance
$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\,\mathbf{H}_k) \cdot \mathbf{P}_k^-$
Initial estimates:
$\mathbf{x}_{k-1}$, $\mathbf{P}_{k-1}$

Figure 3.3: Complete picture of the calculations during the two-step process of the Extended Kalman Filter. For the Initialization the values of the first state vector and the error covariance are needed. A new measurement vector is given to the filter for each Measurement Update step. **Source:** Notation slightly adapted from [9]

The algorithm works in a two-step process as shown below in figure 3.3 which is repeated for every time step which is in my case the next frame. In the prediction step, the Kalman Filter projects the current state variable forward in time using the process model and projects the error covariance ahead. In the second step, called the update step, the previously estimated state variable gets updated using the measurement residual weighted with the so called Kalman gain. Additionally the error covariance is updated. and a new external measurement as described in section x.

To update the state variable the measurement residual is needed which is computed using the current state variable and a new measurement $\mathbf{z}_k$ as described in section 3.4. But the state variable is not part of the measurement space, therefore we have to project the state variable to the measurement space. This projection is basically a camera projection and is not linear due to a division, which is the reason why I use the Extended Kalman Filter (EKF). The reason why I work in measurement space and not in state space is because the measurements are not simply invertible.

3.5.3 Designing the EKF

In the figure 3.3 and in the following text I will use the notation from the book *An Introduction to the Kalman Filter* [9] slightly adapted to the filterpy [3] notation. I will omit the time subscript k for the definitions of multiple variables in this section for the sake of clarity. For coordinates x, y, z I use the subscript w for the 3D space and c_k for the 2D space of the k-th camera. The state and measurement variable $\mathbf{x}, \mathbf{z}$ are written in bold to distinguish from the x and z coordinates. The last note is about the camera matrices which change with every frame. So when I mention them I always refer to the current calibration.

State and Measurement

The EKF is designed to track one person using five different cameras. As mentioned in section 3.3 the OpenPose model *COCO* consists of 15 key-points. For the i-th keypoint the corresponding part of the state vector includes the position in 3D space and their velocities:

$$\mathbf{x}^i := \begin{bmatrix} x_w^i & y_w^i & z_w^i & v_x^i & v_y^i & v_z^i \end{bmatrix}^\intercal, \ \forall i \in \{0, 1, ..., 17\} \tag{3.1}$$

All 18 keypoints together form the state vector:

$$\mathbf{x} := \begin{bmatrix} \mathbf{x}^0 & \mathbf{x}^1 & \cdots & \mathbf{x}^{17} \end{bmatrix}^\intercal \tag{3.2}$$

The measurement vector $\mathbf{z}$ includes for the i-th keypoint the image coordinates x, y for all five cameras:

$$\mathbf{z}^i := \begin{bmatrix} x_{c_0}^i & y_{c_0}^i & x_{c_1}^i & y_{c_1}^i & \cdots & x_{c_4}^i & y_{c_4}^i \end{bmatrix}^\intercal, \ \forall i \in \{0, 1, ..., 17\} \tag{3.3}$$

The complete measurement vector is again for all 18 keypoints:

$$\mathbf{z} := \begin{bmatrix} \mathbf{z}^0 & \mathbf{z}^1 & \cdots & \mathbf{z}^{17} \end{bmatrix}^\mathsf{T} \tag{3.4}$$

Process Model

For the state propagation **F** in the Predict step I use a simple linear model that does not change over time. For the sake of simplicity i assume constant velocity. For several reasons this is not the best choice for modeling the real world. The players move in different directions, the head keypoints behave completely differently to the hand keypoints etc. But due to the high frame rate one can neglect these aspects because the movement from one frame to the next is minimal. The linear motion model for the x coordinates in 3D space is as follows $x = x + v_x \cdot \Delta t$. The time step Δt represents the time between two frames. The resulting state transition model F' propagates the state vector for the i-th keypoint x^i further in time,

$$\mathbf{F}^i := \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \ \forall i \in \{0, ..., 17\} \tag{3.5}$$

so the resulting complete state propagation matrix **F** is a diagional matrix expressed using multiple submatrices $\mathbf{F}^i$:

$$\mathbf{F} := \begin{bmatrix} \mathbf{F}^1 & & \\ & \ddots & \\ & & \mathbf{F}^{17} \end{bmatrix} \tag{3.6}$$

This results in the first equation of our extended Kalman filter to project the state ahead:

$$\mathbf{x}_k^- = \mathbf{F} \cdot \mathbf{x}_{k-1} \tag{3.7}$$

Together with the next equation 3.8, those two form the prediction step of the EKF. The estimation error covariance **P** gets projected from the previous time step $k-1$ to the current time step k.

$$\mathbf{P}_k^- = \mathbf{F}\ \mathbf{P}_{k-1}\ \mathbf{F}^\mathsf{T} + \mathbf{Q}_{k-1} \tag{3.8}$$

To start you have to initialize the covariance matrix **P** and then the filter takes care of updating its value. We know that the position and velocity are correlated but due to the fact that we use constant velocity in our model we just initialize **P** to zero. The matrix is a square matrix of the same dimension as the state vector.

Noise

In both equation 3.8 and 3.11 we add a covariance matrix derived from white Gaussian noise. **Q** is a square matrix of the same dimension as the state vector $\mathbf{x}_k$ and is called the process noise covariance. The matrix **R** is also a square matrix with the dimension of the measurement vector $\mathbf{z}_k$ and is called the measurement noise covariance. They're called white Gaussian noise because they are independent of each other with normal probability distribution. During the EKF iteration they stay constant with values $\sigma_q = 0.3$ and $\sigma_r = 0.02$:

$$\mathbf{Q} := \begin{bmatrix} \sigma_q & & \\ & \ddots & \\ & & \sigma_q \end{bmatrix} \qquad \mathbf{R} := \begin{bmatrix} \sigma_r & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \tag{3.9}$$

The reason for the higher process noise is the inaccurate process model with constant velocity. So the filter tries to rely more on the measurements than on the prediction.

Kalman gain and Error covariance

The residual gets added to the predicted state variable $\mathbf{x}_k^-$ as described in the section below. But the residual is weighted by the Kalman gain. Higher values for the Kalman gain indicate that we give more trust to the measurement and a lower values indicate that we rely more on the prediction. So if the measurement uncertainty $\mathbf{R}_k$ is smaller with respect to the error covariance $\mathbf{P}_k^-$ we rely more on the measurement. The computation for the Kalman gain takes place in the Measurement Update step:

$$\mathbf{K}_k = \mathbf{P}_k^- \, \mathbf{H}_k^\intercal \cdot (\mathbf{H}_k \, \mathbf{P}_k^- \, \mathbf{H}_k^\intercal \; + \; \mathbf{R}_k)^{-1} \tag{3.10}$$

Another equation in the Measurement Update step is the update of the error covariance $\mathbf{P}_k$.

$$\mathbf{P}_k = (\mathbf{I} \; - \mathbf{K}_k \, \mathbf{H}_k) \cdot \mathbf{P}_k^- \tag{3.11}$$

In contrast to **Q** and **R** which are both constant, the error covariance $\mathbf{P}_k$ and the Kalman gain $\mathbf{K}_k$ will stabilize quickly and then remain constant as described in [9]. These two equations involves the matrix **H** which is explained in the section 3.5.3 below.

Measurement Model

The measurement model is designed through a function h which is responsible to relate the state vector $\mathbf{x}_k$ to the measurement $\mathbf{z}_k$. As I mentioned in section 3.5.2 this function projects the state variable into the measurement space in order to be able to compute the residual which leads to this equation for the Measurement Update step:

$$\mathbf{x}_k = \mathbf{x}_k^- + \mathbf{K}_k \cdot (\mathbf{z}_k - h(\mathbf{x}_k^-)) \tag{3.12}$$

I will explain right away in detail how the function h looks like and what the related matrix $\mathbf{H}_x$ is. The filterpy implementation [4] of the EKF requires for the update step the function h with the arguments $\mathbf{x}_k$ and the camera matrices for all five cameras. The output of this function is a vector which looks exactly like the measurement vector $\mathbf{z}_k$ but don't get confused, it's just the same space and no new measurement vector:

$$h(\mathbf{x}_k^-, \text{camera matrices}) = \mathbf{z}_{transformed} \tag{3.13}$$

As mentioned in section 3.1.1 a homogeneous point in the 3D space can be projected to the image screen using the camera matrix $\mathbf{M}$. I will explain this step for the x_w, y_w, z_w coordinates of the state vector $\mathbf{x}_k$ for one keypoint to map to the corresponding measurement space of one specific camera j. I will omit the time subscript k in this section for the sake of understanding.

$$\mathbf{M}_{c_j} \cdot \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \mathbf{A}_{c_j} \left[\mathbf{R}_{c_j} \mid \mathbf{T}_{c_j} \right] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.14}$$

$$= \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.15}$$

$$= \begin{bmatrix} x'_{c_j} \\ y'_{c_j} \\ z'_{c_j} \end{bmatrix} \tag{3.16}$$

In order to get the correct scale and dimension for the screen coordinates x_{c_j}, y_{c_j} one have to divide the result from equation 3.16 by z_{c_j} and drop the last entry for the z coordinate. This forms a part of the function h:

$$\hat{h}\left(\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}, \mathbf{A}_{c_j}, \mathbf{R}_{c_j}, \mathbf{T}_{c_j} \right) = \begin{bmatrix} x_{c_j} \\ y_{c_j} \end{bmatrix} \tag{3.17}$$

$$= \begin{bmatrix} \frac{x'_{c_j}}{z'_{c_j}} \\ \frac{y'_{c_j}}{z'_{c_j}} \end{bmatrix} = \begin{bmatrix} \frac{f_x \cdot (r_{1,1} \cdot x_w + r_{1,2} \cdot y_w + r_{1,3} \cdot z_w + t_x)}{r_{3,1} \cdot x_w + r_{3,2} \cdot y_w + r_{3,3} \cdot z_w + t_z} + x_0 \\ \frac{f_y \cdot (r_{2,1} \cdot x_w + r_{2,2} \cdot y_w + r_{2,3} \cdot z_w + t_x)}{r_{3,1} \cdot x_w + r_{3,2} \cdot y_w + r_{3,3} \cdot z_w + t_z} + y_0 \end{bmatrix} \tag{3.18}$$

To make it clear once again, this was just to transform the 3D coordinates for one keypoint into the image space of one camera. So for the i-th keypoint of the state vector $\mathbf{x}_k$ we drop the velocities and transform the 3D coordinates to screen coordinates for each camera. This gives us a bigger part of h:

$$h_i(\begin{bmatrix} x_w^i \\ y_w^i \\ z_w^i \end{bmatrix}, \text{camera matrices}) = \begin{bmatrix} \hat{h}(\begin{bmatrix} x_w^i \\ y_w^i \\ z_w^i \end{bmatrix}, \mathbf{A}_{c_0}, \mathbf{R}_{c_0}, \mathbf{T}_{c_0}) \\ \vdots \\ \hat{h}(\begin{bmatrix} x_w^i \\ y_w^i \\ z_w^i \end{bmatrix}, \mathbf{A}_{c_4}, \mathbf{R}_{c_4}, \mathbf{T}_{c_4}) \end{bmatrix} \tag{3.19}$$

Finally we can formulate the whole function h:

$$h(\mathbf{x}_k^-, \text{camera matrices}) = \begin{bmatrix} h_0(\begin{bmatrix} x_w^0 \\ y_w^0 \\ z_w^0 \end{bmatrix}, \text{camera matrices}) \\ \vdots \\ h_{17}(\begin{bmatrix} x_w^{17} \\ y_w^{17} \\ z_w^{17} \end{bmatrix}, \text{camera matrices}) \end{bmatrix} \tag{3.20}$$

$$= \mathbf{z}_{transformed} \tag{3.21}$$

Jacobian Matrix H

By the fact that the relationship between state space and the measurement space is a nonlinear function h we need to linearize it by evaluating its partial derivatives with respect to the state vector $\mathbf{x}_k$ like a Taylor series. The resulting matrix is the Jacobian matrix $\mathbf{H}_k$. I will explain how the derivations of one screen coordinate pair (the i-th keypoint) like in equation 3.17 for the camera j looks like:

$$\mathbf{H}_{c_j}^i = \begin{bmatrix} \frac{\partial x_{c_j}^i}{\partial x_w^i} & \frac{\partial x_{c_j}^i}{\partial y_w^i} & \frac{\partial x_{c_j}^i}{\partial z_w^i} & \frac{\partial x_{c_j}^i}{\partial v_x^i} & \frac{\partial x_{c_j}^i}{\partial v_y^i} & \frac{\partial x_{c_j}^i}{\partial v_z^i} \\ \frac{\partial y_{c_j}^i}{\partial x_w^i} & \frac{\partial y_{c_j}^i}{\partial y_w^i} & \frac{\partial y_{c_j}^i}{\partial z_w^i} & \frac{\partial y_{c_j}^i}{\partial v_x^i} & \frac{\partial y_{c_j}^i}{\partial v_y^i} & \frac{\partial y_{c_j}^i}{\partial v_z^i} \end{bmatrix} \tag{3.22}$$

The matrix $\mathbf{H}_{c_j}^i$ would be much wider because of the partial derivatives for the other state variable but they are all zero so we just note this one and build the Jacobian $\mathbf{H}$ with multiple block matrices. So the whole corresponding block of the Jacobian for the *i*-th keypoint is presented:

$$\mathbf{H}^i = \begin{bmatrix} \mathbf{H}^i_{c_0} \\ \mathbf{H}^i_{c_1} \\ \mathbf{H}^i_{c_2} \\ \mathbf{H}^i_{c_3} \\ \mathbf{H}^i_{c_4} \end{bmatrix} \tag{3.23}$$

And finally we can formulate the complete Jacobian $\mathbf{H}_k$ with the partial derivatives for the entire state vector, note that in the steps above i removed the subscript k for readability.

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{H}^0 & & \\ & \ddots & \\ & & \mathbf{H}^{17} \end{bmatrix} \tag{3.24}$$

Like the function h the filterpy [4] implementation also requires the Jacobian of h as input for the update step. So the Jacobian $\mathbf{H}_x$ is also a function with the same arguments as the function h, namely the state vector $\mathbf{x}_k$ and the camera matrices. For the readability I first define some variables which are used afterwards:

$$r_1\mathbf{x}^i := r_{1,1} \cdot x^i_w + r_{1,2} \cdot y^i_w + r_{1,3} \cdot z^i_w \tag{3.25}$$

$$r_2\mathbf{x}^i := r_{2,1} \cdot x^i_w + r_{2,2} \cdot y^i_w + r_{2,3} \cdot z^i_w \tag{3.26}$$

$$r_3\mathbf{x}^i := r_{3,1} \cdot x^i_w + r_{3,2} \cdot y^i_w + r_{3,3} \cdot z^i_w \tag{3.27}$$

So what's left are the partial derivatives for which i'm going to explain the ones from equation 3.22, where the others are analogous.

$$\frac{\partial x^i_{c_j}}{\partial x^i_w} = \frac{f_x \cdot r_{1,1}}{r_3\mathbf{x}^i + t_z} - \frac{f_x \cdot r_{3,1}(r_1\mathbf{x}^i + t_x)}{(r_3\mathbf{x} + t_z)^2} \tag{3.28}$$

$$\frac{\partial x^i_{c_j}}{\partial y^i_w} = \frac{f_x \cdot r_{1,2}}{r_3\mathbf{x}^i + t_z} - \frac{f_x \cdot r_{3,2}(r_1\mathbf{x}^i + t_x)}{(r_3\mathbf{x} + t_z)^2} \tag{3.29}$$

$$\frac{\partial x^i_{c_j}}{\partial z^i_w} = \frac{f_x \cdot r_{1,3}}{r_3\mathbf{x}^i + t_z} - \frac{f_x \cdot r_{3,3}(r_1\mathbf{x}^i + t_x)}{(r_3\mathbf{x} + t_z)^2} \tag{3.30}$$

$$\frac{\partial y^i_{c_j}}{\partial x^i_w} = \frac{f_y \cdot r_{2,1}}{r_3\mathbf{x}^i + t_z} - \frac{f_y \cdot r_{3,1}(r_2\mathbf{x}^i + t_y)}{(r_3\mathbf{x} + t_z)^2} \tag{3.31}$$

$$\frac{\partial y^i_{c_j}}{\partial y^i_w} = \frac{f_y \cdot r_{2,2}}{r_3\mathbf{x}^i + t_z} - \frac{f_y \cdot r_{3,2}(r_2\mathbf{x}^i + t_y)}{(r_3\mathbf{x} + t_z)^2} \tag{3.32}$$

$$\frac{\partial y^i_{c_j}}{\partial z^i_w} = \frac{f_y \cdot r_{2,3}}{r_3\mathbf{x}^i + t_z} - \frac{f_y \cdot r_{3,3}(r_2\mathbf{x}^i + t_y)}{(r_3\mathbf{x} + t_z)^2} \tag{3.33}$$

$$\frac{\partial x^i_{c_j}}{\partial v^i_x} = \frac{\partial x^i_{c_j}}{\partial v^i_y} = \frac{\partial x^i_{c_j}}{\partial v^i_z} = \frac{\partial y^i_{c_j}}{\partial v^i_x} = \frac{\partial y^i_{c_j}}{\partial v^i_y} = \frac{\partial y^i_{c_j}}{\partial v^i_z} = 0 \tag{3.34}$$

State Initialization

TODO

All together result in the final EKF. For a good overview i repeat the five equations for the filter here:

The Update ("Predict"):
(1) Projet the state ahead:

$$\mathbf{x}_k^- = \mathbf{F} \cdot \mathbf{x}_{k-1} \tag{3.35}$$

(2) Project the error covariance ahead:

$$\mathbf{P}_k^- = \mathbf{F}\ \mathbf{P}_{k-1}\ \mathbf{F}^\mathsf{T} + \mathbf{Q}_{k-1} \tag{3.36}$$

Measurement Update ("Correct"):
(1) Compute the Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_k^-\ \mathbf{H}_k^\mathsf{T} \cdot (\mathbf{H}_k\ \mathbf{P}_k^-\ \mathbf{H}_k^\mathsf{T}\ +\ \mathbf{R}_k)^{-1} \tag{3.37}$$

(2) Update estimate with measurement $\mathbf{z}_k$:

$$\mathbf{x}_k = \mathbf{x}_k^-\ +\ \mathbf{K}_k \cdot (\mathbf{z}_k\ -\ h(\mathbf{x}_k^-)) \tag{3.38}$$

(3) Update the error covariance:

$$\mathbf{P}_k = (\mathbf{I}\ - \mathbf{K}_k\ \mathbf{H}_k) \cdot \mathbf{P}_k^- \tag{3.39}$$

Chapter 4

Evaluation and Discussion

Chapter 5

Conclusion

5.1 Future Work

Appendix A

Appendix

Bibliography

[1] FIFA. 2018 World Cup Russia. https://www.fifa.com/worldcup/news/more-than-half-the-world-watched-record-breaking-2018-world-cup. [3-August-2019].

[2] Intel. True view. https://www.intel.co.uk/content/www/uk/en/sports/technology/true-view.html. [3-August-2019].

[3] Roger Labbe. *Kalman and Bayesian Filters in Python*. 2018.

[4] Labbe, Roger. FilterPy Documentation. https://filterpy.readthedocs.io/en/latest/. [3-August-2019].

[5] Labbe, Roger. Github. https://github.com/rlabbe/filterpy. [3-August-2019].

[6] Michael Bane. How Hawk-Eye ball tracking can improve tennis performance. https://phys.org/news/2015-01-hawk-eye-ball-tracking-tennis.html. [3-August-2019].

[7] OpenPose. Github. https://github.com/CMU-Perceptual-Computing-Lab/openpose. [3-August-2019].

[8] Vizrt. The ultimate tools for live sports production. https://www.vizrt.com/sports. [3-August-2019].

[9] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*. 2006.

[10] Wikipedia. Triangulation (computer vision). https://en.wikipedia.org/wiki/Triangulation_(computer_vision). [3-August-2019].

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.