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

June 10, 2018

## Abstract

Professional-grade hand and object racking systems are usually the result of long years of combining specialized hardware and software components to achieve a tracing solution. This comes at the cost of a high technical complexity of the systems and a corresponding high pricepoint. The underlying technical concepts of these systems can be boiled down to rather simple concepts. Therefore this paper will evaluate if it is possible to build a hand and object tracking system on low cost consumer grade hardware components. Therefor the most most common approach to spatial tracking, a stereoscopic camera setup, is chosen as the system base. To keep system cost low, a set of Raspberry Pi's with the matching Camera is used for image acquiring an processing. These processing 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 colored markers are assessed for the prototype application and compared in terms of tracking precision. A relatively new inverse kinematic framework as well as a novel method for input value filtering is also applied and evaluated in the process. 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. The resulting prototype is good base for further research in terms of processing optimization.

## 0.1 Introduction

The standard interface between human and computer has for long years been mouse and 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 and virtual reality 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. With this level of immersiveness, a touch device or just a mouse and keyboard setup is rather hindering for 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's) and furthermore adds the posibility of combining these witch 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 low-res imaging 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[**?**]. Modern day 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 are therefore rather expensive to attain.

This paper assesses, if it is possible to create a real-time hand and object racking 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 grabbing precision and 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.

## 0.2 Realted work

There have been several different approaches on solving the the tracking of the human hands with their maximum of 27 DoF's each. 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 tracking optical system for reduced complexity. Also these systems need to maintain a certain amount of processing speed for achieving a fluent reproduction of the captured motion. This demands the capability of processing large amounts of data in very short intervals. Most prototype testing for the described systems is done in an controlled environment where the background is known. For a more widespread use, these systems need to be capable of registering the hand on an unrestricted background, which may have a wide range of patterns, color and lighting differences. Also human hand motion itself is quite fast, resulting in a higher frame rate demand for the tracking cameras.

Early hand tracking systems, dating back to the 1970's [**?**, **?**], relied on glove systems which used different forms of electronic sensors to measure finger and hand positioning. Newer systems made use of the developements in electronic part size and reduced the mounted sensors sizes[**?**, **?**] Further readings on sensor based hand tracking systems can be found in [**?**, **?**].

With more processing capabilites at hand optical tracking approaches where also evaluated[**?**, **?**, **?**]. These systems take either monocular or stereoscopic images of the hand and process these afterwards to find the hand position. To get a hand tracking from pure visual data, these

approaches all use colored marker on gloves. The approaches vary from only colored fingertips[?] of the glove to fully patterned gloves[?] where the known patter is used for pose estimation. Data retrieved from these markers are then compared against databases of recorded hand position configurations to find a nearest match.
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 RGB-D camera systems are now capable of suppling image data at up to 90fps. The post processing of this data needs more sophisticated image analysis or neural network processing to figure out feature points of the tracked object from the supplied image data[?, ?].
Such an 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 CNNs 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.

## 0.3 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.*"
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*. Their functionality can be classified in the amount of DOFs (*degrees of freedom*)[?] where 1DOF joints can perform a *flexion* or *twist* in one direction. The 2DOF joints can perform *flexion* movement in more than one direction and the 3DOF joints can perform simultaneous simultaneous *directive* and *twist* movements.
Regarding the human hand, each finger sums up to 4 DOF's and the thumb to 5 DOF's. Also considering 6 DOFs for the rotation and position of the whole hand, the total sum add up to 27 DOFs.

### 0.3.1 Constraints in hand motion

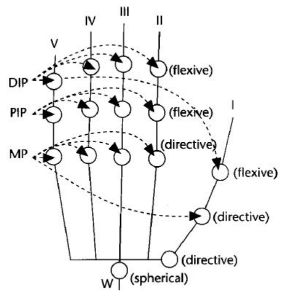

Figure 1: Joint constraints of the human hand ([?])

A full usage of all the declared DOFs 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: 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 the mathematical description of the **Type I** and **Type II** constraints,the following inequalities can be used:

$$\begin{aligned} 0^\circ \le \Theta_{MP_{flex}} \le 90^\circ \\ 0^\circ \le \Theta_{PIP_{flex}} \le 110^\circ \\ 0^\circ \le \Theta_{DIP_{flex}} \le 90^\circ \\ -15^\circ \le \Theta_{MP_{abduct/adduct}} \le 15^\circ \end{aligned} \quad (1)$$

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

$$\Theta_{MP_{abduct/adduct}} = 0^\circ \quad (2)$$

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

$$\Theta_{W_{abduct/adduct}} = 0^\circ \quad (3)$$

Since the DIP,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 that 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 :

$$\Theta_{DIP} = \frac{2}{3}\Theta_{PIP} \quad (4)$$

*Inter-finger* constraints can be imposed between joints of adjacent fingers. *Inter-finger* 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 initital 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 [**?**].

## 0.4 Kinematics

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. 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.

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.
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. 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.

The concept of *Inverse Kinematics*(IK) already describes it's principle in it's name. It takes the reversed approach in comparison to the FK principle. 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.

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. Established solution for solving inverse kinematic problems mostly rely on matrix calculations. These methods include solving via Jacobian Inverse[**?**, **?**], Jacobian transpose, Jacobian pseudo inverse[**?**, **?**], Damped Least Squares[**?**, **?**] and Newton Methods. All of these methods have a rather high computational load and some can suffer from singularities, making a solution not obtainable.

A more recent approach in the kinematics field is the *FABRIK* algorithm proposed by Aristidou and Lasenby[**?**]. 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.

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 $\mathbf{t}$.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 $\mathbf{t}$ is reachable for the end effector.This is done by measuring the distance between the root of the kinematic chain and the target point $\mathbf{t}$. This value is then compared with the sum of the distances $\mathbf{d}_i$.

$$dist_{t,d_1} < \sum_{k=1}^{i} d_i \tag{5}$$

If the summed distance is greater, then the target $\mathbf{t}$ 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.The calculation is started at the end effector, moving to the root of the chain. Therefore we assume that the new position $\mathbf{p}_n^{'}$ with n=4,...,1 is equal to $\mathbf{t}$.

$$\mathbf{p}_n^{'} = \mathbf{t} \tag{6}$$

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

$$\begin{aligned} A &= \mathbf{p}_n^{'} \\ B &= \mathbf{p}_{n-1} \\ \mathbf{l}_{n-1} &= \overline{AB} \end{aligned} \tag{7}$$

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)**).

$$\mathbf{p}_{n-1}^{'} = \mathbf{p}_n^{'} + \left( \frac{\overline{AB}}{|\overline{AB}|} \cdot \mathbf{d}_{n-1} \right) \tag{8}$$

Consecutively, this is done with the remaining joints until the root joint is reached. This finishes the first halve 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. 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 heavyweight calculation operations, 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.

# Chapter 1

# System conception

## 1.1 Technical conception

The technical components of the setup are relatively simple.Instead of using one large dimensioned unit which takes care of all calculations, the image processing steps that have to be done on both stereo images anyway, are outsourced onto the Raspberry Pi's. The image processing can be done on the two Pi's in parallel, which cuts down processing time. Furthermore the image data stays on the device and does not have to be sent to a main unit which would introduce network and data reading latency.
The two **Raspberry Pi 3 Model B** ( Model 3 with 1GB RAM and a 64GB microSD Card running a current Raspbian OS and a C++14 compiler ), which are used as the controllers for the **Raspberry Pi Camera** are connected to the "Master"PC unit via a network Switch. The "Master" takes care of the stereoscopic calculations. The 2D positional data from the slaves and the calculation results from the stereo image disparity is fed into the hand model running on the "Master". The model solution is then applied to the digital hand model and rendered.

## 1.2 Construction Details

For the prototype construction, a seated use case is chosen.The tracking volume that needs to be accomplished for a seated position is limted by our physical limitations of the arm length. A tracking volume of about $100x100x100$ cm should be sufficient to track most of the movement. A seated usecase also benefits the camera system, as this needs to be calibrated for a certain depth to attain correct depth measurements. A simple holder plate was constructed digitally and printed with a 3D Printer. Cut-outs in the mounting plate for the cameras provide the ability to mount the cameras with two machine screws and move their end position along a fixed axis. This ensures that the cameras are mounted wthout axis offsets. It furthermore give the opportunity to change the inter-axial distance of the cameras if needed. For the current setup this distance will be set to 75 mm which corresponds to the inter-axial distance of the human eyes.
As tracking features for the position calculation, colored markers for the fingers are chosen. To ensure a natural haptic feedback, glove based solutions are not valuable. The fingers are instead painted with acrylic colors.A sixth color marker has to be positioned at the connection point between hand and forearm to serve as a global position reference
For the object racking part, a *HTC vive marker* and *Lighthouse* is used. The lighthouse is able to precisely track the position of the tracker in 3D space. The tracker can then be attached to any object that needs to be tracked. The Raspberry's communicate with the main unit via simple UDP protcol over a local network connection.

## 1.3 Image analysis on the Raspberry

For the image analysis part, *OpenCV3*is chosen. The prototype program running on the Raspberry's for tracking the specified color markers is comprised of an image acquisition stage, followed by image optimization and binary mask filtering. The resulting mask-filtered images are used for color contour finding and position calculation.
To reduce image analysis times a *region of interest (ROI)* is defined for each marker after their successful detection. The selection of an appropriate ROI is crucial as it has to incorporate the possibility of large position differences between consecutive frames. First frame ROI calculation is based on a minimum *axis-aligned bounding box(AABB)* fitted onto the marker. The offset value from the image origin 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. 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 resulting rectangle has to be fitted onto the size boundaries of the full image frame by clamping the combination of position, width, height and offset to fit into the available image data boundaries. Should no marker be found in the defined ROI the frame has to be dropped and the next frame should utilize the whole image data.
To compensate for white balance shifting, a non-white ideally diffuse reflecting background should be used for the tracking space and 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. A Gaussian filter is applied to the masks which acts as a low pass filter for the image. After this step an erosion and a dilation[**?**, chapter 3.11-12] is applied to the image to further eliminate unwanted noise. On the cleaned masks, a search for the white areas whch represent the target is done. Under the assumption that all other parasitic objectshave been removed from the image, the marker should be the largest area of positive pixels in the mask frame. This area is taken as the desired tracking marker and the ROI is calculated.

### 1.3.1 RECIEVER SIDE CALCULATIONS

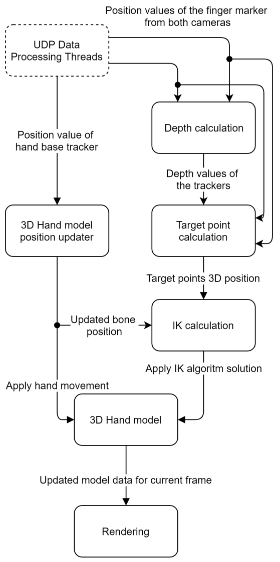
Position values of the finger marker from both cameras
UDP Data Processing Threads
Position value of hand base tracker
Depth calculation
Depth values of the trackers
3D Hand model position updater
Target point calculation
Target points 3D position
Updated bone position
IK calculation
Apply IK algoritm solution
Apply hand movement
3D Hand model
Updated model data for current frame
Rendering

Figure 1.1: Workflow of the reciever side calculations

Figure **??** displays the workflow on the "Master" side. The UDP datastreams are processed to retrieve the 2D Position points. These are fed into the depth calculation which measures the disparity between left and right side image to calculate a depth value[**?**, **?**, **?**]. The resulting 3D datapoints are then used to update the hand model and target points in 3D space. The IK algorithm takes the target points and tries to solve for their position.The tracked object data, which is not displayed here also gets processed and the representing digital models position gets updated before the rendering step.

### 1.3.2 IK CALCULATION

For the inverse kinematics calculation, the *Fabrik* algorithm is used. It does not require complex matrix calculations and has a fast convergence rate. For the implementation of the algorithm on the client side, the *Caliko* Java framework[**?**] is used. For the inverse kinematics calculation to work properly the updated bone positions and the calculated target points from the finger tracking are needed. For each finger a kinematic chain is set up with the correct bone length. The length of each bone has to be measured manually for the first prototype. 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.

## 1.4 System Evaluation

### 1.4.1 Camera hardware and control

The image processing of the camera frames on the raspberry can be a significant bottleneck for the system. With the current system setup running at 30fps, this means that the image processing time, containing frame readout and downstream processing of the images, has to be done in about $1/30s$ or 33 ms to achieve "real time" processing. The cameras use a rolling shutter instead of a global shutter. Since these cameras are designed to be of easy use for the normal consumer, they lack means of external synchronization. Attempting the synchronization of the data via Software further down the processing pipeline is therefore more reasonable.
To synchronize the reading start point 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 parent system via UDP. The parent system send the start message at the end of its initialization phase via a UDP broadcast, ensuing that both Raspberry's get the message at the same time.
All automatic modes are turned of in the camera initialization phase and fixed values are loaded from a JSON file. To get the values for the JSON file a calibration step was implemented into the command line tool. The user-set calibration values can be written into a new JSON file and are directly loaded into the program after calibration is finished. Control over these parameters is crucial for achieving a constant color separation. The threshold values values for the color tracking also need 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 function in the same fashion as the camera calibration was implemented.

## 1.5 Image processing performance

Initial investigation into code timings showed, that a bottleneck was the image optimization feature which applied an erosion an dilation 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, making the algorithm not usable for "real time" application.Removing this feature for whole image scans brought a major speed up in processing time. A test with lower resolution than 640x480 showed that the calculation time can be reduced furthermore. This confirmed 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.
Performance measurement for tracking consistency with stationary color trackers was performed for a time period of 10.000 frames to get qualitative results. The sequential implementation showed an average processing time of 46 ms with a standard deviation of  11 ms. The same measurement were done with a finer time tracking on the single parts of the processing pipeline to get a finer timing resolution. 96% of the pro-

cessing time is taken up by the image stereo rectification (31 ms) and the color tracking(15 ms ).
Stereo rectification is usually done to ensure that the two used images have an aligned horizontal orientation with camera pairs that are not aligned horizontally and/or parallel. Since the camera setup ensures this, 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. The resulting timing values showed to be sufficient to supply a tracking data stream of around 12 to 14 ms processing time with a sequential calculation approach.

### 1.5.1 COLOR ACCURACY AND ROI SIZE

Variations of lighting intensity and color temperature can 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 to achieve a clean separation of the tracker colors and reduce unwanted noise.
System testing with 5 different colored markers showed, that 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 **??** displays such a case where depending on the orientation of the hand, refletion from the scenery lightng can fall int the color tone space of the tracked marker, causing 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. Acrylic paint is available in many color variations, making it possible to stay outide of the skintone color ranges for the marker colors.It showed to have the benefits of being easily to apply to the finger. The finger coating is dry in under a minute after application. Adaption for finger size is automatically included in the application process.

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. Generally, larger ROI's produce more constent tracking results at the cost of higher calculation time.

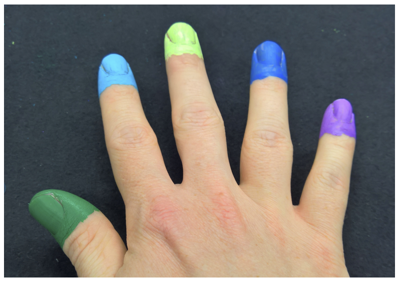
Figure 1.2: Finger marked with suitable acrylic colors

For the colors a mixture of blue and green color tones together with red tones in the purple and magenta section were chosen as the final colors.

### 1.5.2 Depth measurement accuracy

To determine the accuracy of the system for it's depth measurement values a simple test bench setup was used. 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 target for detection. The marker was positioned at altering distances from the camera rig along the it's central axis. The height at which the camera rig is positioned in the prototype 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. The accuracy of the camera system is limited by the number of pixels in relation to the camera view angleas described in [?].For the system setup of 640 px image width and a horizontal view angle of 62.5°, $\Delta\varphi$ is equal to $0.0977\frac{1^\circ}{pixel}$.The system error for a $D = 100cm$ would therefore result in about $2cm$ of possible error.As the measured values showed deviations of up to 5% from the correct value a correcting function [?] was applied for the depth calculation. The depth value is calculated as:

$$D = k * x^z \tag{1.1}$$

with K being:

$$k = \frac{Bx_0}{2\tan(\frac{\varphi_0}{2} + \phi)} \tag{1.2}$$

and x the disparity in pixels. The $\phi$ term in the equation above is used as a compensation for possible alignment errors. A exponential trendline fitted to the measured results represents the the function needed to fulfill equation above.The calculated values for the system are $k = 4543.3$ and $z = -1.035$. With the utilization of the correction function, the accuracy of the depth measurement are acceptable for the prototype application.

## 1.6 Data filtering

Measurements for of the base jitter in the data for the finger markers showed that the systems can only supply a certain stability of position tracking, causing fluctuations in the position results.Jittering of position values also causes the height calculation algorithm to generate incorrect height values.

To get mor stable a results, a filtering of the incoming data-set in several steps with the *1 euro filter* presented by Casiez et al. [?] needs to be applied.It uses a first order low-pass filter with an adaptive cutoff frequency to filter noisy signals for high precision and responsiveness.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*[?].

## 1.7 Inverse kinematics algorithm

Figure 1.3 shows debugging representations of the used kinematic structure in the *Caliko* framework.

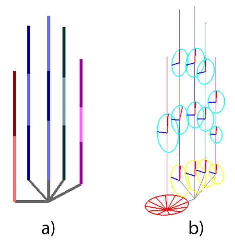

Figure 1.3: Hand model representatio of the used IK Framework: a) Kinematic chains and base structure, b) hand model with visualized movement constraints

It differentiates between base bone constraints and normal bone constraints.The constraints can be applied with either a global or a local reference axis.The reference axis can the be used for the specific type of constraint. The framework supports hinge type constraints ( 1DOF ) and rotor constraints ( 2DOF ). For the rotor constraints only circular curve limitations can be defined. This 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.