KTH Computer Science
and Communication

Real-time filtering for human pose estimation using multiple Kinects

VILLE STOHNE
villes@kth.se

Degree project report
August 2014
School of Computer Science and Communication
Supervisor: Magnus Burénius
Examiner: Stefan Carlsson

TRITA xxx yyyy-nn

Abstract

This Master's thesis proposes a working approach to combining data from multiple Kinect depth sensors in order to create stable pose estimates of a human user in real-time. The multi-camera approach is shown to increase the interaction area in which the user can move around, it gives more accurate estimates when the user is turning and it reduces issues with user occlusion compared to single-camera setups. In this report we implement and compare two different filtering techniques, particle filters and Kalman filters. We also discuss different approaches to fuse data from multiple depth sensors based on the quality of the observations from the different sensors along with techniques to improve estimates such as applying body constraints. Both filtering approaches can be run on a normal laptop in real-time, i.e. 30 Hz. When requiring real-time performance, the computationally efficient Kalman filter performs better than the particle filter overall in terms of stable estimations and performance. The quality of the particle filter is highly dependent on the number of particles that can be used before the frame rate drops below 30 frames per second. The implemented system provides a stable, fast and cost-efficient setup for motion capture and pose estimations of human users. There are important applications in virtual reality and to some extent also 3D games and rendered films that could benefit from the approaches discussed in this report.

Realtidsfiltrering av data från flera Kinectkameror för estimering av mänskliga poser

I denna masteruppsats presenteras en fungerande metod för att kombinera data från flera Kinectkameror med syfte att producera stabila estimeringar av en mänsklig användares pose i realtid. Vi visar att användandet av flera djupkameror ökar interaktionsytan för användaren att röra sig inom och att metoden ger bättre uppskattningar när användaren roterar i detta interaktionsutrymme jämfört med ett system med en kamera. Dessutom visas att systemet med flera kameror hanterar problem som uppstår när delar av användarens kropp är skymd i en eller flera kameror bättre än enkamerasystem. Två filtertyper, partikel- och Kalmanfilter, implementeras och diskuteras i denna rapport. Olika metoder för att kombinera data baserat på kvaliteten på anslutna kamerors observationer av användaren diskuteras också. Detta kombineras med tekniker för att förbättra slutliga användarposer såsom begränsningar av användarens skelett. Båda filterteknikerna konstateras fungera i realtid, här definierat som 30 bilder per sekund, på en vanlig bärbar dator. I tillämpningar som kräver att kunna köras i realtid levererar dock Kalmanfiltret bättre estimeringar jämfört med partikelfiltret vars prestanda är starkt beroende av antalet partiklar som kan användas innan uppdateringsfrekvensen faller under 30 Hz. Sammanfattningsvis ger metoderna i denna rapport ett stabilt, portabelt, snabbt och kostnadseffektivt system för att registrera en användares kroppspose i realtid. Detta har viktiga tillämpningar i virtual reality och till viss del inom animering i film och datorspel.

Acknowledgements

Many thanks to my supervisor Magnus Burénius for important input and insightful discussions. Thanks also to my colleagues at Centive Solutions GmbH for valuable advice and all others who have helped me throughout the project.

Contents

Chapter 1

Introduction

1.1 Report overview

This first introductory chapter gives an overview and background of the project. It defines the goals and summarizes the steps taken to achieve these goals. We also discuss some relevant previous work. Chapter 2 describes the theory of the methods that we use, whereas chapter 3 focuses on the actual implementation. Chapter 4 presents the experiments that we use to evaluate the implemented approach. In chapter 5 we analyze and interpret the experiments and discuss the strengths and weaknesses of the approach. Finally in chapter 6 we summarize our overall conclusions and the direction for future work.

1.2 Background

To make animated characters move in a realistic way, the motion of human actors is often reordered and then applied to animated characters in films and video games [27]. In virtual reality, the user is immersed by wearing a head-mounted display (e.g. Oculus Rift [23]), and does not see his or her own body. As a means of enhancing the experience and facilitating interactions with the virtual environment, it can be useful to render a virtual representation of the user's body that follows the movements performed by the user's real body. We can refer to such a virtual representation of the user's body as an avatar. For such a system to provide a comfortable user experience, the avatar must react immediately to body movements in the real world. In other words, the system has to work in real-time.

There are a number of different techniques available in the market for tracking a moving human body. They include marker-based systems where the user being tracked wears physical markers that are tracked by cameras, see for instance the systems developed by Vicon [31]. There are also sensor-based systems where the tracked user wears sensors (accelerometers, gyroscopes and compasses) that register the user's body motion without relying on cameras. See for instance YEI Technol-

ogy's solution [33]. Another, simpler type of system is markerless tracking systems that do not require the user to wear any equipment for the tracking to work. The Kinect is one such system. Compared to the other types of systems, the Kinect is cheaper and easier to use. These are the main reasons to why we study the Kinect in this project as a method to achieve real-time body pose estimation.

The Microsoft Kinect is an inexpensive and publicly available depth sensor that has made it easy for anyone to start experimenting with depth data. Since its release it has been used in a vast number of applications. One example of a task that can be accomplished with the Kinect is to track a user's body with a skeleton representation consisting of 20 joints and use body movements to interact with computer programs or video games. The level of accuracy needed varies between applications.

In most applications a single depth camera is used and this often works well in applications where the user is always positioned in front of and facing the camera. However, there are applications where this restriction imposes too strict limitations. In such cases it may be justified to extend the system with multiple Kinect cameras. The scope of this Master's thesis is to investigate such multi-Kinect setups in terms of advantages, difficulties and drawbacks. Using multiple depth cameras has several potential advantages including:

1. A single Kinect has a limited field of view, and by using multiple Kinects we can extend the space in which the user can be detected and tracked. With multiple Kinects, different parts of a user's body can also be tracked by different cameras.

2. Using multiple Kinects may in many cases reduce occlusion problems. Parts of the user's body may be occluded by objects seen from certain camera positions, while being visible from other camera positions. Combining information from multiple Kinects can thus provide additional information about the user that would not be available with only one Kinect.

3. Combining information from several sensor may also be a means of improving the overall quality of pose estimates. Having access to more data about the user may help in making better estimates of the user's pose and movements as the influence of e.g. noise may be decreased.

When it comes to disadvantages, a setup with multiple cameras naturally becomes more complex to manage and implement than a single camera system and it also becomes more expensive. In some applications there have also been reported interference issues between individual depth cameras. The fact that each Kinect requires a separate USB controller may impose some practical issues when it comes to connecting multiple Kinects to some computers, especially laptops.

There are also a number of challenges that need to be addressed when it comes

to combining data from multiple sources. Data needs to be given in a coordinate system common to all connected cameras and data from the individual cameras cannot be assumed to be of the same quality, depending on their view of the user. This needs to be taken into account when combining the data. Other potential problems include distinguishing the right side of the user from the left and making sure that the body part labeling is the same in all cameras (depending on the camera setup). Issues of this nature will also be investigated and discussed in this report.

Another important part of this work, which is not directly related to the usage of several cameras, is how to actually provide stable estimates of a human user in real-time. For this part, different filtering techniques and measures to improve estimates such as different motion models and applying body constraints on the user's estimated skeleton will be explored.

On a further note, most of the work in this Master's thesis was done at the headquarters in Aachen of German company Centive Solutions GmbH where I, the author of this report, am a co-founder. The project was supervised by Magnus Burénius, at the time doing his PhD in computer science at KTH - Royal Institute of Technology at the Computer Vision and Active Perception Lab (CVAP). Centive Solutions is a company providing collaborative virtual reality solutions for architecture, construction, design and sales applications. At the time of doing my Master's thesis, there was also another KTH student, Rasmus Johansson, doing his Master's thesis in the company at the same location. His work focused on how to train a system to classify pixels in a depth image so as to estimate the position of different body parts of a user. My thesis uses an already existing system to solve this sub-problem, i.e. the Microsoft Kinect SDK. The focus is instead on combining given body part estimates from multiple Kinect cameras.

1.3 Problem statement and project goals

The objectives of this Master's thesis include exploring techniques for merging skeleton data from multiple Kinect devices. The goal is to create a stable and working program that can be run in real-time taking a real feed of Kinect data from multiple Kinects. The program should be able to handle situations where a user moves between cameras, and make sure that the highest quality data out of the available data is used. The skeleton estimates should be smooth between frames, yet responsive to arbitrary and quick movements of a user in the interaction area.

In addition to this, it should be possible to visualize the results in 3D in real-time for evaluation. Extensive testing will be required in order to tune filters and determine the impact of different approaches and measures taken to improve the output of the algorithms in different usage scenarios with different camera setups. The real-time criteria means that methods cannot be too computationally complex,

introducing further challenges and limits.

This problem is interesting to study as the single-camera approach suffers from several important limitations, as discussed in the background section above. The related challenges of a multi-camera setup have not been fully solved, which leaves room for new approaches. Furthermore, there are many applications that would benefit from the higher quality pose estimations that a well-working multi-camera system with balanced filtering could provide. Examples include: motion capture for animated films, video games and virtual reality applications of different kinds.

1.4 The overall approach

In this project we propose and implement two major filtering techniques allowing an in-depth comparison to be made. Firstly, a particle filter is implemented for dealing with combining data from multiple Kinects and producing pose estimates of the user. Secondly, a traditional Kalman filter is implemented for the same purpose. The particle filter was chosen for its capabilities to be used for very general problems. However, the particle filter requires a lot of computations and one question to look into in this project was whether the particle filter would be fast enough for real-time performance. The Kalman filter was chosen as it is a proven method which is computationally efficient in comparison with a particle filter. In order for the merging algorithms to be properly evaluated in real user applications, we create a test environment with a manager for multiple Kinects working within a global reference system. A 3D visualization displaying all results in real-time along with tools for offline analysis of data makes the evaluation possible. Below is a rough plan for the steps needed to achieve the goals defined in the previous section. This list also acts as a foundation for the theory and implementation chapters presented later in this report.

1. Define a global coordinate system in which the data from all Kinects can be expressed and worked with. This requires an extrinsic camera calibration.
2. Transform the joint position estimates from each of the Kinects into the global reference system.
3. Create a system to evaluate the quality of observations from the different connected cameras.
4. Merge the estimates of joint positions from the Kinects. Two approaches are considered in this project.
5. Apply constraints on the skeleton to avoid stretching and eliminate impossible poses.
6. Draw estimated joints and bones in 3D in real-time to visualize results.
7. Develop methods for verifying and evaluating performance.

1.5 Previous work

Detecting humans and estimating human motion under different circumstances is an active field of research and there are various approaches using different types of hardware in the scientific community. Although the task of following a moving human may appear simple for an actual human, it is a different matter for computers as detecting body parts and or facial features reliably is a challenging problem. Varying lighting conditions, partially occluded body parts or features as well as different appearances between individuals are just a few factors that contribute to the difficulty of the pose estimation problem in arbitrary situations.

In many cases, one wishes to estimate the pose of a human user in 3D from 2D image data. For this problem, there is a wide array of algorithms. Sidenbladh et al. [28] propose a method of fixing a 3D body modeled by cylinders on humans appearing in 2D frames. Kazemi et al. [12] present a method for estimating the 3D pose of football players in a multi-view setting using a random forest. Burénius et al. [6] further show that the pictorial structure framework, which is popular for 2D pose estimation, can be extended to 3D.

Another problem is to estimate the 3D pose of a user, using 3D data, for instance in the form of point clouds. Thanks to the relatively recent introduction of inexpensive hardware, such 3D data can be produced more easily than ever before. The most accurate methods when it comes to tracking human motion still rely on physical markers placed on the human body [13]. Although these methods yield accurate results, it is expensive and cumbersome to use physical markers in practice. For this reason, we will focus on markerless tracking methods in this report.

Despite the relatively short period of time since its launch, the Kinect has been used in various applications outside its initial target of gaming. We find examples in controller-free manipulation of medical image data [9], training applications [32], interactive user interfaces [5] and more. There is also work that combine the Kinect RGB and depth sensor data to fit a skeleton model to a human body using computer vision techniques, such as the work carried out by Kar [11] and Nakamura [20]. By combining color and depth data, tracking of a moving object can be improved over purely vision-based techniques in some situations. Depth data can for instance help distinguish the target from the background in case the target and the background have roughly the same color [20].

Aside from applications where entire human bodies are studies, which is the case in this work, there have also been works studying particular parts of the body, such as hand tracking. One example of the latter is given by Oikonomidis et al. [24] with their algorithm for markerless tracking of a human hand and its articulations. There are also studies of specific body movements such as golf swings [35]. Such approaches may be stable and efficient and work well for the purpose, but they are

at the same time limited to very specific situations.

When it comes to using multiple Kinects, there has been less work done. Williamson et al. have built an application where multiple Kinects are combined to create a training environment for soldiers [32]. This application requires 360 degrees turning mobility of the user and they use multiple Kinects to achieve this. They also investigate a situation where the user carries a weapon during training, which can be used for a more accurate determination of the orientation of the user. Their system uses individual computers for each of the Kinects and a central server for the data fusion. This is an acceptable setup in training applications, but can be cumbersome and expensive in other applications.

Another paper by Asteriadis et al. [1] deals with multi-camera systems where geographically separated users interact in a common 3D environment. They use several cameras as a means of reducing the problems associated with occlusion of users in situations such as running on a treadmill. To estimate joint positions they maximize the sum of energy functions from each Kinect connected to the system. They take into account the motion history of the different body joints and the expected posture from a set of candidate positions to produce final joint estimates. Berger et al. [3] look into different camera configuration methods for multiple Kinects and how multiple Kinects can be used for motion capture. Susanto et al. [29] investigate how objects can be detected in a scene observed by multiple Kinects. They make use of point clouds from each of the Kinects in combination with color information from the Kinects' RGB cameras and report improvements when combining depth and color data compared to using the methods separately. Zhang et al. [36] propose a method to fuse point clouds from multiple Kinect cameras to produce more accurate estimates than with one-camera systems. However, their implementation relies on a GPU implementation and can still only be run in 15 frames per second, which can be limiting in cases where real-time performance is required.

When speaking of the Kinect, it is inevitable to mention the research carried out by Microsoft's research team at Cambridge. In their initial paper they use a random forest trained on a dataset consisting of 3D data of 100,000 human poses [7]. With the obtained decision forest, individual pixels in depth images are classified as belonging to different body parts. As a result, the Kinect can provide estimates of joint positions in real-time of users located in front of a Kinect. This is implemented in the Kinect software development kit (SDK), which we will use in this thesis.

Microsoft's own teams of researchers are also working hard on implementing new functionality for the Kinect device and making this functionality available to developers through their SDK (see section 3.1.2) which is regularly updated. In a recent edition of the SDK, for instance, a feature named Kinect Fusion allowing the user to scan 3D objects or environments from multiple angles using a Kinect and then producing a 3D model from the data was made available [17].

Chapter 2

Theory

In this chapter we look at the theory of the different parts of this thesis. First, we go through a few important probabilistic concept that are needed to understand the filtering techniques presented later in the chapter. The chapter also deals with how to incorporate data from multiple sensors in filtering. We will also briefly go through the human body and its movement constraints.

2.1 Filtering for pose estimations

When an environment is only partially observable, an agent or system still needs to be able to keep track of what state it is in, its belief, with only partial observations at hand. Keeping track of the state is essential in order for rational decisions to be made by the agent. What the state is depends on the application, but one example is the agent's position. The process of computing the belief state, i.e. the posterior probability distribution over the most recent state, when all the evidence up to the current time t is given, is referred to as $filtering$ or *state estimation*. There are various techniques that can be used to tackle this problem [30]. What approach to choose depends on the underlying model for the problem under study.

The Kalman filter can be used in situations where linear motion models can be assumed to apply to the object to be tracked, and where posterior probability distributions are single node Gaussians. A posterior probability distribution will here be understood as the probability distribution of the observed random variable after the evidence obtained from observations or measurements have been taken into account. In cases where the underlying motion model is not linear, but where it is reasonable to assume that a local linearization of the model around the previous estimate is a good approximation, the Extended Kalman Filter (EKF) may be a suitable approach. When speaking of producing the state estimate from a probability distribution, one way to proceed would be to take the maximum of the posterior distribution. This technique is known as a MAP estimate.

In cases where the posterior distribution is not well-described by a Gaussian (i.e. if the distribution is, for instance, multi-modal or discrete) a particle filter may be a more suitable approach [19]. The particle filter is a very general approach that can also deal with non-linear models. However, particle filters are known to be computationally expensive. Depending on the application at hand it can be challenging to find the ideal compromise between accurate approximation of the posterior distributions and associated computational complexity. Generally speaking, the greater the number of particles used, the better the approximation. But more particles also means more computations.

Hidden Markov Model (HMM) filtering can also be used to approach tracking problems such as the one studied in this project. In fact, the Kalman filter can be seen as a special case of the HMM filtering where all distributions are assumed to be Gaussians. In the following sections, some basic theory for these three filter types is given. Understanding HMM filtering can help in understanding the Kalman filter, which is why we cover the HMM filter although it is not used in the final implementation.

2.2 The basic theory

In the following sections we cover some common theory to all the studied filtering techniques. The scope of this chapter is not to provide an in-depth theory section, but rather to help the reader understand the foundation of the different filtering techniques and how they can be implemented.

2.2.1 Bayes' rule

A fundamental rule in probability theory that is an underlying component of most artificial intelligence systems for probabilistic inference is Bayes' rule [26]. It relates conditional probabilities on the form $p(x|y)$ to their "inverses" $p(y|x)$. If we assume that x is a quantity that we want to infer from our data y, then the following holds:

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(y|x)p(x)}{p(y)}$$

provided that $p(y) \neq 0$. Furthermore,

$$p(y) = \sum_x p(x,y)$$

Here, $p(x)$ is referred to as the prior and reflects the knowledge about X before measurements, i.e. sensor data y, have been taken into account. $p(x|y)$ is the posterior, which is often the distribution of interest, and thanks to Bayes' rule it can be computed from the prior and $p(y|x)$. The distribution $p(y|x)$ is referred to as the generative distribution describing the probability of measuring y if x was the

case. $p(y)$ is independent of x and can thus be treated as a normalization function.

Bayes' rule can also deal with conditional probabilities involving more than two random variables. Introducing $Z = z$ we have:

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)}$$

The concept of conditional independence may be useful for simplifying joint probability distributions that arise in many applications. Furthermore, x and y are conditionally independent given z if and only if

$$p(x,y|z) = p(x|z)p(y|z)$$

which can be inserted into Bayes' rule.

2.2.2 State and belief state

The concept of state is important for the models to be discussed in coming sections. The state, which will be denoted x throughout this chapter, can be described by a state vector that contains relevant information about the environment and the pose of the user or a robot or something else depending on the application. The state vector can change over time and the state at time t will be denoted x_t. In this project, the state will typically be the x-,y- and z-coordinates of all the joints of the skeleton model provided by the Kinect. In some models, the state also contains the first order derivatives of the position coordinates, describing the velocity in each coordinate.

In the following, let measurement data be denoted by z. One or more sensors provide information about the environment and the measurements by all the connected sensors will be contained in z. At time t the measurement data is denoted z_t.

Important to note is that the state x_t cannot be measured directly. We thus have to rely on sensor data and a model describing as closely as possible the movements of the object to track and then infer the most likely underlying state at each time t. The belief state is the distribution of the underlying state which is used for the state estimation:

$$belief(x_t) = p(x_t|z_{1:t})$$

The belief is thus the posterior probability distribution reflecting the knowledge about the state after the measurements up to and including time t have been incorporated. One can also speak of the posterior before the measurement z_t has been taken into account.

$$\overline{belief}(x_t) = p(x_t|z_{1:t-1})$$

The actual state estimation from the belief distribution can be done in different ways. The maximum a posteriori estimate (MAP) is a simple one, which consists in finding the argument x for which the belief distribution has its global maximum.

2.3 Hidden Markov Models (HMM)

A Hidden Markov Model (HMM) is a temporal probabilistic model where the modeled process is assumed to be a Markov chain with states that are hidden, i.e. not directly observable. The Markov chain has a transition model that defines the probability for the system to go from state one state to the next:

$$p(x_t|x_{0:t-1})$$

For Markov chains one makes the assumption that the current state depends only on a finite number of previous time steps. Usually one deals with first-order Markov chains where the current state only depends on the previous state. This means that

$$p(x_t|x_{0:t-1}) = p(x_t|x_{t-1})$$

The transition model for a first-order Markov chain is thus

$$p(x_t|x_{t-1})$$

As mentioned above, the states of an HMM cannot be directly observed. The HMM therefore has an associated observation model (sometimes called a sensor model) that gives the probability distribution of a measurement yielding an output z_t if the underlying state is x_t.

$$p(z_t|x_{0:t}, z_{0:t-1})$$

Similarly as for the transition model we make the assumption that the current measurement only depends on the current state:

$$p(z_t|x_{0:t}, z_{0:t-1}) = p(z_t|x_t)$$

For the HMM to be fully defined, we also need the prior probability distribution for the state at time $t = 0$

$$p(x_0)$$

to be known. Note that whereas the hidden states are discrete for an HMM, the observations may be discrete or continuous. If they are discrete, the observation model is usually expressed by a an observation matrix, and in the continuous case by a probability distribution, such as a conditional Gaussian. For an HMM, the joint probability distribution, which is the basis for inference, is given by:

$$p(x_{0:t}, z_{0:t}) = p(x_0)p(z_0|x_0) \prod_{k=1}^{t} p(x_k|x_{k-1})p(z_k|x_k)$$

where $p(x_k|x_{k-1})$ and $p(z_k|x_k)$ are the transition and emission probabilities described above.

2.4 State space models

We can now define a state space model with hidden states that are continuous. The state space model is in fact just like an HMM, but whereas the HMM has discrete hidden states, the space state model has continuous hidden states. The space state model can be expressed as:

$$x_t = g(x_{t-1}) + \epsilon_t$$
$$z_t = h(x_t) + \delta_t$$

Here, ϵ_t and δ_t represent system noise and measurement noise at time t respectively. The function g is the transition model and h is the observation model. The state x_t is hidden in the sense that it cannot be observed directly, but noisy information about the state can be obtained through observations via the observation model. The functions g and h may or may not be linear affecting the choice of filter type.

The parameters governing the noise terms above can change the way a filter behaves. These errors are assumed to follow a probability distribution with parameters depending on the chosen distribution. In the case of a normal distribution, for instance, the mean and co-variance matrices are chosen for the errors. These covariance matrices are referred to as process covariance and measurement covariance for the process and measurement noise respectively in this report.

Below we look at two examples of state space models that we later use with the filter algorithms.

2.4.1 Zero velocity model

With velocity assumed to be zero, the state vector for a 3 dimensional problem is given by (below, vectors are in bold to stress the fact that they are vectors):

$$\boldsymbol{x_t} = [x_t, y_t, z_t]^\intercal$$

and the state space model is given by

$$\boldsymbol{x_t} = A_t \boldsymbol{x_{t-1}} + \boldsymbol{\epsilon_t}$$
$$\boldsymbol{z_t} = C_t \boldsymbol{x_t} + \boldsymbol{\delta_t}$$

where the transition matrix A is given by

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The observation matrix, C, taking into account the observed position's coordinates is given by

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.4.2 Constant velocity model

For a problem with 3 spatial dimensions and velocities in each of the coordinate directions, the state space becomes 6-dimensional. The state vector is represented as

$$\boldsymbol{x_t} = [x_t, y_t, z_t, \dot{x_t}, \dot{y_t}, \dot{z_t}]^\mathsf{T}$$

and the motion model has the same form as in the zero velocity case above:

$$\boldsymbol{x_t} = A_t \boldsymbol{x_{t-1}} + \boldsymbol{\epsilon_t}$$

$$\boldsymbol{z_t} = C_t \boldsymbol{x_t} + \boldsymbol{\delta_t}$$

where the matrix A is chosen as

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where Δt is the time interval between time steps. The velocities in the state vector are thus multiplied by the update frequency of the observing cameras, i.e. the time interval between frames.

In the case where only positions (and not velocities) are observed, the observation matrix, C, is given by:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

2.5 Hidden Markov Model filtering

Filtering consists in computing the belief state online as measurement data z_t streams in. In other words, what we want to do is to compute the posterior for the next time step given the current state distribution and the new evidence (see also figure 2.1 for an example on how distributions can be represented in different filter types):

$$p(x_t|z_{1:t}) = f(z_t, p(x_{t-1}|z_{1:t-1}))$$

In the case of an HMM, the algorithm used for this is known as the forward algorithm and consists of two steps:

1. Prediction step
   The current state distribution is projected forward onto the next time step using the transition model of the HMM.

2. Update step
   The projected distribution for the next time step is updated with respect to the new observation (evidence).

Mathematically speaking, we have

$$p(x_t|z_{1:t}) = p(x_t|z_{1:t-1}, z_t) = \alpha p(z_t|x_t, z_{1:t-1})p(x_t|z_{1:t-1}) = \alpha p(z_t|x_t)p(x_t|z_{1:t-1})$$

Above we have used Bayes' rule and the Markov assumption. α is a normalizing constant introduced in order for probabilities to sum to 1. Here, the factor $p(x_t|z_{1:t-1})$ represents the prediction step and $p(z_t|x_t)$ takes into account the new observation (this factor is obtained from the observation model).

We can condition on the state x_{t-1} in order to obtain a recursive formulation.

$$p(x_t|z_{1:t}) = p(x_t|z_{1:t-1}, z_t) = \alpha p(z_t|x_t) \sum_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})$$

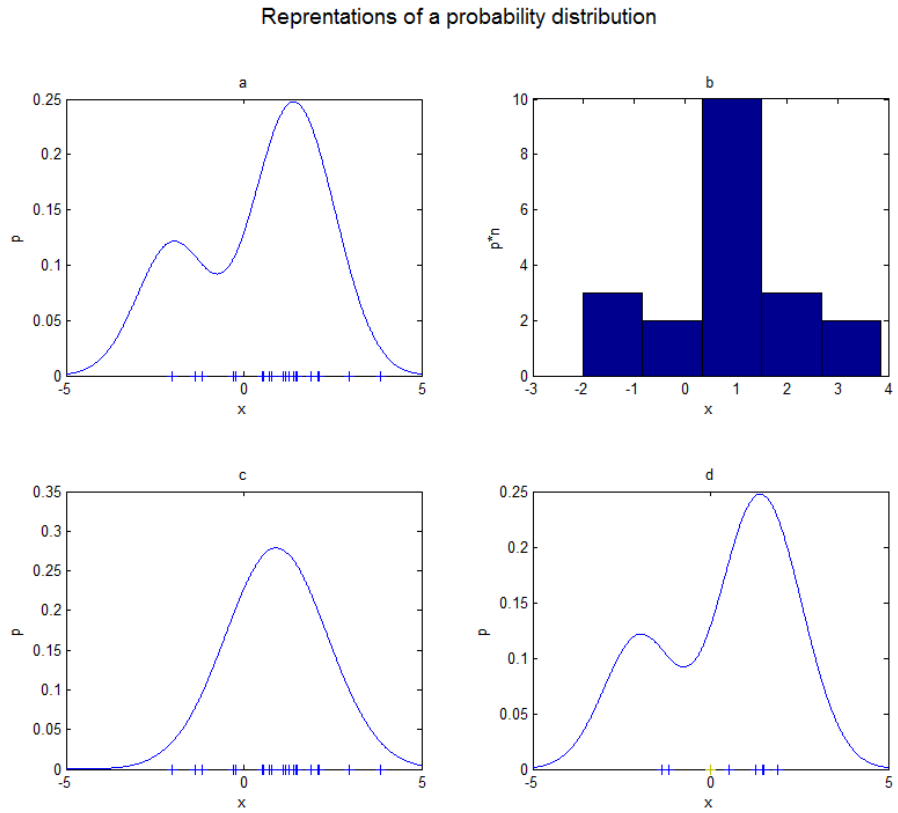

Figure 2.1. A continuous distribution as the one seen in a) can be represented in different ways. a) Here, the probability density function of a mixture of Gaussians is drawn. In the bottom of the graph, n random samples from the distribution are shown as horizontal lines. These samples can be considered as measurements. b) Here, the n samples are shown as a histogram. If all the values are divided by the number of samples, n, then the histogram constitutes a probability distribution with a finite number of possible states. This representation is used for Hidden Markov Models, where the number of hidden states are finite. c) In this figure, a single normal distribution has been fitted to the samples drawn in a). This illustrates a probability distribution as they appear in a Kalman filter, where the probability distributions are assumed to be Gaussians. d) Here, we see the result of a subsampling step where samples are drawn from the original n samples with a probability proportional to each sample's weight. In this example the weights are computed as a function of the distance to the sample mean for illustration purposes. After the subsampling step, we see that certain samples with a low assigned probability (weight) have been eliminated. This illustrates how probability distributions can be represented in a particle filter.

2.6 Kalman filtering

2.6.1 The Kalman filter

The Kalman filter is used for performing exact Bayesian filtering for linear-Gaussian state space models. In the case of Kalman filters, the state space model introduced in section 2.4 is subject to certain conditions and simplifying assumptions.

1. The transition and observation models are assumed linear:

$$x_t = A_t x_{t-1} + \epsilon_t$$

$$z_t = C_t x_t + \delta_t$$

   where A_t and C_t are matrices that may or may not vary over time.

2. Furthermore, the system noise ϵ_t and the measurement noise δ_t are assumed to be Gaussian, i.e.

$$\epsilon_t \sim \mathcal{N}(0, Q_t)$$

$$\delta_t \sim \mathcal{N}(0, R_t)$$

   where Q_t and R_t are the associated covariance matrices that may be constant or vary over time.

3. The initial belief, $belief(x_0)$, must also be Gaussian in order to ensure that the posterior, $belief(x_t)$, will remain Gaussian.

If these conditions are met, the Kalman filter goes through the same prediction update cycle presented in the previous section on HMM filtering. The difference is that every distribution here is Gaussian and will remain so through the cycles. See also an example probability distribution represented by a Gaussian in figure 2.1.

The multivariate Gaussian distribution is given by:

$$p(x) = det(2\pi\Sigma)^{-\frac{1}{2}} \exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

where μ is the mean vector and Σ the covariance matrix. The mean vector has the same dimension as the state vector and the covariance matrix is a symmetric, positive-semidefinite quadratic matrix of dimension state vector x squared.

One further remark is that the Gaussian distribution is unimodal, meaning that it only has one peak. When the posterior and the data/measurements can be described by unimodal Gaussians, the Kalman filter is a very efficient method. When the mentioned distributions are not well-described by Gaussians, the assumptions of the Kalman filter may be too strict. When it comes to the Kinect, Lindbo Larsen et al. [13] have shown that such a stereo camera makes it possible to use approximately unimodal likelihood models, making the use of Kalman filters possible.

The Kalman filter algorithm takes as input the mean and covariance from the previous time step, i.e. μ_{t-1} and Σ_{t-1}, along with the new observation, z_t, and outputs the new mean and covariance, μ_t and Σ_t. With the notations provided in the beginning of this section, the following steps make up the algorithm:

1. Prediction of mean vector
$$\overline{\mu}_t = A_t\mu_{t-1}$$

2. Prediction of covariance matrix
$$\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + Q_t$$

3. Computation of the Kalman Gain
$$K_t = \overline{\Sigma}_t C_t^T (C_t\overline{\Sigma}_t C_t^T + R_t)^{-1}$$

4. Update mean vector with observation
$$\mu_t = \overline{\mu}_t + K_t(z_t - C_t\overline{\mu}_t)$$

5. Update covariance with observation
$$\Sigma_t = (I - K_tC_t)\overline{\Sigma}_t$$

The mathematical derivation of the above steps falls outside the scope of this report and will be omitted. For a complete derivation please refer to [30].

2.6.2 The Extended Kalman filter

The goal of the Extended Kalman Filter (EKF) is to make it possible to use the filter in cases where the assumptions of the normal Kalman filter are too strict. When the underlying model is not linear we cannot directly apply the exact inferences discussed in the previous section. Instead we can use an approximate method such as the EKF. This is for instance the case when the movement of an object between observations cannot be approximated by movement along a straight line.

The extension to the normal Kalman filter consists in linearizing the model around the state estimate from the previous time step. This can be done in different ways where the simplest one is to use a first-order Taylor expansion. With the linearization done, the steps are the same as for the normal Kalman filter, see section 2.6.1.

Important to note is that the noise variances in the model equations are left unchanged when carrying out the linearization, i.e. the error resulting from linearization is not taken into account. This is acceptable when the linearization error can

be assumed to be small, otherwise other methods should be considered.

There are also further improved versions of the EKF, for instance the Unscented Kalman Filter (UKF) [19] and also different ways of providing support for multimodal probability distributions that are not further treated in this report.

2.7 Particle filtering

Exact inference may sometimes be impossible for complex problems with highly nonlinear underlying models. Below are a few cases where methods such as the Kalman filter may not be an appropriate choice [8]:

1. Systems that have multimodal error models (i.e. having more than one value with high probability).

2. Systems with observation or error models that are highly skewed (i.e. having a mean and most likely value that are far apart).

3. Systems with discontinuities (for instance if the model has been formed by fitting experimental data).

4. Systems that are nonlinear in the sense that the observation model is sensitive to the predicted state values around which an EKF can be linearized.

In these cases, there is a need for approximate solutions. Particle filtering is a family of stochastic algorithms that can be used for approximate online inference. Here, parameters for describing the posterior probability distribution are not used, but instead sampling from the previous step posterior distribution is done. The result can in many cases be more accurate than when using a Kalman filter, but at the cost of heavier computations. The computational complexity is due to the fact that many particles are needed to provide accurate approximations of the associated probability distributions.

Particle filtering is based on likelihood weighting, which can be used for approximate inference in Bayesian networks [26]. In likelihood weighting, one only generates events that are consistent with the evidence. This way, one can avoid rejecting samples that are not consistent with evidence leading to a smaller number of samples required compared to other similar methods.

Using the likelihood weighting directly for dynamic Bayesian networks, however, is not efficient without a few modifications. Firstly, we use the samples themselves to approximate the current state distribution. Secondly, we focus samples on regions of high probability in the state space. This means throwing away samples of low weights given observations. Particle filters are designed to incorporate the above ideas and work as follows:

In the first step, produce N samples from the prior distribution $p(x_0)$. Then for each time step the following cycle is run through:

1. Propagation step
   For each of the N samples, sample the next state value x_{t+1} given the current state x_t and the transition model $p(x_{t+1}|x_t)$.

2. Weighting step
   Each of the N samples are then weighted by the likelihood it assigns to the new evidence $p(z_{t+1}|x_{t+1})$, i.e. through the observation or sensor model.

3. Resampling step
   A new set of N samples is created by resampling from the weighted set from the previous step. All samples are drawn with replacement from the population in the previous step, and the probability for a sample to be selected is proportional to its weight. All the samples in the new population can be unweighted for the next time step in the cycle. An illustration of resampling from data can be seen in figure 2.1.

2.8 Incorporating data from multiple sensors

The filters described in the previous sections are directly applicable for single sensor cases, but in order for them to be used in multi-sensor cases (i.e. with multiple Kinect cameras or other sensors connected to the system) some additional considerations are necessary.

2.8.1 Approaches to fusing data from multiple sensors

There are different methods that can be used to include multiple sensors in a system, below we look at three approaches [8]:

1. One approach, referred to as the Group sensor model, is to consider a group of sensors as one single sensor. This is done by combining measurements from the different sensors into one combined measurement. This approach often works well for a small number of sensors and less so if the number of sensors is large. As this method combines the measurements before inputting a combined measurement to a filtering algorithm, the observation matrix will not increase in complexity when adding more sensors. This in term makes the method computationally efficient and the approach is useful if applicable to the problem under study.

2. Another approach is to treat measurements from the different sensors in a sequential manner for each time step. This means that for each time step there are several sub-time steps, one for each sensor. There is thus a need to go through the predict and update cycles in the filtering methods several

times per time step which leads to heavier computations. Here, the matrices and vectors in the filtering algorithms remain the same as in the single-sensor case for each prediction-update cycle, but on the other hand, several cycles per time step are required.

3. A third way to go about the problem of multiple sensors is to derive model equations that take the different sensor measurements into account in a common state estimation. This can be useful when the number of sensors is large as methods such as the inverse-covariance form provide a complexity which only increases linearly with the number of sensors. The method is, however, more complicated in itself and in the cases where the number of sensors is relatively small, the gain is limited.

2.8.2 Sensor-confidence considerations

In this particular case, we are working with multiple Kinects with only partially overlapping observation spaces. This means that there may be situations where the user will not be observed by all cameras in the system. Here, the combination of measurements will only concern those cameras, one or several, that observe the user. If a user is partially observed by a Kinect, the device has a system of inferring those joint positions that it cannot observe directly.

This is useful in many applications, but the inferred joint positions will naturally have a lower confidence level than those actually observed. Therefore, the confidence level of each of the Kinects should be taken into account when producing the final estimate. Assigning weights based on whether a given joint is tracked or not in each connected Kinect may be one way. This approach could be combined with an overall weight for each Kinect at each frame by determining the ratio of tracked joints or some other measure reflecting the overall quality of observations of a given camera at a given time.

2.8.3 The left and right problem

Kinect cameras are designed to have the user positioned in front of the unit and facing it. In situations with multiple cameras, this will not always be the case depending on the camera setup. For instance, if the user is facing one camera and has one camera placed behind him/her, the camera that the user is facing will label the skeleton joints correctly with respect to left and right as shown in figure 2.2. However, the camera behind the user will produce a skeleton where the labeling of joints is mirrored. When combining measurements of joints, the labeling between the same joint seen from different cameras will not correspond, causing the merging to fail. Depending on the camera setup and the usage scenario, different approaches can be used to approach this problem.

1. In simple camera setups it could be possible to simply flip the labeling with the respect to left and right from one camera in order to get the labeling consistent between cameras.

2. We can also imagine a distance function that checks for instance the position of the joint labeled 'right hand' for one camera and computes the distance to the position of hand joints labeled 'right hand' and 'left hand' for another camera. If the distance to 'left hand' is smaller than that to 'right hand' in the second camera, the hand joints are relabeled for one of the cameras. If this is not the case, then the hand joints are assumed correctly labeled already. The procedure is repeated for all connected cameras and all joints that are affected by right- and left labeling. The camera with the highest overall weight as discussed in section 2.8.2 can be chosen as the reference that the other joints are labeled with respect to.

3. Another way to deal with the problem could be to have a means of determining the orientation of the user inside the interaction space observed by the cameras. This would make it possible to label joints consistently between cameras. A system of markers or a sensor detecting the orientation of the user could be ways to determine the user's orientation.

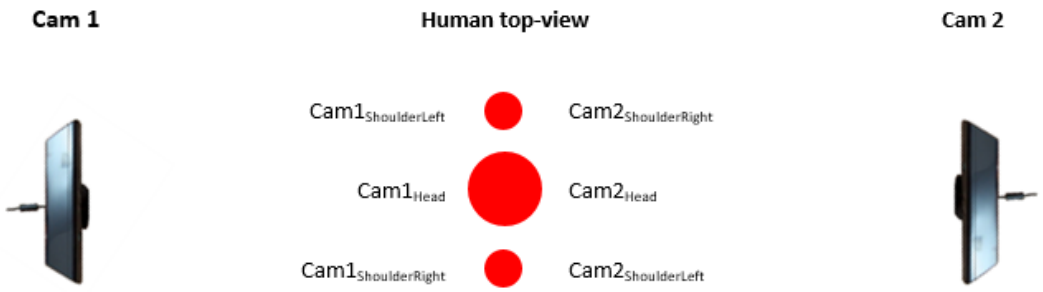

Figure 2.2. Figure shows a user located between two Kinect cameras. The head is labeled consistently between the two cameras, but the upper shoulder in the figure is labeled as ShoulderLeft by camera 1 and as ShoulderRight by camera 2. In order for data to be correctly combined in a multi-camera setup, the labeling needs to be consistent between all cameras connected to the system.

2.8.4 The data association problem

If multiple users are being tracked by a system, there is yet another challenge introduced, namely that of assigning measurements to the correct user. In this report, the focus lies on applications where only one user is tracked at a time, meaning that this problem does not arise, but it remains a possible future extension.

Methods to deal with the data association problem include the Probabilistic Data Association Filter (PDAF) [8].

2.9 From joint positions to a human skeleton

Human motion can be complex or less so depending on the application and the body part(s) under study. Given that we are interested in the entire human body in this application, we need to consider all of its joints. Given that different joints move and behave differently, it may be appropriate for them to be modeled differently. If the filtering methods described in previous sections are applied to each joint individually, then the underlying motion model for each joint can be independently specified. Concretely, this means that the different joints can have different state space models, as defined in section 2.4.

2.9.1 Kinematic constraints

The accuracy of joint data can be further improved by taking into account the anatomy of the human body and its movement constraints [2]. For instance, some joints, such as knees and elbows, only bend along one axis. They are called hinge joints as they only have one degree of freedom. Certain joints, such as the hand joints, can move faster than others, like the head. Joints may also have a limited range within which they can be bent. All this information can be used to correct impossible pose estimates. The pose is here considered as the entire skeleton with each joint position being estimated. Bone lengths can also be used in a similar fashion to check that joints are not located too far away from their parent joints.

One way to deal with the constraints is to define a hierarchical structure for the skeleton, i.e. to define a root node in a tree corresponding to a joint in the skeleton. Then the bones connecting the root node to first-level joints in the skeleton define the structure of the tree that will have a depth equal to the number of levels in the hierarchy. Once this has been done, the imposed body constraints can be defined between a parent node and its child nodes as bone length constraints as well as joint angle constraints [34]. For more information on how this structure was defined in this project, please refer to section 3.2.8 in the implementation chapter.

2.9.2 Ways of visualizing the pose

A simple visualization where the skeleton is represented by plotted joint positions in 3D along with lines connecting relevant joints acting as bones works well for evaluating a pose estimate, see figure 2.3. More advanced visualization options can however also be considered. One example is mapping the estimated skeleton to a visualized 3D avatar. The avatar is a visual character of the person for which the skeleton is estimated so that the avatar follows the movements of the user as precisely as possible.

In order for the avatar to be visualized on a screen or another type of visualization device, a 3D model of a character (human or other) is needed. Once all the necessary information from the estimated skeleton has been extracted and the kinematic constraints of the body have been taken into account, the final joint position estimates are tied to the joints of the 3D model. The number of joints of the 3D model's skeleton can be different from our skeleton. If this is the case, there is a need of a retargeting step where the joints from our skeleton is mapped to the appropriate corresponding joints of the 3D model.

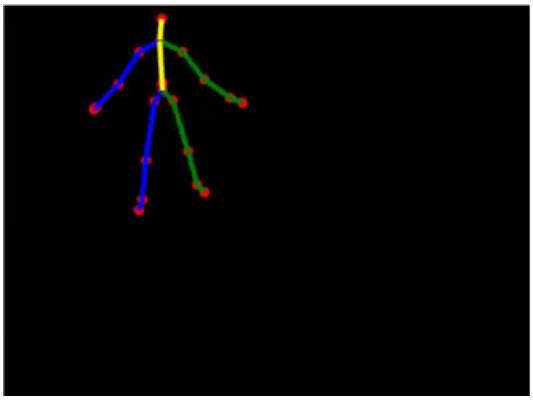

Figure 2.3. A view of a merged skeleton in the global coordinate system. The joints of the user are represented by dots and bones between two joints are drawn as lines (here the right and left side of the user are drawn in different colors.)

2.10 Latency

Latency is an important concept when working with filters. Latency can be defined as the time it takes for an input to a system to produce the corresponding output. One example would be the time it takes for the avatar to respond to a movement of the user. For most people, latency becomes a problem if it exceeds 100 milliseconds. Above this limit, the response of the system to the user's input begins to feel too delayed to be comfortable [2]. A related concept is joint filtering latency. This is a measure of how long it takes for the filter output to catch up with the actual joint position (if the joint is moved for real). There is generally a tradeoff between the smoothing effect of a filter and the delay it introduces. In this particular application, the latency is of major importance for the user experience and the chosen method must not introduce too much latency, or it will not be useful, regardless of its precision.

Chapter 3

Implementation details

In this chapter we go through the steps to implement the chosen algorithms and methods. We also look at initial choices related to the implementation and development work as well as the main components of the resulting program.

3.1 General considerations

3.1.1 Requirements for implementation

For this application, one of the most important characteristics is that the algorithms need to be executable in real-time. More concretely, we define real-time to be when computations can be carried out quickly enough for 30 frames per second to be within reach. This is needed for the application to be free from perceived lag. Another goal is to improve the tracking over the tracking offered by the Kinect out of the box. More specifically, the algorithm should provide more stable estimates over a larger interaction area than one single Kinect device. It should also benefit from the increase in available observation data provided by multiple Kinects. Another important aspect is to find a good balance between smoothness between frames at one hand, and reactivity to rapid user movements on the other. Perfecting both aspects at the same time is often not within reach, which is why we need to speak about finding the right balance between the two.

3.1.2 Kinect for Windows SDK and Visual Studio

One important resource to this project for the implementation is the Microsoft Kinect for Windows SDK (Software Development Kit). This is a library that makes it easier for programmers to access data from the Kinect and it also comes with various built-in features to help exploit the data. For this project, the latest version available at the time of writing, the SDK 1.8 [15] was used. Aside from providing an interface between the hardware and the software side, the SDK also comes with a rather comprehensive documentation to ease development.

The actual coding was done in C# using Microsoft Visual Studio 2012. When programing with the Kinect using the Kinect SDK, C# is a natural choice as it highly compatible with the Kinect SDK. The choice of using Visual Studio was made in order to simplify the actual implementation on a Microsoft Windows-based platform as it provides a long list of tools making C# programming efficient.

Thanks to the Kinect SDK, programmers have access to an estimation of up to two users' joint positions estimated from depth data. The obtained skeletons consist of 20 joints as can be seen in figure 3.1 and the position estimates of each joint are available at a frame rate of 30 frames per seconds [14]. Each joint is labeled to be easily accessible for developers. If the Kinect is not able to view a given joint, its position cannot be estimated directly, but will be inferred by the Kinect estimation software. Such an inferred joint position may be less accurate than a properly estimated one and to mark this fact, the joint is given a tracking state "inferred". For more detailed technical specifications and information on how the Kinect works, please refer to Appendix A.

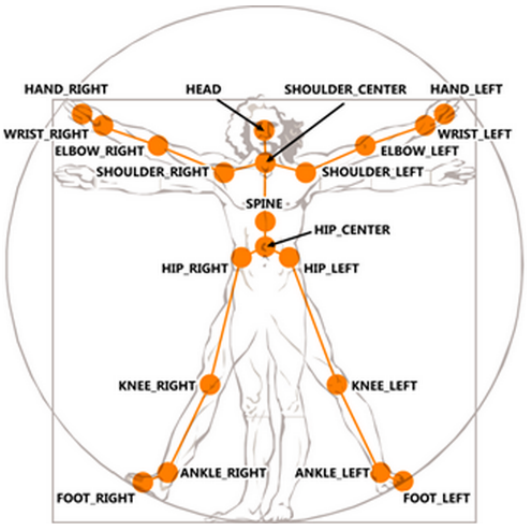

Figure 3.1. Image shows the joints estimated by the Kinect SDK. Image source: [16]

3.1.3 Hardware and restrictions

All aspects of the development, testing and evaluation of the system implemented in this project were carried out on a portable Ultrabook computer with an Intel Core i7-3517U CPU with 2 physical cores (4 virtual cores) running at 1.90 GHz. The computer has two separate USB 3.0 controllers (and a total of three USB 3.0 ports). This limits the number of Kinects that can be connected to the computer to two

devices as each Kinect requires its own USB controller. Although the algorithms themselves can handle a greater number of Kinects, the system has due to the hardware restrictions only been tested with two simultaneously connected Kinects. This is, however, sufficient for testing and evaluating the performance and efficiency of the system.

3.1.4 The camera setup

Two main camera setups have been tested and compared throughout this project, they can be seen in figures 3.2 and 3.3 below. The calibration of the cameras is possible as long as the field of view of the cameras are partially overlapping. As discussed in the section on hardware and restrictions, only two Kinects can be simultaneously connected to the computer. An interesting setup to test would consist of four Kinect cameras placed at 90-degree angles between them resulting in a full 360-degree view of the user with the potential of providing even more reliable tracking.

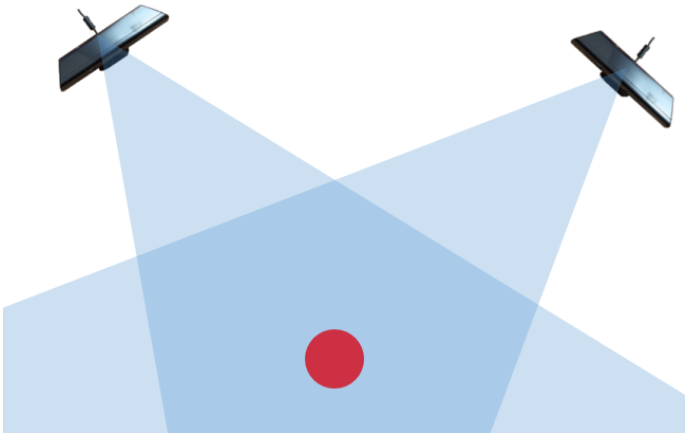

Figure 3.2. One possible camera setup that was used in the project. The cones show the field of view of the Kinects. The user, represented by a dot, can position himself in regions viewed by at least one Kinect. The angle between the cameras can be varied but is around 90 degrees in this setup.

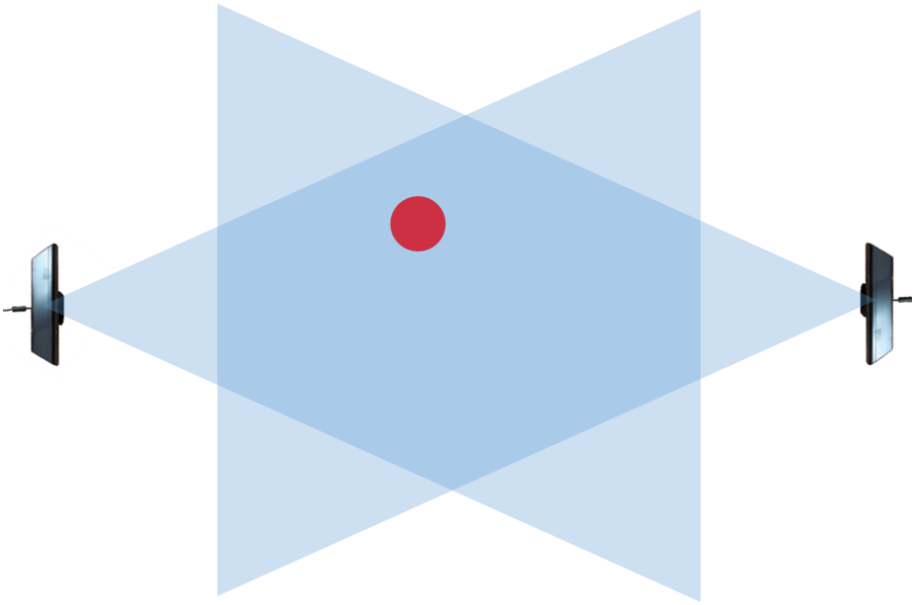

Figure 3.3. Another possible camera setup that was used in the project. The cones show the field of view of the Kinects and the user, represented by a dot, can position himself in regions viewed by at least one Kinect.

3.2 The chosen methodology

3.2.1 Overview

The implementation of a working system for pose estimation using multiple Kinect devices is based on the methods discussed in the theory chapter. In this section, the focus is on the actual implementation details. However, some details are omitted to increase readability and keep the report on a suitable technical level. Below is the overall approach chosen for this project along with a brief explanation of why a certain part of the algorithm is necessary or why it may improve the overall quality of the algorithm's output.

1. Define a global 3D-coordinate system in which the data from all the Kinects can be expressed and worked with. This requires an extrinsic camera calibration and for this, a calibration algorithm not requiring a printed chess board was chosen as it is quicker and easier to use.

2. Transform the joint position estimates from each of the Kinects into the global reference system. This is required since the individual Kinect cameras provide their skeleton data in local camera coordinate systems.

3. Create a system to evaluate the quality of observations from the different connected cameras. This step is justified as we want to let the most reliable data influence final joint estimates more than less reliable data.

4. Merge the joint position estimates from the two Kinects using filters and different ways to combine data. Two main approaches for the filtering part are considered in this project, namely a particle filter and a Kalman filter. It was not known beforehand which of these would perform the best, and implementing the two would make it possible to compare them and find the most suitable approach for the application considered in this project.

5. Apply constraints on the merged skeleton to avoid stretching and eliminate impossible poses. This step acts as a means of correcting the joint estimates when they are considered as a whole human skeleton and not only a collection of individual joints. With this more global perspective, impossible poses can be corrected before generating the final skeleton estimate.

6. Draw the estimated skeleton (i.e. its joints and bones) in 3D in real-time to visualize results. This step is needed to make sure that the estimations can be run in real-time. Being able to see the final result instantly as a user moves in the interaction area of the Kinect cameras makes it possible to assess the feel of the output and how well the algorithms work in real usage scenarios.

7. Implement a module for recording camera data to enable offline analysis and plotting of results. This is useful for getting a more precise appreciation of the results. At the same time, the offline analysis allows different filter configurations to be compared on the same sequence of recorded data. This is important both for understanding results and for the process of tuning the filters.

3.2.2 Program architecture

In order to implement the approach listed above in working code, a simple architecture was chosen which can be seen in figure 3.4 below. The most important components are described in the following sub sections.

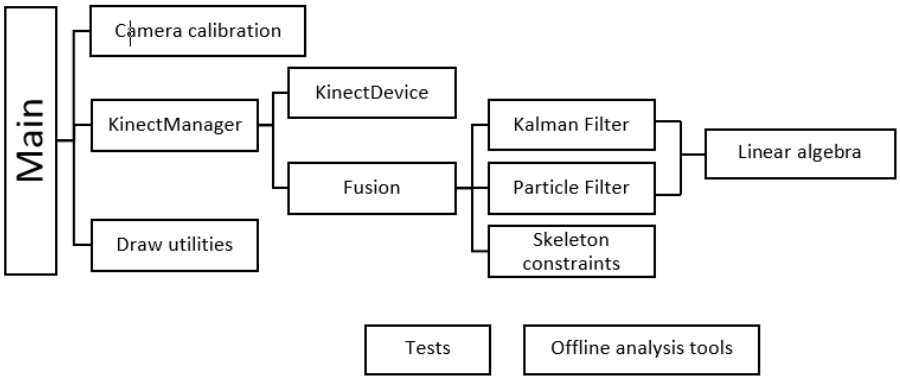

Figure 3.4. A simplified diagram showing the program architecture. Only the main components are included for clarity.

3.2.3 Manager for multiple Kinect devices

When using multiple Kinect devices there is a need to synchronize frames from each connected device so that the combination of data is done on data from the same time step. Here, a Kinect manager following an observer design pattern was implemented to make sure that the algorithm, at each time step, runs only when all connected cameras have signaled that data is available. Custom events have been defined to notify other components when, for instance, a merged skeleton is available.

Each of the Kinects are handled as KinectDevice objects that contain data specific to the Kinect in question. The information includes the device ID, the transformation matrix for converting data from local to global coordinates as well as the joint data in local and global coordinates.

3.2.4 Camera calibration algorithm

For this project, a calibration algorithm developed by Mikael Hedberg at Centive Solutions GmbH was used. The algorithm does not require a printed chess board for the calibration and is therefore user-friendly and suitable for applications where users cannot be assumed to be at ease with calibrating a system using conventional methods. The algorithm consists of the following steps:

1. The floor plane is detected using built-in routines in the Kinect SDK.

2. The user manually selects an origin and coordinate axis orientations by clicking on the screen.

3. The user then moves around in an area within the field of view of all connected cameras. Data is collected from each of the cameras during a few seconds and stored temporarily.

4. One of the cameras in the setup is chosen as the reference. For the remaining camera(s), the translation vector and rotation matrix that minimize the mean square error with respect to the reference camera are obtained using singular value decomposition and the iterative closest point algorithm. For a more detailed description of these algorithms, see for instance [19] and [4].

5. For the reference camera, the transformation matrix is a simple rotation and transformation of the local camera coordinate system to the user-defined global coordinate system.

6. For each of the non-reference cameras, the resulting transformation matrix is the product of:

   a) the transformation from the given camera's local coordinate system to the coordinate system of the reference camera

   b) the transformation matrix of the reference camera to the global coordinate system

Each camera's calibration matrix is stored in its corresponding KinectDevice-object as described in section 3.2.3 above.

3.2.5 Sensor fusion

One important aspect of successfully merging data from multiple measurement devices is how to combine the data from the different sources. In this project a few different approaches to combining measurements from multiple Kinects were implemented and tested to see how the methods compared under different circumstances. Below we go through areas that were considered for the sensor fusion algorithms.

Taking joint tracking status into account

Regardless of the approach, it is useful to have a measure of the confidence levels of observations from each of the connected Kinect devices. The position estimate of each joint in a skeleton provided by the Kinect SDK has a parameter indicating whether the joint in question is tracked or inferred. If a joint is clearly seen by the camera (that is, it is not outside the field of view of the camera or occluded by an object or by the user) then that joint is considered to be *tracked*. If this is not the case, the joint in question is given the status $inferred$. In short, the estimated position of an inferred joint is thus less certain than the estimated position of a tracked joint. This means that if there are cameras that see one joint properly, and others that do not, then the inferred joint estimates can be discarded as they are likely to decrease the accuracy of the final joint estimate. If on the other hand,

several cameras see a joint as tracked, then each of the observations are taken into account. If none of the cameras see a given joint as tracked, then inferred joint estimates are taken into account.

Choosing the best camera at a particular time

The Kinect system has been trained to work well in cases where the user is facing the camera straight on. It can therefore be useful to compute a camera's overall confidence level and not only on the individual joint level. A camera's confidence level is determined by the number of joints out of the maximum of 20 that it is able to see and track. This is based on the assumption that a camera that tracks a large number of joints can be considered to have a good view of the user. The camera that achieves the highest percentage of tracked node at a given time is assigned the status of best camera. The best camera can be given a higher weight for its tracked joints, when combining them with observations from the other cameras.

For certain camera setups, it may also be reasonable to assume that one camera focusing on the left side of the user is likely to provide more accurate estimates of the left part of the skeleton and the camera focusing on the right side of the user providing more accurate estimates of the right side. Taking into account this a priori knowledge, rather than assigning a single best-camera based on the entire skeleton, is done in an approach referred to as the extended best-camera approach in the results chapter.

Mirroring skeletons for certain camera setups

For certain camera setups it is important to relabel joints so that detected joints in the different cameras correspond to the same actual node. In the camera setup where two cameras are facing each other, a simple solution is to change left and right nodes for one of the cameras. Such a function has been implemented and the user can choose to activate it in cases where this resolves labeling problems. In other setups, such as the one indicated in figure 3.2, this is not necessary as all joints already have the same label in the connected cameras. This simple mirroring solution was chosen as it is simple to implement and does not require any additional hardware or markers. It was also a choice based on available time for the project. It would be interesting to look into more advanced solutions as future work.

How measurements from multiple cameras are combined

Below we go through the implemented ways of combining measurements from several cameras in the case where at least one camera provides joint estimates with a *tracked* status:

1. Sequential updates, i.e. running through the update steps in the filtering algorithms once for each sensor at each time step.

2. Unweighted average, i.e. taking the geometric average for the position given by each sensor and then running through the update step for the filtering algorithms once with this combined observation.

3. Weighted average, i.e. taking the weighted geometric average of the position given by each sensor, where the weights are determined by the overall confidence level for the camera. Then the update step is run once for the combined observation.

4. If only one camera delivers a tracked position of a given joint, then only this observation is taken into account as it is likely that taking inferred joint positions into account would decrease the accuracy of the final position estimate of that joint.

If a given joint is not tracked by any of the cameras, then an inferred joint position from the best camera (highest confidence score) is used for updating the filter. This situation may occur if a joint is outside the field of view of all connected cameras, or occluded in a way that no camera can see the joint in question. In this case, it is difficult to provide a good estimate, so the inferred joint position is only used as a last resort. In this situation, the best camera is likely to give the best inferred position as we assume that that camera has the best view of the user. The inferred positions from the other camera(s) are therefore not taken into account.

Discarding outliers

In order not to take into account highly noisy measurements, there is a simple threshold based on the distance from the new observation to that from the previous time step. The threshold is adjustable on a per-joint basis so that joints that are often noisier, such as the lower legs and feet of the user, can be treated differently with respect to outliers compared to joints that are more often observed with higher accuracy. In addition to the threshold, we can also, on a per-joint basis, force noisy measurements to be taken into account after a number of frames. This way the filter can be reinitialized if tracking is temporarily lost and estimates lag behind the actual position of the user. Depending on the used motion model for a given joint, an alternative to the above mentioned distance measure is to consider the distance between the predicted position for time t with the corresponding observation(s) at time t and discard the observation if it is deemed too far off the prediction.

For the particle filter, an additional outlier suppression can be applied to particles that turn out to stray too far off the measurement. This may be useful in situation where there are relatively few particles and where outliers can have a disproportionally big influence on the final estimate. With sufficiently many particles, however, these far-off particles will be given small enough weights not to play a big role in the final estimate in any case.

3.2.6 Pose estimation using particle filtering

The Particle filter implemented in this project is of the standard type described in the theory chapter, section 2.7. As motion model, a simple zero velocity model was implemented. This would serve as a starting point for evaluating whether the algorithm could be run in real-time before considering more advanced models. The zero-velocity motion model is described in section 2.4.1.

For a particle filter to work properly, a sufficiently large number of particles are needed to accurately describe the probability distributions on which estimations are based. As the required number is not usually known before-hand, the number of particles can easily be modified in the program so as to find the optimal balance between filter accuracy and speed. When it comes to the update and resampling steps used for the particle filter, the equations are given in the theory chapter, section 2.7. Variance parameters are modifiable online as the program is running so that the effects of different parameter values can be studied in real-time.

As a means of stabilizing the particle filter output in real-time runs, we allow the user to discretize the state space. This effect is achieved by rounding off final particle positions to a user defined number of decimals which works in the same way as creating a 3D grid for the interaction space. The finer the grid is made, the more precise the position estimate can potentially become, but there is a risk that a greater number of particles are needed to provide stable estimates at that level of grid fineness. Note that the particle filter can also be run without this discretization of the state space.

For the final joint position estimations, two different approaches for extracting an estimate based on the probability distribution made up of the particle samples were implemented. The first is a standard histogram approach where the estimate is given by the position indicated by the biggest number of particles. The alternative approach is a weighted mean of all particles' positions with their respective weights taken into account. This approach is computationally less demanding, but beforehand expected to yield lower accuracy.

In an attempt to boost performance of the particle filter, the possibility to run parts of the code in parallel on multiple cores was implemented as well.

3.2.7 Pose estimation using a Kalman filter

The Kalman filter that was implemented in this project is a standard 3D Kalman filter which is explained with all equations presented in section 2.6. Here, a motion model taking velocity into account was studied as the Kalman filter does not require a lot of computations compared to the particle filter. This choice of motion model means a higher dimensional state vector than the one used with the particle filter

with 6 rather than 3 elements. The constant velocity model is described in section 2.4.2. The covariance of the filter can be adjusted while running the program so as to be able to evaluate the effects of different parameter values immediately.

3.2.8 Application of body constraints

Once the estimates of all the joints of the user's body have been obtained by any of the filtering and merging techniques discussed in the previous sections, body constraints are applied to correct impossible or highly unlikely poses. Bones are defined as lines connecting two joints. However, all joints are not connected to every other joint, so the skeleton is defined by the hierarchy in figure 3.5 below.

When it comes to bone length constraints, different individuals are of different height and have different bone lengths. In order to take this into account, the developed algorithm initializes itself by measuring the bone lengths of a user in a frame where the entire skeleton is tracked so as to get reliable results. Once this has been done, the bone lengths are stored for that user and applied to all frames from that moment on. A 5 % error margin is allowed before detected bone lengths are adjusted.

The bone constraints are applied to the merged and filtered skeleton as a last step of treatment, before the estimated skeleton is finally visualized or used otherwise. Here we have gone for a simple approach and there are more advanced methods available that may yield more accurate results.

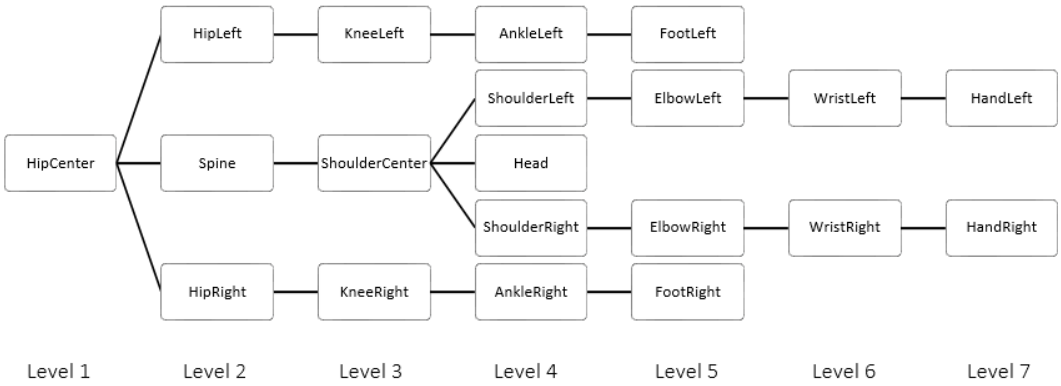

Figure 3.5. The defined hierarchy for joints in the skeleton. There are 7 levels in the tree and bones are defined as the connections between two nodes in the tree structure.

3.2.9 Offline analysis tools and tests

For analyzing results in detail, a module that can record captured Kinect data that can be analyzed offline (i.e. without the Kinects being connected) was implemented. This makes it possible to study different filters and filter parameters as well as different fusion techniques offline on the same data sequence in great detail. It also makes it possible to analyze filter configurations that are too computationally complex to be run in real-time. Such an analysis is useful in understanding the characteristics of a filter to a degree which is not possible only through online analysis. A module for manually defining test sequences and plotting results was implemented. This was done in Matlab as it provides good tools for interaction with and visualization of data. Test modules allowing to test the different units (filter algorithms as well as the implemented methods for matrix operations) have been implemented to make sure that there are no errors that could affect the resulting output.

3.2.10 Visualization

We want real-time visualization to be able to analyze and compare the different approaches developed in this report. As the actual visualization is not the most important part of this project, a simple 3D visualization was implemented. Here, joints are drawn as spheres, and the bones connecting relevant joints in a skeleton are drawn as lines, see figure 3.6 below.

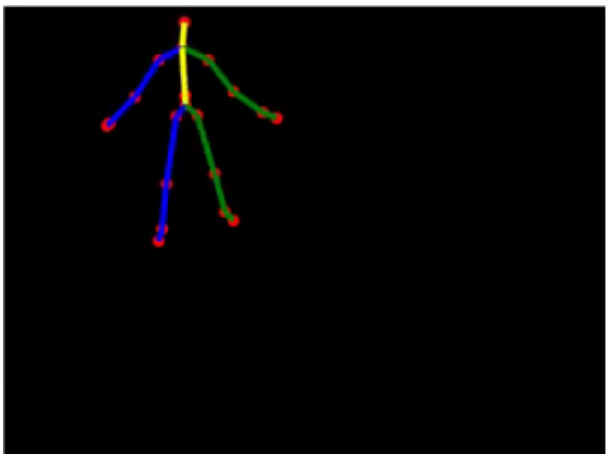

Figure 3.6. Image shows a merged skeleton in the global coordinate system. Joints are represented by dots and bones are drawn as lines. The right and left side of the user are colored differently to help distinguishing left from right.

Chapter 4

Results and experiments

In this chapter we go through the results of the conducted experiments to evaluate the algorithms described in the previous chapter. The evaluation consists of a quantitative and a qualitative part aimed at evaluating different aspects of the algorithms. The chapter is divided into two parts accordingly to make the presentation easier to follow.

4.1 Quantitative evaluation

In this section we go through the evaluation of the algorithms, which was carried out on a sequence of frames where joint positions had been manually defined for each frame. This manual estimation of joint positions was done on depth data in the form of point clouds from the connected cameras. For the test sequence, all available data from two connected Kinect cameras was recorded, including the point clouds and the joint estimations of the user. The camera setup used when recording the sequence was the one illustrated in figure 3.2. The length of the sequence was limited to 91 frames due to large quantity of data that had to be stored for each frame. This corresponds to a sequence of approximately 3 seconds, which is enough to comprise a sufficiently complex movement pattern (including rapid arm and leg movements of the user) to make the evaluation meaningful.

The recorded point clouds from the individual cameras were then merged. A Matlab script that allowed a human user to click on the screen to manually classify each joint position in each merged frame was used to define the test sequence, see figures 4.1 and 4.2. The performance of each filter type with different parameters and combinations of activated features could then be evaluated. This was done by simply comparing the output from the filter with the manually defined joint positions of the test sequence.

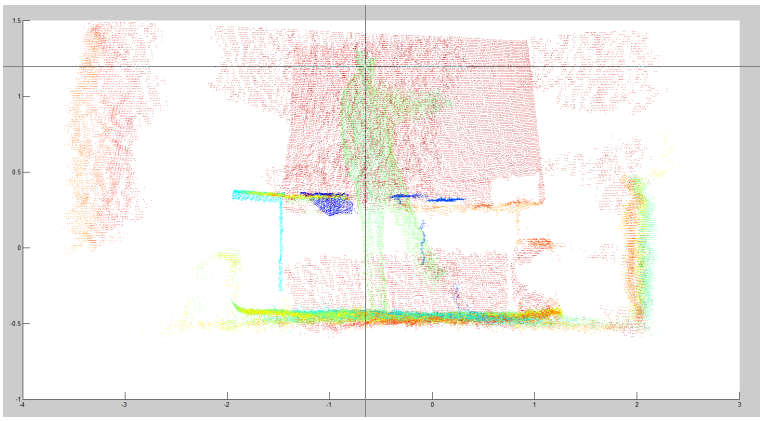

Figure 4.1. The Matlab module created to manually define joint positions as a reference for evaluation. The user clicks on the screen to define joint positions in each frame seen from different angles for accuracy.

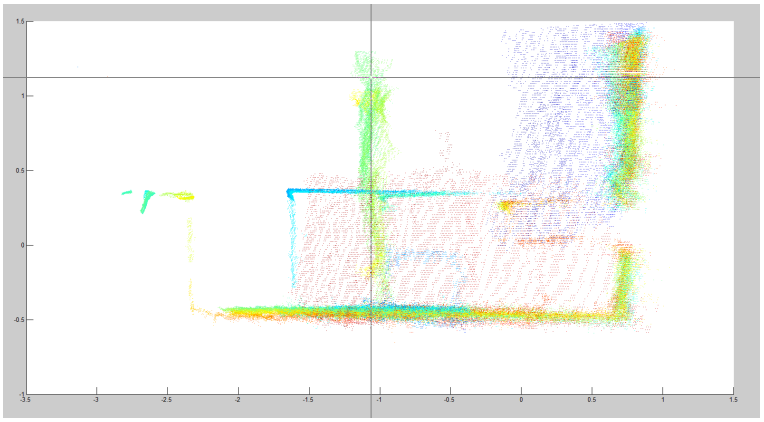

Figure 4.2. The Matlab module with the point cloud seen from the side to define the depth from the camera plane.

4.1.1 Results of quantitative evaluation

We will now go through the output of the filter algorithms on the test sequence with different filter configurations. The results of the runs with the Kalman filter are summarized in figures 4.3, 4.1 and 4.4 and those of the particle filter in 4.5, 4.7 and 4.6. These figures all show the sum of squared discrepancies in the X-, Y-, and Z-coordinates between the manually classified joint positions and the output from the filter algorithms. The error sum is taken over all joints and all frames in the evaluation sequence to produce the total error. To make these numbers meaningful, the error between the individual camera estimates and the reference sequence are computed and presented in the same way.

It is interesting to note that one of the cameras performed considerably better than the other, with a total error of 3.79 m^2 compared to 26.02 m^2. The average camera error was 14.91 m^2. When analyzing the sequence of frames closer, it could however be noted that the camera yielding the biggest error, did not get a good view of the left side of the user, resulting in a high total error. Furthermore, the rate of un-tracked nodes for camera 1 was 4.8 % and 2.3 % for camera 2. This shows that the fact that a node is tracked is not necessarily a guarantee for accurate estimates.

The best performing approach for the Kalman filter is the extended best-camera approach as can be seen in figure 4.3. This approach lets one of the cameras deal with the left-hand side of the user, and the other camera with the right-hand side. This approach also performed better than the camera that yielded the lowest error, meaning that information extracted from the low-performing camera could actually improve final estimates over a one-camera setup. The remaining ways to combine data did not perform as well, but they all returned considerably lower errors than the average camera error. The best camera approach, where the best camera for the entire skeleton is assigned, achieved a second place in terms of total error, closely followed by the weighted average approach. The weights were chosen as follows: 0.8 for the best camera and 0.2 for the remaining one. The sequential update rule assigning equal importance to tracked joints from both cameras performed worse, but still considerably better than the average camera.

When it comes to filter parameters, figure 4.3 shows that a process covariance of 0.1 for the Kalman filter returned a lower total error values than smaller parameter values. The smaller the value of this parameter, the higher the degree of filtering between frames. This makes the transitions from one frame to the next less responsive to changes in observed data. In the test sequence, the rate of tracked nodes was high for both cameras, meaning that random flickering between frames was limited. Such flickering, if present, could otherwise be reduced with a higher degree of filtering.

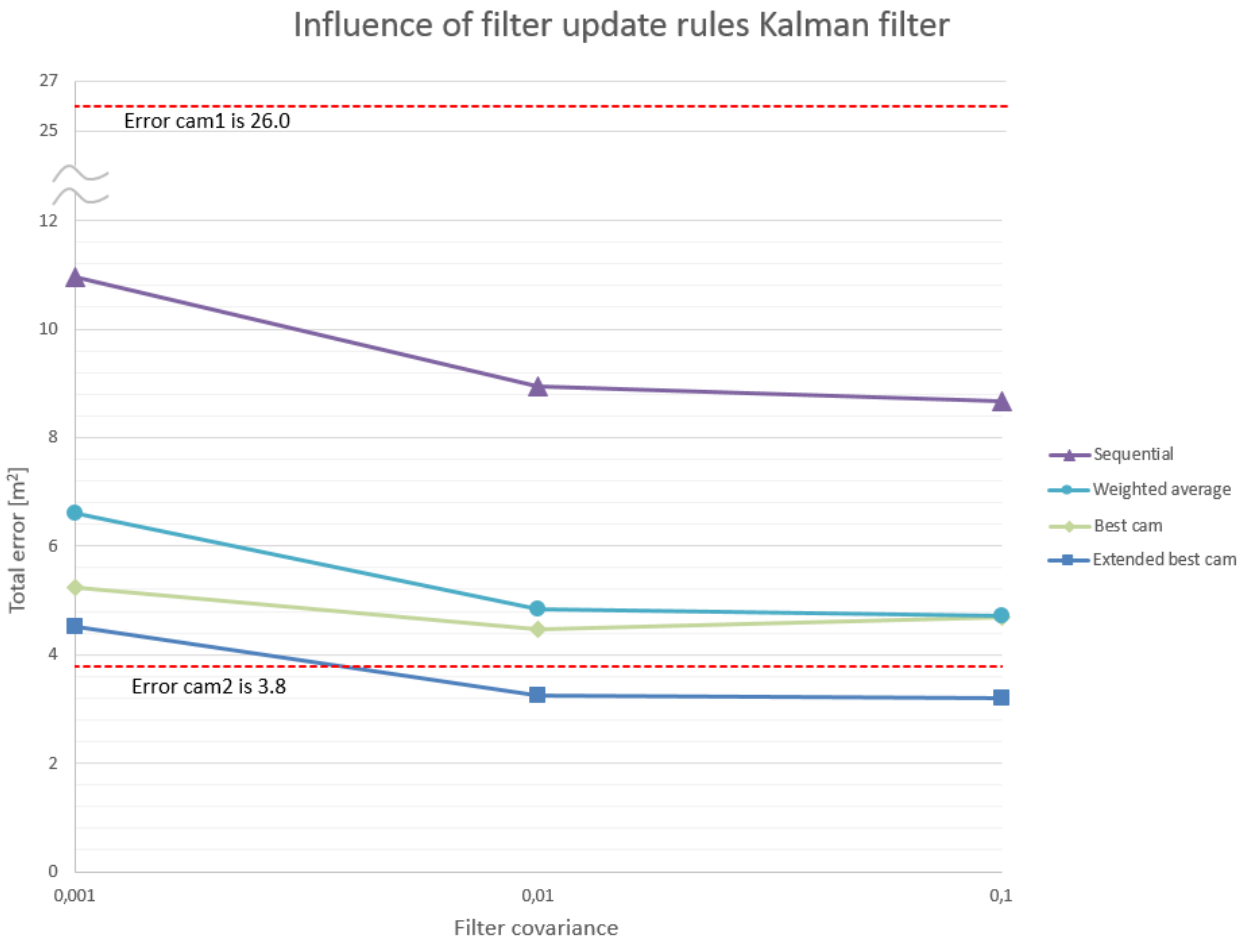

Figure 4.3. Four different update rules for incoming observations to the Kalman filter are applied and compared in terms of total error with respect to a test sequence with manually defined joint positions. The extended best camera approach produced better estimates than the single camera with the smallest error when the filter process covariance was set to 0.1 or 0.01. All approaches performed considerably better than the average camera error (14.91 m^2).

Furthermore, it was noted during experiments that the choice of motion model for the Kalman filter did not make a significant difference with a process covariance value of 0.1. At this level, observations are taken into account more than when the covariance is smaller, meaning that the filter relies more on the motion model and less on actual observations. However, with lower process covariance values, the differences become greater favoring the model with constant velocity compared to the model where the velocity is assumed to be 0. As the differences are in many cases small, the results for the comparison of the two motion models along with different update rules are presented in a table, see table 4.1. The trend that the difference in total error between motion models is small for a covariance value of 0.1-0.01 and becomes bigger the smaller the covariance value holds true for all the studied update rules.

Figure 4.4 shows the results of applying outlier suppression and skeleton constraints with the Kalman filter. When enabling the outlier suppression, the results are identical for the test sequence compared to when the feature is disabled, indicating that there were no outliers above the chosen threshold for changes in joint positions between frames. Note that the algorithm does not take into account the manually defined position when determining whether an observation should be discarded or not, as this information is only available for the test sequence. Applying body constraints, adjusting the lengths of bones in case they are deemed too long or short, actually increased the total error slightly. The bone lengths are determined from the actual observation frames, meaning that if the frames used for determining the bone lengths contain incorrect joint positions, it would result in unwanted corrections for the remainder of the sequence.

Table 4.1. Two motion models with the Kalman filter are compared in terms of total error over the test sequence. Differences between the constant velocity model and the zero velocity model are small for a process covariance of 0.1 and 0.01. The difference becomes larger with the lowest tested covariance value of 0.001. This holds true for all evaluated update rules.

Update rule	Filter Co-variance	Motion model	Error cam1 [m^2]	Error cam2 [m^2]	Error cam avg. [m^2]	Error merged [m^2]
Ext. Best cam	0.1	v=const.	26.02	3.79	14.91	3.18
	0.01					3.24
	0.001					4.52
Ext. Best cam	0.1	v=0	26.02	3.79	14.91	3.19
	0.01					3.25
	0.001					4.95
Best cam	0.1	v=const.	26.02	3.79	14.91	4.70
	0.01					4.46
	0.001					5.24
Best cam	0.1	v=0	26.02	3.79	14.91	4.70
	0.01					4.47
	0.001					5.72
Sequential	0.1	v=const.	26.02	3.79	14.91	8.66
	0.01					8.93
	0.001					10.97
Sequential	0.1	v=0	26.02	3.79	14.91	8.68
	0.01					9.13
	0.001					12.09
Weighted avg.	0.1	v=const.	26.02	3.79	14.91	4.72
	0.01					4.83
	0.001					6.61
Weighted avg.	0.1	v=0	26.02	3.79	14.91	4.74
	0.01					5.00
	0.001					7.76

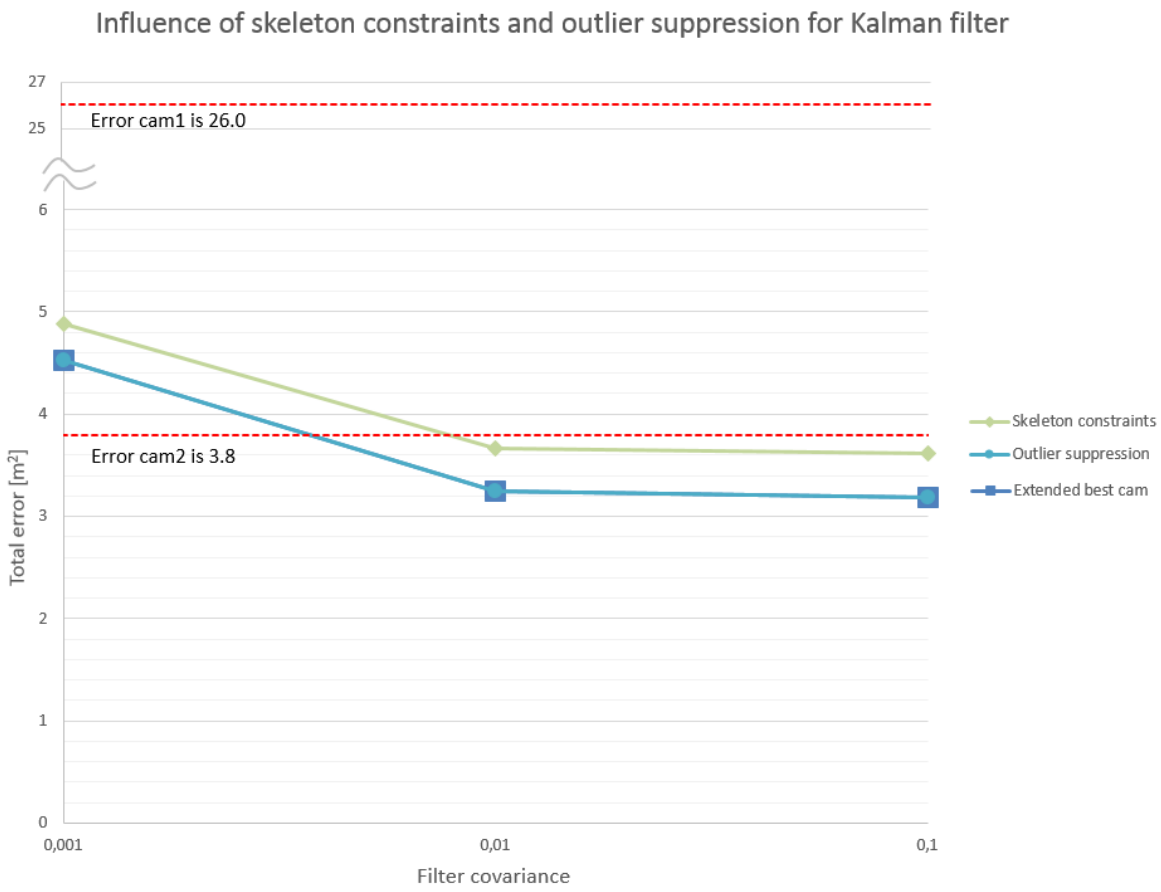

Figure 4.4. Figure shows total error over the test sequence of frames with skeleton constraints and outlier suppression separately enabled compared to a filter with both features disabled. The outlier suppression did not produce any difference in total error. Consequently, the graphs with and without outlier suppression enabled are superposed in the figure. Activating skeleton constraints actually increased the error over the test sequence.

When it comes to the particle filter, three measurements were carried out for each tested filter configuration and the average total error was computed in the end. This is due to the probabilistic nature for the particle filter, meaning that one run will not produce the same results as the next. As for the Kalman filter, the extended best-camera approach produced the smallest total error for the particle filter out of the evaluated update rules, as can be seen in figure 4.5. For the other update rules, the results were the same as for the Kalman filter, i.e. the best-camera approach performed slightly better than the weighted average approach, and the sequential approach performed the worst on the test sequence. The latter had a bigger total error than the average camera on all particle count levels.

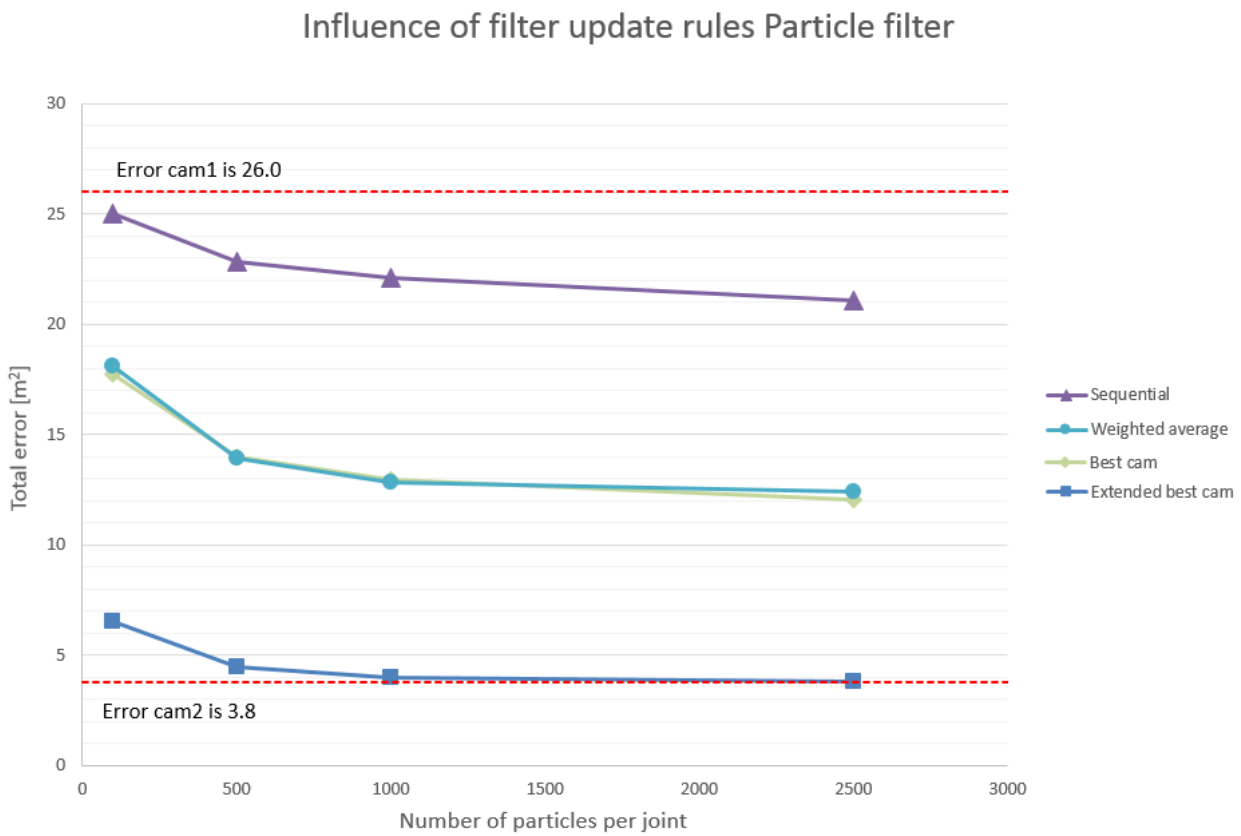

Figure 4.5. Four update rules were evaluated with the particle filter on the test sequence of frames. The extended best camera approach performed on par with the individual camera with the lowest error at 1000 particles per joint and slightly better at 2500 particles per joint. The weighted average and best camera approaches performed better than the average individual camera from 500 particles per joint and up.

To achieve similar or better performance than the best of the individual cameras, 2500 particles per joint were required. For 2500 particles or more, it also turned out to make sense to increase the fineness of the grid for the particle simulations to a mm level rather than cm level in order to decrease round-off errors, see figure 4.6. For a lower particle count, the number of particles was not sufficient to benefit from the finer grid and as a result performing worse than with a cm grid.

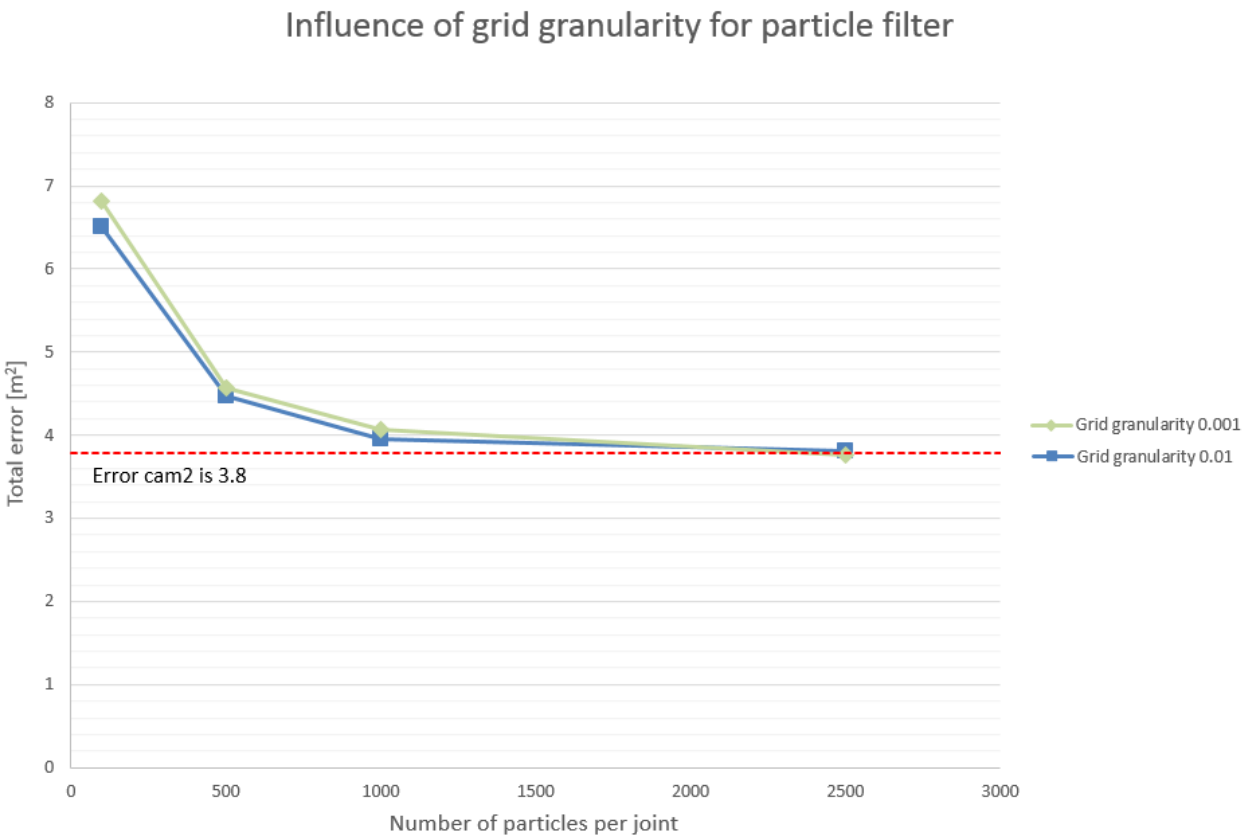

Figure 4.6. Figure shows the impact of changing the (optional) grid size for the particle filter (0.01 corresponds to centimeter-level accuracy and 0.001 millimeter-level). At less than 2500 particles per joint, the 0.01 level performed better, but at 2500 the number of particles were sufficient to produce a smaller total error on the test sequence.

The results when enabling skeleton constraints in combination with the particle filter yielded similar results compared to the Kalman filter under the same conditions, see figure 4.7. This is expected as the same way of adjusting bone lengths was applied to both filters. As for the Kalman filter, enabling outlier suppression produced no noticeable difference in terms of total error on the test sequence.

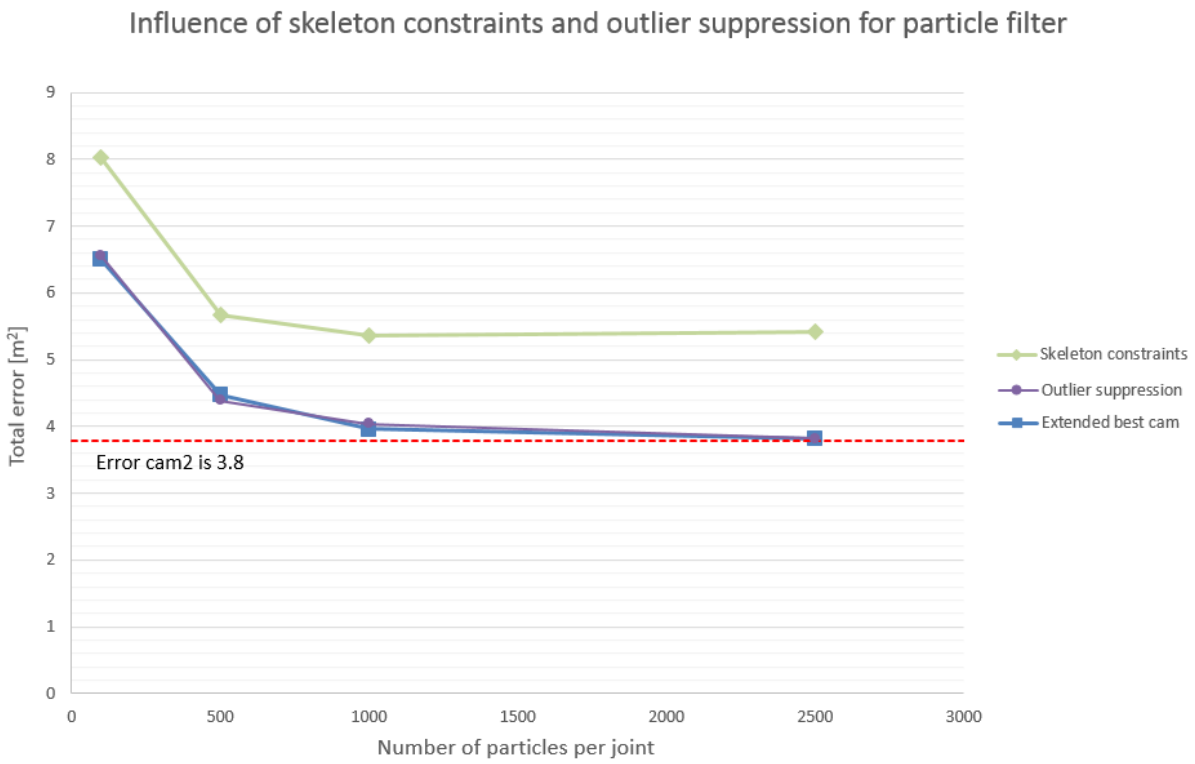

Figure 4.7. Figure shows impact of separately enabling skeleton constraints and outlier suppression for the particle filter. Outlier suppression made no difference beyond the probabilistic variations introduced by the particle filter. Enabling skeleton constraints increased total error on the test sequence.

4.1.2 Influence of the particle count in the particle filter

In the above sections, we have established how particle filter performance depends on the number of used particles per joint. However, we also need to take into account the time required to carry out the necessary computations at a given particle count per joint level. Generally speaking, there is a tradeoff between the number of particles and computation time. In offline applications, the problem of computation time is less critical than in real-time applications where there is a limit on how long computations per frame are allowed to take before the frame rate drops below acceptable levels. In order to evaluate the performance of the implemented particle filter as a function of number of particles per joint, timed runs with different particle counts were conducted. The results shown in figure 4.8 were performed on a computer with hardware as described in section 3.1.3. Note that these computations were carried out in offline mode on recorded sequences of data, without visualization overhead.

Our target output frame rate of 30 frames per second translates into approximately

33 ms per frame for the full skeleton (including visualization overhead). In full skeleton runs with real-time visualization, the limit for maintaining 30 frames per second was found to be around 200 particles per joint, compared to around 500 particles per joint during offline calculations without visualization. Beyond this point, the algorithm could not keep up with the feed of observations from the Kinects. Comparing the particle filter to the Kalman filter in terms of computation time per frame, the Kalman filter is 5 times faster than the particle filter with 100 particles per joint – a considerable difference. For greater particle counts, the difference increases further.

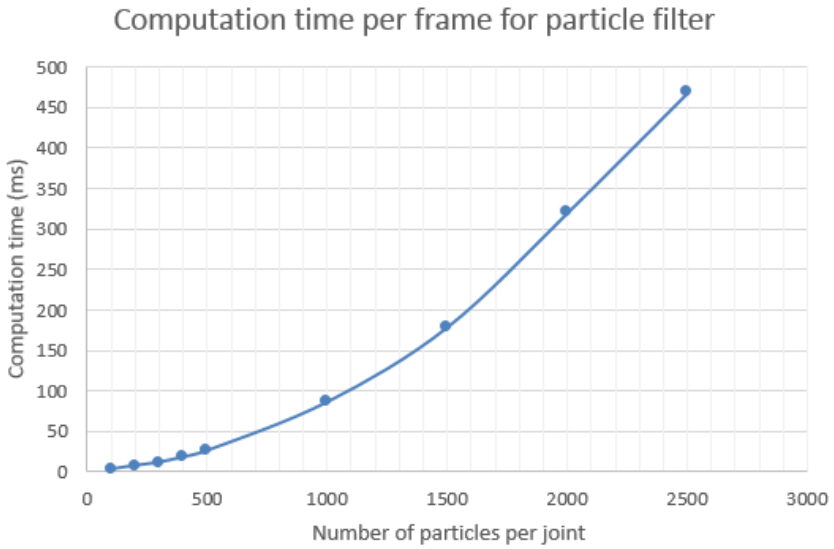

Figure 4.8. Graph shows the required computation time for the particle filter to estimate the position of all 20 joints for one frame.

4.2 Qualitative evaluation

When evaluating the usability of the developed methods in real applications, the quantitative analysis above is not sufficient for evaluating every aspect of the algorithms at hand. For instance, one key criteria that needs to be fulfilled for the algorithm to be usable is that all components can be run in real-time, i.e. at least 30 frames per second. As the computations in the quantitative section above are carried out offline, this aspect is not taken into account. Since the implemented algorithms are designed to provide a smooth yet sufficiently reactive user experience, we need a way to assess these aspects as well. This was done using real-time visualization of the input to and resulting output from the algorithm being evaluated.

Figure 4.9 below shows a screenshot from the test program used for evaluation.

The camera data from the individual cameras connected to the system is shown along with the output from the evaluated merging algorithm. The merged skeleton is drawn from the point of view of camera 1 for reference. In the skeleton data from the individual Kinects, bones between joints where at least one of them have an inferred position are colored red.

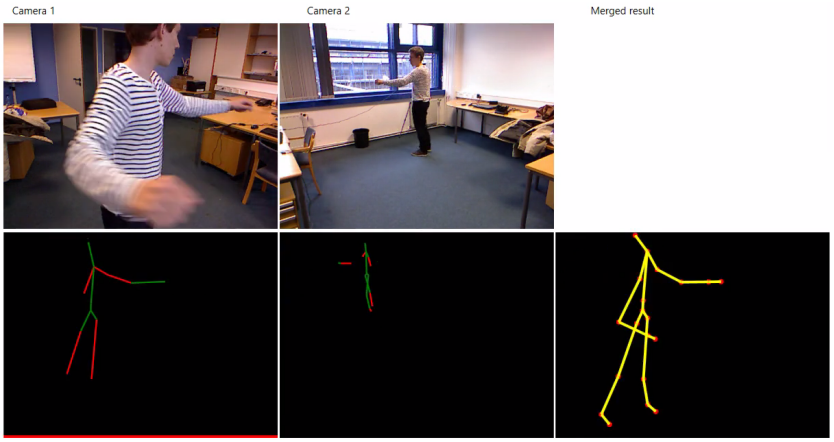

Figure 4.9. The views from two connected cameras. From the top left to the bottom right corner we find the following information: video stream from Kinect 1, video stream from Kinect 2, skeleton data from Kinect 1, skeleton data from Kinect 2 and finally the output from the fusion algorithm.

In the following sections, we go through different qualitative aspects of the results. We treat scenarios where a single camera can be compared to a multi-camera setup in terms of occlusion handling, size of interaction area and how well the system handles situations where the user is turning (i.e. changing his orientation within the interaction area). The influence of different filter parameters and filter features is described as well.

4.2.1 Influence of the camera setup

According to tests, the Kinect system is not capable of determining whether a user is facing the camera or turning his/her back to it. This can be seen in figure 4.10 below where the left-side bones are drawn in a different color to the right-side bones. Camera 1 correctly identifies that the left arm is held up, whereas camera 2 identifies the arm being held up as the user's right arm. This would be correct if the user was actually facing the second camera, which is not the case. This has implications for the camera setup shown in figure 3.3. In order to correctly combine data from the two cameras with that camera setup, the joints of one of the cameras need to be relabeled before taking into account its observations.

When running experiments with the two proposed camera setups, it became clear

that they both have their advantages and drawbacks. The camera setup shown in figure 3.2 provides good visibility of the user in all positions in a 180-degree-range as the user is roughly facing at least one of the cameras when within this range. However, this camera setup provides lower visibility of the user when neither of the cameras observe the user facing the camera. When this is the case, self-occlusions of arms become more frequent. Combined with the fact that the Kinects are not trained on data where the user is not facing the camera, the result is less accurate estimates. The second camera configuration, refer to figure 3.3, provides a large interaction area, but introduces issues when the user is turned 90 degrees with respect to both cameras. In this position neither of the cameras have a good view of the user, resulting in unstable joint position observations.

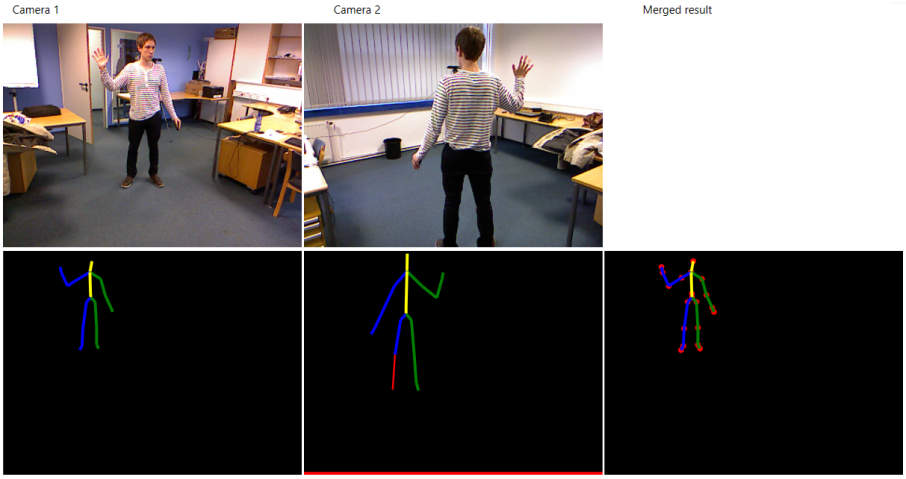

Figure 4.10. The image shows that the Kinect system cannot distinguish the front from the rear of a user. A user in front of a Kinect is assumed to be facing it.

4.2.2 Situations of partial occlusion

Situations where parts of the user's body are not in view for all cameras occur frequently. In the case of a one-camera-layout, the system does not get any observation data for an occluded region, leaving no other option but to infer the position of the affected joints. In a multi-camera system, there may be sufficient information about the entire skeleton if at least one of the connected cameras gets a view of a region occluded in the view of the other cameras. Occlusions can be due to either objects obscuring parts of the user's body, or the user being in a position where parts of the body are hidden from view in one or several cameras. Below, we find a couple of situations where the implemented algorithms successfully produced a full skeleton estimate in spite of the user being partially obscured. In figure 4.11 the legs of the user are occluded by objects. In figure 4.12 the user's body is not viewed in full by

any of the connected cameras. By combining data from the tracked joints in the individual cameras, the implemented system managed to produce a merged skeleton taking based on the partial information available through the cameras respectively. Both the Kalman and particle filter had similar performance in terms of dealing with partial occlusions as the occlusion handling relies on reliably combining data from all the connected Kinects and this was no problem for either of the algorithms.

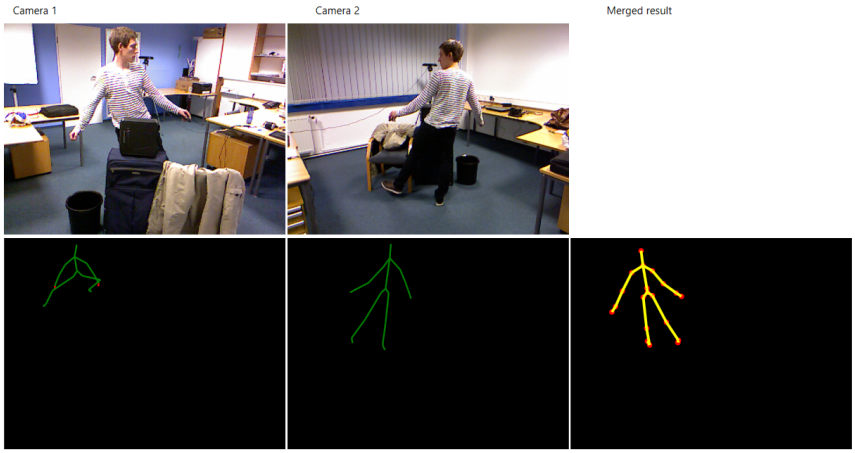

Figure 4.11. The user's lower body is occluded from view in camera 1. Thanks to camera 2 being able to view the legs of the user from behind, the resulting skeleton estimate is complete.

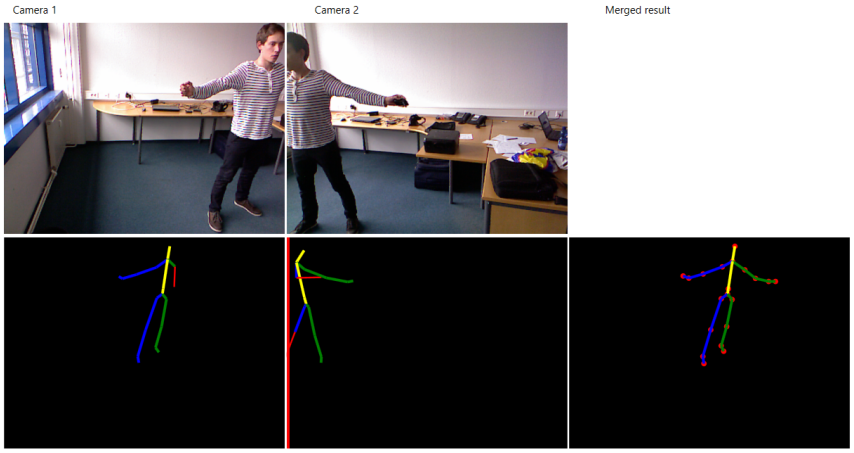

Figure 4.12. Here, the skeleton cannot be observed entirely by either of the cameras individually, but by merging the data from the individual cameras, the resulting merged skeleton can be constructed in full.

4.2.3 Increase in interaction area

Depending on the camera setup, the interaction area could successfully be made larger by adding additional cameras to the system compared to a one-camera setup. In figure 4.13 below, the user is seen leaving the interaction area of one of the cameras and entering a part of the interaction space only seen by the other camera. Not only did the size of the interaction area increase, but there is also a region of overlap where the user is seen by multiple cameras and where there is more information available about the user.

A common situation is otherwise that the user's feet fall outside the field of view of one of the cameras as the user gets too close to that particular camera. Another camera connected to the system can, depending on the camera setup, view the user from a greater distance and thus potentially get a view of the feet, again improving the skeleton estimate compared to a one-camera system. In terms of increase in usable interaction area, the above results are valid for both the particle filter and the Kalman filter approach. The increase in interaction area requires data from multiple sources to be correctly combined, which both algorithms manage.

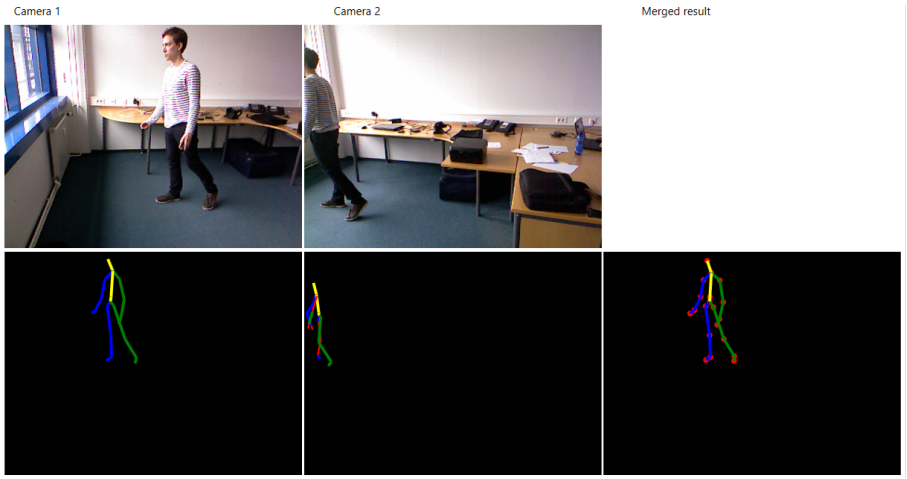

Figure 4.13. The user is moving into a part of the total interaction area not seen by camera 2. Camera 1 provides an extension of the possible area in which the user can be detected compared to a single-camera system.

4.2.4 Influence of how measurements are combined

We have already investigated the influence of filter update rules in the quantitative evaluation section. Here we look at the results from real-time evaluation of

the different update rules in terms of perceived responsiveness and smoothness for arbitrary human motion.

The Best camera approach only takes one out of potentially several available observations per joint into account. The choice is the camera with the highest overall score, i.e. the highest ratio of tracked joints. In cases of only one tracked joint, the tracked observation is taken into account irrespective of which camera has the highest score. This simple approach worked well in situations where the overlapping camera region was relatively small. Some jumping or flickering in the merged skeleton could be observed in the transition region between cameras as the overall best camera could quickly change in this region. If the camera calibration is not perfect, this may give rise to jumpy behavior of certain joints. This behavior could be reduced by applying decreasing filter covariance for the Kalman filter and thus smoothing out the rapid jumps in data. The presence of flickering still shows an underlying drawback of the simple best camera approach in certain situations.

The extended best camera approach makes the assumption that the left side of the user is better viewed by a camera viewing the user from the left, and that the right side of the user can equally be better viewed from a camera to the right. This is the case in the camera setup depicted in figure 3.2. This assumption was seen to improve accuracy in the quantitative evaluation and it also reduced problems with flickering in the region of overlap described above.

The sequential update rule takes all data of high quality into account, meaning that the observations from all cameras that see a joint as "tracked" are taken into account. During real-time evaluation, this way of combining data provided smooth and stable estimates overall as no data of high quality was rejected. The sequential update rule also turned out to be less sensitive to inaccuracies in the camera calibration and jumping estimates were considerably reduced. The quantitative evaluation showed, however, that the fact that a joint is tracked does not necessarily mean that the corresponding joint estimate is accurate. This means that it is not always advantageous to take all data deemed to be of quality into account. The sequential update rule is also computationally more expensive than the other update rules as it carries out one update cycle per connected sensor. This fact makes the sequential update rule unsuitable for the particle filter (more on this below).

The average and weighted average approaches are based on the best camera approach, but may not reject tracked observations from the camera not having the highest ratio of tracked joints. These approaches were less sensitive to flickering than the best camera approach, but not as stable as the sequential update rule during real-time testing. The behavior of the filter with the weighted average update rule can be changed by adjusting the weights. Choosing weights far apart yields a

behavior similar to the best camera approach and weights that lie close give more stable, but sometimes less accurate estimates.

For the particle filter the choice of how to combine measurements proved critical in terms of performance. The sequential approach reduced performance considerably, given that the update function of the filter had to be applied two times at each time step. The already low number of particles per joint thus had to be reduced further to maintain real-time performance. It was thus preferable to use one of the other approaches. To assess the differences between these remaining update rules for use with the particle filter turned out to be difficult as the number of particles could not be increased beyond 200 for real-time performance with enabled visualization to be maintained.

At this particle count per joint level, the particle filter worked reasonably well overall and the filtered skeleton was able to combine data from the connected Kinect cameras and output a merged skeleton that could follow the user closely, even during rapid body movements. However, given the relatively small number of particles that could be used in real-time, the merged skeleton suffered from some flickering due to the probabilistic nature of the filter. This behavior could be reduced or almost eliminated entirely by introducing a 3D grid for the state space. Limiting the grid to cubes of dimension 0.1m×0.1m×0.1m provided stable estimates without flickering, but the resulting pose estimates were not as precise as with finer grids. The resulting skeletons were somewhat distorted due to the lower precision.

4.2.5 Influence of different motion models

For the Kalman filter, two motion models were tested, one with constant velocity, and the other with zero velocity. Given that observations are available 30 times each second, the need for a motion model between time steps turned out to be limited during testing. In order to better understand wherein the small differences lie, figure 4.14 shows the two motion models applied to a sequence of observations of the user's hand making a loop motion. The zero velocity model follows the observations more closely, whereas the constant velocity model smoothens out quick changes between frames in the observation sequence.

One parameter that considerably changed the filter behavior both for the particle filter and the Kalman filter is the process covariance matrix. During real-time testing, the covariance greatly affected the response to changes in observations between frames, making it a key parameter to modify when tuning the filter to obtain a certain behavior. For applications where responsiveness is the main priority, a low degree of filtering with rapid response to observations can be chosen. In other applications, where smoothness is of greater importance, the filter can be tuned accordingly. In figure 4.15 below, the results of using different process covariances for the Kalman filter can be seen for a user performing a loop with the left hand.

4.2.6 Influence of applying body constraints

During testing it was found that adding body constraints could in some situations help eliminate impossible poses that could otherwise occur. One example is when tracking of a joint is temporarily lost, and when the joints connecting to that joint remain tracked. This sometimes led to situations where the untracked joint did not follow the tracked ones sufficiently well, and the discrepancies could in these cases be corrected by having fixed bone lengths. When a high degree of smoothing was applied, this problem was especially prominent.

As the quantitative analysis section has shown, bone constraints may actually provide less accurate estimates in some situations. It is very important that bone lengths are correctly set for the user being tracked. If this is not the case, later corrections may be applied incorrectly, reducing the quality of estimates. The method

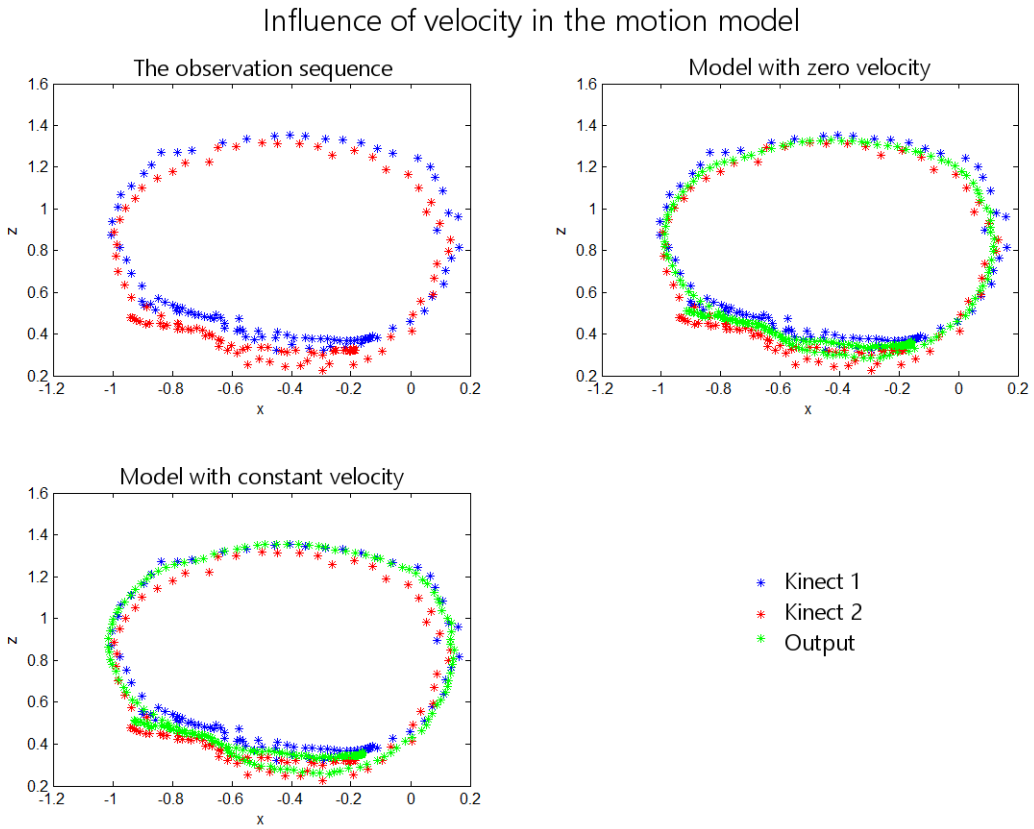

Figure 4.14. The graph shows the influence of velocity in the motion model for a Kalman filter. With velocity taken into account, the resulting sequence of estimations becomes smoother than the zero-velocity model. Without velocity, the filter takes the observations into account to a higher degree, and the estimate becomes somewhat more noise-sensitive.

to measure bone lengths of the actual user directly in skeleton frames worked well in practice provided that the user's joints were all tracked and stable when carrying out the measurement. By carrying out the measurements on several frames and then taking the mean could reduce the influence of noise, resulting in a more stable bone length initialization.

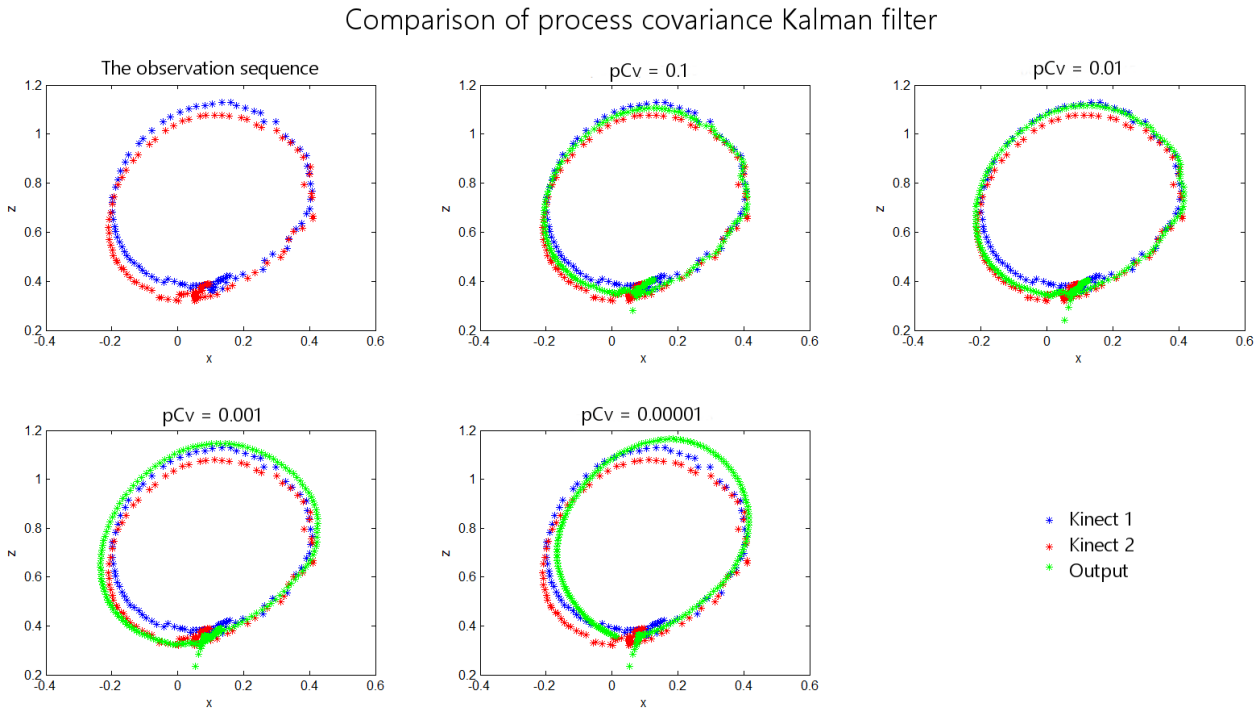

Figure 4.15. Here, the influence of the process covariance is shown for a Kalman filter. A higher value gives a filter that is more reactive to observations, but at the same time more noise-sensitive. Smaller covariance gives a smoother trail of estimates, but the filter responds less to rapid joint movements.

Chapter 5

Discussion

In this chapter we discuss the results from the previous chapter and their implications. We also go through the strengths and weaknesses of the implemented system and areas in need of further work.

5.1 Overall system performance

The quantitative analysis has showed that the multi-camera approach developed in this report performs better than a single camera in terms of total error on a sequence of frames with manually defined joint positions. For the studied test sequence, one of the cameras performed considerably better than the other, but the output from the merging algorithm with suitably chosen parameters proved better than the best of the cameras. This, in turn, shows that some information from a less accurate camera can actually help improve the overall estimate. Although only one of the update rules (extended best camera) actually outperformed the best of the individual cameras in terms of total error, the other investigated update rules performed considerably better than the average camera. It should be noted that the test sequence does not capture all possible scenarios that the merging algorithms could face, but it still provides a fair way of evaluating the performance in a controlled environment.

The qualitative analysis furthermore showed that the multi-camera setup increases the size of the interaction space in which the user's skeleton can be accurately estimated. There is also an increase when it comes to the range in which a user can change the orientation of his or her body and still remain tracked. With two cameras, a full 180-degree coverage could be achieved by placing the cameras as shown in figure 3.2. The analysis also showed that multiple cameras makes occlusions less frequent. If a joint is not observable in one of the cameras, there is always the chance that the joint in question can be observed by a differently positioned camera. The algorithms are also capable of merging data in situations where two cameras each see only a part of the user's body. An example is when one camera only sees the left part of the user and another camera only the right part.

In real use cases, the implemented system was shown to work well in providing stable estimates of the user's body pose when data is available for a majority of joints. If several joints fall outside the field of view of all cameras, final skeleton estimates are generally of low quality. This is to be expected as without data for a given joint from any of the connected cameras, the position of that joint must be estimated without any corrective observations serving as a base for the estimate.

During experiments it also became clear that the Kinect has only been trained for situations where the user is facing the camera. It does not perform as well in situations where the user is turned away from the camera. In order for enough data to be available in close to all situation and user poses, 4 or more cameras would be preferable over the two-camera-setups studied here. It would allow for a 360-degree-view of the user, regardless of the direction the user is facing. The implemented system was developed in a way that makes it possible to increase the number of connected cameras, but this could not be tested due to the hardware restrictions discussed in section 3.1.3.

One point that was surprising at first in the quantitative analysis was that the use of body constraints actually had a tendency to reduce accuracy compared to when disabled altogether. This is believed to be due to the fact that the algorithm relies on one or several frames to determine the bone lengths of the user. In the tested sequence it is likely that the frames from which the bone lengths were calculated did not correspond to real bone lengths meaning that joints were incorrectly adjusted in later frames. The bone constraints worked well during real-time testing if bone lengths were set properly.

The choice of motion model from among the ones tested in this project did not affect performance to a great extent. This does not mean, however, that there are not situations where a well-chosen motion model can make a difference. In cases where observations are not available for a certain amount of time, the algorithm has to base its estimation on the predictive models of the filters until new observation data is available. This is the case when tracking of a node is temporarily lost in all connected cameras. For arbitrary human motions it would, however, be very difficult to construct a model suitable for all situations that could arise. One of the goals of a multi-camera setup is also to decrease the number of situations where a joint cannot be observed to prevent the problem from arising in the first place.

One drawback with using multiple depth sensors is the need to calibrate the system which adds complexity to it. However, with calibration algorithms that do not need the use of a chess board, a calibration that is good enough for practical use can be obtained by simply walking in the area viewed by all connected cameras. The proposed merging algorithms were not very sensitive to the accuracy of the camera calibration. Ideally, however, the calibration should be made as precise as possible,

but some discrepancies could be efficiently dealt with in filtering steps. Without smoothing altogether, some flickering as a result of a somewhat imprecise calibration and inaccuracies in data could be noticed when the user was in the region of overlap between cameras, especially with the best camera approach for updating the filter. With a correctly parameterized filter, this problem could, however, be solved.

It has been proposed by Microsoft [18] that multiple Kinects may interfere in the zone of overlap between two or more sensors. In the camera setups tested in this project, however, problems related to interference between Kinects were not experienced.

5.2 Comparison between particle and Kalman filters

The filtering techniques studied in this report have both been shown to work when it comes to the task of combining and smoothening incoming data from multiple cameras. Both the particle filter and Kalman filter can be run in real-time, but having studied the performance of the two filter types above, the Kalman filter is the fastest of the two by a considerable margin. The qualitative analysis provided evidence that in real-world usage, the Kalman filter provides more stable and usable pose estimates than its particle counterpart when real-time performance is required. This was confirmed by the quantitative analysis where the output from the filters were compared to a manually defined user skeleton in each frame. Here, the Kalman filter performed better than the particle filter while at the same time requiring considerably less computations to arrive at the results.

For the particle filter, the number of particles turned out to be the main factor determining the performance of the filter, as expected. The filter could be made to work in real-time on a normal laptop, but in order to achieve responsive and accurate estimates, the particle count needs to be increased beyond the real-time limit. This limits the utility of the method, at least on the hardware specification used in this project. The quantitative analysis still showed the potential of the particle filter, as it was possible to achieve better accuracy than the most accurate individual camera in the system if the number of particles was high enough.

The attempts to increase performance of the particle filter by parallelizing the execution of the filter turned out to be inefficient, as more time was spent on overhead and synchronization of threads than the actual gains in performance from carrying out computations for partial particle sets in parallel. Aside from the performance issues, the particle filter is very general in comparison with the Kalman filter and is not limited to certain classes of probability distributions, which may prove essential in some applications. In this application, the Kalman filter turned out to work well with the Kinect data in spite of its Gaussian limitations.

5.3 Choosing a filter

Regardless of the chosen filter type, an important question is how to tune the filter to make it behave in a way suitable for the application at hand. For the tracking application studied here, this required considerable amounts of time for testing and tweaking in the implementation and evaluation phases. One helpful feature during this work was to be able to modify filter parameters while running the tracking application so that the impact could be observed in real-time. The possibility to record sequences and apply different filter parameters and compare output on the same data was also helpful in finding appropriate filter parameters.

To select a suitable filter for a given application, several aspects need to be considered. Firstly, we need to be aware that certain filters rely on assumptions that may not always be fulfilled. Kalman filters in their basic form rely on unimodal probability distributions, and if this assumption is not fulfilled, more general filter types, such as particle filters are required. As was mentioned in the theory chapter, it has been established that it is possible to use approximately unimodal likelihood models with Kinect data [13], meaning that it is possible to use Kalman filters in this case. Lack of information and ambiguities in data normally gives rise to multimodal probability distributions and if this is the case, a particle filter would be more suitable.

Secondly, the performance requirements may also eliminate some choices of filter. In such cases, a computationally less expensive filter such as the Kalman filter may be suitable. In this project, the performance of the particle filter was not known beforehand, and it was not certain in the early phases whether the criteria allowing the use of a Kalman filter were completely fulfilled for the Kinect data. These are the reasons why the particle filter appeared to be the most promising candidate and the technique that was implemented first. The Kalman filter is a very different approach to particle filtering, and it was therefore thought to be an interesting object of comparison to implement and compare to the particle filter. The Kalman approach was selected partly on the basis that it is known to require less computations, which was considered important after the particle filter had been implemented and tested at first.

5.4 Areas of improvement

User rotation in a 360-degree interaction space

One issue with the current 2-camera-setups is that under some circumstances, it is difficult to consistently distinguish between left and right. This is especially the case when full 360-degree rotations are allowed. As shown in section 4.2.1, the Kinect cannot determine this correctly on its own as it is assumed that the user is always facing the camera. One way to get around this problem would be to determine

the orientation of the user inside the interaction space more reliably than currently done. One approach could be to increase the number of cameras and combine this measure with a technique to determine the orientation of the user.

The latter could potentially be achieved by using a face detection technique to determine in which direction a user is looking. With this information, a more reliable way of assigning the same label to a given joint across cameras could be achieved. In virtual reality applications where a user is wearing a head mounted display (HMD), the display could serve as a marker indicating the orientation of the user's head in a similar fashion. In cases where a head tracker is used in combination with the HMD, information from the head tracker could also be used in determining the orientation of the user after an initial calibration of the head tracker in the camera system.

In this case, four cameras would not be needed for the determination of the user's orientation. But they would still be useful in increasing the amount of available data about the user to produce better estimates as the user turns around freely. With four cameras positioned with 90-degree angles between them, the observed information should also be sufficient to avoid the situation of ambiguities in the data, yielding unimodal probability distributions for the combined observations, meaning that Kalman filters can be applied [13].

Support for multiple users

One thing relevant for certain applications is the possibility for multiple users to be tracked simultaneously in the same interaction area. This has not been studied in-depth in this thesis as it was not a priority for the application at hand. In order to create such multi-user support we would need to associate each user with an id and make sure that the id is the same in all connected cameras. The association could be achieved in different ways. A simple solution would be for each user to initialize themselves in all cameras by for instance raising their arms in turn. Alternatively, users could be assigned an id in the order they enter the interaction area, provided that they do not all enter at the same time. However, these approaches would not work flawlessly if tracking was lost in one or more of the cameras for some reason. A marker-based solution would get around this problem, but would make the system more complicated and less user friendly. A system that identifies users based on the color of their clothing using the RGB camera of the Kinect could also be imagined, but this would not work in situations where multiple users wear similar garments.

Another problem is that multiple users would occlude one another inside the interaction area. With four cameras positioned with 90-degree-angles between them, the problem would be less serious than with fewer cameras, but there would still be situations where the estimations of the individual users would be adversely affected by occlusions. Worth noting is also that the current Kinect SDK only has support

for skeleton tracking of up to two simultaneous users, imposing a limit as well.

Developments in tracking hardware

In this project, the focus has been on the software side of the tracking problem, but good tracking also relies partly on good hardware. The Kinect has been tested in many different scenarios and in different camera setups throughout the project. Although the device performs reasonably well when it comes to tracking under favorable circumstances, there is definitely room for improvement. The Kinect has been available since 2010 and Microsoft has already launched its predecessor, at the time of writing only shipping with its new generation gaming console, the Xbox One. This new version of the Kinect cannot yet be used for pc and the algorithms have therefore not been tested with the new hardware.

The specifications of the new Kinect have been reported by several technology resources and every aspect of the new device have been improved over the current Kinect [22]. A 1080p camera with a frame rate of 60 frames per second, skeleton tracking with 25 joints of up to 6 users simultaneously are mentioned along with improved field of view. According to tests, the skeleton tracking is also reported to be considerably improved [10]. The improved hardware is thus likely to improve the performance of the algorithms presented in this thesis, as higher quality data will improve the input to the algorithm, resulting in output of higher quality as well.

5.5 Utility in real-world applications

Systems for tracking a human body in motion are used in many applications today. Examples can be found in animations for films and games, interactive user interfaces and a range of virtual reality applications. Improved accuracy and freedom to move without constraints, not thinking about being in the correct position with respect to a depth sensor is an improvement over many current systems. Also, not having to wear markers or physical sensors registering the movement of the body allows for less expensive systems that are easier and more convenient to use. Although the implemented system is not perfect, it offers considerable benefits over one-camera-systems. In addition to this, the proposed algorithms work with depth cameras that are both cheap and available off-the-shelf. Given that the necessary computations can be carried out on a normal laptop computer, the system is portable as well.

Chapter 6

Conclusions and future work

In this report a working algorithm that enables data from multiple Kinect cameras to be combined to improve user pose estimation in real-time has been presented. Compared to a single-camera-system, the implemented multi-camera system is able to improve accuracy to levels beyond that of the most accurate of the connected cameras. It also manages to reduce situations of partial occlusion (including self-occlusion) of the user, to increase the possible interaction area and to make it possible for the user to turn around freely in a range of 0-180 degrees using two cameras. The results of tests of the system with two cameras indicate that it would be possible to track full 360-degree turns in a reliable and stable way with four connected cameras.

Two main filter types, a particle filter and a Kalman filter, were implemented in the project and then discussed and compared throughout this report. An important goal of the project was to create a system that could be fed joint position from multiple cameras and process the information to output a final skeleton estimate at a frame rate of 30 frames per second. This could be achieved with both the particle filter and the Kalman filter. However, the Kalman filter estimates were considerably more accurate in real-time applications. The real-time criteria meant that the number of particles in the particle filter had to be reduced to levels where the precision of the estimates would suffer.

Offline analysis showed that the particle filter could provide more accurate estimates than any of the cameras connected to the system treated individually. To achieve that level of accuracy, however, the number of particles had to be greater than the real-time limit. The approach as such was thus shown to work if a high enough number of particles are used. It can furthermore be concluded that simple motion models with constant or even zero velocity are well-adapted to the complex movements of a human body when data is available at a rate of 30 frames per second, at least in the situations studied in this project.

Although the methods outlined in this report work well in real usage scenarios, there is always room for further improvements. A reliable determination of the orientation of the user in 360 degrees instead of 180 would still need to be implemented to open up for completely unrestricted motion. Support for multiple users in the interaction area at the same time is currently unsupported as this would need users to be correctly and coherently identified with the same id across all connected cameras (the system as such can otherwise handle multiple users). The camera calibration used could also be improved for even better end results, but this is not critical as the filters handled minor discrepancies between cameras relatively well. Implementing a more advanced joint constraint model also taking into account bone orientations is another area to explore further. This could help eliminating impossible poses to a higher degree than in the currently implemented system which only takes into account bone lengths in a joint hierarchy.

Another thing would be to investigate whether more refined versions of the Kalman filter such as the extended Kalman filter or unscented Kalman filter would make a difference in the case of tracking arbitrary human motion at 30 frames per second. For the particle filter it would furthermore be interesting to increase real-time performance by implementing the program in C++ with parallelization on the GPU through OpenMP [25] or CUDA [21]. Last but not least, it would be interesting to try the program with the second generation Kinect, to evaluate how much of an improvement could be achieved with more refined hardware.

Bibliography

[1] Stylianos Asteriadis, Anargyros Chatzitofis, Dimitrios Zarpalas, Dimitrios S. Alexiadis and Petros Daras. 2013. Estimating human motion from multiple Kinect Sensors. *Proceedings of the 6th International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications*, p. 3. ACM.

[2] Mehran Azimi. Skeletal Joint Smoothing White Paper. *Microsoft Developer Network*. Retrieved from `http://msdn.microsoft.com/en-us/library/jj131429.aspx`, last seen on September 26, 2013.

[3] Kai Berger, Kai Ruhl, Yannic Schroeder, Christian Bruemmer, Alexander Scholz and Marcus Magnor. 2011. Markerless Motion Capture using multiple Color-Depth Sensors. *VMV*, pp. 317-324.

[4] Paul J. Besl and Neil D. Mckay. 1992. A method for registration of 3-D shapes. *Robotics-DL tentative*, pp. 586-606. International Society for Optics and Photonics.

[5] Maged N. Kamel Boulos, et al. 2011. Web GIS in practice X: a Microsoft Kinect natural user interface for Google Earth navigation. *International journal of health geographics* 10.1 (2011): 45.

[6] Magnus Burenius, Josephine Sullivan and Stefan Carlsson. 2013. 3D Pictorial Structures for Multiple View Articulated Pose Estimation. *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference* pp. 3618-3625.

[7] A. Criminisi, J. Shotton and E. Konukoglu. 2011. Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. *Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114, 5(6)*, 12.

[8] Hugh Durrant-Whyte. 2001. Multi Sensor Data Fusion. Australian Centre for Field Robotics, The University of Sydney. Retrieved from `http://www.acfr.edu.au/pdfs/training/multiSensorDataFusion/dataFusionNotes.pdf`, last seen on November 14, 2013

[9] Luigi Gallo, Alessio Pierluigi Placitelli and Mario Ciampi. 2011. Controller-free exploration of medical image data: Experiencing the Kinect. *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on. IEEE.*

[10] Ben Gilbert. 2013. Xbox One review: a fast and powerful work in progress. *Engadget*. Retrieved from `http://www.engadget.com/2013/11/20/microsoft-xbox-one-review/`, last seen on January 2, 2014

[11] Abishek Kar. 2010. Skeletal Tracking using Microsoft Kinect. *Methodology* 1, 1-11.

[12] Vahid Kazemi, Magnus Burenius, Hossein Azizpour and Josephine Sullivan. 2013. Multi-view Body Part Recognition with Random Forests. *Proceedings of BMVC 2013.*

[13] Anders Boesen Lindbo Larsen, Søren Hauberg, and Kim Steenstrup Pedersen. 2011. Unscented Kalman Filtering for Articulated Human Tracking. *Image Analysis*, pp. 228-237. Springer Berlin Heidelberg.

[14] Microsoft. Kinect for windows sensor components and specifications. *Microsoft Developer Network*. Retrieved from `http://msdn.microsoft.com/en-us/library/jj131033.aspx`, last seen on September 3, 2013

[15] Microsoft. Kinect for Windows SDK. *Microsoft Kinect for Windows Dev Center*. Retrieved from `http://www.microsoft.com/en-us/kinectforwindowsdev/Downloads.aspx`, last seen on September 26, 2013

[16] Microsoft. JointType enumeration. *Microsoft Developer Network*. Retrieved from `http://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx`, last seen on September 26, 2013

[17] Microsoft. Kinect Fusion. *Microsoft Developer Network*. Retrieved from `http://msdn.microsoft.com/en-us/library/dn188670.aspx`, last seen on September 26, 2013

[18] Microsoft. Skeleton Tracking With Multiple Kinect Sensors. *Microsoft Developer Network*. Retrieved from `http://msdn.microsoft.com/en-us/library/dn188677.aspx`, last seen on January 5, 2014.

[19] Kevin P. Murphy. 2012. *Machine Learning: a probabilistic perspective*. MIT Press, Cambridge Massachusetts.

[20] Takayuki Nakamura. 2011. Real-time 3-D Object Tracking Using Kinect Sensor. *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference*, pp. 784-788.

[21] Nvidia CUDA. *Nvidia*. Retrieved from `http://www.nvidia.com/object/cuda_home_new.html`, last seen on July 29, 2014

[22] Terrence O'Brien. 2013. Microsoft's new Kinect is official: larger field of view, HD camera, wake with voice. *Engadget*. Retrieved from `http://www.engadget.com/2013/05/21/microsofts-new-kinect-is-official/`, last seen on September 26, 2013

[23] Oculus VR. Development Kit 2. *Oculus VR*. Retrieved from `http://www.oculusvr.com/dk2/`, last seen on August 27, 2014

[24] Iason Oikonomidis, Nikolaos Kyriazis and Antonis A. Argyros. 2011. Efficient Model-based 3D Tracking of Hand Articulations using Kinect. *BMVC*, pp. 1-11.

[25] OpenMP. *OpenMP Architecture Review Board*. Retrieved from `http://openmp.org/wp/`, last seen on July 29, 2014

[26] Stuart Russel and Peter Norvig. 2010. *Artificial Intelligence: A Modern Approach*, Third Edition. Pearson Education, Upper Saddle River New Jersey.

[27] Sanders, Adrien-Luc. Creating Unreality from Reality *About Technology*. Retrieved from `http://animation.about.com/od/moviemagic/a/motioncapmagic.htm`, last seen on August 27, 2014

[28] Hedvig Sidenbladh, Michael J. Black and David J. Fleet. 2000. Stochastic Tracking of 3D Human Figures Using 2D Image Motion. *Computer Vision-ECCV 2000*, pp. 702-718. Springer Berlin Heidelberg.

[29] Wandi Susanto, Marcus Rohrbach and Bernt Schiele. 2012. 3D Object Detection with Multiple Kinects. *Computer Vision-ECCV 2012. Workshops and Demonstrations*, pp. 93-102. Springer Berlin Heidelberg.

[30] Sebastian Thrun, Wolfram Burgard and Dieter Fox. 2006. *Probabilistic Robotics*. MIT Press, Cambridge Massachusetts

[31] Vicon Motion Systems Ltd. Markers and Suits. *Vicon Motion Systems Ltd.* Retrieved from `http://www.vicon.com/System/Markers`, last seen on August 27, 2014

[32] Brian M. Williamson, Dr. Joseph J LaViola Jr., Tim Roberts and Pat Garrity. 2012. Multi-Kinect Tracking for Dismounted Soldier Training. *Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, Vol. 2012, No. 1. National Training Systems Association

[33] YEI Technology. 3-Space Mocap Starter Bundle *YEI Corporation*. Retrieved from `http://www.yeitechnology.com/productdisplay/3-space-mocap-starter-bundle`, last seen on August 27, 2014

[34] Liang Zhang, Guido Brunnett and Stephan Rusdorf. 2011. Real-time Human Motion Capture with Simple Marker Sets and Monocular Video. *Journal of Virtual Reality and Broadcasting*, 8.1.

[35] Lichao Zhang, Hsieh Jui-Chien, and Wang Jiangping. 2012. A Kinect-based golf swing classification system using HMM and Neuro-Fuzzy. *Computer Science and Information Processing (CSIP), 2012 International Conference*, pp. 1163-1166.

[36] Licong Zhang, Jürgen Sturm, Daniel Cremers and Dongheui Lee. 2012. Real-time Human Motion Tracking using Multiple Depth Cameras. *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference* IEEE

Appendix A

The Kinect hardware

Microsoft launched the Kinect in November 2010. The device was primarily designed for the Xbox 360 gaming console allowing the user's body to become the game controller. Thanks to this mass-market approach with a reasonable price point, the Kinect is within the reach for almost everyone. Today, numerous research projects use the Kinect for tracking, human pose estimations, object recognition or interfaces of different kinds. A few examples have been given throughout this report. What is important to note is that the tracking of the user is done without sensors or markers mounted on the body of the user, contrary to some other tracking systems.

A.1 How the Kinect works

A normal camera takes an image without having any idea of what the images is portraying. In order for the Kinect to work, the device needs to have an understanding of what its sensors are viewing so as to be able to detect users and estimate their joint positions. The Kinect thus analyses images to detect humans and identify the different body parts of the user. For this to work, Microsoft has used a machine learning approach where hundreds of thousands of images of human poses have been used to train the system. The training of the system is a big and time-consuming task but once the training is done, the knowledge can be stored in a random forest, making it possible for the system to find the joint positions of human users in real-time. For developers, it also means that they are not required to estimate the position data directly from raw data themselves, which makes working with the Kinect easier.

The Kinect is not a simple camera, but includes several different sensors in one package making the tracking of users possible. It has an IR emitter and an IR depth sensor that together produce a point cloud which can be used to estimate the position of a user standing in front of the Kinect as well as the position of his/her body joints. When it comes to the depth sensor output from the Kinect, data is available not only as raw data in the form of point clouds but also as estimated

joint positions directly, as mentioned above.

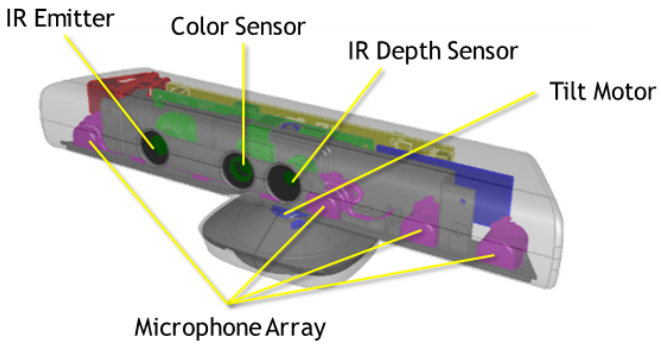

Figure A.1. The Kinect contains 3D IR depth sensors, an RGB camera and an array of microphones. Image source: [14]

The Kinect also offers an RGB camera with a resolution of 640x480 pixels and an array of four microphones. The depth map and RGB output both have a frame rate of 30 frames per seconds [14]. This makes the system usable in real-time applications, provided that the computer meets the system requirements. The depth sensor actually has a maximum resolution of 1280x1024 pixels, but in order to achieve the 30 frames per second, the resolution needs to be limited to 640x480 pixels. The horizontal field of view of the Kinect is 57 degrees and the vertical field of view 43 degrees. When it comes to the the depth range, the user needs to be situated between 0.8 and 8.0 m away from the Kinect in order to be detected properly.