Masterthesis Medientechnolgie

A real-time hand and object tracking system with low cost consumer-grade hardware

vorgelegt von

Oliver Kalbfleisch

Erstgutachter: Prof. Dr. Arnulph Fuhrmann(TH Köln)

Zweitgutachter: Prof. Dr. Stefan Michael Grünvogel(TH Köln)

Juni 2018

Fakultät für
Informations-, Medien-
und Elektrotechnik

Technology
Arts Sciences
TH Köln

Masterarbeit

Titel: Ein echtzeit Hand- und Objektrackingsystem basierend auf kostengünstiger Hardwarekomponenten

Gutachter:

- Prof. Dr. Arnulph Fuhrmann (TH Köln)
- Prof. Dr. Stefan Michael Grünvogel (TH Köln)

Zusammenfassung:

Diese Abschlussarbeit beschäftigt sich mit der Frage der Umsetzbarkeit eines Hand- und Objekt-Tracking Systems, basierend auf günstigen consumer-grade Hardwarekomponenten. Um einen niedrigen Systempreis zu erzielen, wurden zwei Raspberry Pi Minicomputer mitsamt kompatiblen Kameras für die Bildaufnahme und Verarbeitung verwendet. Die aufgenommenen Daten werden an einen „Master“ PC kommuniziert, welcher sich um die Tiefen- und Positionsberechnung sowie das finale Rendering kümmert. Verschiedene Formen von Fingermarkierungen wurden im Zuge der Arbeit vorgestellt und in Bezug auf die Präzision des Trackings und der Einschränkung der haptischen Fähigkeiten der Finger mit der Markierung bewertet. Die finale Auswertung des entstandenen Prototyps zeigt, dass mit der ausgewählten Hardware grundsätzlich ein, mit gewissen Einschränkungen, funktionierendes Trackingsystem realisiert werden konnte.

Stichwörter: Echtzeit, Inverse Kinematik, Hand-tracking, color-tracking,Objekt-tracking, günstig

Datum: 29. Juni 2018

Masters Thesis

Title: A real-time hand and object tracking system with low cost consumer grade

Reviewers:

- Prof. Dr. Arnulph Fuhrmann (TH Cologne)
- Prof. Dr. Stefan Michael Grünvogel (TH Cologne)

Abstract:

This paper will evaluate if it is possible to build a hand and object tracking system on low cost consumer grade hardware components. For image acquisition and processing, a set of Raspberry Pi's with the matching camera is used to keep system cost low. These units are linked to a master unit which takes care of stereoscopic depth calculations, the position value processing and the final rendering of a hand and object model in digital space. Several forms of colored marker types are assessed for the prototype application. The final results show that it is possible to achieve a tracking system solution with the used hardware components that has a suitable tracking update frequency and precision.

Keywords: Hand-tracking, Object-tracking, inverse kineamtics, based on low-cost, realtime

Date: June 29th, 2018

Contents

1 Introduction

For long years, the standard interface between a human and a computer has been a mouse and a keyboard. But with the advance of technology, new interfacing methods were developed in the last few years. Touch technology for interfacing with mobile devices and desktop computers has become a reliable technology and has been integrated into our everyday lives. Advances in capabilities of CPU as well as GPU hardware has built a foundation for the use of advanced *Augmented* (AR) and *Virtual Reality* (VR) technology. 3D and stereoscopic rendering can now be accomplished even on mobile devices (with some limitations). For an intensely immersive experience, VR goggles are used to explore digitally created worlds.
A touch device or just a mouse and keyboard setup is rather hindering for the immersivity of the user experience and the logical and more natural way of interfacing with our digital technology would be by utilizing the built-in human control elements that we carry around with us every day.
The human hands offer a form of control element that provides a large number of "*Degrees-of-Freedom*" (DOF) and furthermore adds the posibility of combining these with a tactile component. The first more humble attemps of solving this problem already began in the early 1980's, where a lack of computational power and suitable display hardware made more complex systems impossible. These systems were mostly aimed at registering simple hand gerstures like pointing as an intefrace method and were not able to reconstruct full hand positions in realtime [**?**]. State-of-the-art elaborate system for hand tracking use infrared or stereoscopic cameras to achieve a more performant tracking result, utilizing the calculation capabilities of standard consumer computers. These systems are mostly specialized for the task they are doing and therefore are rather expensive to attain.
This thesis assesses, if it is possible to create a real-time hand and object tracking system based on easily accessible consumer grade hardware and open source programs. The system should provide an experience for the user that is as natural as possible in terms of tracking precision. The components should not hinder the motion capability of the hand. Furthermore the system should aim at being comfortable in terms of design as a cumbersome system would not be used in everyday life. Before evaluating a prototype system, an introduction to the anatomical properties of the human hand will be given. The understanding of these properties plays a major role in the setup for a hand simulation. As a follow-up, an overview of existing technologies in terms of tracking and display of the human hand will be given before the system conception, construction and analysis parts.

2 Description of the hand in digital space

Tracking of the human hand has always been a challenging problem. In comparison to other larger body parts like the arm or the head, the human hand itself consists of a large variety of smaller components, namely bones and muscles. These components have to be taken into account when trying to replicate the natural motion of the hand in digital space as they all influence the result of the hand motion with their properties.

2.1 Physiological structure of the human hand

Lee and Kuni [**?**] described the human hand as "*an articulated structure with about 30 degrees of freedom [which] changes shape in various ways by its joint movements*"

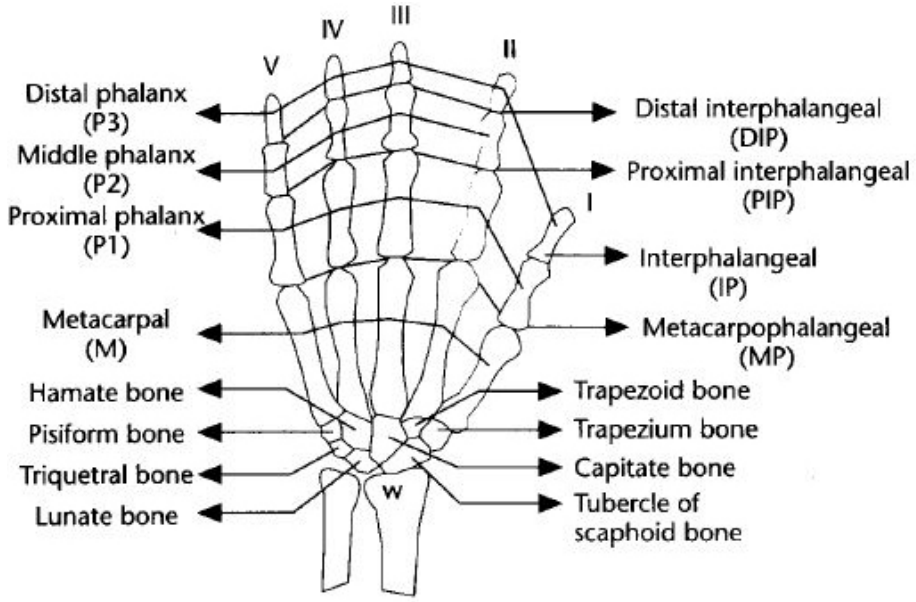

Figure 2.1: Bone structure of the left human hand [**?**]

All of the hand parts are connected to at least one neighboring part via a *joint*. The *joints* affect the position of the connected part. To describe the movement of the hand parts, we can use the rotation angles of the joints to correlate to a specific position. To do so, we define a local coordinate system for each of the exiting hand joints. Through these coordinate systems, we achieve a sequence of rotations in the local coordinate systems of the joints. Such a sequence can then be used to describe a specific movement and/or position of a part. Not all of the joints in the human hand have equal *Degrees of freedom* (DOF's). Their functionality can be classified in the amount of DOF's [**?**]:

- 1 DOF
  - A joint movement that can perform a **flexion** or **twist** in one direction
- 2 DOF
  - A joint movement that can perform **flexion** in more than one direction (**directive**)

- 3 DOF
  - A joint movement that permits simultaneous **directive** and **twist** movements.(**spherical**)

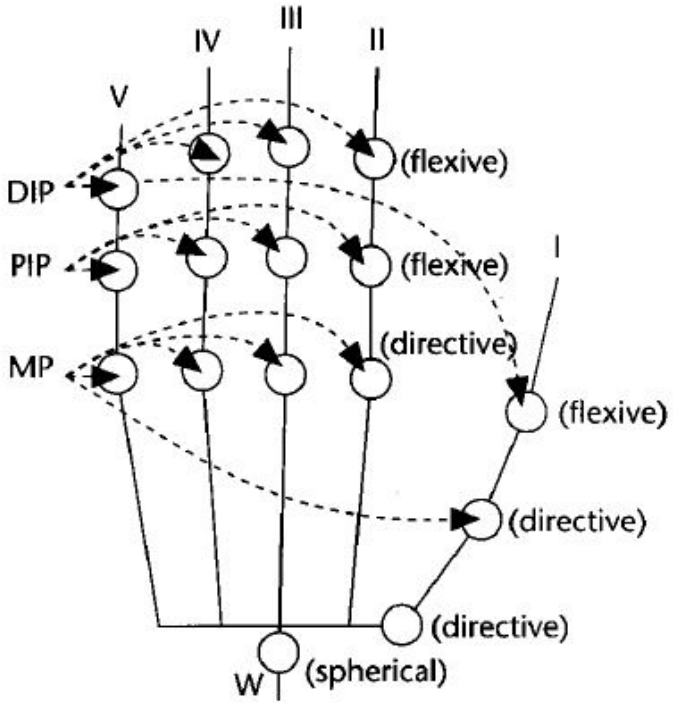

Figure 2.2: Types of degrees of freedom for the joints of the human hand

When looking at the DOF's displayed in Figure 2.2, each finger (II-V) sums up to 4 DOF's and the thumb to 5 DOF's. Also considering 6 DOF's for the rotation and position of the whole hand, the sum of all components adds up to 27 DOF's.

2.1.1 Constraints in hand motion

A full usage of all the declared DOF's would lead to a large amount of possible combinations. Since the hand is not only made up of bones but also muscles and the skin, we can impose some constraints [**?** **?**] to the movement of the joints. Ling, Wu and Huang [**?**] proposed following classification for the constraints:

- **Type I constraints**
  - A constraint that limits the range of finger motions based on hand anatomy
- **Type II constraints**
  - A constraint that limits the position of the joints during finger movement
- **Type III constraints**
  - A constraint that limits position based on natural hand motions

The **Type I** and **Type II** constraints rely on the physiological and mechanical properties of the human hand. **Type III** constraints are results of common and natural movements. Therefore these constraints will be differing form person to person. However, these movements are to a certain degree similar for everyone. Accordingly, a broad grouping can be applied. For example, the curling of the fingers at the same time when forming a fist is way more natural than curling each finger by itself. Here the motion of the hand is quite similar between different persons, but the constraints cannot be described in a mathematical form. The examples for the other two constraint types are less complicated.

For the **Type I** constraint, there is no doubt that the position of the fingertip is limited by the length of the other finger segments and thereby can only reach as far as the combined length. If you want to touch your hand palm with your fingertip, all joints in the finger have to be bend to achieve this position, which is a constraint example of the **Type II**. For the mathematical description of **Type I** and **Type II** constraints, the following inequalities can be used:

$$\begin{aligned} 0^\circ \leq \Theta_{MP_{flex}} \leq 90^\circ \\ 0^\circ \leq \Theta_{PIP_{flex}} \leq 110^\circ \\ 0^\circ \leq \Theta_{DIP_{flex}} \leq 90^\circ \\ -15^\circ \leq \Theta_{MP_{abduct/adduct}} \leq 15^\circ \end{aligned} \tag{2.1}$$

For the middle finger, a further specific constraint is applicable, as the middle fingers *Metacarpal* (MP) joint normally does not abduct and adduct much. Therefore an approximation is applicable, resulting in the removal of 1 DOF from the model:

$$\Theta_{MP_{abduct/adduct}} = 0^\circ \tag{2.2}$$

The same behavior can be seen in the combination of hand parts labeled **W** in Figure 2.2 (the connection point between hand and lower arm). This approximation also eliminates 1 DOF on the connected thumb:

$$\Theta_{W_{abduct/adduct}} = 0^\circ \tag{2.3}$$

Since the *Distal Interphalangeal* (DIP), *Proximal Interphalangeal*(PIP) and MP joints of our index, middle, ring, and little fingers only have 1 DOF for flexion, we can further assume that their motion is limited to movement in one plane.

The **Type II** constraints can be split into *inter-finger* and *intra-finger* constraints. Regarding *intra-finger* constraints between the joints of the same finger, human hand anatomy implies certain dependencies. To bend the DIP joints on the specific finger,the corresponding PIP joints of that finger must also be bent. The approximation for this relation [**?**] can be described as :

$$\Theta_{DIP} = \frac{2}{3}\Theta_{PIP} \tag{2.4}$$

Inter-finger constraints can be imposed between joints of adjacent fingers. These constraints describe that the bending of an MP joint in the index finger forces the MP joint

in the middle finger to bend as well.
When combining the constraints described in the above equations, the initial number 21 DOF's of the human hand can be reduced to 15. Inequalities for these cases, obtained through empiric studies, can be found in [**?**].

2.2 Kinematics

The preceding sections gave an overview of how we can describe a model of the human hand and introduced some limiting constraints. With the model and the constraints, we can now start to build a kinematic system for the animation of the model.
Kinematic systems contain so called *kinematic chains*, which consist of a *starting point* or *root*, kinematic elements like *joints*, *links* and an *endpoint*, also called *end-effector*. Applied to the human hand, the whole hand model represents the kinematic system. This system contains several *kinematic chains*, namely the fingers of the hand with the fingertips being the *end-effectors* of each of these chains.
Each of these components has it's set of DOF's which can be described mathematically. The established convention for describing this kind of linked systems is the *Denavit-Hartenberg* convention (D-H) [**?**]. It was originally introduced in order to standardize the coordinate frames for spatial linkages in the field of robotics, but has since then been applied in other fields as well [**?**]. As hand movements starts, the states of the kinematic chains begin to change. Joint angles and end-effector positions are modified until the end position is reached. To represent the new position and angle values of our physical hand with a kinematic system, two major paths for achieving a solution can be taken.

2.2.1 Forward Kinematics

The *Forward Kinematics* (FK) approach uses the knowledge of the resulting angles and positions after the application of known transformations to the kinematic chain.The resulting new positions of the *joints* and *links* between the *root* and the *end effector* is used to solve the problem of finding the *end-effector's* position.
We can denote the existing *end-effectors* relative position to an origin as $s_1, ..., s_k$. The s_i position is the result of the combination of all the joint angles in the corresponding kinematic chain. Respectively, we define the target position of the *end-effectors* as $t_1, ..., t_i$, with t_i being the target position for the *end-effector* s_i. The required positional change for the *end-effector* can now be described as $e_i = t_i - s_i$. In systems with more than one end-effector, like our hand system, the components can be described by vectors.

$$\begin{aligned} \vec{\mathbf{s}} &= (\mathbf{s}_1, ..., \mathbf{s}_n)^T \\ \vec{\mathbf{t}} &= (\mathbf{t}_1, ..., \mathbf{t}_n)^T \\ \vec{\mathbf{e}} &= \vec{\mathbf{t}} - \vec{\mathbf{s}} \end{aligned} \tag{2.5}$$

As the vector components of $\vec{\mathbf{s}}$ are results of the chain joint angles $\theta_1, ..., \theta_n$ and therefore are effected by them, we define:

$$\begin{aligned} \vec{\mathbf{s}_i} &= f_i(\boldsymbol{\theta}) \\ \vec{\mathbf{s}} &= f(\boldsymbol{\theta}) \end{aligned} \tag{2.6}$$

With $\boldsymbol{\theta}$ being the column vector $\boldsymbol{\theta} = (\theta_1, ..., \theta_n)^T$. The second vector equation displayed in (2.6) is also called the *Forward Kinematics* (FK) solution.

The advantage of an FK solution is that there is always an unique solution to the problem. In consequence, this approach is commonly used in the field of robotics, where the information on the chain elements is easily available. The tracking of the human hand and all of its chain components is rather complicated when trying to collect all current joint and bone position states. Therefore a solution which takes a known position of the *end-effector* and calculates the parameters for the rest of the chain is more desirable.

2.2.2 Inverse kinematics

> "Inverse Kinematics (IK) is a method for computing the posture via estimating each individual degree of freedom in order to satisfy a given task" [**?**]

The IK concept already describes it's principle in it's name. It takes the reversed approach in comparison to the FK principle in section 2.2.1. Instead of knowing the states of the chain elements and calculating the resulting position of the *end-effector*, we take the position of the *end-effector* and try to retrieve the possible states of the other chain elements.

$$\boldsymbol{\theta} = f^{-1}(\vec{\mathbf{s}_d}) \tag{2.7}$$

The result of this equation is the vector $\boldsymbol{\theta}$ for which the values of $\vec{\mathbf{s}}$ coincide to the desired configuration $\vec{\mathbf{s}_d}$. In the case of an optimal result, this configuration would have the same position values as the target positions.
The main problem with this method occurs in the calculation of the f^{-1} function, due to it being a highly non linear operator which is not easily invertible. In contrary to having a unique solution with the FK approach, the IK approach can end at the point of not finding a suitable solution. Figure 2.3 displays three possible outcomes for the IK approach.

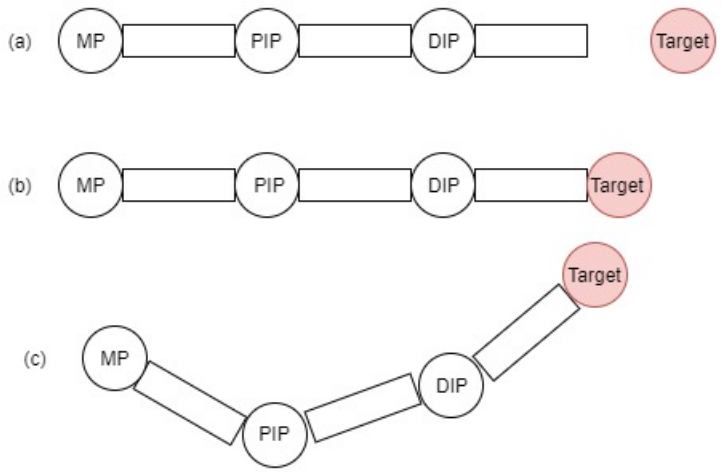

Figure 2.3: Possible solution for an IK problem of a human finger: (a) The given target position of the end-effector can not be reached. (b) The given target can only be reached by one solution. (c) The target position can be reached with multiple different solutions.

2.2.3 Jacobian-based methods

Jacobian inverse

One common approach to solve the IK problem is the utilization of a *Jacobian* matrix and an iterative calculation process. This matrix contains the partial derivatives of the

chain systems relative to the end-effector **s**. When using the *Jacobian*, a linear approximation of the IK problem will be applied for solving. The approximation's components model the end-effector's relative movement to changes in transitions of the systems link translations and joint angles. Therefore, the resulting function is dependent on the joint angles $\boldsymbol{\theta}$ values and can be defined as:

$$J(\boldsymbol{\theta})_{ij} = \left(\frac{\partial \mathbf{s}_i}{\partial \theta_j} \right)_{ij} \tag{2.8}$$

i=1,...,k and *j*= 1,...,n. Further readings on methods for the calculation of *Jacobian* matrices can be found in [**?**]. Based on Definition (2.8), an entry for the j-th rotational joint would be calculated as follows:

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j) \tag{2.9}$$

Here $\mathbf{p}_j$ is the position of the joint, and $\mathbf{v}_j$ is the unit vector pointing along the current axis of rotation for the joint. Taking the derivative of Definition (2.6) with respect to time gives the basic equation for forward kinematics that describes the velocities of the end-effectors:

$$\dot{\vec{\boldsymbol{s}}} = \boldsymbol{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{2.10}$$

Now having all the values for the angles, the *end-effector* position and the target positions, we can compute the resulting *Jacobian* matrix. Thereafter we seek an update value $\Delta\boldsymbol{\theta}$ for incrementing the current joint values:

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{curr} + \Delta\boldsymbol{\theta} \tag{2.11}$$

The idea here is that the chosen value for $\Delta\boldsymbol{\theta}$ should lead to the resulting $\Delta\vec{\boldsymbol{s}}$ being approximately equal to $\vec{\mathbf{e}}$ from (2.5). The $\vec{\mathbf{e}}$ can be approximated by:

$$\Delta\vec{\boldsymbol{s}} \approx \boldsymbol{J}(\boldsymbol{\theta_{curr}})\Delta\boldsymbol{\theta} \tag{2.12}$$

Using this approximation, we can reformulate the FK problem as $\vec{\boldsymbol{e}} = J\Delta\boldsymbol{\theta}$ and therefore our *Inverse Kinematics* problem form 2.7 can be expressed as $\Delta\boldsymbol{\theta} = J^{-1}\vec{\boldsymbol{e}}$.
The problem we run into with this solution is the construction of the inverse *Jacobian* matrix. The *Jacobian* matrix **J** may not be square or invertible. In the case of being invertible, the result may only work inferiorly because of it being nearly singular. Being singular means that no change in joint angle values may achieve the desired end-effector position as an outcome.

Jacobian transpose

One approach to calculating the value of $\Delta\boldsymbol{\theta}$ without having to calculate the inverse of **J** is done by replacing the inverse with the transpose of **J**.

$$\Delta\boldsymbol{\theta} = \alpha\boldsymbol{J}^T\vec{\boldsymbol{e}} \tag{2.13}$$

Of course the transpose and the inverse of **J** are not the same thing. When using the

theorems displayed in [**?** **?**] we can show that:

$$\langle JJ^T\vec{e}, \vec{e}\rangle = \langle J^T\vec{e}, J^T\vec{e}\rangle = \|J^T\vec{e}\| \geq 0 \tag{2.14}$$

Under a sufficiently small $\alpha > 0$ the updated angles from 2.11 will change the end-effector positions by approximately $\alpha JJ^T\vec{e}$. They also state that the value of α can be calculated by minimizing the new value of the error vector $\vec{e}$ after each update.

$$\alpha = \frac{\langle \vec{e}, JJ^T\vec{e}\rangle}{\langle JJ^T\vec{e}, JJ^T\vec{e}\rangle} \tag{2.15}$$

Jacobian pseudo inverse

Instead of calculating the normal inverse of the *Jacoboian*, which can lead to the problems described before, we can use the so called *pseudo-inverse* [**?**] for the calculation. The *pseudo inverse* is defined for all matrices **J**, even ones which are not square or not of full row rank.

$$\Delta\boldsymbol{\theta} = J^{\dagger}\vec{e} \tag{2.16}$$

The *pseudo inverse* represents the best possible solution for $J\Delta\boldsymbol{\theta} = \vec{e}$ in respect to least squares, but does suffer from instability near singularities. It has the property that the matrix $(I - J^{\dagger}J)$ performs a projection onto the null space of **J** . Therefore, for all vectors $\varphi, J(I - J^{\dagger}J)\varphi = 0$. With this knowledge, $\Delta\boldsymbol{\theta}$ can be calculated by:

$$\Delta\boldsymbol{\theta} = J^{\dagger}\vec{e} + (I - J^{\dagger}J)\varphi \tag{2.17}$$

for any vector φ and still obtain a value for $\Delta\theta$ which minimizes the value $J\Delta\boldsymbol{\theta} - \vec{e}$. The pseudo inverse of **J** can be derived as follows:

$$\begin{aligned} \boldsymbol{J}^{\dagger} &= (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T \\ \Delta\boldsymbol{\theta} &= (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T\vec{e} \\ (\boldsymbol{J}^T\boldsymbol{J})\Delta\boldsymbol{\theta} &= \boldsymbol{J}^T\vec{e} \end{aligned} \tag{2.18}$$

Using $\vec{\boldsymbol{y}} = \boldsymbol{J}^T\vec{e}$, we can now solve the equation $(\boldsymbol{J}^T\boldsymbol{J})\Delta\boldsymbol{\theta} = \vec{\boldsymbol{y}}$

2.2.4 Singular Value Decomposition

The usage of a *singular value decomposition* is another method of calculating the *pseudo inverse* of a *Jacobian* matrix. A *singular value decomposition* of **J** consists of expressing **J** in the form

$$\boldsymbol{J} = \boldsymbol{UDV^T} \tag{2.19}$$

For an *m*×*n* Jacobian matrix, **U** is *m*×*m* and the columns are the *Eigenvectors* of JJ^T. **D** is *m*×*n*, and the entries are zero except along the diagonal where $d_{ii} = \sigma_i$ with $\boldsymbol{\sigma_i}$ being the singular value and $\sigma 1 \geq \sigma 2 \geq ... \geq \sigma m \geq 0$. Singular values are the non-negative square roots of the *Eigenvalues* of JJ^T and J^TJ. **V** is *n*×*n* and the columns are the

Eigenvectors of $J^T J$. The construction of the pseudo inverse is done as follows:

$$\boldsymbol{J^{\dagger} = V D^{\dagger} U^{T}} \tag{2.20}$$

The pseudo-inverse of the diagonal matrix **D** is obtained by replacing each positive diagonal entry with it's reciprocal [**?**].

2.2.5 Damped Least Squares

The *Dampened Least Squares* method is applied for inverse kinematics as it avoids the singularity problems that can occur when using the *pseudoinverse*. It further provides a numerical stable method for selecting $\Delta\theta$. First occurences for inverse kinematics can be found in [**? ?**]. Instead of finding the minimum for $\Delta\theta$ as a best solution for the equation displayed in 2.5, the goal of this method is to find the value that minimizes the following quantity with $\lambda \in \mathbb{R}$ as a dampening factor.

$$\| J\Delta\boldsymbol{\theta} - \vec{\boldsymbol{e}} \|^2 + \lambda^2 \| \Delta\boldsymbol{\theta} \|^2 \tag{2.21}$$

which can be rewritten to the corresponding

$$(\boldsymbol{J^T J} + \lambda^2 I)\Delta\boldsymbol{\theta} = \boldsymbol{J^T}\vec{\boldsymbol{e}} \tag{2.22}$$

and therefore

$$\Delta\boldsymbol{\theta} = (\boldsymbol{J^T J} + \lambda^2 I)^{-1}\boldsymbol{J^T}\vec{\boldsymbol{e}} \tag{2.23}$$

The selection of a correct λ value is essential for this method, small values of λ give accurate solutions but low robustness to the occurrence of singular and near-singular configurations, while large values of λ result in low tracking accuracy even when a feasible and accurate solution would be possible [**? ?**].

2.2.6 Newton Methods

The family of *Newton methods* is based on a second order Taylor series expansion of the object function **f(x)**:

$$f(x+\sigma) = f(x) + [\bigtriangledown f(x)]^T\sigma + \frac{1}{2}\sigma^T H_f(x)\sigma \tag{2.24}$$

where Hf(x) is the *Hessian matrix*. The downside of this approach is that the calculation of the *Hessian matrix* bares high computational cost for each iteration as a result of its complexity. Like the approximation of the *Jacobian Inverse*, several attempts to lower the computational cost have been made by utilizing an approximation of the *Hessian matrix* based on a function gradient value. Since the *Newton methods* are posed as a minimization problem, they return smooth motion without erratic discontinuities. The *Newton methods* also have the advantage that they do not suffer from singularity problems, such as that which occurs when finding the *Jacobian Inverse*.

2.2.7 FABRIK Algorithm

A more recent approach in the kinematics field is the *FABRIK* algorithm proposed by Aristidou and Lasenby [**?**]. As shown in section 2.2.2, most approaches depend on computational intensive matrix operation like calculating an inverse and may have problems with matrix singularity.

The *FABRIK* algorithm does not depend on these matrix operations as is solves for the position of a point on a line to retrieve the new joint positions. This is done in an forward and also inverse solving approach, iterating these steps until the calculated position converges towards the target position from the tracking data.

Figure 2.4 illustrates the steps that are contained in one iteration step.The chain joints are denoted as $\mathbf{p}_i$ with the distance $\mathbf{d}_i$ being $|\mathbf{p}_{i+1} - \mathbf{p}_i|$. The *target point* for the *end-effector* is denoted as **t**. Step **(a)** displays the starting point of the iteration. The joint positions are taken from either a previous iteration or from an initial calibration. But before calculations can begin, the algorithm has to check whether the intended target point **t** is reachable for the *end-effector*. This is done by measuring the distance between the root of the *kinematic chain* and the *target point* **t**. This value is then compared with the sum of the distances $\mathbf{d}_i$.

$$dist_{t,d_1} < \sum_{k=1}^{i} d_i \qquad (2.25)$$

If the summed distance is greater, then the target **t** is within the reach of the system and the calculation can continue, otherwise the calculation has to be aborted and the error has to be manged otherwise. Assuming this requirement to be met, we can now begin with the first calculation.

The inverse calculation step is the first step, which is displayed in **(b)** and **(c)**. The calculation is started at the end effector, moving to the root of the chain.

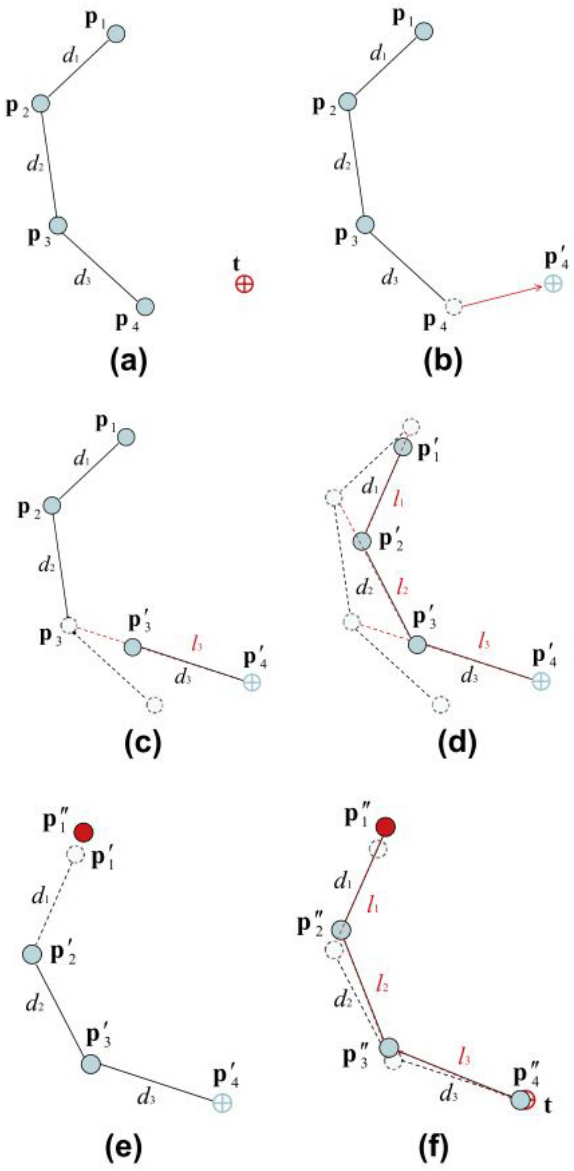

Figure 2.4: Forward and backward calculation steps for one iteration of the **FABRIK**algorithm. (a) Initial position of the system. (b) End effector is moved to target position. (c) Deterimne position of next joint on constructed line. (d) Repeat until root is reached. (e) Move root joint to initial position. (f) Repeat calculation in reverse direction [**?**]

Therefore we assume that the new position $\mathbf{p}_n^{'}$ with n=4,...,1 is equal to $\mathbf{t}$.

$$\mathbf{p}_n^{'} = \mathbf{t} \tag{2.26}$$

From this new point, we can construct a line that intersects $\mathbf{p}_n^{'}$ and $\mathbf{p}_{n-1}$.

$$\begin{aligned} A &= \mathbf{p}_n^{'} \\ B &= \mathbf{p}_{n-1} \\ \mathbf{l}_{n-1} &= \overline{AB} \end{aligned} \tag{2.27}$$

The resulting position of the new $\mathbf{p}_{n-1}^{'}$ point is located on this line with the distance of $\mathbf{d}_{n-1}$ from $\mathbf{p}_n^{'}$ (see **(c)**).

$$\mathbf{p}_{n-1}^{'} = \mathbf{p}_n^{'} + \left(\frac{\overline{AB}}{|\overline{AB}|} \cdot \mathbf{d}_{n-1} \right) \tag{2.28}$$

Consecutively, this is done with the remaining joints until the *root joint* is reached (see **(d)**). This finishes the first half of the iteration step. With the calculated positions, we now perform a forward calculation, starting from the root until we reach the *end-effector*. Since the *root* of the system normally does not move from it's initial position, we have to reset the *root joint* to this value before starting to calculate the new positions of the subsequent joints (see **(e)**).
Analogous to the procedure in the inverse step, we construct the lines between the points and determine the new position values of the joints. The end result of this step is shown in **(f)**. At this point, we can decide if the result position of the *end-effector* is appropriate in comparison to the value of $\mathbf{t}$. A simple threshold value for this case could be the position difference between these two points.
Since this algorithm does not rely on the heavy-weight matrix operations that the previously described methods incorporate, it converges much faster and in less iterations. Furthermore the incorporation of constraints to the calculation does not interfere with this values and is rather easy.

3 Tracking in real space

The previous sections provided information on the structure of the human hand and how to represent it in the digital space. To apply the algorithms presented in Section 2.2 to our digital hand model, we need input data from our real world representative.

3.1 General tracking technologies

The methods for gaining positional data can be roughly categorized into two major groups, glove based methods and vision based methods. Glove based methods have already been in developement since the 1980's [**?**] and since then have resulted in several solution attempts. Sturman and Zeltzer gave a survey on the existing tracking methods in their paper [**?**]. They distinguish between two areas of tracking, first the 3D positional tracking of the hand (and also other bodyparts) without regard to the hands shape and secondly the tracking of the hand shape with glove technologies. These tracking technologies presented are still applied today in modern tracking solutions [**? ?**]. They account for solutions with optical tracking based on marker detection, magnetic detection via measurements of an artificial magnetic field [**?**] and acoustic measurements via triangulation of ultrasonic pings.

3.1.1 Optical tracking

The components for an optical tracking systems consist of several cameras for object detection and a tracking characteristic of the object to be tracked. These characteristics can either be artificially applied ones like active flashing infrared LED on key tracking positions of the body or infrared reflective markers. A series of cameras positioned around the tracking subject will track these markers inside their visual fields.
The second method uses a single camera to capture the silhouette image of the subject, which is analyzed to determine positions of the various parts of the body and user gestures. The image data is handed over to special software, which correlates the marker positions in the multiple viewpoints and uses the different lens perspectives to calculate a 3D coordinate for each marker. These image interpretation and correlation tasks require computationally costly operations. The marker tracking is also prone to errors through variation in lighting of the scene, material reflection properties and also marker occlusion as the trackers are moved. Also most of the systems rely on several tracking cameras for a complete coverage of the tracking space. This leads to a higher system complexity in terms of setup and calibration.

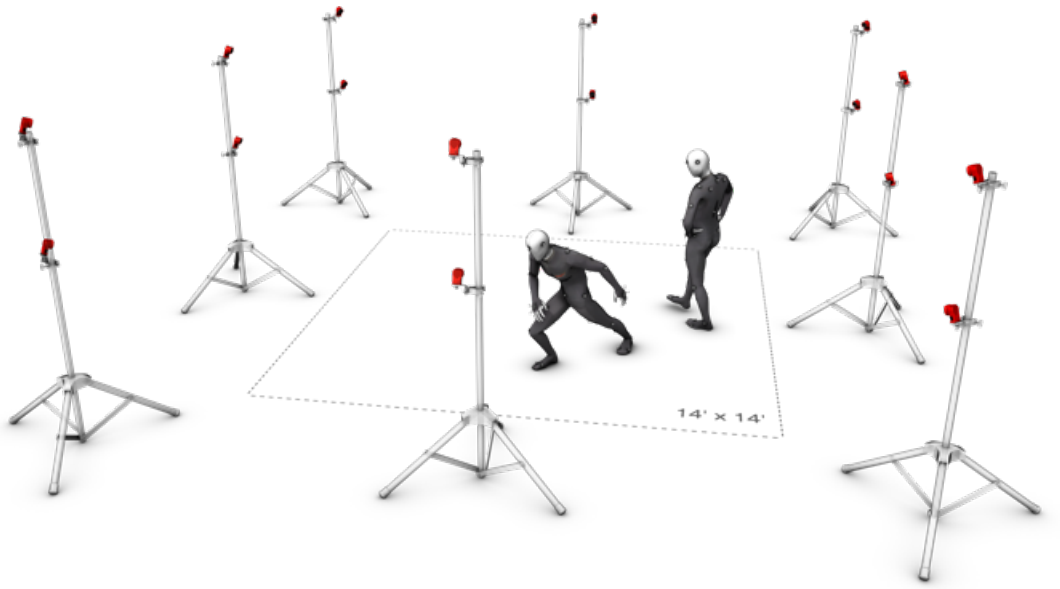

Figure 3.1: Example for an optical tracking system with passive infrared reflector markers [**?**]

3.1.2 Magnetic tracking

The usage of a magnetic field for position tracking is a relatively uncomplicated technology. The earth already provides us with a magnetic field to orient on. Similar to the optical trackers, magnet field sensors can be placed at key tracking positions. These sensors measure field strength in 3 orthogonal oriented axes to produce a 3D vector of the units orientation with respect to the excitation. As the earths magnetic field is prone to changes based on geographical location, data corrections have to be applied to the measurements.

Another solution is to generate an local magnetic field via a multi-coil source unit [**?**]. The source unit coils are energized in sequence and the corresponding magnetic field vector is measured in the sensor unit. With three such excitation's, one can estimate the position and orientation of the sensor unit with respect to the source unit. The downside of this technology is that ferromagnetic and conductive material in the surrounding environment will have an effect on the generated magnetic field. These distortion effects will form small unwanted secondary source units through the induction of small *Foucault currents* from the main sources magnetic field.
Magnetic fields have an inverse cubic falloff as a function of distance from the source, which limits the range of operation for the system. Position resolution in the radial direction from source to sensor depends on the gradient of the magnetic field strength, and thus the positional jitter grows as the fourth power of the separation distance. In comparison to other tracking technologies, the magnetic field tracking solution has the advantage of not suffering from line of sight problems from tracker occlusion. The magnetic fields are capable of passing through the human body. Also the sensor size for measuring magnetic fields is rather small, giving the trackers a small volume. Furthermore, only one source unit is needed for the tracking of multiple sensor units.

3.1.3 Acoustic tracking

The principles of acoustic tracking are very similar to those of optic tracking technologies. Instead of using light waves, the systems utilize acoustic pulses of ultrasonic wavelengths to measure the time of flight between emitter and sensor for range measurement. To get a good measuring result from the systems, the used acoustic transducers have to be as omnidirectional as possible, so that the signal can be detected no matter how the emitter is positioned or oriented in the tracking volume. For the speakers to achieve a wide beam width, their size has to be small.
To be able to build the microphones into the tracker, they can only have active surfaces of a few millimeters in diameter. This leads to a reduction in range as the efficiency of acoustic transducers is proportional to the active surface area. Also acoustic systems may have problems with ambient noises occluding the signal.
This becomes even more critical when using such a system outdoor. Sound waves travel at a much slower speed than light waves which brings benefits and downsides with it. Sound waves can be reflected from objects, producing echos which arrive at the receiving sensor at a later point in time. Here the slower speed can be beneficial as the system can await the first sound occurrence to arrive at the sensor and filter out all later reflections from the data. The reflections of the previous pulse also have to be subsided before a new measurement can be made, lowering the update rates of the system. The air the sound waves travel through is also a limiting factor like humidity, air pressure and air currents can influence the traveling sound waves. In comparison to the optical systems, the acoustic systems are not as prone to occlusion errors since sound waves have a better ability to bend around obstacles than light waves. Most of these downside can be adressed with a combination of these systems with another form of tracking like in [**?**].

3.2 Hand tracking systems

In comparison to the tracking of the whole human body, the tracking of the human hands with their maximum of 27 DOF's each in a small space is rather difficult to handle. The systems that try to achieve this have to encounter several difficulties. Self occlusion plays a great role in the system design, especially when working with only one optical tracking system for reduced complexity. These systems need to maintain a certain amount of processing speed for achieving a fluent reproduction of the captured motion. This requires the capability of processing large amounts of data in very short time intervals. Most prototype testing for the described systems is done in an controlled environment with a well known background. A known background reduces the amount of image preprocessing needed to be able to find the hand in the image. For a more widespread use, these systems need to be capable of registering the hand on an unrestricted background. This can have a wide range of patterns, color and lighting differences. Additionally, human hand motion itself is quite fast. This results in a higher frame rate demand for the tracking cameras. The following section will describe different approaches to solve the aforementioned difficulties.

3.2.1 Sensor based

Tracking systems that make use of sensors mounted to the hand via straps or a glove have alread been in use since the 1970's. First prototype glove systems included the *Sayre Glove* [**?**] and the *Digital Entry Data Glove* [**?**].
The *Sayre Glove* utilizes a light source and a photocell which are connected via a flexi-

ble tube. These components are mounted along each finger of the hand. The light that passes through the tube is influenced by the bending angle of the tube which corresponds to the finger bending. A large bending angle of the finger induces a larger bending angle of the tube, reducing the intensity of the light that reaches the photo diode. The resulting voltage in the photocell can then be mapped to a specific bending angle of the finger.

The *Digital Entry Data Glove* was patented by Gary Grimes in 1983. Instead of using only one type of sensors for measurement, this glove-based system used several sensor types for different measurements. Touch or proximity sensors were used for determining whether the user's thumb was touching another part of the hand or fingers. To measure the flexion of the joints in the thumb, index, and little finger, four *knuckle-bend* sensors were utilized.

Figure 3.2: Acceleglove system with the described sensor positions [**?**]

To get measurements for the tilting of the hand in respect to the horizontal plane, two tilt sensors were mounted. Finally two inertial sensors for measuring the twisting of the forearm and the flexing of the wrist were utilized. This glove was intended for creating alphanumeric characters from hand positions. Recognition of hand signs was performed via a hard-wired circuitry, which mapped 80 unique combinations of sensor readings to a subset of the 96 printable ASCII characters.
These early systems only provided a limited amount of sensors as the electronic parts were much larger than today. This also resulted in the operation not being very user friendly. As they were developed to serve very specific applications, they were used only briefly, and never for commercial use.
Modern developements in this sector include [**? ? ?**]. The *AcceleGlove* [**?**] consists of six dual-axis accelerometers, mounted on the fingers and the back of the palm, reporting position with respect to the gravitational vector. Sensors are placed on the back of the middle phalanges, on the back of the distal phalange of the thumb, and on the back of the palm.

Kuroda et al. [**?**] introduced the *StrinGlov*, which can obtains full *Degrees of Freedom* of the human hand using 24 *Inductcoders* and 9 contact sensors, and encodes hand postures into posture codes on its own *Digital Signal Processor* (DSP). The bending angles of the fingers are measured with the *Inductcoders*.
These sensors relate the finger movement to a change in magnetic flux induced by the movement of the sensor parts that are attached to the finger. The sensor functionality is similar to the functionality of the light sensors from [**?**]. The 9 contact sensors, also based on magnetic fields, are put on the fingertips and on the inside of the hand to be able to measure contact between two fingertips or the fingertips and the hand. The system furthermore benefits from simple structure, which leads to low production cost.

Majeau et al. [**?**] proposed a systems that uses optical flexion sensors to determine the bending angles of the fingers. The optical flexion sensors consist of a *Light Emitting Diode* (LED), a photodetector and an optical fibre. The LED emits light which travels through the optical fibres. The intensity that results as the output from the fibre end is measured by the photosensor. This measuring principle also corresponds to the principle of [**?**].

Figure 3.3: Color glove setup used by Wang and Popovic [**?**]

Furthermore, the system is also capable of measuring the abduction of two fingers. Here, the LED is mounted to one finger and the detector to the other. The optical fiber is run from the LED down the finger to the knuckle base and then up to the detector. When abducting the two fingers, the angle at the knuckle base changes: This results in an intensity change at the detector through the loss of intensity at the curvature of the optic fiber. The change in intensity of both measuring techniques can be mapped to corresponding hand movements. Further readings on sensor based hand tracking systems can be found in [**? ?**].

3.2.2 Appearance based - Optical marker

Optical approaches use properties of the human hand to hand to estimate the current position. The evaluation of the hand pose is usually split up into two steps. The first one is the registration of the "real" hand.
This is mostly done by at least one camera which is aimed at the hand. The camera then supplies a continuous stream of image data to an evaluation software. In the second step, the evaluation software tries to find key spots in the provided images. This data is then used for the search of the matching digital hand pose. The definition of the key spots for the pose matching can be achieved by several techniques.

An example is the use of markings for finger segments as displayed in [**? ? ?**]. In the method described by Wang and Popovic [**?**], a glove with a color pattern is used. The glove is segmented into ten segments, colored randomly from a pool of ten distinct colors (see Figure 3.3). The pattern is created by selecting twenty seed triangles on a 3D hand model that have a maximum distance to each other. The remaining triangles are assigned into patches, based on their distance to each of these seeds. Each patch is assigned one of ten colors at random. The jagged boundaries between the patches are artifacts from the low triangle count of the used hand model.
Algortihms that use other characteristics for tracking like texture or shading [**?**] rely on an accurate pose from a previous frame to constrain the posture search in the current frame. When using bare hand pose estimation, two different hand poses can map to similar images. This can lead to an inaccurate pose estimation and the breakdown of the algorithm, if the wrong estimation is made.
The benefit of the color glove is that with the unique patch pattern, hand poses always map to distinguishable images. Therefore it can effectively recover itself at each frame without the need of a previous frame. With the colored glove as a tracking feature, the

images from the camera are prepared for a database sampling step. The database that was used consisted of 18,000 finger configurations.
A distance metric between two configurations was defined as the *Root Mean Square* (RMS) error between the vertex positions of the corresponding skinned hand models. With this distance metric, a low-dispersion sampling was used to draw a uniform set of samples from the over-complete collection of finger configurations.
The selected configurations are rendered at various 3-D orientations using a synthetic hand model at a fixed position from the virtual camera. The camera image was then compared to the database images. This is done via a nearest neighbor lookup using a *Hausdorff-like* distance [**?**] and penalizing the distance to the closest pixel of the same color in the other image. The resulting pose from the database can then be applied to the digital hand model.

The method presented by Fredriksson and Ryen [**?**] also uses a color coded glove. In contrast to the fully colored glove described before, the used glove only has colored fingertips. Color markers for the palm and a colored band for the wrist are also used. The palm markers and the wrist band are used to retrieve the 3D position of the hand. The wrist marker is furthermore used to define the bounding box of the the tracking area in a calibration process. Defining a bounding greatly simplifies further image analysis operations by taking away a large amount of unneeded image data.
After determining position and orientation of the hand in 3D by utilizing wrist band and palm marker position, the system tracks the grip angle and the lateral tilt angle for each finger. The grip angle is defined as the curling of the finger towards the palm. The second angle is defined as the lateral tilt angle, which measures the spread of the fingers.
The grip angle is calculated by comparing the density of finger pixels around an estimated knuckle line. From the density, an estimation for the fingers y-axis position around an origin position at the knuckles base is made. This value is then transformed into an angle value using an inverse tangent operation.

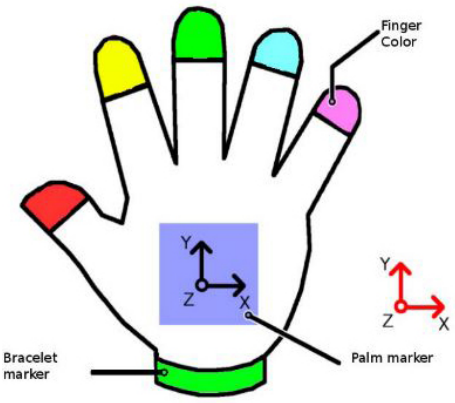

Figure 3.4: Color glove setup used by Fredriksson and Ryen [**?**]

One downside of this tracking method is that it does not incorporate the movement of the thumb. The thumb has other movement possibilities and therefore needs a different algorithmic approach. Also the approach only focuses on the the tracking of the hand data and does not implement a digital counterpart.

3.2.3 Model based - Image analysis

Hand tracking with image analysis of optical marker positions has the flaw that these markers have to be put onto the hand. Using a glove for this purpose imposes just a little bit of inconvenience, but has the drawback of not fitting every hand shape equally. Gloves also impose the problem of having to deal with hygiene when switching between users.
The most natural way of tracking the human hand would be by simply using the properties of the human hand like skin color and shape for tracking. This would remove the need for any artificially added tracking features and has the greatest amount of comfort

for the human user. A full overview of the technological advances in this area can be found in [**?** **?**]. The next part will only highlight some of the main topics displayed.

As already explained before, the human hand motion leads to a large variety of shapes with many self-occlusions. This becomes even more critical when trying to reproduce the human hand pose via computer vision. Also the separation between background image information and the relevant hand information is a major topic, which is usually also the first step in the algorithmic process.

An established method for the extraction of background data was introduced by Stauffer and Grimson [**?**]. It uses the idea of representing each pixel value as a *Mixture of Gaussian functions* (*MoG*) and updating these pixels during run time with new *Gaussian functions*.
The algorithm stores the mean, the variance, and the likelihood value, based on previous values, for each pixel location and distribution. A new input is compared to the stored mean values for each distribution at this location.
A result that is less than a constant times the variance declares that this pixel is part of that distribution and should be updated accordingly. For a higher value result, a new distribution is created which replaces the least probable distribution. A new input belongs to the background, if the distribution that it is part of is one of the most likely distributions. A foreground pixel that keeps the same color over time will slowly be incorporated into the background.
The decision rule, whether a new pixel belongs to an existing distribution or not, can be written as [**?**]:

$$if\ (|P_{new} - P^d_{mean}| \leq KP^d_{std}) \quad then \quad P_{new}\ \in\ Distribution\ d \tag{3.1}$$

Where P_{new} is the new pixel value. P^d_{mean} and P^d_{std} are the stored mean and standard deviation of distribution *d* values. *K* is a constant. The convenience of this method is that it is also able to subtract background information from uncontrolled environments. This makes it possible for systems to work outside of lab conditions. After the segmentation of foreground and background is achieved,the hand pose has to be retrieved from the left-over image data. Solutions for retireving a full set of DOF as hand pose estimation data can be split into two main categories [**?**], *Model-based tracking* and *Single frame pose estimation*.

Model-based tracking refers to a top-down tracking method based on parametric models of the 3D hand shape and it's kinematic structure. The root idea here is to execute a search at each frame of an image sequence to find the pose of the shape model that best matches the features extracted from the images. A prediction based on previous motion and dynamics data from the object is set up as a base for the search. Optimization features can be incorporated, like employing a local search around the calculated prediction to produce a single best estimate at each frame. The downside of this method is the weakness of accumulating errors over longer run time periods due to occlusions and complexity of hand motion. These can be overcome by keeping track of multiple outcome hypotheses at each frame for error reduction.

Single frame pose estimation in contrast to the previous method, does not utilize assumptions on time coherence. At each iteration of the search algorithm the whole data

set is searched for a match. On one side, this introduces more computational work for achieving a match. On the other side, this lowers the amount of constraints that have to be put on the user when initializing the tracker system. Furthermore, the possibly rapid motion of the human hands and fingers acts as failure factor for time coherence dependent approaches, where a single frame approach is not as vulnerable.

3.3 State of the art technologies

The currently most used type of system for tracking hardware is a setup of a combination of RGB and infrared sensors for imaging an depth measurement (RGB-D). The first commercial system using this technology is the *Microsoft Kinect* system[**?**]. The system uses a infrared projector and an infrared sensor to measure distances in the captured scene.

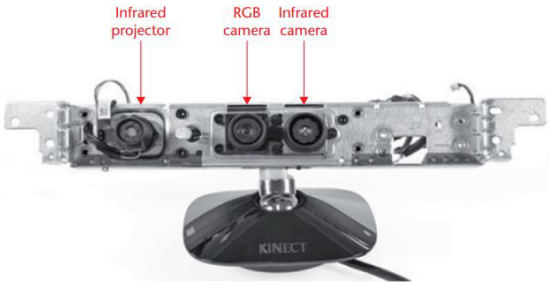

Figure 3.5: Microsoft kinect camera system with the three used sensor units [**?**]

State of the art systems that also incorporate depth sensing through infrared distance measurement are the *Intel RealSense* system and the more consumer focused *Leap Motion Controller*. The *Leap Motion Controller* does not supply a separate RGB sensor and does therefore only provide depth measurement data within a specific range from the device [**?**]. It combines two infrared cameras with three infrared emitter LED's.

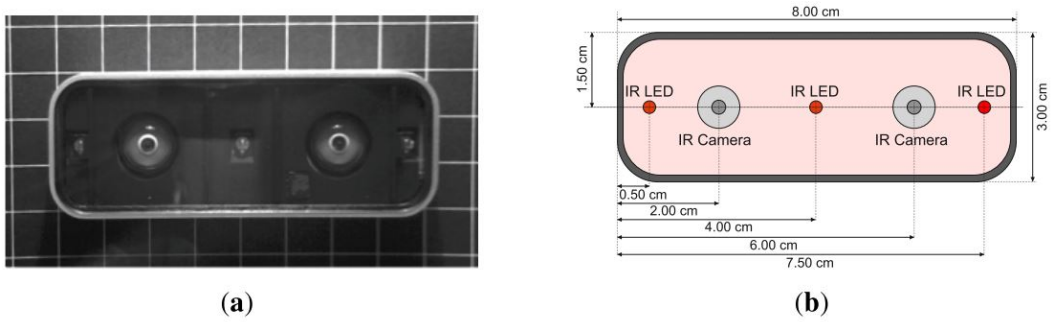

Figure 3.6: Visualization of a (a) Real (using infrared imaging) and (b) Schematic View of the *Leap Motion Controller* [**?**].

The latest product in this category are the *Intel RealSense* camera systems [**?**]. The Intel systems, like the *Leap Motion Controller* utilize a stereoscopic infrared system setup, and can also provide RGB sensor data, depending on the selected model.

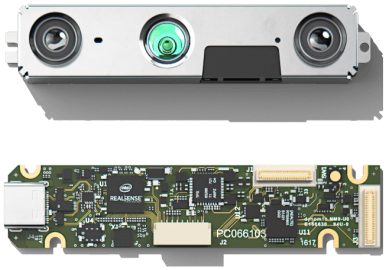

Figure 3.7: Visualization of a *Intel ReaslSense* depth module[**?**]

The *Kinect* can supply depth data at around 30 fps. It's modern descendants from Intel can supply data at frame rates of up to 90 fps. The systems provide the convenience of not having to apply any form of distinguishable marking to the surface that has to be tracked. The downside of this feature is that the devices only supply SDKs for accessing the collected data from the device. The post processing of this data needs additional sophisticated image analysis or neural network processing to figure out feature points of the tracked object from the supplied image data [**? ?**].

Image analysis of large images can be expensive for real time applications and training of new neural networks for image analysis is also time consumable. The tweaking of the network parameters and network retraining until the trained result fits the application also consume greater amounts of time. An recent approach of combining RGB-D Data and the usage of *CNNs* (Convolutional Neural Networks) to estimate hand pose and render a correct digital representaion of the hand is shown in [**?**].
Their approach uses two subsequently applied *CNN's* to localize the hand and regress 3D joint locations. The first *CNN* estimates the 2D position of the hand center in the input. The combined information of the hand position, together with the corresponding input depth value, is used to generate a normalized cropped image. This image is then fed into a second *CNN* to regress relative 3D hand joint locations in real time. Pose estimation is furthermore refined by using a kinematic pose tracking energy to achieve more accuracy, robustness and temporal stability.
They also introduce a new photo-realistic data set that uses a merged reality approach to capture and synthesize large amounts of annotated data of natural hand interaction in cluttered scenes for CNN training data.

4 System Perquisites

Before elaborating a final system conception, some perquisites have to be defined.
System components to be defined:

- Display hardware
- Tracking technology
- Tracking hardware
- Data cleaning
- Image analysis
- Inverse kinematics algorithm
- Hand model

4.1 Display hardware

The first component definition that has to be done is actually the last component of the processing chain. The final display hardware properties have to be defined initially, as these are the determining factor for all other components. Since the result of hand and object tracking is the most immersive when using a virtual reality headset rather than a normal display, these do not need to be taken into consideration.
Another device category, which may be in the focus of interest, contains current mobile device platforms like *Google cardboard* or the *Samsung Gear*. In terms of creating a low cost consumer-grade solution, these devices should be the first category to look at. They are more widely available and do not need extra hardware except the smart-phone as a display.
Most modern devices have native support of WebGL in the browser and the emerging *WebXR* standard makes it fairly easy to display VR content. They also come with the convenience of already having a set of integrated sensors for position an rotation integrated. The downside of this device class reveals when taking a look at the hardware specifications. Although the newest high-tier consumer grade devices do have a good performance output for their size, they are still no comparison for a desktop PC setup. Also these devices, as they are first of all mobile phones, show a lack of wire connections. This makes it necessary to mostly communicate over wireless protocols. This leads to a growth in infrastructure (server, wi-fi-hotspot, etc.) and every further node in the processing chain leads to a physical signal delay.

After these first preliminary steps, only pure VR-headsets like the *HTC Vive* [**?**] or *Occulus Rift* [**?**] should be the main focus. They utilize the processing capabilities of a dedicated GPU and high-power CPU capabilities of modern desktop computers. External sensors provide high precision tracking results. Furthermore, they provide a variety of connection ports for wired connections, eliminating the need for wireless transmissions and providing fast signal transport. In terms of cost, these devices do lie on the more costly

side as you need the headset itself and a hardware setup which is capable of providing the required output frame-rate. But with the latest generations of GPU's and CPU's, this required category was opened towards the normal consumer.

The *HTC Vive*, as well as the *Occulus Rift* are able to display 90 images per second on their two displays, each having a resolution of 2160x1200 pixels.The resulting resolution for the user corresponds to a little over full HD resolution per eye. The duration of one frame at 90 frames per second is around 11 ms. This is the time-frame in which the rest of the components must theoretically supply the image data, do the image analysis to retrieve the tracking data, recalculate the inverse kinematics model, calculate the object position and render the whole scene before sending it to the headset to be displayed.

4.2 Tracking technology

Physical sensors placed on the hand tend to give more precise tracking results compared to optical methods. The cost for the data precision is that the sensors have to be mounted to the hand in somehow. Cables or fibers, depending on sensor type, can hinder hand movement. Utilizing a glove-based system brings the downside of the "one size fits all" problem. Human hands can differ largely in size. Therefore, one glove will not be able to be used for a wide range of users. Furthermore, cloth gloves tend to get dirty while being used and are therefore unsuitable in terms of hygiene.
Color markers can be utilized very easily. The simplest form could be colored electrical tape wrapped around the fingertips. These markers can be made disposable after usage. The size of the marker can be adapted to be large enough to realize a robust tracking, while being small enough to not constrain hand movement. They also provide the possibility to perform a person-specific calibration of the hand model in an initialization step. The easiest way to detect color markers is to use an optical tracking system. Monocular optical tracking system are relatively easy to set up, as they only need one camera. The complexity in the following processing steps of evaluating the marker positions from only a single frame rises drastically.
Stereoscopic system, in comparison, have a higher initial effort in setting up and calibration. The following processing steps are fairly easy in comparison.

4.3 Tracking Hardware

As explained in the previous chapters, the motion of the human hand is complex and in some cases really fast. Therefore, the optical tracking hardware should ideally be able to record images with the same or higher frequency as the display medium that is to be used. Prosumer and professional grade cameras are able to produce these kind of frame rates (60 fps and above), but most of the time store these high frame-rate videos directly to a hard drive rather than broadcasting them somewhere. Also, the hardware that is needed to record such specific high frame rates is relatively expensive. Affordable consumer-grade cameras like the *GoPro* can record at a frame-rate of 120 fps, but the live transmission of this material has a time delay of several frames.

When trying to utilize low cost hardware, the limitation of recording and sending a video signal from a camera to a processing hardware will therefore be limited to 60 fps.

Another option that can be taken into consideration is to completely eliminate the need to transport the image data from camera to another device. This can be achieved when image capturing and processing happen on the same device.

The *Raspberry Pi 3* microcomputer provides this capability. It combines a quite powerful computation capacity with an on-board hardware connector for a camera. The camera hardware is able to record in a 1080x1920 resolution at 60 fps and at lower resolutions up to 90 fps. The camera module can be directly connected to the circuit board, providing direct access for further processing.
For a complete three dimension spatial tracking, a setup with only one camera is not sufficient. It lacks the data for the depth position of the object. To get this data, further hardware has to be used. One option would be the utilization of a *time-of-flight* based depth sensing system like the ones described in Section 3.3, but these system do come at a rather high entry level price.
A second option could be the tracking pods that HTC offers as an expansion to it's *Vive* system. These trackers could provide 3D positional data, but with the downside of being relatively bulky. This would hinder the movement space of the hand and the tracker would have to be positioned on the arm as near as possible to the wrist. This would only provide the position of the wrist. But as hindering as these trackers are on the hand, they could be easily used to provide tracking data for used objects. The tracker could be positioned onto the object, where it does not hamper the function.

The hardware setup of the *Raspberry Pi* with the camera should be relatively easy and cost efficient.Therefore, the usage of a second *Raspberry Pi* with a second camera is good leverage point. The two cameras can be set up to create a stereoscopic camera setup. This can provide information about depth position through the disparity of the two camera images. Furthermore, the image operations that need to be independently applied to each of the cameras frames can be done on the two separate device. This eliminates the need for sending data traffic with image information through a network for post processing.
A downside of this method is similar to all stereoscopic camera setups. The two cameras need to have some kind of frame synchronization to get the corresponding two frames for evaluation.

4.4 Data Filtering

Every system that deals with the computation of live data is suspect to data fluctuation in input and output values. It is also to be assumed, that the system will not be able to achieve an ideal synchronization between the cameras. This will probably lead to a degradation of tracking performance, which has to be handled. It has to be evaluated at which point a data filtering will lead to the best end results.

4.5 Image analysis

The images recorded by the camera system need to be processed further down the line to get the positional data of selected hand features. The most common library to do such image processing is the *OpenCV* [**?**] library. It is originally written in C and was ported into several other programming languages. It has built-in features for camera calibration,

color and object detection as well as several image optimization features that might be helpful for the processing.

4.6 Inverse kinematics

After the tracking hardware, the inverse kinematics model is the second essential part of the setup. It receives the position data from the image post-processing and calculates the resulting current hand model position. The applied algorithm for the IK solution should be able to keep up with the data-flow from the tracking algorithm and should produce a fluid optical result. As the calculation of the IK solution and the rendering of the scene with the models should be done on the same machine, computation power and memory consumption should be critical criteria. As the system will rely on optical tracking data, an algorithm that is capable of dealing with tracker occlusions and recovery from this data loss should be prioritized.

4.7 Hand model

After retrieving all needed data from the real world counterpart and calculation of pose estimation is finished, the returned solution has to be displayed in the digital world. Therefore, a digital hand model is necessary to represent the calculated data. Modern-day computer animation techniques make it possible to create nearly photo realistic digital replicas of human skin wrapped onto anatomically precisely modeled body parts. These highly complex models require a lot of calculation resources and are mostly not appropriate for a real-time rendering approach.
When adapting calculation data to digital hand models, those of lesser complexity are preferred. Sacrificing the quality of the resulting output for speed and more fluent motion results is acceptable.

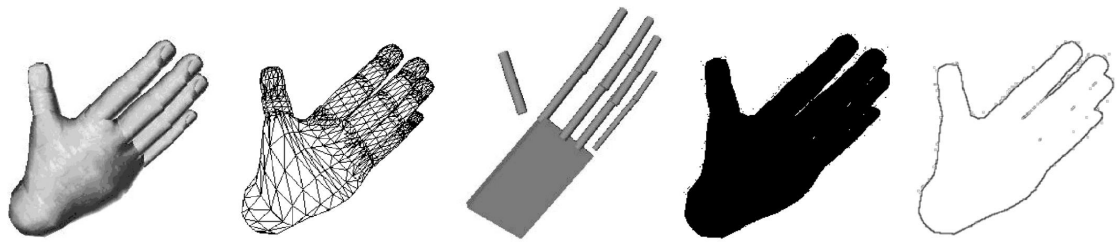

Figure 4.1: Different hand models can be used to represent the same hand posture. (a) 3D Textured volumetric model. (b) 3D wire-frame volumetric model. (c) 3D skeletal model. (d) Binary silhouette. (e) Contour. [**?** , p. 682]

Figure 4.1 shows different model representations of the same hand posture. The models of **(a)** and **(b)** represent the aforementioned types of models that would be used in modern computer-graphics for displaying hand models realistically.
These models are based on complex 3D surfaces like *NURBS* (nonuniform rational B-splines) which are wrapped around an underlying skeleton. The wire-frame model of **(b)** give a good overview of how many vertex calculation have to be made for the surface in each frame, causing these models to hardly run in real time if not supplying a large amount of computational power.
In applications where the 3D orientation of each individual finger is not necessary for

processing and the general silhouette is sufficient (e.g general hand position tracking or sign language detection), even more reduced representation forms like **(d)** and **(e)** can be chosen. The model displayed in **(c)** is a simplification of the human hand structure with geometric primitives like cylinders and boxes. These geometric primitives have the virtue of being much easier to describe than a *NURBS* model. A cylinder, for example, is fully described by it's height and it's radius under the assumption that the position and orientation data is generally needed for all the described models. This type of model is also referred to as *skeletal models*, as they mostly try to reproduce the structure of the human hand as described in Section 2.1.
For evaluation purposes, a rather simple model form should be used for the representation of the tracked hand in digital space. The outcome results for tracking accuracy, update speed and the resulting match of physical and digital position of the hand are of more relevance than a highly realistic hand model.
To receive a correct optical result of the hand model finger and hand dimensions have to be measured for each hand specifically and stored into a data model for correct bone dimensions of the hand model. Ideally, this step should be automateable and included into the system spin up process.

5 System conception

The following section will describe the conceptions of the technical and software parts for the real prototype system. A broader description of the system setup will be given, before going deeper into the conception of single important parts of the system.

5.1 Technical conception

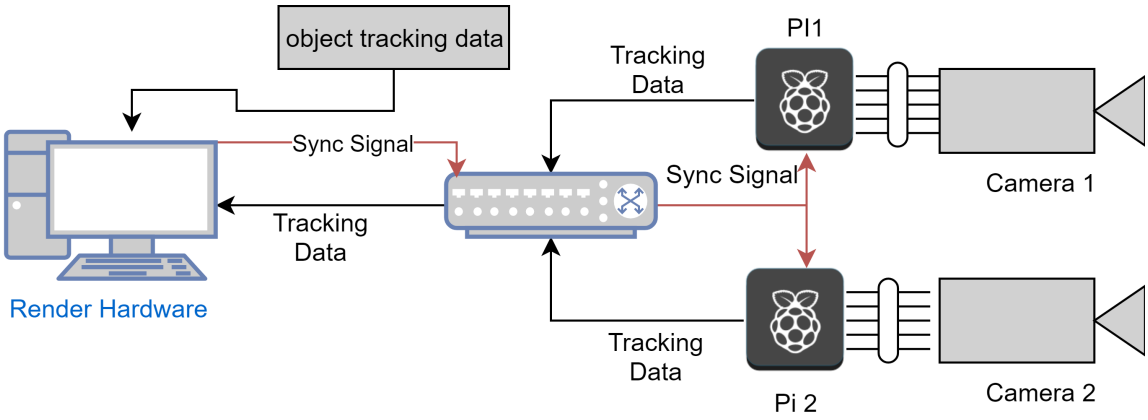

Figure 5.1: System setup concept with all technical parts

As depicted in Figure 5.1, the technical components of the setup are relatively simple. The same image processing steps have to be done on the images for both cameras. Instead of using one large dimensioned unit which takes care of all calculations, the calculations are outsourced onto the Raspberry Pi's. The image processing can be done on these two in parallel, which cuts down processing time. Furthermore the image data stays on the devices and does not have to be sent to another unit. Sending the data would introduce network and data reading latency.

The two **Raspberry Pi 3 Model B**, which serve as the controllers for the **Raspberry Pi Camera** are connected via network cable to a network switch. A connection over wireless network would be possible with the on-board capabilities. This might create latency problems and/or signal interference with other existing networks. Therefor a cable connection is the better solution.
The switch also connects to the "Master" PC to which the "Slave" Pi's communicate their data. The "Master" also takes care of the stereoscopic calculations, as it is dimensioned with far more computational power than the "Slaves". The 2D positional data from the "Slaves" and the calculation results from the stereoscopic image disparity is fed into the hand model which is running on the "Master". The model's calculation solution is then applied to the digital hand model and rendered.

5.2 Construction details

For the system construction, it is important to know what the primary use case of the system will be. Set up as a mobile system, the Raspberry Pi's could be run with battery power. A wireless communication leads to the beforehand described problems and a long enough cable connection is rather cumbersome. Furthermore, the cameras need to be calibrated at a fixed position for correct depth measurements. Some kind of mounting harness is needed for the cameras to stay in a fixed position.
A more easier approach is defining a seated use-case for the user. The systems main chore is to track the hand of the user. A seated position reduces position tracking errors, as it does not have to take account of large positional changes of the rest of the body. The tracking volume that needs to be accomplished for a seated position is also reduced the limitations of the arm length. A tracking volume of about $100x100x100$ cm should be sufficient to track most of the movement.
A frame of 30x30 mm construction profile is used as a base to mount all the components of the system. To reduce possible color reflections from the background of the tracking space, black fabric is used as background material. The other components of the system are:

- 2 Raspberry Pi 3 Model 3 with 1GB RAM and a 64GB micro-SD card running a current Raspbian OS and a C++14 compiler
- Compiled version of OpenCV and Boost library
- Java LWJGL 3 and CALIKO library
- 2 Raspberry Pi Camera Modules V2 with a 100 cm connector cable
- Ethernet Switch
- Monitor for calibration and debugging of the Raspberry side software
- PC with decent graphics card and a current Java version 8+
- Mounting peripherals
- Active cooled case for the Raspberry
- Two micro USB power supplies with 3.0 A output current
- HTC Vive Lighthouse and Tracker

The active cooled Raspberry Pi cases are an optional part of the construction, but the real time image processing on the CPU of the Raspberry together with the running camera produces larger amounts of heat than in normal operation mode. The attached heat sinks and the active cooling fan helps with faster heat dissipation. The 3.0A power supply units are recommended, as lower output currents can lead to a power brown-out of the system. This can cause system damage over longer usage times and even make the used SD card unreadable.

Mounting peripherals

The used cameras are basically camera sensors mounted on a PCB with an optic. They have 4 drilled holes as fixing points. A simple holder plate was designed and printed with a 3D printer for faster prototyping. The holder was designed in two parts to be able to change the mounting distance of the camera setup from the construction profile. Cut-outs in the mounting plate for the cameras provide the ability to mount the cameras with two machine screws and move their end positions along a fixed axis (see Figure 5.2). This guarantees that the cameras are mounted without a vertical position offset. For the current setup, the distance between the two cameras is set to 75 mm. This corresponds to the inter-axial distance of the human eyes.

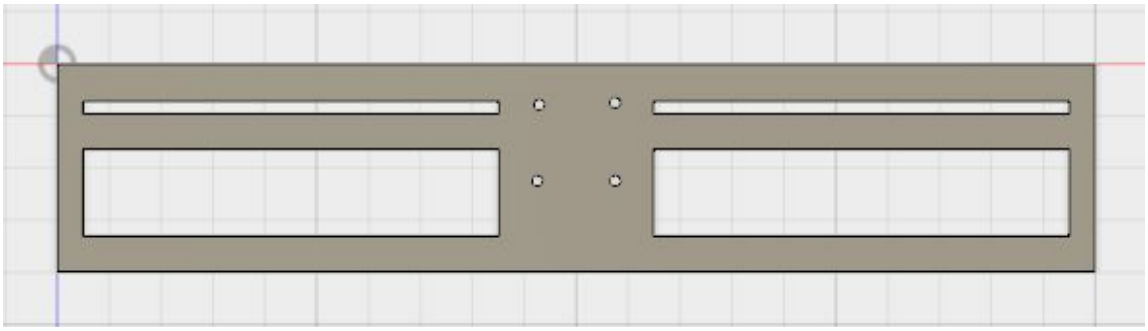

Figure 5.2: Technical drawing of the camera holder

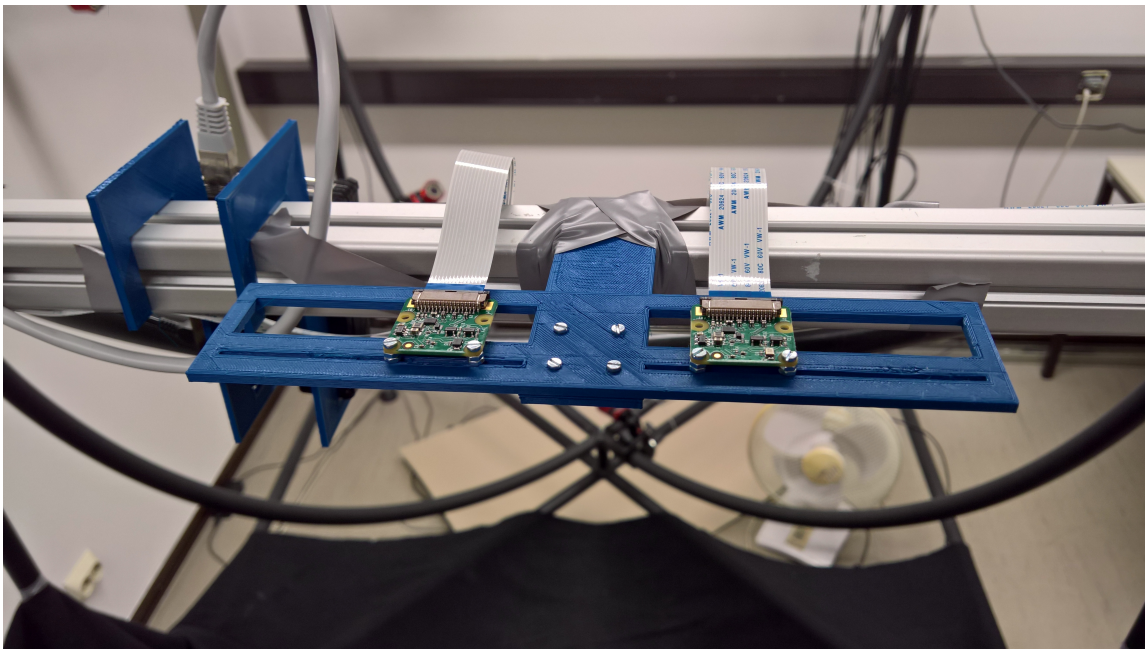

Figure 5.3: Printed camera holder with mounted cameras

5.3 Image analysis on the Raspberry

For the image analysis part, *OpenCV3* with it's *Python 3* bindings is chosen. The *OpenCV* package needs to be downloaded and compiled onto each device before use. Before image analysis part can start, the used cameras need to be calibrated. As they are basically pinhole cameras, they introduce amounts of radial and tangential distortion to the images. These distortions need to be compensated for, especially when using them as source for

the stereo images. Image distortion in these pictures would lead to incorrect calculations for image depth. *OpenCV* supplies the tools to calculate the distortion parameters as well as the camera matrix [**?**]. The camera matrix is basically a description for the transformation of points in 3D object space to the 2D image space of the camera. For the used cameras, we can consider a central projection of points in space onto our image plane. The center of the camera is considered the center of an euclidean coordinate system and the projection plane **z** is located at distance *f* equal to the focal length of the camera.

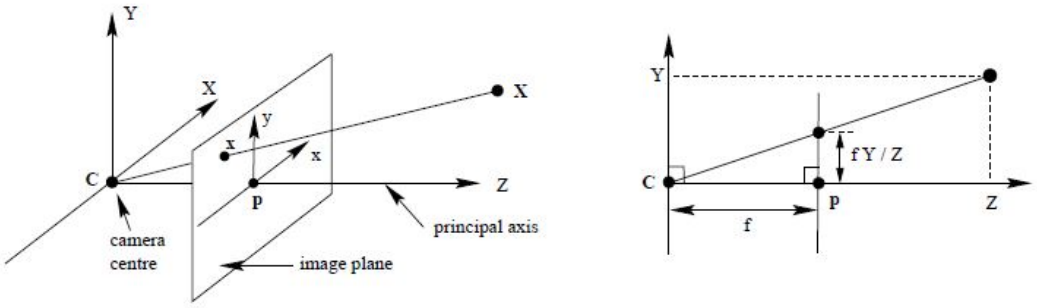

Figure 5.4: Image point projection for pinhole camera systems [**?**]

As shown in Figure 5.4, a point from the object space can be projected onto the image plane by drawing a ray from the object space point to the camera center. The intersection point of the ray with the image plane defines the projection point. With this knowledge the projection can be described as :

$$(X, Y, Z)^T \mapsto (fX/Z, fY/Z)^T \tag{5.1}$$

For the calculation of the image distortion, *OpenCV* uses a chessboard image for calibration. The printed image is held in front of the camera at a constant distance and rotated and positioned differently. After every position/rotation change an image of the pattern should be taken. These images are then used to find the corners at the intersection of the black and white squares. With the previous knowledge of the square sizes, the projection errors in radial and tangential directions can be calculated:

$$\begin{aligned} x_{tanDistorted} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 + r^6) \\ y_{tanDistorted} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 + r^6) \\ \\ x_{radDistorted} &= x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{radDistorted} &= x + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{aligned} \tag{5.2}$$

As can bee seen from the above equations, the parameters that need to be calculated are k_1, k_2, p_1, p_2, k_3. These parameters are called the *distortion coefficients*. Together with the *intrinsic* and *extrinsic* parameters of the camera, the camera images can be distorted to get better image results. This undistortion operation has to be done for every image frame that the camera produces.

When using a stereoscopic setup, the cameras have to be stereo rectified to account for vertical deviation between the camera images. *OpenCV* provides all required methods for doing so. Instead of taking images with each camera separately, both cameras

are triggered simultaneously to take an image of the chessboard pattern used for camera calibration. With this set of image pairs, the calibration utility is able to calculate a remapping transformation for each camera. After the application of this remapping, corresponding image points should be on the same horizontal level in the image.
After the cameras are calibrated, the resulting image optimization can be applied after image retrieval.
The rest of the process is now comprised of:

- Image conversion to HSV colorspace
- Mask construction and filtering
- Contour finding an position calculation
- Sending of positional data via network as UPD package

5.3.1 Image acquisition from the camera

As the computation time for image filtering and mask generation may vary, it makes sense to separate the image acquisition from the computation part. Therefore, the loading of the image data frame from the camera could be outsourced into its own thread. The synchronicity of the two frame reading threads on the Pi's is assumed for the first implementation and has to be tested on a finished setup. These treads will take the data from the current image frame on the camera and hand it over to the main thread. There the image processing is handled. For the first frames, the full image area is needed since the position of the markers is not known. Performance-wise, this introduces a lot of overhead. Each mask generation step for the specific color hast to scan the complete image area. Therefore every pixel has to be read out 5 times to get the tracking for all finger markers.

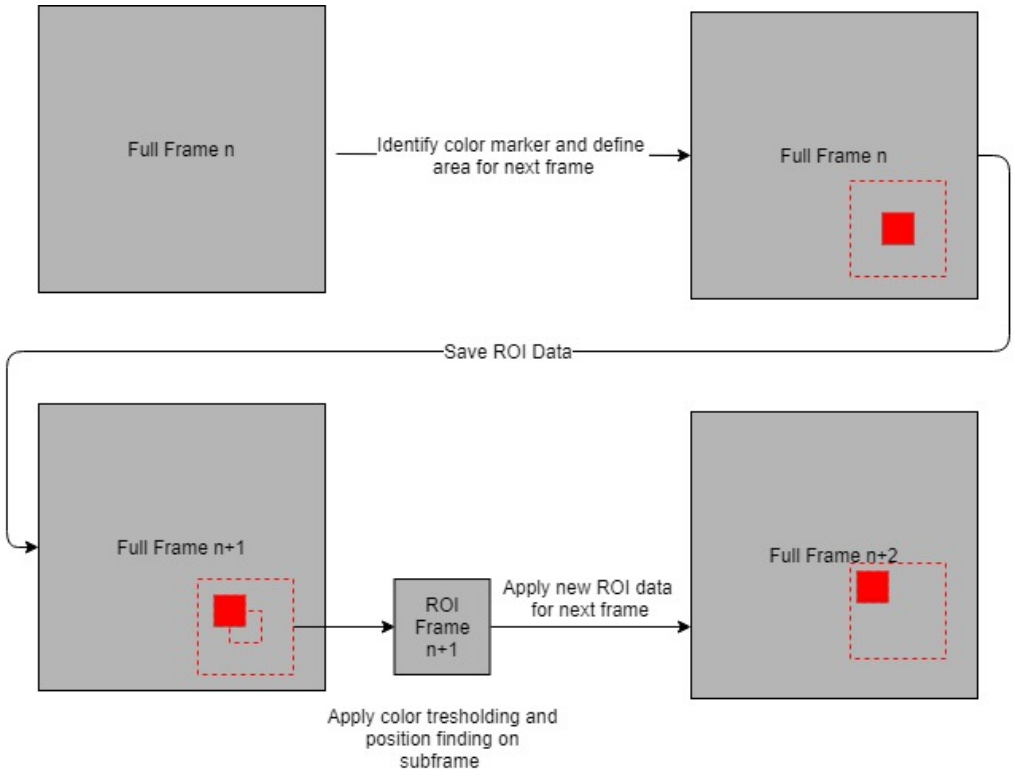

Figure 5.5: ROI Area workflow for consecutive frames

After a few frames, we get hold of the marker position and can define a reduced region of interest (ROI) on the image (Figure 5.5). The new ROI results from the positional data of the previous frame. It has to be dimensioned correctly as to incorporate the possibility of large position differences between consecutive frames.
Furthermore, the offset position, preferably the top left corner of the region, has to be kept track of. By taking a sub-region of the frame as input for further processing, the resulting coordinates relate to the top left corner of this sub-region as origin. Consequently, only the first set of coordinates has the correct origin as reference point for the next frame. Following frames need this information to retrieve the correct sub-region from the full frame. This is done by adding the offset to the calculated positional data and using the new value for the position.
First frame ROI calculation is based on a minimum *axis-aligned bounding box (AABB)* fitted onto the marker. The offset value for the ROI is added to the bounding box values resulting in a rectangle data-set containing the position of the offset top-left corner $\vec{\mathbf{tl}}$ and the offset **width** and **height** of the ROI rectangle:

$$\begin{aligned} \vec{\mathbf{tl_0}} &= (AABB_{x_0} - \text{offset}_x, AABB_{y_0} - \text{offset}_y) \\ \mathbf{width_0} &= AABBw_0 + (2 \cdot \text{offset}_x) \\ \mathbf{height_0} &= AABB_{h_0} + (2 \cdot \text{offset}_y) \end{aligned} \tag{5.3}$$

These values are taken to extract the sub-region of the consecutive at $\vec{\mathbf{tl}}$ with size of width and height. The $\vec{\mathbf{tl}}$ value is also saved as offset value $\vec{\mathbf{OffPos}}$. The position calculation for the next frame will utilize the offset value as follows:

$$\begin{aligned} \vec{\mathbf{tl_1}}_x &= x_1 + \vec{\mathbf{OffPos}}_x - \text{offset}_x \\ \vec{\mathbf{tl_1}}_y &= y_1 + \vec{\mathbf{OffPos}}_y - \text{offset}_y \\ \mathbf{width_1} &= AABB_{w_1} + (2 \cdot \text{offset}_x) \\ \mathbf{height_1} &= AABB_{h_1} + (2 \cdot \text{offset}_y) \end{aligned} \tag{5.4}$$

This should result in the correct positional data for all frames following the initial frame. Finally,the resulting rectangle has to be fitted onto the size boundaries of the full image frame. When applying a fixed offset value, the resulting region that will be extracted from the following frame might reach outside the size boundaries of the full frame. This can happen when the markers are positioned near the edges of the frame. Trying to read out pixels outside the available size of the image will result into an error. At these points, no data is available to retrieve. Therefore the combination of position, width, height and offset has to be clamped to fit into the available image data:

$$\begin{aligned} \textbf{width} &= (max(0, min(\mathbf{width}, image_{width})) \\ \textbf{height} &= (max(0, min(\mathbf{height}, image_{height})) \end{aligned} \tag{5.5}$$

Should no marker be found in the defined ROI or the certainty of the result is not high enough, the frame has to be dropped and the next frame should utilize the whole image data.

5.3.2 Image conversion to HSV colorspace

The image data that is supplied by the cameras is delivered in an RGB data format. This format could already be used for the further calculation. It does though make more sense to convert the input data into the HSV colorspace. Since we are not using high precision cameras, it is necessary to define a range of color values around the desired color which we want to track. The HSV colorspace can be displayed as a cone, in comparison to the RGB colorspace, which can be displayed as a cube. The color values for the HSV space are all located on a circle spanning form 0 to 360 degrees. The hue value (H) corresponds to the angle on the circle describing the cone base. A $0°$ angle corresponds to a reddish color,a $120°$ angle lies in the area of green and a $240°$ angle and above correspond to blue.

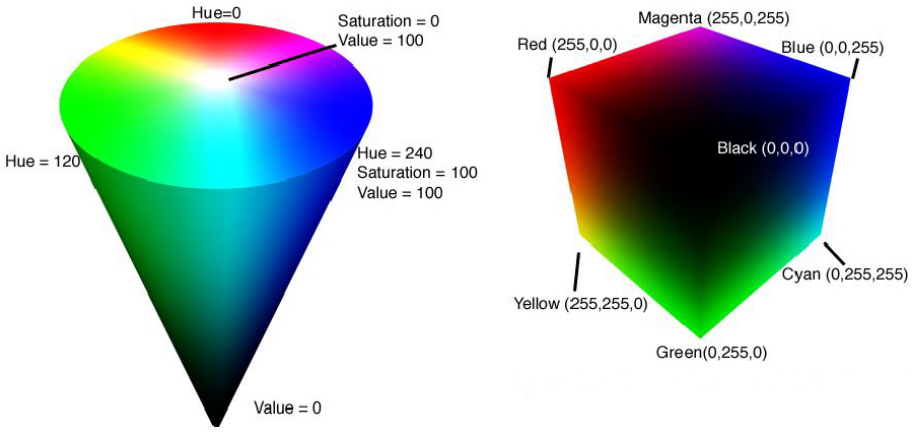

Figure 5.6: Color cone of the HSV colorspace with marked hue (H), saturation (S) and value (v) components. RGB colorspace cube with red, green and blue components [**?**]

The saturation value (S) corresponds to the amount of white the color contains where 0 is pure white and 100 corresponds to the fully saturated color. For optimal results, only highly saturated colors should be used to ensure correct color detection. The last component is the value component (V) which describes the intensity of the color. Alike the value settings for the saturation, value ranges of at least 50 should be used for tracking precision. For tracking the five fingers of the hand, we need five distinguishable colors. Here the primary colors red, green and blue will be the choice for the first three colors. The other two selected colors are orange and yellow.
The color conversion from the input RGB values to the desired HSV colorspace is done as follows:

$$\begin{aligned} hsv_{low} &= (hue_{targetcolor} - sensitivity, 100, 100) \\ hsv_{up} &= (hue_{targetcolor} - sensitivity, 100, 100) \end{aligned} \tag{5.6}$$

To be able to clearly distinguish these colors in the video frame, a constant and homogeneous lighting is needed. A neutral color temperature also helps with color consistency.

5.3.3 Mask construction and filtering

The HSV color converted values are used as the parameters for a binary mask generation on the current frame. In this mask, all pixels who's values lie outside of the specified range are set to zero (black) and the remaining are set to 255 (white). For these masks to work properly, any other larger objects that might contain similar colors has to be removed from the scene to eliminate wrong tracking data.

The used cameras run on an automated white balance and exposure mode. The variation of these values can cause shifts in the appearance of the colored markers and therefore alter the generated masks. To compensate for white balance shifting, a non-white ideally diffuse reflecting background should be used for the tracking space. Also the auto-modes should be turned off. Appropriate values for white balance and exposure have to be determined at the initialization step of the system. As all digital cameras tend to have signal noise in the sensor data, high frequency noise in the color channels will be present. To reduce the base noise level, a good lighting situation should be provided. Low level light situations require a higher signal gain. While this produces a brighter image, it also amplifies the base signal noise of the camera. The remaining noise should be filtered out before any further computation on the image data can be done.

The first step in this progress is to use a *Gaussian filter* to blur the mask. The *Gaussian filter* acts as a low-pass filter for the image. The downside of applying a blur onto the generated image is a loss of detail. As the markers will not be utilizing complex geometric forms or fine patterns, the loss of details is acceptable.
After this step, a combination of two morphological operations, erosion and dilation, is used to further improve the data of the thresholded binary image [**?** , chapter 3.11-12]. Erosion and dilation operation on thresholded images perform the tasks of removing light spots on dark backgrounds as well as dark spots on light backgrounds. These spots are usually the result of the remaining image noise after the blurring or incorrect pixel values from the camera which causes the pixel to be incorrectly thresholded.

To apply the specific operators to the image a mask (often also called kernel) needs to be defined. These masks normally consist of an n x n shaped matrix. The values in the matrix rows and columns are either one or zero for binary images. An odd number of rows and columns is usually used to be able to define a center pixel. The central entry of the matrix corresponds to the current pixel of the image for which the morphological operation is to be applied. As the data of the pixel should be preserved, the value is set to one.

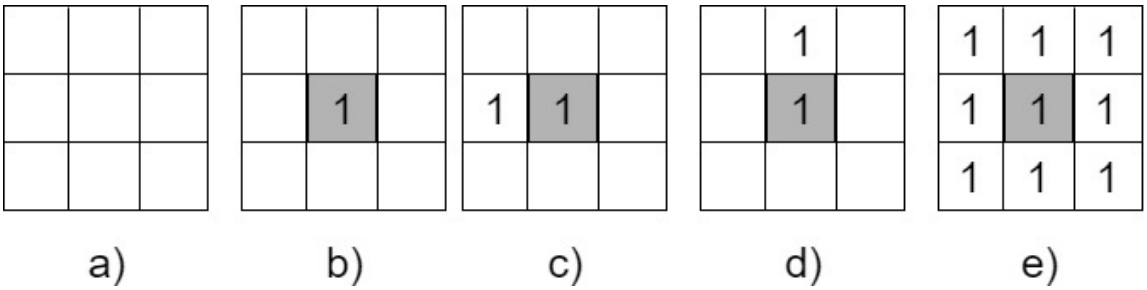

Figure 5.7: Example mask of 3x3 size : **a)** Empty mask **b)** Mask that keeps pixel value **c)** Combination of b) with a left shift **d)** Combination of b) with an up-shift **e)** Isotropic mask

In the further processing, the central pixel value and the values of the neighboring pixels are taken and summed up. The ones in the matrix outside of the center value define if the pixel is used for the operation. A threshold value is then defined. The result of the sum operation is compared to the threshold value. If the result value is above the threshold, the central pixel value is altered, depending on the image operation. If the result value is lower, the current pixel value is kept.

input: threshold $threshVal$ value for the calculation, image segment $imgSegment$ of size $n \times n$, mask $morphMask$ of size $n \times n$, operation value (0 or 1) $opVal$, value of central pixel $centVal$

Result: New value for pixel

```
for i ← 0 to maskHeight do
    for j ← 0 to maskWidth do
        if morphMask[i, j] = 1 then
            result ←result +morphMask[i, j]
        end
    end
end
if result >= threshVal then
    centVal = opVal
end
```

Algorithm 1: Pseudo-code for the the pixel value calculation

When using the isotropic matrix displayed in Figure 5.7 **e)** and setting a threshold of 8 on an dilation operation, the result would be the filling of a white pixel when surrounded by black pixel. The same operation on an erosion would lead to an removal of a black pixel surrounded by white pixels.
The combination of the morphological operator *erode* and *dilate* leads to two methods called *morphological opening* and *morphological closing*. The idea behind these two is to use the two basic operation in a certain order to enhance the binary image quality [**?** , chapter 3.12]. The formal definition for the operation:

$$\begin{aligned} O &= Original\ image \\ \oplus &= Dilation\ operation \\ E &= Erosion\ mask \\ \ominus &= Erosion\ operation \\ D &= Dilation\ mask \\ \circ &= Morphologic\ opening \\ \bullet &= Morphologic\ closing \end{aligned} \tag{5.7}$$

$$O \bullet D = (O \oplus D) \ominus E$$
$$O \circ D = (O \ominus E) \oplus D$$

The *morphological closing* operation is applied first to eliminate so called "salt noise" (white pixels in black area), narrow cracks, channels and small holes in the original image. The major part of the work for this operation is done on the areas where not tracking marker is found. It removes the remaining noise left over from the blur or signal distortion.

The application of the *morphological opening* removes so called "pepper noise", fine hairs and small protrusions. The outcome of the practical application of the opening is the removal of wrong pixels inside the area of the color marker in the image. This helps with the following calculation of the bounding boxes as the algorithm searches for the largest closed area int the image.
Generally, the *morphological closing* of an image enlarges it while filling bright defects inside of an black area. The *morphological opening* makes it smaller while removing bright defects on a dark surface. As the system will not be handling large image data, the mask sizes can be kept low, utilizing the above described 3×3 mask size. The kernel structure options provided by *OpenCV* contain a cross structure, an elliptical structure and a isotropic version. As it would be useful to get all surrounding image data and the tracking markers will not have complex geometric structures, the isotropic mask structure should be sufficient.

5.3.4 Contour finding and position calculation

With the cleaned mask, we can continue and search for the white areas in the mask which might represent our target. Under the assumption that we have removed all other parasitic objects from the image, the marker should be the largest area of positive (white) pixels in the mask frame. Therefore only the largest area found in the search is taken as the desired tracking marker. For the selected area, a fitting bounding box is calculated and the center of this box is used as the positional parameter of the tracking marker.

5.3.5 Object tracking

For the object tracking part, a setup consisting of an *HTC Vive Tracker* and a *Lighthouse Base station* is chosen. Optical tracking via the camera system would be possible, but each object would require a further distinguishable color marker. Each marker would need it's own calibration before the system can start tracking. The HTC system uses infrared signals to measure the position of the marker in space in relation to the position of the lighthouse. These should not interfere with the camera image, as the cameras have a built-in infrared filter. This setup enables the system to quickly change the tracked object used without the need for re-calibration. Only the tracker position in relation to the hand tracking coordinate system has to be set in the initial start-up phase.

5.4 Systems communication

The data that is generated by the two Raspberry's has to be communicated to the "Master" unit where it is further processed. The data transport time delay between the different endpoints should be minimal. As the data is streamed in real time, protocols like TCP, which utilize control mechanisms for package order and receiving of data, use to much overhead. After the time-frame needed for the protocol to request a missing package again, the data may already be obsolete.
Furthermore, the amount of data that is send in each data frame is small. It only contains the position value of the specified number of tracking points and some separators for string processing. Since the devices will be connected to a local network via a switch, network workload from other devices and protocols can be assumed minimal and not impacting transport time.

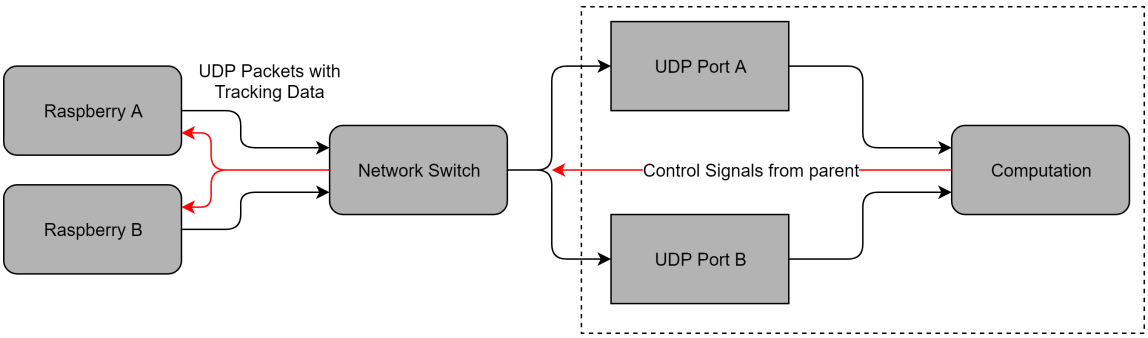

Figure 5.8: Work-flow for the UDP Data packages that are send from the Raspberry's to the parent PC

Regarding all these points, a simple UPD protocol setup seems to be the best choice. Package loss that might appear with UDP is not critical as it results in only the loss of one frame data-set.
It has to be ensured that the UDP data streams from both devices are recorded separately to ensure that the data can be separated correctly for the stereoscopic calculations. The possibility to send multi- and/or broadcast messages via UDP also opens up an easy way of control communication between the "Master" and the "Slave" Pi's.

5.5 Data processing on the receiver side

To ensure that the two data streams are separated and therefore distinguishable, each stream receives a dedicated UDP listening port on the "Master" side (Figure 5.8). This makes it possible to separate the UDP listeners to separate running threads in the background. The positional data will be encoded as comma-separated x and y vectors. Data separation is done by a simple semicolon as a separator between the vectors. After the string decoding, the extracted values are written into thread-safe variables. This allows the system to read current data without conflicting the writing process of the UPD threads.

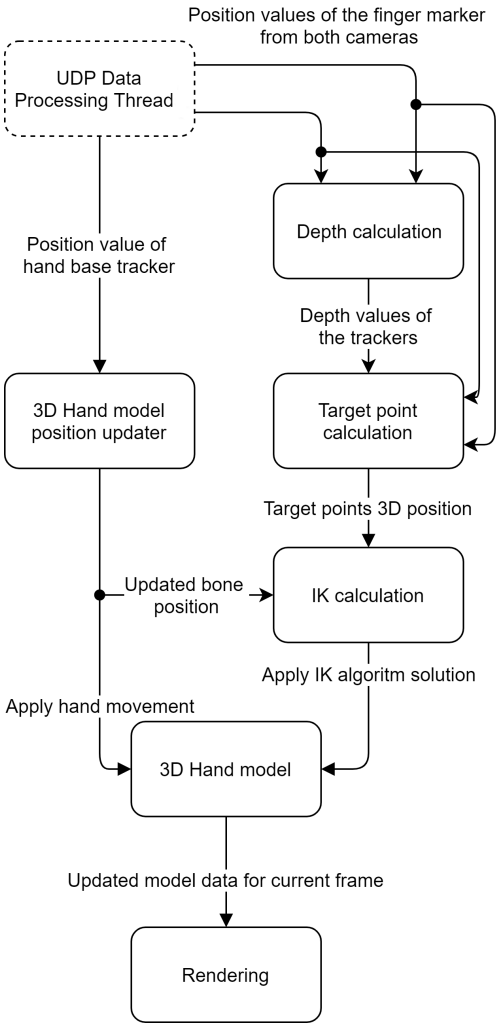

Figure 5.9: Overview diagram of the processing steps for the hand model data on the "Master" side

5.5.1 Stereoscopic calculations

The positional data that is extracted in the UDP data processing only consists of 2D position data. To retrieve depth information for correct spatial placement of the marker, the image disparity of the two cameras can be utilized to calculate the object distance from the cameras [**?**].

Before diving into the mathematical concepts, some essential terms have to be explained. These are:

- Inter-axial distance
- Parallax
- Zero-point plane

Inter-axial distance

Stereoscopic imaging systems are based off of our own human visual system. Therefore, such a system always consists of at leas two cameras. The distance difference between the two cameras is called the *"inter-axial distance"* of the system. The usual distance for such systems is the mean distance of the human eyes. This corresponds to around 75 mm. Shifting the cameras closer together or further apart will change the resulting stereoscopic effect. Shifting the cameras closer together will enlarge objects. Shifting the cameras further apart will make objects seem much smaller than they are.

Parallax

Through the separation distance of the cameras, a *disparity* in the generated images will occur. This means that the objects in the picture of the right camera will have a different position in terms of perspective than the images that result from the left camera. This resulting disparity is used to generate the 3Deffect when displaying the images. The term "*parallax*" is used to describe the comparison of the two images. The two images can have a negative, positive or zero *parallax*.

Zero-point plane

The *Zero-point plane* is a special theoretical plane in a stereoscopic system. The location of the *Zero-point plane* defines the distance from the camera system at which the image points from both cameras coincide. As Figure 5.10 shows, objects that lie before the *zero-point plane*(closer to the camera) will have a negative *parallax*. They will appear closer to the viewer. On the opposite side, objects with positive *parallax* will lie behind the zero point plane(further away from the camera). They will appear further away.

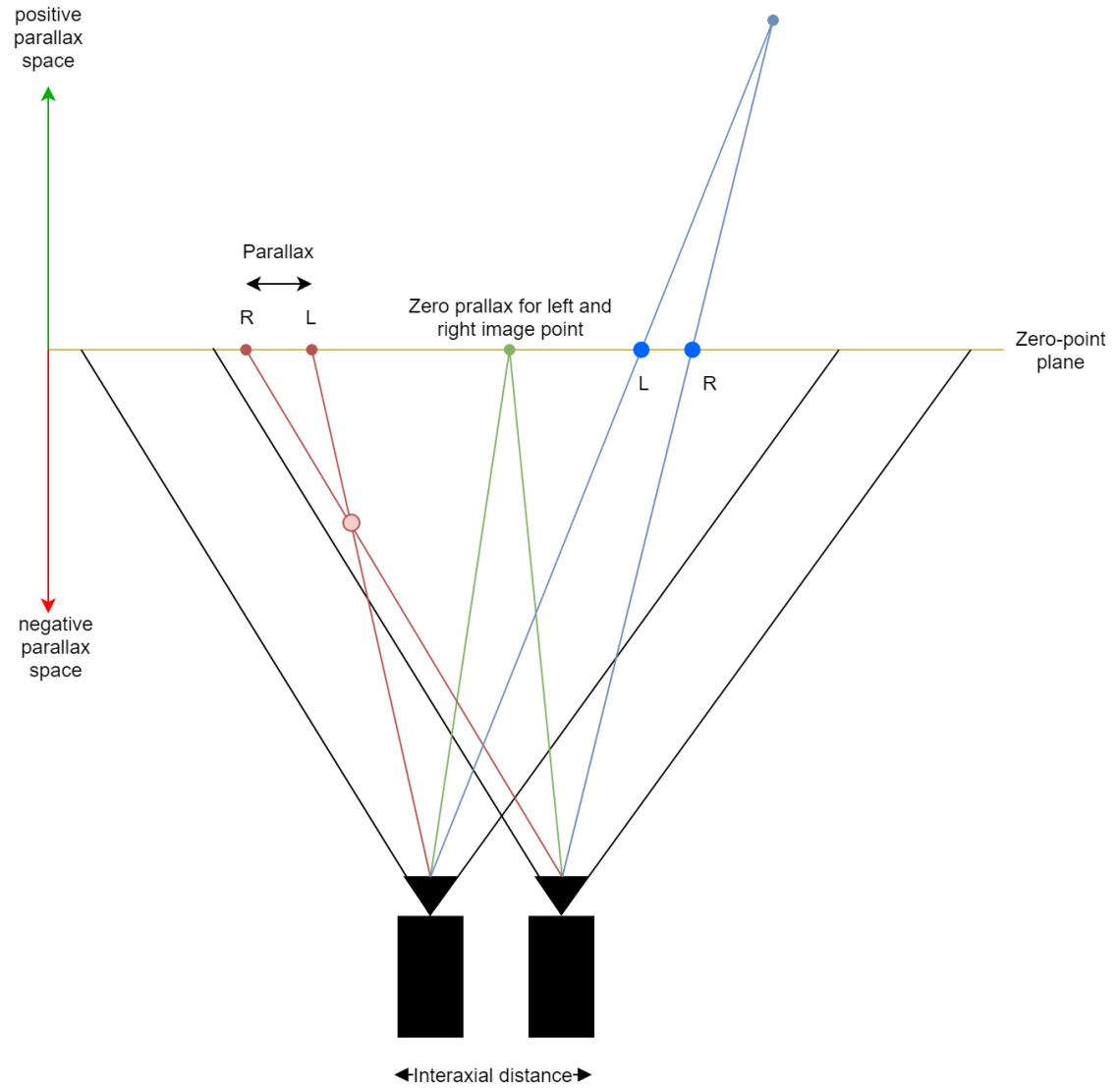

Figure 5.10: Two camera setup with the marked stereoscopic areas

When setting up parallel stereoscopic setups, additional attention needs to be put in the camera placement before calculating depth form disparity. Horizontal misalignment of the cameras can lead to incorrect calculation results when calculating depth from disparity.

5.5.2 Depth calculation

With a correctly aligned system, we are able to calculate the distance of an object from the cameras using common trigonometric fundamentals [**? ?**].
The position for the right camera will be denoted as S_R and the position of the left camera will be denoted as S_L. The resulting distance of $S_L - S_R$ is the *inter-axial distance* B of the system. The view angle θ of the camera, which should be the same for both cameras, will respectively be θ_1 and θ_2. From the camera positioning, the assumption of parallel optical axes is made. The angles of ϕ_l, ϕ_r describe the resulting angle between the optical axis of the camera and the object. When looking at Figure5.11a), we can express that:

$$B = B_1 + B_2 = D \tan \varphi_1 + D \tan \varphi_2 \tag{5.8}$$

Rearranging for D gives us:

$$D = \frac{B}{\tan\varphi_1 + \tan\varphi_2} \tag{5.9}$$

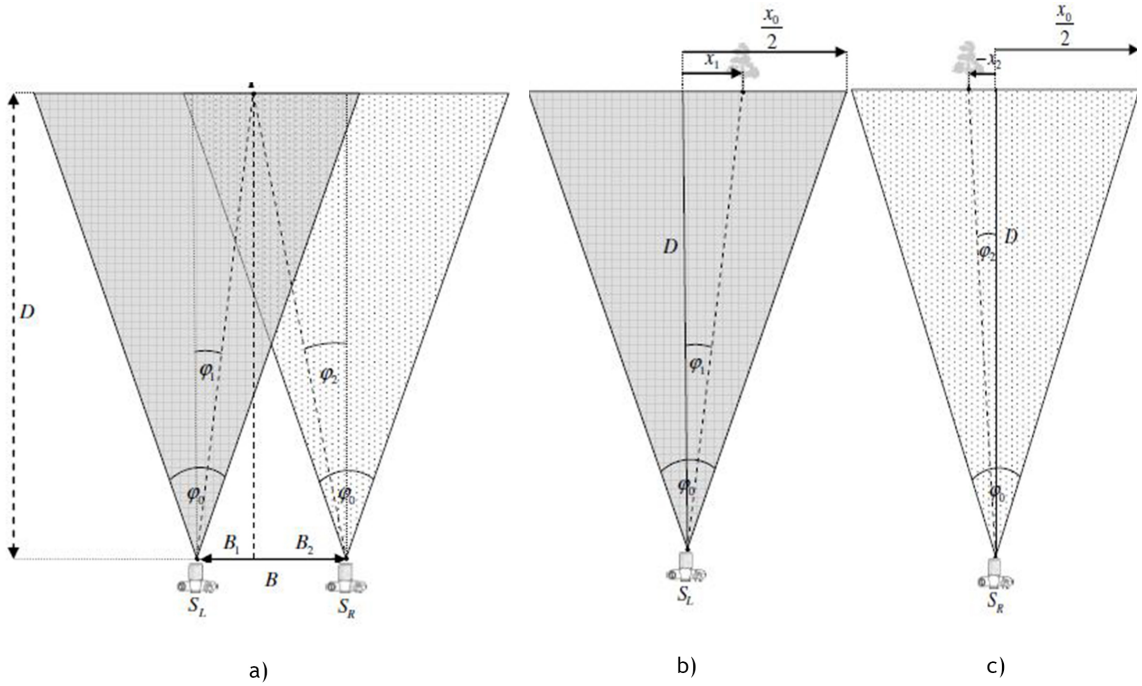

Figure 5.11: Camera viewing angles and distance values for calculation [**?**].

With Figure 5.11b)-c) one can find that:

$$\frac{x_1}{\frac{x_0}{2}} = \frac{tan\varphi_1}{\tan(\frac{\varphi_0}{2})} \tag{5.10}$$

$$\frac{-x_2}{\frac{x_0}{2}} = \frac{tan\varphi_2}{\tan(\frac{\varphi_0}{2})} \tag{5.11}$$

Resulting in the calculation of D to be expressible as:

$$D = \frac{Bx_0}{2\tan(\frac{\varphi_0}{2})(X_L - X_R)} \tag{5.12}$$

Where B is the distance between the cameras. x_0 corresponds to the horizontal width of the picture format in pixels. φ_0 represents the viewing angle of the camera and $(X_L - X_R)$ represent the disparity between the two images in pixels.

5.5.3 Model position update

Before a solution with the *Inverse Kinematics* algorithm can be calculated, the hand movement in real space has to be matched with that on the digital side. To achieve this, a further marker besides the finger markings is required. A sixth color marker has to be positioned on top of the hand to serve as a global position reference. The position difference between the last and current frame can be calculated from the tracking values and then be applied to the whole model.

5.5.4 Inverse Kinematics

For the inverse kinematics calculation, the *FABRIK* algorithm that is described in Section 2.2.7 was selected. It does not require complex matrix calculations and has a fast convergence rate. For the implementation of the algorithm on the "Master" side, the *Caliko* Java framework [**?**] is used. The usage of a Java framework provides a operating-system independent implementation of the solution. The library supplies all the needed classes and function to create *kinematic chains* and *structures* as well as the implementation of the algorithm itself. It furthermore ships a visualization package which utilizes the *Lightweight Java Game Library 3* (LWJG3). The provided classes can be used optionally for debug visualization or even as a base for a more sophisticated visualization as it is possible to load further components for i.e. VR into LWJGL.

For the inverse kinematics calculation to work properly, the updated bone positions and the calculated target points from the finger tracking are needed. The length of each bone for the hand model has to be measured manually for the first prototype. Further iterations should provide a more elaborated and maybe even automated calibration procedure. The movement of each bone in the kinematic chains has to be restricted by constraints to restrict the algorithms solutions to physiological possible hand poses. The values for finger constraints that are described in Section 2.1 should provide sufficient starting values.

6 System Evaluation

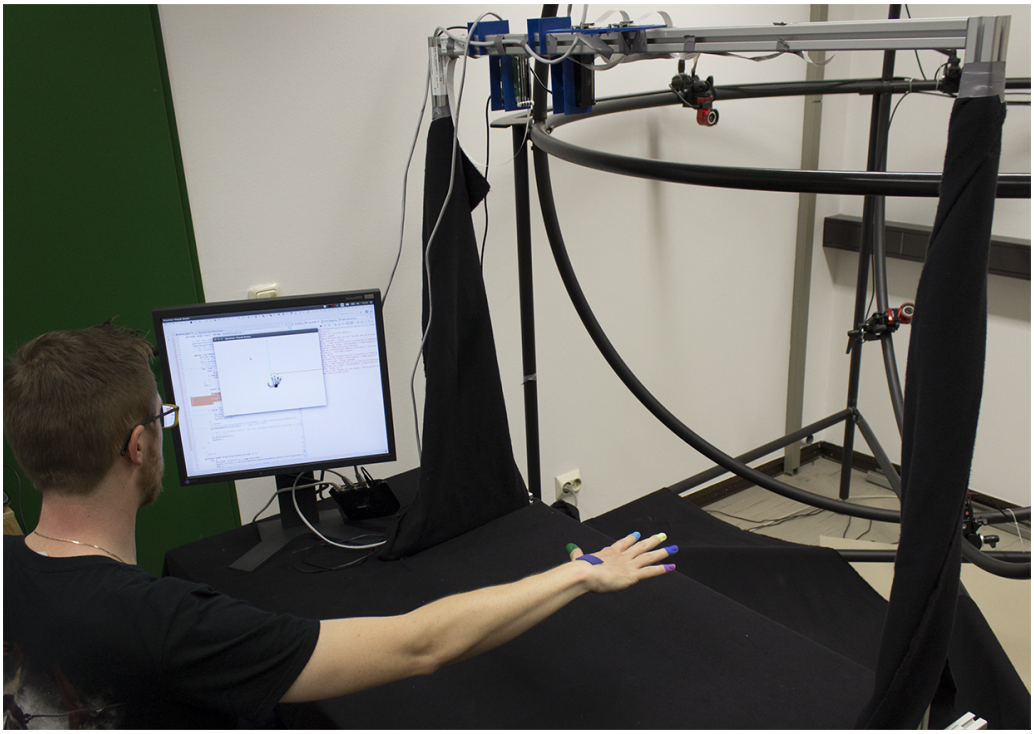

Figure 6.1: Final system setup with cameras on construction profile, color markers and visualization.

6.1 Camera hardware and control

The Raspberry Pi image processing of the camera frames can be a significant bottleneck for the system. "Real-time" evaluation demands the processing time of a frame to be in the range of the time of the camera FPS to utilize all available frames. With the current system setup limited to 30 fps by the camera framework, the image processing has to be done in about $1/30s$ or 33 ms to achieve "real-time" processing. Professional stereoscopic setups use cameras with global shutters and a synchronization signal. This enables a sub-frame synchronization of the two camera systems. The cameras used in the prototype setup use a rolling shutter instead. Furthermore, these cameras are designed to be of easy use for the normal consumer, therefore their design does not incorporate a synchronization feature. To achieve a form of hardware synchronization, a large amount of reverse engineering would be needed as the electrical drawings as well as the controlling software are closed source. The multi-layered circuit board of the camera would also complicate the reverse engineering step. For these reasons, attempting the synchronization of the data further down the processing pipeline was more reasonable.
To synchronize the frame reading of both cameras, the system boots up to the point where all needed components are initialized. At this point, the program waits for a start message from the "Master" system via *UDP*. The "Master" system send the start message at the end of its initialization phase via a *UDP* broadcast.This ensures that both Raspberry's get the message at the same time. In normal usage mode, the camera uses

automatic modes for it's parameter setup. This continuous measuring mode produced fluctuation in image brightness and color saturation. To eliminate this problem, all automatic modes were turned off in the camera initialization phase. Fixed values are loaded from a *JSON* file instead. To get the values for the *JSON* file, a calibration step was implemented. This calibration step enables the user to preview and manipulate the following values for the camera:

- Saturation
- Brightness
- Gain
- Exposure time
- Contrast
- White balance values for blue and red channel

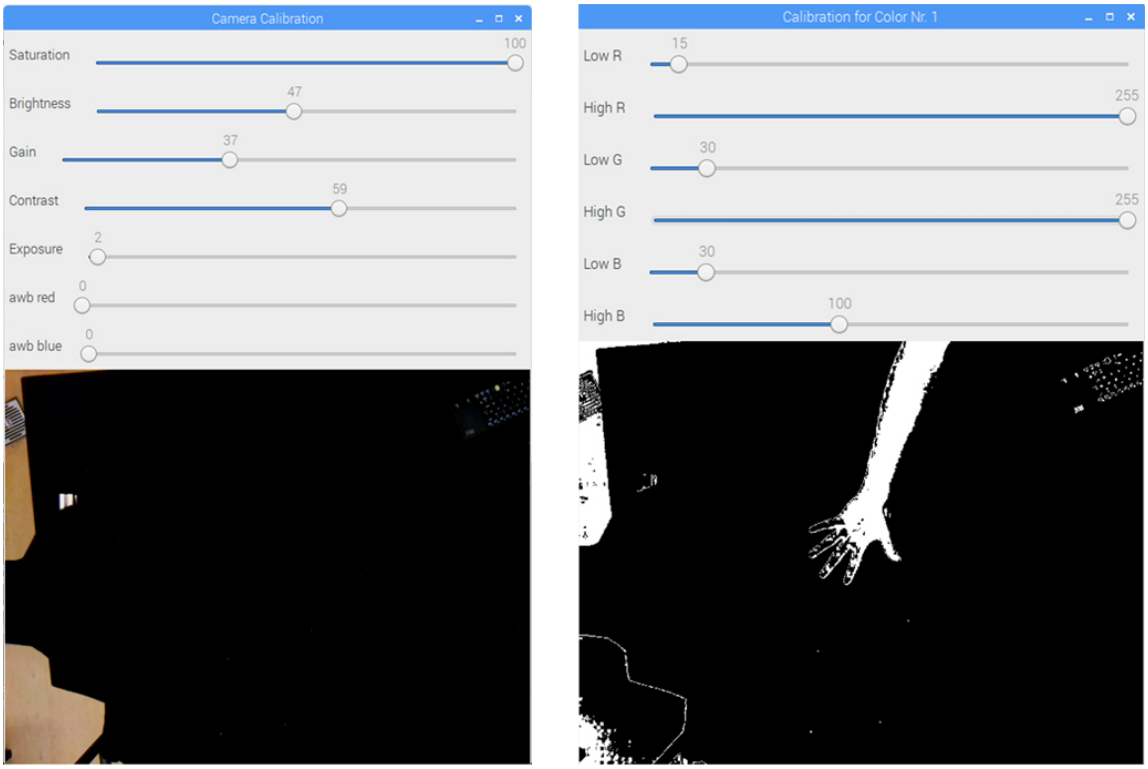

Figure 6.2: Camera and color calibration utility window

The user-set calibration values can be written into a new *JSON* file. They are directly loaded into the program after calibration is finished. Control over these parameters showed to be crucial for achieving a constant color separation. The threshold values for the color tracking also needed to be initially calibrated. These values are dependent on the lighting situation and any change results in the need for re-calibration. To simplify this calibration, a calibration functionality in line with the camera calibration was implemented. The implemented feature provides the user with a preview of the resulting thresholded black and white image with applied camera settings. Range sliders for lower and upper boundaries for the three RGB components make it fairly easy to adjust and view thresholding results in "real-time". The calibration values can also be stored into a *JSON* file. This file can then be read out on following program start-ups.

6.2 Image processing performance

The first prototype implementation with *OpenCV* in *Python* showed, that a speed of 33 ms was achievable when only tracking one color. The implementation reached around 30 - 40 ms processing time when scanning whole images. An implementation of an *"Region of interest"* feature reduced the processing time to around 30 ms for a single color. The downside of the implementation was revealed when implementing the other 4 colors needed to do a full five finger tracking. This approach ramped-up processing time to around 100 ms per frame, making this solution run at 10 FPS.

The sequential algorithm also showed another design flaw. When not utilizing the ROI approach, the algorithm would scan the whole grabbed camera frame for each color to create the corresponding threshold map. The *OpenCV* color threshold methods are designed to only search for one color threshold at the time. They therefore need this procedure. One solution option is the already mentioned data reduction through ROI usage. This solution could furthermore benefit from a multi-threaded implementation. The original image data is only read. Parallel memory readout is not a problem and the generated mask can be saved separately.

Another approach which could help speed up the process is implementing an own method for thresholding the grabbed frame. This would result in the possibility to do all the needed color thresholds in one image loop. With this solution, the overhead would be reduced by 4 image iterations. The thresholding operations would stay the same as these still have to be run on each pixel. The output of this method would then return the 5 needed threshold mask for further processing.

A second performance issue that arose from the first prototype was that the input camera frames are RGB coded. For better options of color separations, the image was initially intended to be converted into HSV colorspace. This operation turned out to be second longest operation after frame grabbing.
Testing of the cameras indicated that the color separation in the original RGB images of the cameras is good enough for the used test colors. This caused this step to be omitted and shaved off several milliseconds of processing time. The rest of the processing steps are performed in the sub-millisecond range and are therefore not as valuable for performance optimization.
The results from this prototype indicated, that the *OpenCV-Python* combination is not suitable for reaching the 30 ms target time. The OpenCV library used by python is actually just the ported version of the C code library with bindings for python. As *C* and *C++* are more hardware near and therefore more performant, a second prototype implementation was done in *C++*.

The recreated workflow of the Python code in *C++* initially showed similar performance measurements when using a sequential approach. This was to be expected, as it is the same code the Python bindings are using. Further investigation into code timings showed, that another bottleneck was the image optimization feature which applied erosion and dilation operations to remove high frequency noise in the image. This operation brought the time up to 100 ms processing time when processing the whole frame area.This makes the implementation not usable for "real-time" application. Removing this feature when scanning full frames resulted in a major speed up of processing time.

The implementation of a thread pool, which handles all the color detection jobs, brought down the calculation time for 5 colors to around 120-140 ms. Although these times are still not usable for real time, it brought a significant performance boost. Figure 6.3 shows the initial structure design of the program with a camera frame buffer and the thread pool approach. The cyclic buffer was intended to hold images frames for readout where camera frame rates are much higher than the achieved processing rates of the source code.Since the frame-rates of the current setup are slower than the time spend on processing, the image-buffer for the camera is not needed. The feature is left in the code but is not actively used at the moment.

Since the thread pool approach still took too long, a test with lower resolution than 640x480 was done and showed that the calculation time can be reduced. This indicated that the ROI idea would be feasible for further performance optimization. A test at a frame resolution of 320x240 pixels reduced the processing time for all five colors to below 10 ms.
Utilization of the knowledge from these tests led to the implementation of the aforementioned *"Region of interest"* feature. The first frame is analyzed as a whole frame, optimally resulting in the detection of a marker. As the marker detection creates a rectangle around the tracked area, we already have the coordinates for the ROI and only have to add an offset to this area.
The value of the ROI offset has to be set high enough to ensure that in the following frame, the tracking marker is still contained. It also has to be ensured, that the generated ROI is clamped to the frame dimensions. If not doing so, parts of the readout are can be located outside of the image plane, causing readout errors when used for the subsequent frame.
The following frames will then be processed with the input of the ROI, selecting only the defined section of the image and updating the ROI with the new data for that frame. The *C++* implementation brought down the processing time as assumed. This made it possible to apply the image improvement operations that were left out on the python implementation.
Performance measurement for tracking consistency with stationary color trackers was performed for a time period of 10.000 frames to get qualitative results. A sequential implementation with the usage of the ROI and image optimization was used for better analysis options and showed an average processing time of 46 ms with a standard deviation of 11 ms.

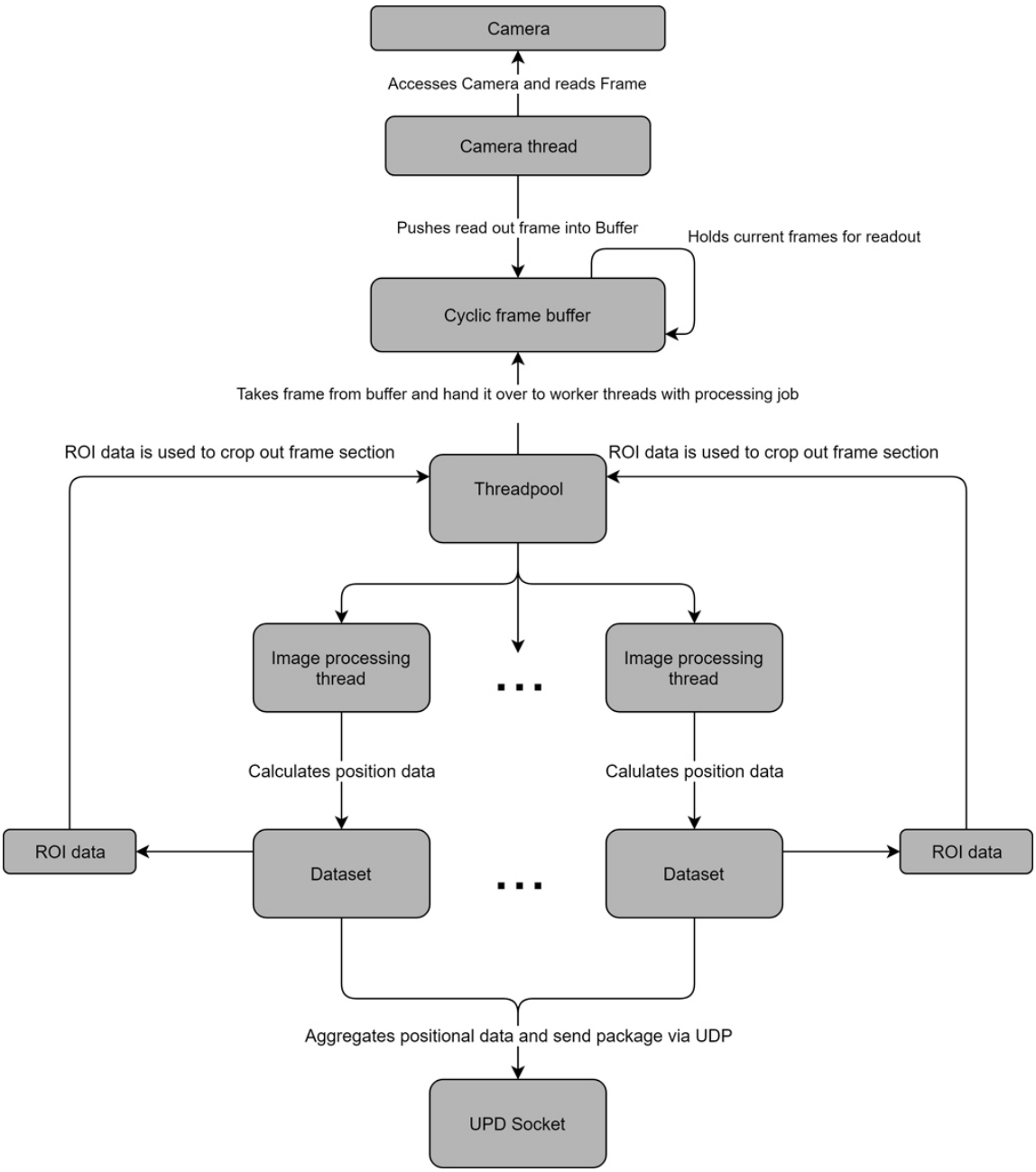

Figure 6.3: Intended work-flow of the C++ implementation with thread pool solution and ROI for high frame-rate solutions

Further analysis showed that the high standard deviation came from peaks in performance times that took up to 400 ms when loosing tracking.
Filtering all values over 50 ms out from the data set showed that only 1% (~ 1000 out of 10.000 frames) of the recorded processing times were lying outside of this time frame. Filtering out these values form the data set and recalculation dropped the average processing time down 3 ms to around 43 ms. The standard deviation was dropped drastically from its previous 11 ms to around 1.6 ms. This shows that a filtering of processing times on the Raspberry could lead to a more consistent frame processing time. To do so, after every processing cycle the time should be checked. If it is over the defined time limit, no data should be sent.
Since this performance measurement does not give deeper insights on which part of the process consumes large amounts of processing time, the same measurement was done with a finer time tracking on the single parts of the processing pipeline.

Table 6.1: Measurement results for sequential processing timing

Time in ms	Image retrieval	Stereo rectification	Color detection (6 colors)	UDP sending	Summed time
Average	0.92	31.39	14.55	0.12	46.97
Std.Deviation	0.17	7.19	5.06	0.06	10.77

Table 6.1 highlights the performance bottlenecks of the system. Image retrieval from the camera as well as the time to process and send the data results via UDP are in the sub millisecond range. Therefore they impose no significant delay. Without these two steps, the remaining 96% of the processing time is taken up by the image stereo rectification and the color tracking. When comparing these two, the far larger time consuming part is taken up by the stereo rectification part with nearly two thirds of the processing time.
The use of the stereo rectification in the system was done to ensure that the two used images have an aligned horizontal orientation. This is needed for camera pairs that are not aligned horizontally and/or parallel. Since the camera mounting ensures that the cameras are aligned parallel and at the same horizontal height, the rectification step can be omitted for performance reasons. It cuts down 67% of the processing time of the system. As the stereo calculation does not use the vertical position difference for calculations, the possible resulting differences can be dealt with via filtering on the receiver side. A solution for the color tracking time improvement was already described before. Utilizing a multi-threaded approach could lead to further reduction of the processing time.

A comparison between the before mentioned sequential algorithm with ROI and a multi-threaded implementation with ROI was tested as well. The multi-threaded implementation was designed to run the same processing stack as one loop of the sequential approach on each thread. Measurements over 10.000 frames for stationary as well as moving targets were made. One camera was run with the sequential implementation, while the second one ran the multi-threading approach. Thereby it is ensured that both cameras have the same measurement conditions.
The comparison of the measurements showed that in the current implementation state, the multi-threaded approach seems to have a much longer processing time. Table 6.2

Table 6.2: Measurement values of implementation comparison

Measurement	Average (ms)	Std. Dev (ms)
Single-thread static	16.96	4.03
Multi-thread static	32.09	10.64
Single-thread dynamic	19.41	22.44
Multi-thread dynamic	35.80	22.88

shows that the measured time are roughly twice as long as the code running on a single thread. This behavior may result from an design or implementation fault in the multi-thread code and should be evaluated in future works to improve processing performance of the system. For the current state of the prototype, the single-threaded implementation with it's average processing time of around 19 ms is sufficient.

6.2.1 Color accuracy and ROI size

As already mentioned in the conceptional phase, the lighting of the tracking area is a major factor. Variations of lighting intensity and color temperature cause the color of the trackers to shift their color. This can cause a fluctuation in the calculated tracker positions, as the defined color threshold ranges are kept as small as possible. This was done to achieve a clean separation of the tracker colors and reduce unwanted noise.

System testing with the 5 different colored shrink tubing markers proved, that the color tracking for colors outside of the color range of the human skin tones (green,blue) is rather unproblematic. The color markers falling into the colorspace of possible light reflection from skin may cause problems, depending on the lighting situation of the setup.

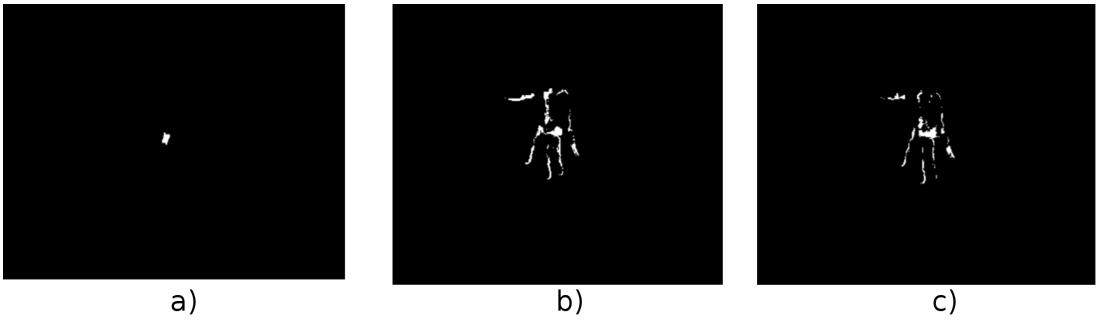

Figure 6.4: Images showing reflection problems with orange color thresholding. a) Correct threshold with only the marker being thresholded b-c) Depending on the rotation of the hand, skin reflection color falls into tracking threshold.

Figure 6.4 displays such a case where depending on the orientation of the hand, reflection from the scenery lighting can fall into the color tone space of the tracked marker. This can cause large parts of the hand to light up in the image. Since the calculation of the tracking marker position relies on finding the largest closed "white" area in the thresholded image, this can cause a "jumping" of the calculated position.

The first solution for this problem is to narrow down the color threshold spans. By doing so, one can ensure that only the wanted colors are tracked. This method does have a downside effect. When the markers are moved in the tracking space, light variations and shadowing will cause the colors to shift their appearance slightly. With broader threshold ranges, this does not seem to be a large problem as these changes still lie inside the ranges. Narrowed down color ranges might not include these color variations. It could also create cavities in the thresholded image.
The applied morphological operations on the image can fix these problems up to a certain point, depending on filter kernel size, but for larger areas this could vary position tracking results.

The implemented ROI feature speed up the whole system calculations once the colors are found. Before this point, the system scans whole frames to find colors, which takes more time than the much smaller ROI regions. Empirical evaluations showed, that for the used tracking space, a ROI region size offset of 40 px in x and y directions produces the best results in terms of consistency. Smaller region offsets caused the system to use tracking for faster hand motion, which causes system slow down until the tracking has recovered. Larger ROI's showed to produce more consistent tracking results at the cost of higher calculation time.

6.2.2 Finger markers

The initial color marker size was selected to be relatively large, spanning most of the last segment size of each finger. The used material of the colored shrink tubing was selected because it seemed to be relatively diffuse and therefore unsusceptible to errors resulting from the reflection of the scene lighting. The material turned out to be mostly diffuse reflecting, but depending on color and angle of light incidence, reflections in the color of the scene light did occur. A test-set of smaller markers was also evaluated, showing the same color reflection problems.
The benefit of a smaller marker size would be much lower restriction of the finger movements and also unveiling the fingertips for better haptic interaction.

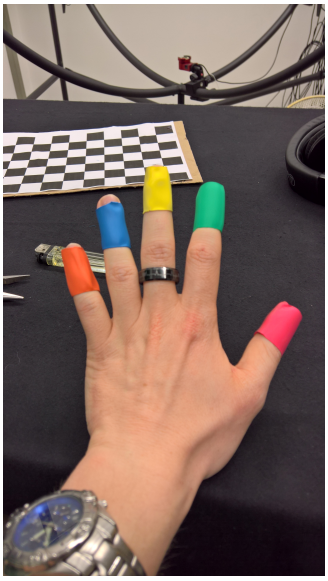

Figure 6.5: First set of shrink tube color markers

As both markers showed the above described problem of mixing marker area with possible skin reflections, an improvement to the markers was applied. At both ends of the larger markers, a piece of gray colored tape was applied to create an artificial border for the tracking algorithm.

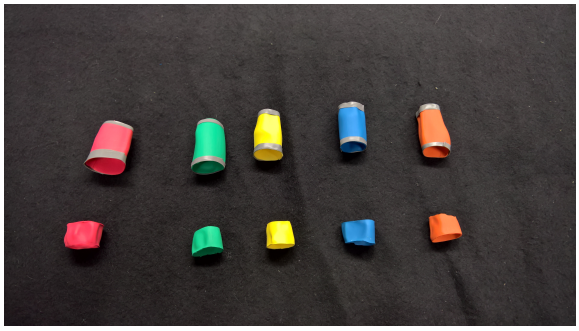

Figure 6.6: Improved larger and smaller set of shrink tube color markers

The idea behind this solution was to create a fixed border between the color area of the tracker and the color area of the skin tones. The grey color of the border ensures that this area will not be tracked. The usage of black instead of gray would even furthermore ensure this.
Since the threshold algorithm searches for the largest closed color area in the binary threshold image, this border should cause a separation of unwanted skin reflection area and actual tracker color area. The problem that the resulting color marker area might not be the largest area still persists after this improvement, but this case has to be handled separately.

Optical results showed that for the colors lying outside of the red to orange color spectrum that skin reflections produce, the artificial border stabilized the color detection.
The idea of using shrink tubing as marker material showed to be suitable except in the cases described above. One downside of the material is that the displayed colors are already the highest variety of available colors on the market. Other colors from the green

and blue color areas like purple are simply not available or have to be ordered as a special request in larger amounts.

The original diameter is much larger than a human finger. The used shrink tube is able to be shrunken to one third of its size. The reason for this was to be able to adapt to the varying diameters of the human fingers. The shrinking procedure requires a constant amount of heat to be applied to the material to activate shrinking. The amount of heat that is needed surpasses the heat production of a standard hair dryer. The original application of shrink tubing is on heat resistant cables. The heat that needs to be applied can easily be more than 100 degree Celsius, which makes a direct application on the users hand highly insecure.

For the prototype, the shrink tube parts were shrunk with a standard cigarette lighter and continuously fitted onto the user finger until the desired form was reached. This method showed to be sub-optimal, since the flame of the lighter produces only a punctual heat source. This caused uneven shrinkage of the heated parts. This can lead to cavities or bumps in the material. It also changes the reflection properties at these points. A regulated heat gun would probably generate better results.
To get smoother shrinking results, parts of PVC tubing with the correct diameter could be used as dummy parts for the main part of the shrinking.

Another approach that was tested was the usage of matte acrylic paint to cover parts of the finger. The paint comes with the benefit of being easily to apply to the finger. Acrylic paint is available in much more color variations than the shrink tube. The finger coating is dry in under a minute after application. Adaption for finger size is automatically included in the application process. If the tracking system does not need to evaluate for hand rotation, i.e. the working principle is a top on view, a nail polish style application of the color on the fingernail can be sufficient for the tracking system. The thumb needs to be treated separately, as it has more rotation capabilities than the other fingers. It should be marked completely on all sides to achieve a constant tracking. If hand rotation is tracked, the other fingers should be completely marked as well.

Figure 6.7: Finger marked with suitable acrylic colors

As the final marker material, a mixture of blue and green color tones together with red tones in the purple and magenta section based on the acrylic color was chosen.

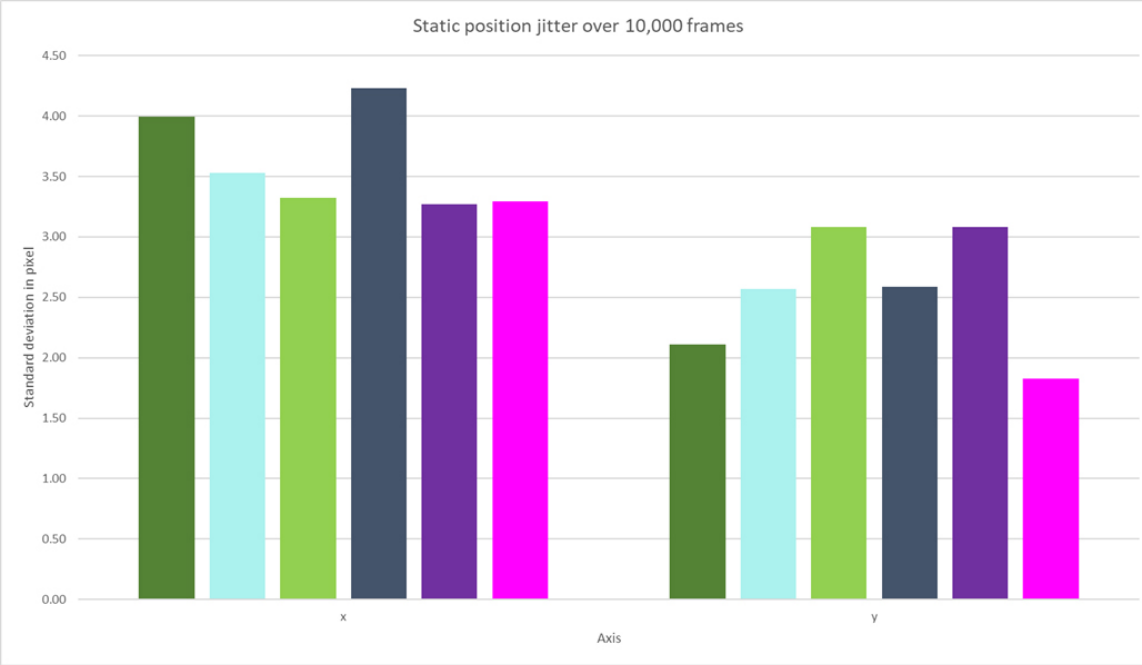

Figure 6.8: Bar chart showing position jitter of final colors in static position

The usage of the acrylic color also provides the least degradation in terms of haptics on the fingers. Figure 6.8 shows the measured static position jitter of the system with the final colors. The reachable system precision lies between 2 and 5 pixel deviation. At a system resolution of 640x480 pixels, this results in a deviation of $\sim 0.7\%$ on the x axis and $\sim 1.0\%$ deviation on the y axis.

6.2.3 Depth measurement accuracy

To determine the accuracy of the system, the camera rig was aligned horizontally and fixed to a test bench. A large sized colored marker (75 mm x 50 mm) was used as tracking target. The marker was positioned at altering distances from the camera rig along the central axis. The height at which the camera rig is positioned in the experiment setup will be around 100 cm, so the measured distances started at 100 cm from the camera and were decremented in steps of 5 cm until 20 cm in front of the camera rig (see Table 6.3 and Figure 6.9).

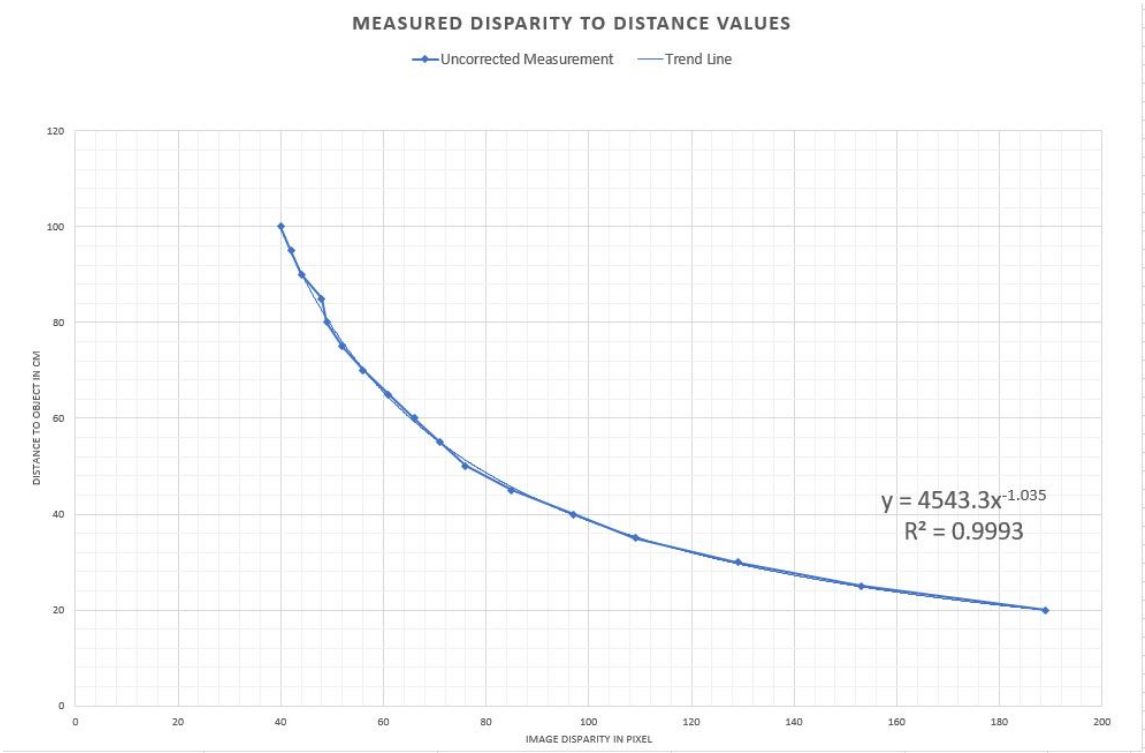

Figure 6.9: Chart showing the results of the disparity measurement with a plotted trend line

The accuracy of the camera system is limited by the number of pixels in relation to the camera view angle [**?**]:

$$\Delta\varphi = \frac{\varphi_0}{x_0} \tag{6.1}$$

For the system setup of $x_0 = 640$ px image width and a horizontal view angle of $\varphi_0 = 62.5°$, $\Delta\varphi$ is equal to $0.0977\frac{1°}{pixel}$.

The resulting distance error would be:

$$\frac{\tan\varphi}{\tan(\varphi - \Delta\varphi)} = \frac{\Delta D + D}{D} \tag{6.2}$$

$$\Delta D = \frac{D^2}{B} tan(\Delta\varphi) \tag{6.3}$$

The system error for a target distance $D = 100cm$ would therefore result in about $2cm$

of possible error. As the measured values show deviations of up to 5% from the correct value, the function from the fitted trend line displayed in Figure 6.9 was taken as a correcting factor for the depth calculation.

Equation 6.3 uses the uncorrected values for the calculation. Values lying on the trend line of Figure 6.9 should correspond more to the correct distance values [**?**]. The equation can be rewritten to:

$$D = k * x^z \tag{6.4}$$

with K being:

$$k = \frac{Bx_0}{2\tan(\frac{\varphi_0}{2} + \phi)} \tag{6.5}$$

and *x* the disparity in pixels. The ϕ term in the equation above is used as a compensation for possible alignment errors. The trend line, which is fit to the measurements in Figure 6.9 represents the the function needed to fulfill Equation 6.4. The calculated values are $k = 4543.3$ and $z = -1.035$. With the utilization of the corrected value formula, the accuracy of the depth measurement results is acceptable for the prototype application. For future work, the distance measurement procedure could be applied with more measurement points to refine the resulting function for more precision. The measurement values that were retrieved can be found in Table 6.3.

Table 6.3: Depth measurement values

Real distance from camera (cm)	Left camera x-value	Right camera x-value	Disparity in pixels	Calculated distance in cm	Deviation in %(abs.)	Corrected values	Deviation in %(abs.)
100	302	342	40	99.463	0.54	99.670	0.33
95	301	343	42	94.727	0.29	94.760	0.25
90	300	344	44	90.421	0.47	90.304	0.34
85	298	346	48	82.886	2.49	82.523	-2.91
80	296	345	49	81.194	1.49	80.780	0.98
75	296	348	52	76.510	2.01	75.960	1.28
70	293	349	56	71.045	1.49	70.349	0.5
65	289	350	61	65.222	0.34	64.388	0.94
60	289	355	66	60.281	0.47	59.344	1.09
55	286	357	71	56.036	1.88	55.022	0.04
50	285	361	76	52.349	4.7	51.279	2.56
45	280	365	85	46.806	4.01	45.668	1.48
40	265	362	97	41.016	2.54	39.831	0.42
35	258	367	109	36.5	4.29	35.300	0.86
30	244	373	129	30.841	2.80	29.650	1.17
25	288	381	153	26.003	4.01	24.848	0.61
20	206	395	189	21.050	5.25	19.965	0.17

6.3 Data filtering

The finger marker base jitter measurements showed that the systems can only supply a certain stability of position tracking. This results in the fluctuation of the measurement results. When rendering this unfiltered data directly, a large optical jitter in all positions is the case. Jittering of position values also causes the height calculation algorithm to generate incorrect height values.
To counter this data jitter, the incoming data-set is filtered on the "Master" in several steps with the *1 euro filter* presented by Casiez et al. [**?**]. It uses a first order low-pass filter with an adaptive cutoff frequency to filter noisy signals for high precision and responsiveness. This means that at low speeds, a low cutoff reduces jitter at the expense of lag. At high speeds, the cutoff is increased to reduce lag rather than jitter.
The filter was chosen, beacuase of a relative simple and easy implementation as well as an uncomplicated setup and tuning process. It also produces faster and better results in comparsion to the normally used *Kalman filter* [**?**], *moving average filter* or *low-pass filters and exponential smoothing filters* [**?**].
The filter needs three base values to function correctly. The first one is the frequency at which the data values are delivered. This can easily be measured from the time stamp differences of the incoming data packages. The other two values are the minimum cutoff frequency $f_{c_{min}}$ of the low pass filter and the slope of the filter curve β.
These values have to be calibrated in a manual process for each marker. In the first step the value of $f_{c_{min}}$ is set to one and β is set to zero. The markers are held in a static position or moved very slowly to check for jitter. The value of $f_{c_{min}}$ is then adapted to a point where the jitter is minimized without creating to much lag for these slow movements. In the second step, the tracker is moved fast and the β value is increased until lag for fast movements is minimized.

In a first step, the incoming data for each finger and the base marker are filtered separately and component-wise. This allows for specific fine tuning of the filter values, as each marker shows different jitter. The incoming values are also passed to the depth calculation algorithm where the z position of the respective marker is calculated. The result is also filtered before all components are written into the final result position vector.

6.4 Inverse kinematics algorithm

Figure 6.10 shows the a debugging representation of the used kinematic structure of the *Caliko* framework. The non-black colored lines of the left picture show the kinematic chains for each finger. The black colored lines are used as an offset structure from the hand base marker. The kinematic chains all have their own target point for which the algorithm can solve for. All chains are grouped into a kinematic model for easier data handling. The kinematic model provides a container for easier access of the base and the connection structure through which the fingers are connected to the base. With this data, the algorithm is capable of calculating a solution for each kinematic finger chain and applying these values to the position. If the target is reachable, the digital finger position should match the pose of its real world counterpart up to a specified threshold value.

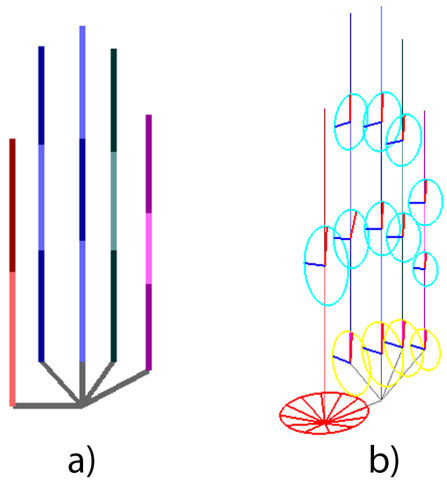

Figure 6.10: Hand model representation of the used IK framework: a) Kinematic chains and base structure, b) Hand model with visualized movement constraints

The right image shows the same model with applied constraints to each chains. The framework differentiates between base-bone constraints and normal bone constraints. The base-bone constraints are treated separately as the base-bone is the starting point of the kinematic chain and therefore effects all following bones. In the 3D mode constraints with either a global or a local reference axis can be applied. The local reference axis usually refers to a axis of the previous bone. Global constraints are used for base-bone constraints. The reference axis can the be used for the specific type of constraint. The framework supports hinge type constraints and rotor constraints. The hinge types define a rotation axis around which the bone can rotate (1 DOF). The rotor-based constraint defines an angle for a cone on a defined axis which limits the movement of the joint in multiple directions (2 DOF). This circular curve limitation is a downside of the framework as a parabolic curve description for a rotor-based constraint would better fit the movement capabilities of the fingers.

6.5 Grabbing test

To test the tracking limitations of the system , a grabbing test with the tracked *HTC Vive* marker and the tracked hand was done. Before the test was started, it was ensured that the infrared signals from the object tracker do not interfere with the camera images for tracking.

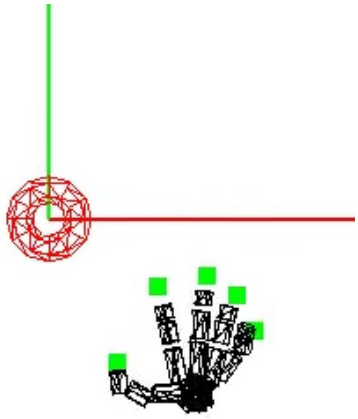

Figure 6.11: Hand and tracker model visualization used for testing

The built-in infrared filters of the camera manage to filter out the signals, as no image distortion was visible. The test chore was to pick up the tracked marker based on the position of the marker representation in digital space. A precise initial position calibration for the object tracker is done to ensure matching real and digital positions. The tracking limitations of the system showed when bending the fingers around the marker to pick it up. The bending causes the fingers to be occluded from the cameras and no new position can be tracked. This occlusion problem results from the positioning of the cameras. The top-down view limits the movement areas of the hand where all fingers markers can be tracked. The hand position marker is not affected from this occlusion problem as long as the hand is not turned around. For the system to handle grabbing movements correctly, the cameras would have to be positioned at a different angle than the top down-view or a further pair of cameras recording from a different angle has to be added.

7 Future Work

The system evaluation showed that the principal functionality of the tracking system can be realized with the selected components. One major point of improvement that could be applied to the system would be improvements to the used camera control framework. At the current state, a maximum frame-rate of only 30 FPS readout is reachable. The hardware is theoretically able to supply up to 90 FPS readout. Large parts of the processing time on the Raspberry side are accounted for by costly image optimization like blurring or transformations. These operations on whole images are classical candidates to be outsourced onto the GPU of the Raspberry for faster computation. A conversion of the existing *OpenCV* code into the assembler-like code for the raspberry GPU should be a future goal.
The calculation results for the depth values were already improved through the correction formula. To get even better results, a measurement session with more measurement points should be done to improve the values furthermore. The color threshold, which is done by *OpenCV*, could also be optimized to threshold all colors in a single run instead of running an operation for each color. This could also be candidate for GPU outsourcing. In the current state, a lot of values have to edited manually and are only visible on the command line. A graphical user interface should be build to ease the access. The hand data that is used for prototyping is still hard coded and only manually configurable. A calibration procedure for the system as well as a suitable data format for representing and translating these values for the IK algorithm is still to be done.
The system limitation that showed in the grabbing test should also be addressed to improve the system performance when having to handle partial or complete marker occlusions.

8 Conclusion

The constructed prototype showed, that it is possible to construct a basic hand and object tracking system with consumer grade hard and software. The hardware capabilities of the used Raspberry Pi showed to be able to supply enough processing power to run live image processing operations. A downside of the used camera hardware is that the used sensor utilizes a rolling instead of a global shutter. This makes the synchronization of the two cameras for the stereoscopic setup harder. Furthermore, the cameras were not intended to be used in such a scenario, therefore they lack a frame synchronization feature. The only way of achieving a form of synchronization is on the software side.
Timing measurements showed that at lower frame rates, the described multi-threaded processing approach in it's current implementation does not reach lower processing times than the sequential approach. The sequential process is in comparison less error prone since it does not need the additional synchronization wrappings that the multi-threaded approach needs. Should higher readout frame rates need to be achieved from the camera, the multi-threaded approach should definitely be further improved. At the selected resolution of 640x480 pixels, the system is able to track 6 color markers at around 50 fps. The selection of the usable color markers is rather limited. Colors from the orange yellow and red color spectrum showed to be rather difficult to track. The reflections that can occur when the hand is moved inside the tracking space can fall into these color ranges. This introduces an unwanted error in the marker position calculation. Shrink tubing, although being easy to adapt to finger sizes showed to have the downside of not being available in the needed tracking colors. The selected application of acrylic paint to the fingers as color markers has the benefit of being available in a wide variety of colors. It is also the least restricting type of all used markers in terms of haptics. The used inverse kinematics framework is a good starting point for the display of the digital hand model. It does lack some features like inter-finger and parabolic constraints which would better match the final result to the real world hand position.
The grabbing test showed the limitations of the prototype. The system provides a stable object tracking performance even when partially occluding the object tracker with the hand. The hand-tracking can provide a acceptable tracking if all markers are visible. Certain movements of the hand can cause the finger markers to become occluded. Especially when grabbing objects, this can lead to a tracking loss and degrade tracking performance.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorgelegte Abschlussarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Köln, 29.Juni 2018

Oliver Kalbfleisch