

# CAPTURING DETAILED HAND MOTION USING THE KINECT SENSOR AND MAX-SUM BELIEF PROPAGATION.

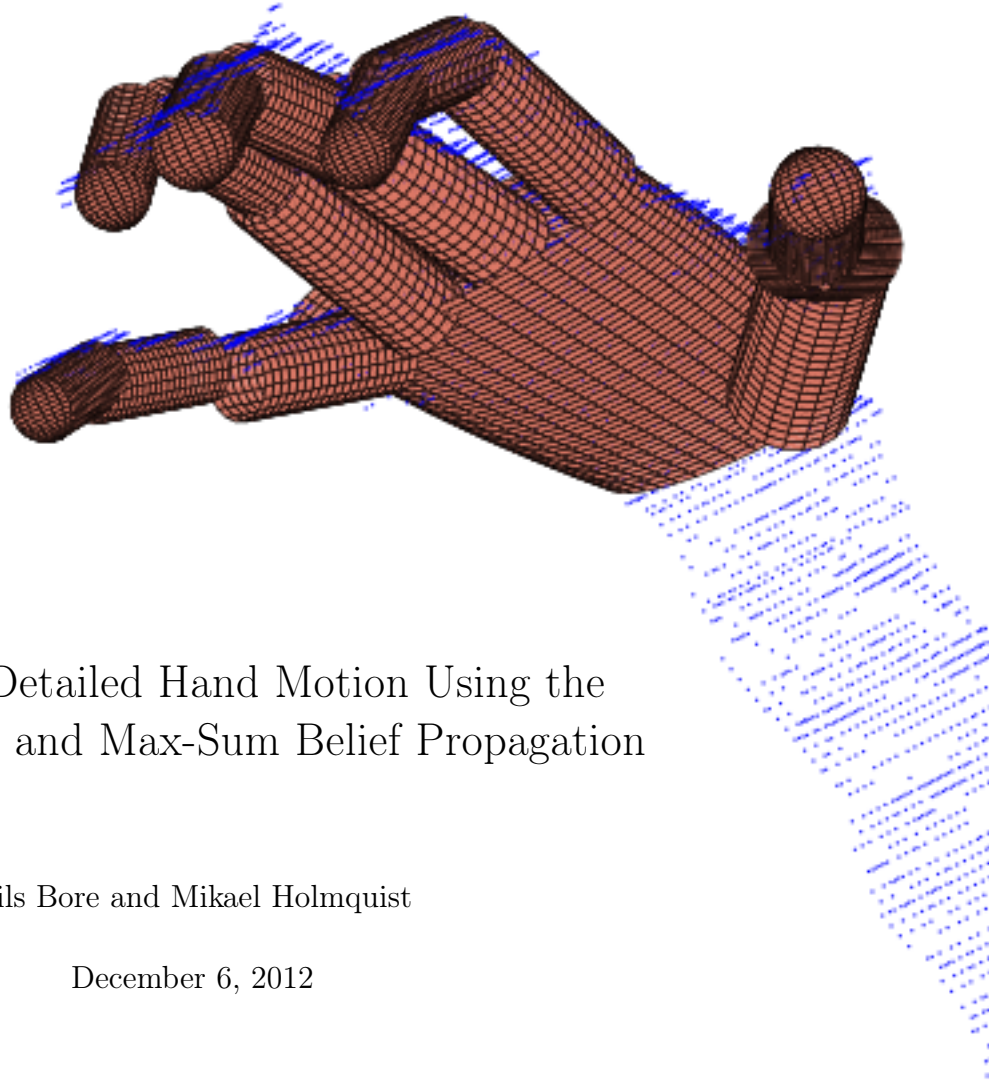
NILS BORE AND MIKAEL HOLMQUIST

Master's thesis  
2012:E42



LUND UNIVERSITY

Faculty of Engineering  
Centre for Mathematical Sciences  
Mathematics



# Capturing Detailed Hand Motion Using the Kinect Sensor and Max-Sum Belief Propagation

Nils Bore and Mikael Holmquist

December 6, 2012



## Abstract

Recent research indicates that several neurological diseases that affect motor functions could be diagnosed using analysis of detailed arm and hand motion. This analysis has earlier been carried out manually by looking for certain motion patterns in patients and animals performing a *skilled reaching task*. In this thesis we investigate the possibility of performing these tests in a more automated fashion by implementing image analysis methods for capturing arm and hand motion data from RGBD recordings. We have used the Microsoft Kinect sensor to capture motion both on a precise level, describing movements around individual joints of the hand, and on a coarser level, finding directions and positions of the lower and upper arm.

Our methods take advantage of both the RGB photos, detecting skin colour and finding arm/hand pixels, and the depth images, constructing 3D point clouds that we try to match to a simple geometrical model of the hand. Our approach is to model each phalanx of the hand individually, draw hypotheses for each of these around their pose from the previous frame and then optimize to find the most likely hand configuration using a Belief Propagation based algorithm.

We present results from running our algorithms on a few test sequences. The algorithm works well under favourable circumstances but has problems giving the correct pose for example when fingers occlude each other. Possible additions to the framework that might help to overcome these issues are also discussed.





## Acknowledgements

We would like to thank professor *Per Petersson* for suggesting this thesis. It is fantastic to use mathematics but even more so when one can work with real-world problems. Many thanks also go to our excellent supervisor *Olof Enqvist* who has supported us throughout our thesis work and provided us with invaluable input and knowledge. We would also like to thank *Gunnar Sparr* for starting the Engineering Mathematics program that we now, after 5 enlightening years, graduate from - It has been a thrilling journey!

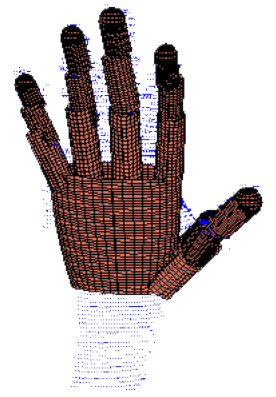
Finally, as citizens of the European Union, we would like to thank *Norway* for awarding us the *Nobel Peace Prize*.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Medical Motivation . . . . .	2
1.3	Aim of the Thesis . . . . .	3
<b>2</b>	<b>Methods</b>	<b>4</b>
2.1	Kinect Software . . . . .	4
2.2	Arm Detection . . . . .	6
2.2.1	Skin Detection . . . . .	6
2.2.2	Depth Segmentation . . . . .	7
2.3	Arm Fitting and Hand Localization . . . . .	9
2.3.1	Computing Point Normals . . . . .	9
2.3.2	Arm Fitting Using Random Sample Consensus . . . . .	10
2.4	Kinematic Hand Model . . . . .	11
2.4.1	Redundant Hand Model . . . . .	13
2.5	Belief Propagation . . . . .	14
2.5.1	Max-Sum Belief Propagation . . . . .	16
2.6	BP on a Redundant Hand Model . . . . .	17
2.6.1	Hypothesis Generation . . . . .	18
2.6.2	Model Generation . . . . .	19
2.6.3	Calculating Evidence . . . . .	21
2.6.4	Calculating Evidence - Exception for Fingertips . . . . .	21
2.6.5	Constraints . . . . .	21
2.6.6	Calculating Kinematic Angles . . . . .	22
2.6.7	Angular Constraints . . . . .	24
2.6.8	Improving Algorithm Efficiency . . . . .	24
2.6.9	Algorithm Summary . . . . .	25
2.7	Collision Correction . . . . .	25
2.7.1	Modified Evidence Function for Proximal Links . . . . .	26
2.7.2	Structural Constraints . . . . .	26
2.8	Tracking System Summary . . . . .	27
2.9	Test Setup . . . . .	27
<b>3</b>	<b>Evaluation and Results</b>	<b>29</b>
3.1	Evaluation Methods . . . . .	29
3.2	Abduction Movements . . . . .	30
3.3	Abduction Movements with Collision Correction . . . . .	33
3.4	Flex Movements . . . . .	36

3.5	Time Complexity . . . . .	39
<b>4</b>	<b>Conclusions</b>	<b>41</b>
<b>5</b>	<b>Discussion</b>	<b>43</b>
5.1	Initial Guess . . . . .	43
5.2	Using Multiple Kinect Sensors . . . . .	43
5.3	Handling Self-Occlusion and Object-Occlusion . . . . .	44
5.4	Smarter Hypothesis Generation . . . . .	44
<b>A</b>	<b>Belief Propagation Proof</b>	<b>47</b>



# Chapter 1

## Introduction

This is a master thesis that has been carried out at the Centre for Mathematical Sciences at Lund University. In this chapter we give some background and examples of previous work to the problems and theory we have considered. We also describe our aim of the thesis and give some background and motivation for the proposed medical application.

In Chapter 2 we have explained our methods and the theory behind them in detail. The methods are explained in the chronological order of the algorithm and the full structure of the algorithm can be found at the very end of Chapter 2.

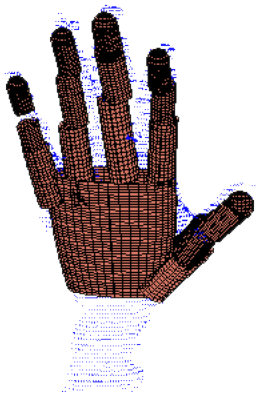
In Chapter 3 we give account of our evaluation methods and present some results.

Finally, Chapter 4 contains a discussion on improvements and some conclusions that can be drawn from the results.

### 1.1 Background

Hand pose estimation using camera sensors is an inherently difficult task, due to several different factors. The hand has more than 20 degrees of freedom [1] and a parametrization typically has non-trivial dependencies between the parameters. The intricate tree-like structure of palm and fingers makes projections contain self-occlusions which make it hard to reconstruct a three-dimensional configuration. The hand model having so many degrees of freedom also makes it exceedingly computationally expensive to use a brute-force technique, e.g. to sample guesses from each parameter space and combine them to see how well the values fit the data. To overcome these problems, many different techniques have been proposed. The approaches can be divided into two main categories: single-frame estimation and model-based tracking [1]. Of those two the latter often yields better results since it uses temporal restrictions on hand and arm motions.

For many years, hardware for capturing Depth of Field images was expensive equipment. It is only recently that it has become affordable, foremost in the



form of the Microsoft Kinect sensor. Due to the low cost, the sensor has some limitations, among others limited resolution and speed. Despite, or because of, this it has become a common tool among academia to tackle computer vision problems. Several papers has been written to specifically target the hand pose estimation problem since the arrival of the Kinect [2, 3]. The ability to easily and relatively precisely reconstruct 3D surfaces from the Depth of Field images makes the Kinect sensor better suited to tackle the hand pose estimation problem than an ordinary camera.

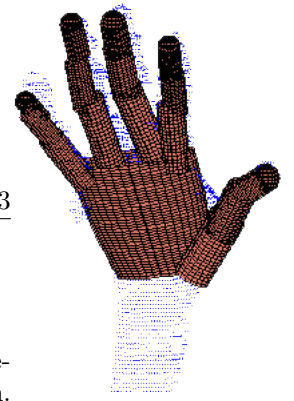
## 1.2 Medical Motivation

This thesis was suggested by professor *Per Petersson* from the BioMedical Centre at *Lund University* who wanted to adapt the work by *Palmér et al.*, on motor control in rats [4], to humans. The *skilled reaching task* for rats was developed by Whishaw and colleagues some 20 years ago and has since then been thoroughly characterized by this group [5]. The task is for the subject to reach for a small piece of food, grasp it with its paw and eat it.

On a gross level, a great resemblance between human and rodent movements in this task has been established [6] and specific motor deficits have been shown to be present in rat models of Parkinson's disease and in other models of neurological conditions involving changes in motor control such as Attention Deficit/Hyperactivity Disorder (ADHD) [7]. Interestingly, initial studies indicate that clinical tests involving this motor task could in fact be used to detect parkinsonian symptoms in patients even before they can be identified using standard neurological evaluation methods [8].

In [6], the use of rodent skilled reaching is evaluated as a translational model for investigating a number of neurological diseases. More or less manual frame by frame analysis of the skilled reaching test is performed, comparing healthy rats and humans to rats and humans suffering from diseases. The above mentioned article has made these comparisons for Parkinson's disease, Huntington's disease and to stroke patients. The results show that the human movement and its associated deficits can in fact be well understood by studying experimental lesions to the same systems in rodents.

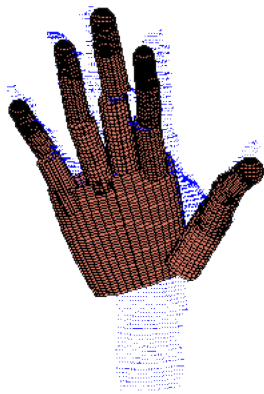
To analyze detailed reaching kinematics in rats with a higher spatial and temporal resolution than previous techniques have allowed for, Palmér et al. have recently developed an automated motion tracking system that can track detailed movements in individual joints in rats during skilled reaching [4]. This system is currently being used to investigate how different brain structures are involved in the gradual acquisition of this motor skill in rats. By adapting the system to work with human subjects, it would be possible to validate that the learning currently being investigated in rats indeed corresponds to human motor adaptations. It would also be possible to extend the studies of motor dysfunction in rats to patients suffering from various motor disorders. Automated systems for this would provide more precise information about the movements and would make it possible to perform the tests on larger groups of subjects.



### 1.3 Aim of the Thesis

Our aim with this thesis is to develop an automated system for collecting detailed hand motion data from RGBD video recordings of human hand motion. We intend to develop an image analysis system to track motion on a fine scale for the hand, where we track precise motion around individual joints, but also upper-/lower arm motion on a larger scale. We will use the Microsoft Kinect Sensor to capture the data. Our aim is to make the system work in a fully automated fashion with minimal requirements on the test environment and with no, or as few as possible, physical markers on the test subjects. Our goal is to implement a system that could be used to track arm and hand motion of human subjects performing a *skilled reaching task*, which for example means that we want to be able to track both free hand motion and hand motion when interacting with simple objects.





## Chapter 2

# Methods

### 2.1 Kinect Software

The Microsoft Kinect was originally developed for use with the Xbox gaming platform. It uses an infrared laser that projects a structured pattern of dots onto the scene in front of the Kinect. An algorithm then recognizes individual parts of the deformed pattern and can compute a depth for each portion. The extracted depths are then interpolated in order to form a depth image. Patterns of different resolution are used for different regions of the scene depending on how far away the device estimates that the regions are. This makes it possible to capture depth at ranges between 0.8-3.5 m, although with greater uncertainty when further away. At the normal working distance of our algorithm, about 1.2-2 m, the depth resolution is about 1 mm [9]. The laser and the infrared sensor in the device sit some distance apart, something that some areas where there is pattern may be occluded from the sensor. This results in areas where we have no information, typically appearing at one side of objects standing out in the scene, see for example around the contours of the arm and hand in Figure 2.1.

Before long it became clear that the Kinect had interesting applications within other fields than gaming. Many people hacked it to get to the built-in depth camera. For this reason, independent drivers were developed to be able to communicate with the Kinect using an ordinary personal computer. Microsoft later released their own software for use with the Kinect, the Kinect SDK. In this thesis we have used other packages, namely the OpenNI/NITE software interface for accessing the functions of the Kinect and the SensorKinect driver for communicating with the device. The OpenNI interface has several components that we will reference throughout the thesis. First of all, we use the package to record our RGBD video at 30 FPS. These videos are really two; one ordinary video containing colour values (RGB) and another for the depth values (D) of the corresponding pixels. Both videos are VGA resolution ( $640 \times 480$  pixels) but the depth part does not contain any information near the edges. The depth component is what really makes this thesis possible. With ordinary sensors you need multiple cameras set up to capture the scene from different angles to do a 3D reconstruction. In this thesis, we will rely on a single Kinect sensor for reconstructing 3D points of a hand. The OpenNI also handles the conversion

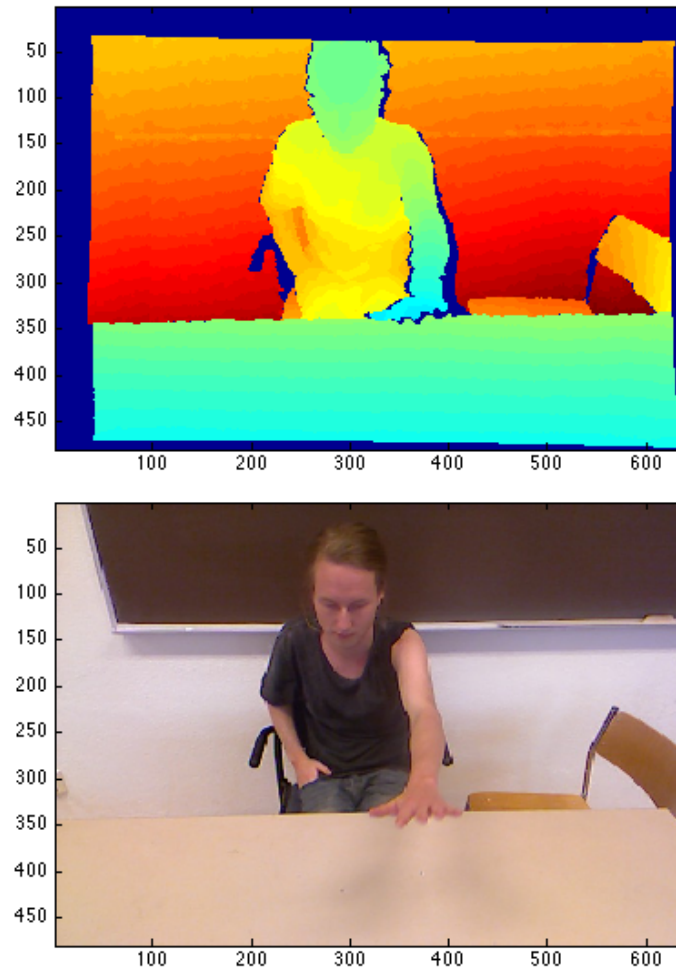
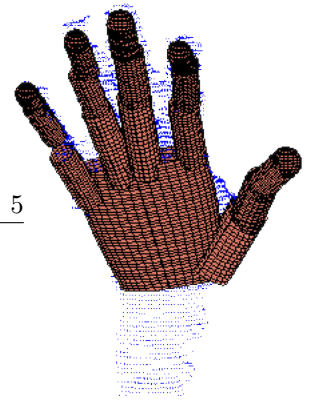
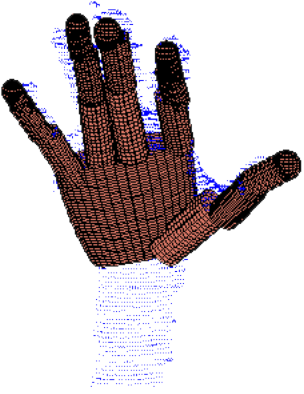


Figure 2.1: The depth component and the RGB component of a frame in a Kinect video capture. This frame is from one of the test sequences we present results for in Chapter 3. The pixels in the depth image where there is no information are blue.

from pixels in the depth image to 3D points in a coordinate system defined by the software. We will reference this conversion wherever it is used in the algorithms. To make our task a little less monumental, we have also decided to use the hand-tracking built in to the software package. This seems to be a very simple algorithm that tries to identify the middle point of the palm, often inexactly. We will refer to this point as *the Kinect hand point*. The tracking output is used to identify which pixels in the images correspond to the hand. All of the functionality that we use in the packages have to be accessed through the common programming language `c++`. To be able to get this data into `MATLAB`, we have written wrapper functions for much of the functionality that we need.



## 2.2 Arm Detection

### 2.2.1 Skin Detection

In order to classify which pixels in each image that belong to the arm/hand we perform skin colour detection. First of all, we take care of the problem that the colour of the table used in many of our test setups is very skin-like. To get rid of the points of the table we therefore use RANSAC (see Section 2.3.2) to fit a plane to the 3D points captured by the Kinect and find the inliers. We then delete all of the corresponding pixels in the depth image as well as in the colour image and continue with the skin detection.

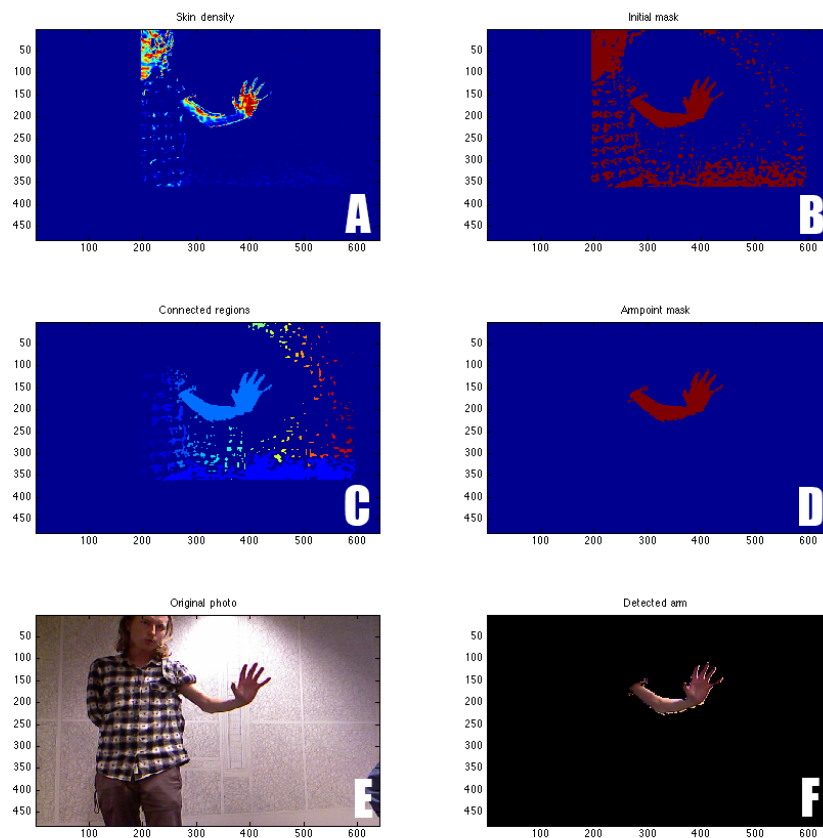
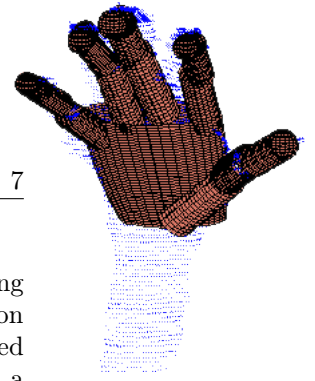


Figure 2.2: Skin- and arm detection

Our approach is to classify each pixel as being skin coloured or not based on how well the colour in each pixel fits to a parametric model of skin colour, with parameters estimated from training data. In order to get a good model for the skin colour, invariant to different lightning conditions, we transform the RGB values into a chrominance-luminance separated colour space and use only the chrominance components for our model. We have experimented with two such colour



spaces, namely HSV (or HSB) and  $YC_bC_r$ . A number of surveys on modelling skin colour using different colour spaces and their performance in skin detection algorithms have been made [10, 11]. The results vary a bit but the two mentioned above seem to be among the best performing ones that can be achieved by a simple linear transformation from RGB. They both separate chrominance and luminance effectively and when only considering the chrominance components, they also share the property of invariance to surface orientation relative to the light source [11, 12]. We compared these colour spaces using images captured with the Kinect sensor and came to the conclusion that  $YC_bC_r$  performed best.  $YC_bC_r$  is represented by *luma*  $Y$  constructed as a weighted sum of the RGB values, and two colour difference values  $C_b$  and  $C_r$ , formed by subtracting luma from the red and blue component of the RGB,

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= B - Y \\ C_r &= R - Y \end{aligned}$$

Discarding the luminance component of this colour space we then model the skin colour distribution as an elliptical Gaussian joint probability density function (pdf), defined as

$$p(\mathbf{c}|skin) = \frac{1}{2\pi \det(\mathbf{\Sigma})^{1/2}} \cdot \exp\left(-\frac{1}{2}(\mathbf{c} - \boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{c} - \boldsymbol{\mu})\right), \quad (2.1)$$

where  $c$  is the 2D colour vector and  $\boldsymbol{\mu}$  and  $\mathbf{\Sigma}$  are the standard estimates of the mean and covariance matrix of the skin colour distribution. These estimates are built from training data consisting of skin coloured patches. The training data is built beforehand by extracting small patches around *the Kinect hand point* in each frame of a sequence. These model parameters are then used for all frames of a sequence (one at the time) to model skin colour and detect arm pixels.

We calculate the skin density (Figure 2.2A) for every pixel in a certain frame according to (2.1) and then threshold the result to get an initial skin mask (Figure 2.2B). This mask will contain ones for pixels that have been detected as being skin coloured and zeros otherwise. To get rid of unwanted skin colour regions in the mask (such as face and the "wrong" arm) we divide the mask into connected regions (Figure 2.2C) and consider the connected region containing the Kinect hand point to be the arm (Figure 2.2D).

### 2.2.2 Depth Segmentation

In the example shown in Figure 2.3 the skin detection algorithm managed to get rid of other skin coloured body parts than the arm of interest. This is however only true if these other body parts are not connected to the arm in the image. It might very well be the case that the arm or hand is in front of e.g. the face and that will make the face and arm connected in the skin colour mask. To take care of this we perform a depth segmentation step built on the requirement that the depth values in two neighbouring patches of the arm should not differ more than some threshold.

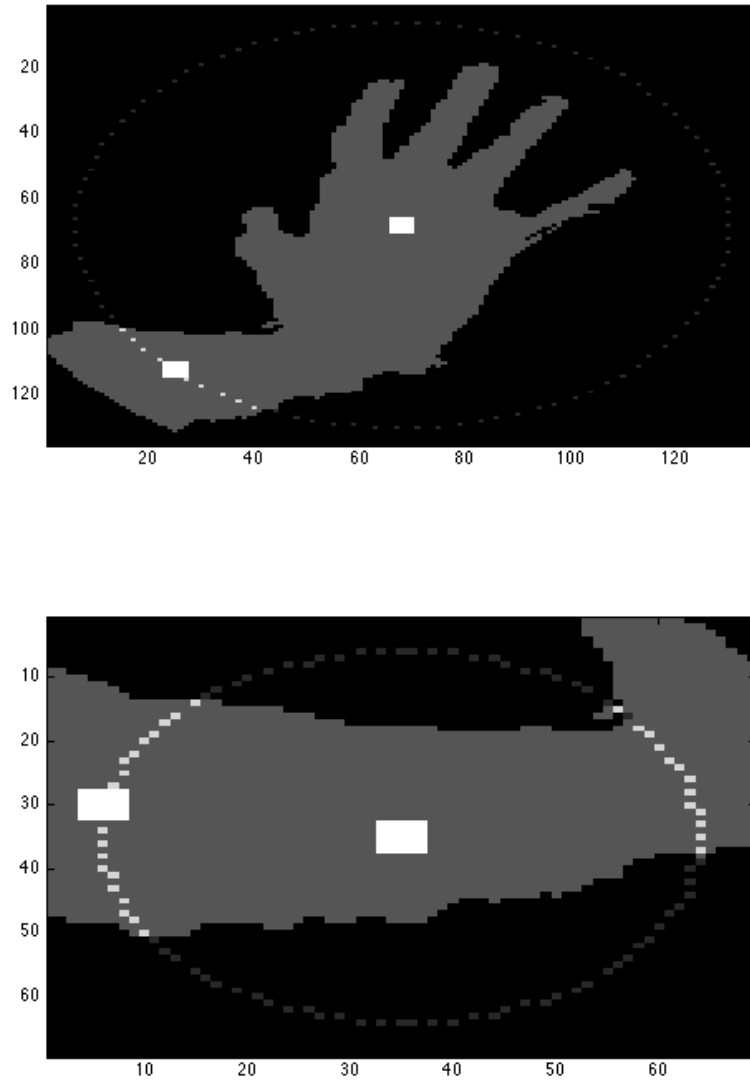
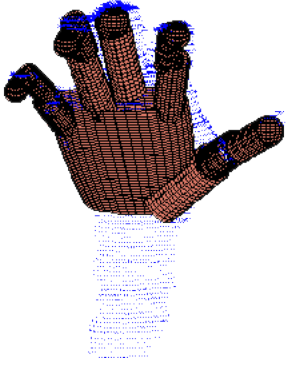
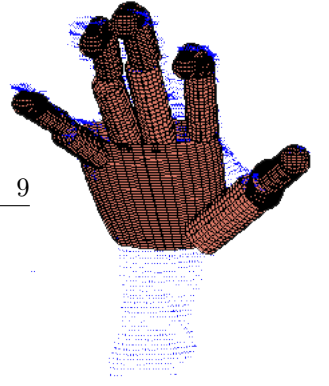


Figure 2.3: First and second iteration of depth segmentation.

We start by selecting the first centre point  $\mathbf{c}_0$  to be the Kinect hand point. Then, for each iteration, a circular patch  $\mathbf{P}$  is selected from the depth image  $\mathbf{D}$  around the current centre point  $\mathbf{c}_n$  of radius  $r$ . We threshold the patch to



create a mask

$$M_{\mathbf{p}} = \begin{cases} 1, & \text{if } |D_{\mathbf{p}} - D_{\mathbf{c}_n}| < \tau \\ 0, & \text{otherwise} \end{cases}, \quad \mathbf{p} \in \mathbf{P}.$$

The threshold  $\tau$  is set to be slightly larger than the radius of the patch and the masks are for every iteration added to a total arm mask. The next centre point should now be selected as a point on the circular boundary of  $\mathbf{M}$ , i.e.  $\mathbf{c}_{n+1} = \mathbf{c}_n + r\hat{\mathbf{v}}_n$ . To determine  $\hat{\mathbf{v}}_n$ , we examine the connected regions on the boundary of  $\mathbf{M}$  and pick out the centre points  $\mathbf{x}_i$  of the regions  $i = 1, \dots, N_{br}$ , where  $N_{br}$  is the number of boundary regions larger than some threshold  $\tau_{br}$ . We finally set

$$\hat{\mathbf{v}}_n = \frac{\mathbf{x}_k - \mathbf{c}_n}{|\mathbf{x}_k - \mathbf{c}_n|},$$

with

$$k = \arg \max_{i=1, \dots, N_{br}} |\mathbf{c}_{n-1} - \mathbf{x}_i|,$$

i.e. pick the point  $\mathbf{x}_k$  furthest from the previous centre point. The reasoning for this is to propagate the mask in a direction along the arm. Figure 2.3 shows the masks  $\mathbf{M}$  from the first two iterations together with the connected boundary regions (white) and the selected centre points.

## 2.3 Arm Fitting and Hand Localization

In each frame we want to fit a model to the upper and lower arm before fitting the hand model. Both because it could be of interest in the motion analysis but also because if we can get a good fit on the arm we will know with more precision which points belong to the hand. We will model the arm with two cylinders and use a RANSAC-type approach for the fitting.

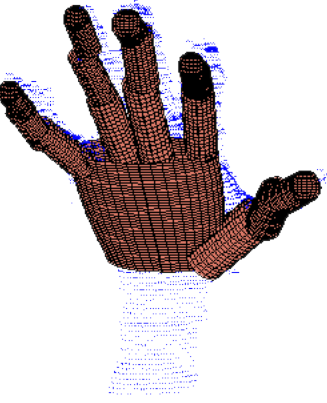
The output from the skin colour and depth segmentation is a depth image

$$D_{i,j} = \begin{cases} d_{i,j}, & \text{if pixel } (i,j) \text{ was detected as an arm/hand point} \\ 0, & \text{otherwise} \end{cases},$$

where  $d_{i,j}$  is the depth value measured by the Kinect sensor. We then transform the points  $\{(i, j, d_{i,j}); D_{i,j} \neq 0\}$  into real world 3D coordinates  $\mathbf{X}$  using the Kinect Package and note  $\mathbf{x}_{i,j}$  to be the 3-dimensional cartesian point corresponding to the pixel at position  $(i, j)$  in the depth image. We will refer to  $\mathbf{X}$  as the *point cloud*.

### 2.3.1 Computing Point Normals

To be able to more efficiently estimate the arm configuration we want to compute an approximate surface normal for each pixel in the depth image. To compute the normal for a depth value  $D_{i,j}$  at position  $(i, j)$  in the discrete image we form a square patch  $I = \{(m, n) \in \mathbb{N}^2; |i - m| \leq k, |j - n| \leq k, |D_{m,n} - D_{i,j}| < T\}$ , with  $k = 5$ . The depth threshold  $T = 20$  cm is applied to be sure that the points



are part of the same surface. We then pick out the points  $\mathbf{x}_{m,n}$  in the point cloud corresponding to the pixels  $(D_{m,n})_{(m,n) \in I}$  and form the matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_a^T & 1 \\ \mathbf{x}_b^T & 1 \\ \vdots & \end{bmatrix}, \quad a, b \in I,$$

where each row corresponds to exactly one pixel position in  $I$ . One way to find a plane to fit the points  $\mathbf{x}_{m,n}$  is to solve the minimization problem

$$\min_{\|\mathbf{p}\|=1} \|\mathbf{X}\mathbf{p}\|_2,$$

where  $\mathbf{p}$  is the implicit representation of the sought plane. This can be done using the singular value decomposition  $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$  of  $\mathbf{X}$  by extracting the vector of  $\mathbf{V}$  corresponding to the smallest singular value in  $\mathbf{\Sigma}$  [13]. If there is a plane that perfectly fits the points that value will be  $\sigma_i = 0$ . The normal direction  $\mathbf{n}_{i,j}$  is the first three elements of the plane  $\mathbf{p}$ . We choose the one pointing towards the camera since the surfaces captured by the Kinect are the ones facing the camera.

### 2.3.2 Arm Fitting Using Random Sample Consensus

The *RANdom SAMple Consensus* (RANSAC) algorithm is an iterative parameter-estimation method designed to cope with a large proportion of outliers in the input data. It was first proposed by Fischler and Bolles in 1981 [14] and has been a very popular model fitting technique in image analysis and computer vision applications ever since. The basic idea is to generate a candidate solution by randomly selecting the minimum number of data points needed in order to estimate the parameters of the model, evaluate candidate solutions and repeat until a good enough solution is found. The algorithm can easily be summarized in the steps outlined below:

---

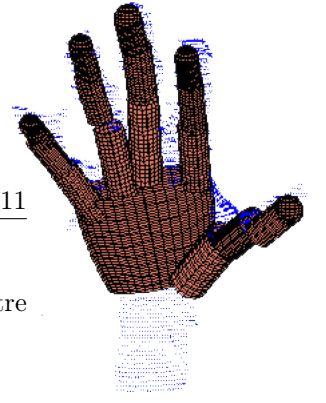
#### Algorithm 1 RANSAC

---

1. Select randomly from the data the minimum number of points required to determine the model parameters.
  2. Calculate model parameters from the selected points.
  3. Evaluate how many points from the entire data fit well (within a predefined threshold  $\epsilon$ ) to the candidate model. We will refer to these points as *inliers*.
  4. If the proportion of inliers exceeds a predefined threshold  $\tau$ , re-estimate the model using all the inliers and terminate.
  5. Otherwise, repeat Steps 1 through 4 ( a maximum of  $N$  times).
- 

We model the arm with two connected cylinders of fixed radius and length, one for the lower and one for the upper arm. We need three points and their corresponding point normals to uniquely determine a model configuration. We therefore start by randomly selecting three points  $\mathbf{x}'_1, \mathbf{x}'_2, \mathbf{x}'_3$  from a modified point cloud  $\mathbf{X}'$ . The points in  $\mathbf{X}'$  are formed by subtracting the arm model radius  $r$  from the point  $\mathbf{x}_{i,j}$  along the corresponding normal direction  $\mathbf{n}_{i,j}$ ,





$\mathbf{x}'_{i,j} = \mathbf{x}_{i,j} - r \cdot \mathbf{n}_{i,j}$ . This way, we hope to place all the points at the arm centre line, see Figure 2.4.

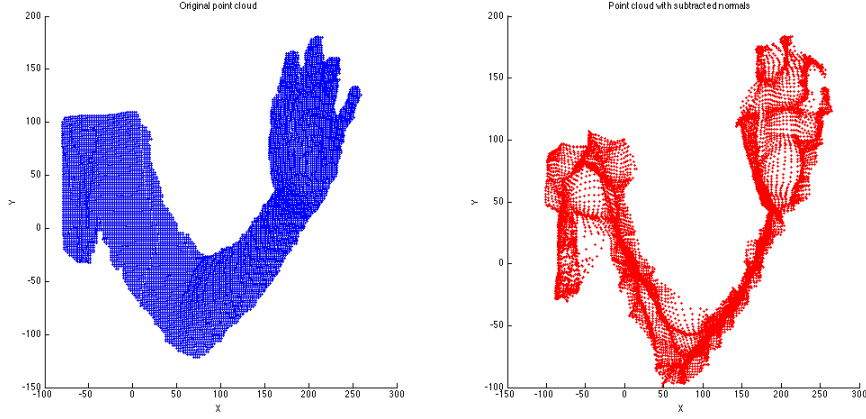


Figure 2.4: Original point cloud  $\mathbf{X}$  (left) and Normal subtracted point cloud  $\mathbf{X}'$  (right).

We then determine which of these points are lying on the same line by examining which of the corresponding point normals  $\mathbf{n}_1$ ,  $\mathbf{n}_2$  and  $\mathbf{n}_3$  are pointing in (roughly) the same direction. In the case of no similar pairs we break the iteration and draw new random points. If they all have similar directions, we consider the arm to be straight with direction  $\mathbf{v}$  pointing along the points towards the *Kinect Hand Point*  $\mathbf{h}$ . We then calculate the elbow point as  $\mathbf{e} = \mathbf{h} - L_L \cdot \mathbf{v}$ , where  $L_L$  is the predetermined length of the lower arm. Otherwise, and most commonly, two of the points, which we will now call  $\tilde{\mathbf{x}}_1$  and  $\tilde{\mathbf{x}}_2$ , will lie on the same line  $\mathbf{l}_1$  and the third,  $\tilde{\mathbf{x}}_3$ , on a significantly different one. We then calculate  $\mathbf{e}$  as the intersection between  $\mathbf{l}_1$  and the plane formed by  $\tilde{\mathbf{x}}_3$  and its corresponding normal direction. The line  $\mathbf{l}_2$  on which  $\tilde{\mathbf{x}}_3$  lies is determined by  $\mathbf{e}$  and  $\tilde{\mathbf{x}}_3$ . We determine which of the lines represent the upper and lower arm respectively by examining which is the closest to  $\mathbf{h}$ . The endpoints of the cylinders are then determined by the distances  $L_L$  and  $L_U$  (length of the upper arm) from  $\mathbf{e}$  along the respective lines. We then continue as in the standard RANSAC algorithm, calculating the proportion of inliers in  $\mathbf{X}'$ . A point is considered an inlier if it is closer than a predefined threshold  $T = 1.5 \text{ cm}$  to either one (or both) of the candidate upper- and lower arm lines. Figure 2.5 shows the matched arm cylinders projected into the original photo. We have also projected and marked the points that we have classified as hand points with green colour.

## 2.4 Kinematic Hand Model

The human hand skeleton consists of 27 bones; 8 inside the wrist, 4 to make up the palm and another  $3 \times 5 = 15$  phalanges for the 5 fingers, see Figure 2.6. Our kinematic hand model (inspired by [1, 3]) ignores the wrist bones and thus consist of 19 links that imitate the corresponding human bones and 24 degrees of freedom (DoF) that represent the joints between them. Links and joints



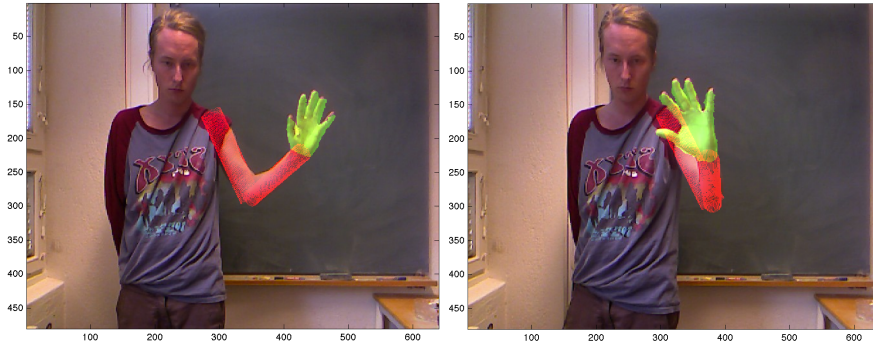
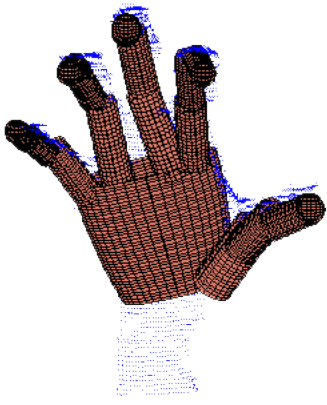


Figure 2.5: Upper arm, lower arm and hand points matched using RANSAC.

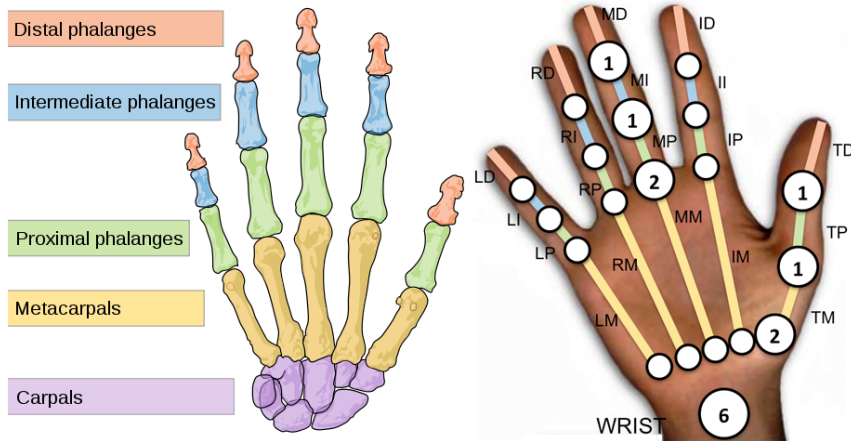
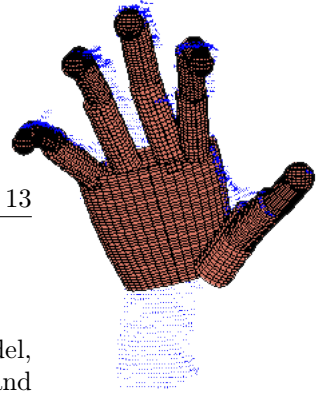


Figure 2.6: Hand model

are denoted  $L_{i,j}$  and  $\Theta_{i,j}$  respectively, where  $i$  represents a finger ( $i$ =**T**humb, **I**ndex, **M**iddle, **R**ing or **L**ittle) and  $j$  its corresponding link. The indexing of a joint is the same as for the link above it. The four different links are named  $j =$  (**M**etacarpal, **P**roximal, **I**ntermediate and **D**istal). In Figure 2.6 all links are named according to the above definition and each joint is marked with the corresponding degrees of freedom. Moreover, the DoF's represent the number of different possible rotations for the corresponding joint. The different rotations are **flexion**, **abduction** and **twist**. Note that the thumb has a different kinematic configuration than the four fingers. For the four fingers, the proximal joints have 2 DoF, supporting both flexion and abduction, while the other ones are only able to rotate in one way (flexion). The thumb only has three links (no intermediate link) and here it is the metacarpal joint that has 2 DoF. In the same figure we have also put a wrist joint. The wrist joint has 6 DoF (3D position  $(x,y,z)$  and all three rotations flex, abd, twist).



### 2.4.1 Redundant Hand Model

In the algorithm we are using our own representation of the hand. In this model, we first of all simplify by ignoring the four metacarpal links in the palm and represent the whole palm as a single segment. This will make it impossible to model the deformation of the palm that movement of the metacarpal links results in. We believe these deformations to be small enough to ignore. This leaves us with 16 phalanges/sections and in order to match and optimize our geometrical model with the data, we need to be able to optimize and draw hypotheses for these phalanges independently. We therefore "separate" them and let each phalanx have 6 DoF. We model the phalanges with cylinders for the fingers, cylinders with spherical caps for the fingertips and a cut-off ellipsoid for the palm. In total, this hand model will have 96 DoF, hence the name "redundant".

Each phalanx is modeled with variables  $(x, y, z) \in \mathbb{R}^3$  for the spatial coordinates of the centre point. The  $z$  coordinate can only be positive since they can only be detected in front of the camera. In reality there are more bounds determined by the allowed ranges of the depth camera and the field of view.

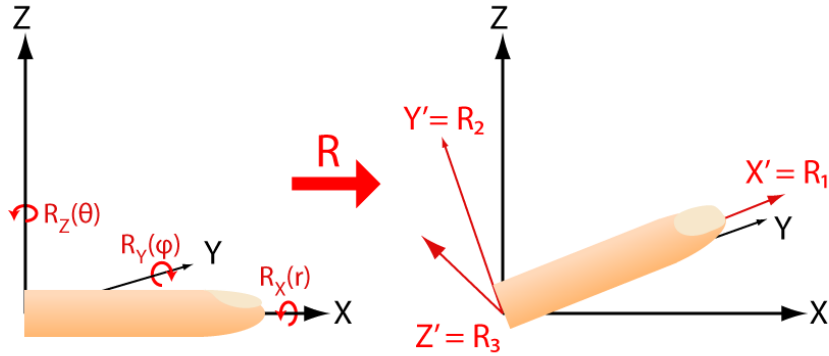
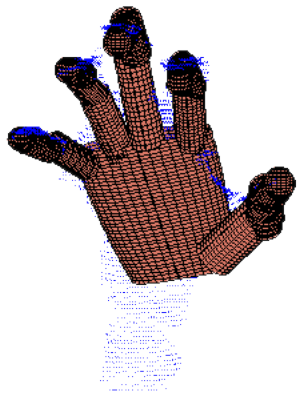


Figure 2.7: The phalanx is placed along the  $x$ -axis with normal pointing upward along the  $z$ -axis. It is then rotated with the rotation  $\mathbf{R}$  determined by the Euler angles  $\theta$ ,  $\phi$  and  $r$ .

The orientation is modeled via standard Euler angles  $\theta \in (-\pi, \pi]$ ,  $\phi \in [\frac{\pi}{2}, \frac{\pi}{2}]$  for the direction and  $r \in (-\pi, \pi]$  for the orientation around the direction axis. Euler angles are chosen because they are well documented and fairly easy to handle. It is also important that the rotation around the direction,  $r$ , can be factored out of the total rotation in this representation. This allows the algorithm to compute the model fit for a phalanx modeled with a cylinder simultaneously for all hypotheses with the same value of  $\theta$  and  $\phi$ . For those same angles, both the endpoints will also be the same (since they are also invariant to the rotation around the axis), something that will be important to the efficiency of our algorithm. The phalanx is placed along the  $x$ -axis with the normal direction (the



up side of a finger or palm) pointing in the direction of the z-axis. Each angle defines a counter-clockwise rotation around their respective axis.  $r$  determines the rotation

$$\mathbf{R}_x(r) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(r) & -\sin(r) \\ 0 & \sin(r) & \cos(r) \end{bmatrix} \quad (2.2)$$

around the x-axis. Similarly  $\phi$  and  $\theta$  describe the rotation about the y-axis

$$\mathbf{R}_y(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \quad (2.3)$$

and z-axis

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

respectively. The total rotation  $R$  is given by

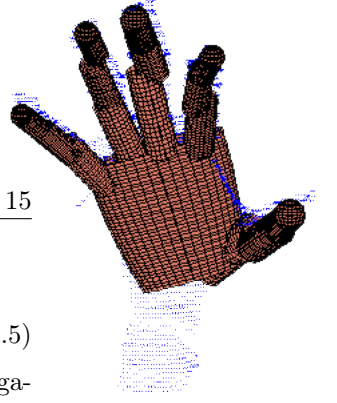
$$\mathbf{R}(\theta, \phi, r) = \mathbf{R}_z(\theta)\mathbf{R}_y(\phi)\mathbf{R}_x(r).$$

With this notation, the direction in which a phalanx is pointing is the rotated x-axis, e.g. the first column  $\mathbf{R}_1$  of the total rotation matrix. The normal direction of the phalanx corresponds to the rotated z-axis,  $\mathbf{R}_3$ . One can get a sense of how this system works by looking at Figure 2.7.

## 2.5 Belief Propagation

Our hand model can be viewed as a pairwise Markov Random Field, with dependencies between adjacent phalanges. Exactly what those dependencies are, will be discussed later on. In order to use a redundant hand model we need to be able to fit individual phalanges to data and decide what combinations of phalanx configurations form a possible hand. To do this we consider the Belief Propagation algorithm to perform inference in the graphical model. The algorithm will allow us to effectively decide which configuration maximizes the MRF joint probability function. We begin with an overview of the BP algorithm and continue explaining how we have used it.

Belief Propagation is a message-passing algorithm for performing inference in graphical models such as Markov random fields or Bayesian networks. We will discuss the case of a pairwise Markov random field (MRF), in which each two non-adjacent random variables  $X_m$  and  $X_n$  satisfy the pairwise Markov property



$$X_m \perp\!\!\!\perp X_n | X_{V \setminus \{m,n\}}, \quad (2.5)$$

i.e. are conditionally independent given all other variables. The Belief Propagation algorithm, proposed by Pearl in 1982 [15], efficiently calculates the marginal distributions  $p_{X_i}, i \in V$  for MRFs without cycles, i.e. tree graphs. For each iteration of the algorithm, every node passes messages to the neighbouring nodes. These messages are real valued functions containing information about how relatively likely one node finds the states of its neighbouring nodes. Each random variable  $X_i$  in the MRF also has observations  $y_i$  from a random variable  $Y_i$ . We denote the so called *evidence* of  $x_i$  given the observation  $y_i$

$$\Phi_i(x_i) = p_{X_i|Y_i}(x_i|y_i). \quad (2.6)$$

These observations of  $Y_i, i \in V$  are also taken into account when computing the messages. The joint probability of a pairwise MRF is given by [16, 17]

$$\begin{aligned} p(x_1, x_2, \dots, x_n) &= P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \\ &\propto \prod_{(i,j) \in V, i>j} \psi_{i,j}(x_i, x_j) \prod_i \Phi_i(x_i), \end{aligned} \quad (2.7)$$

where  $\psi_{i,j}(x_i, x_j)$  is a real valued function of two individual states  $x_i, x_j$  of adjacent nodes  $i$  and  $j$  measuring how likely the states are to occur together. It is often called the *compatibility function*.

The message  $m_{i \rightarrow j}$  from a node  $i$  to an adjacent node  $j$ , concerning how likely  $i$  finds the state  $x_j$  of  $j$ , is computed by

$$m_{i \rightarrow j}(x_j) = \sum_{x_i} \left( \Phi_i(x_i) \psi_{i,j}(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{k \rightarrow i}(x_i) \right), \quad (2.8)$$

where  $N(i)$  denotes the neighbours of node  $i$  in the tree graph. After propagating all messages over the graph as shown in Figure 2.8, the algorithm calculates a so called *belief* about the different states based on the incoming messages and local observations. The belief for state  $x_i$  of node  $i$  is given by

$$b_i(x_i) = \frac{1}{Z} \Phi_i(x_i) \prod_{j \in N(i)} m_{j \rightarrow i}(x_i), \quad (2.9)$$

and signifies how likely  $x_i$  is in comparison to the other states of  $i$ .  $Z$  is a normalization factor that makes sure that the beliefs of  $i$  sum up to one. In order to compute a message from  $i$  to  $j$  we need to have received messages from all neighbours apart from  $j$  (see the product in (2.8)). The algorithm therefore has to start at a node  $i$  that only has one neighbour, i.e. one of the leaves.

For tree graphs, as studied in this thesis, the beliefs can be shown to coincide with the marginal distribution of the variables  $X_i$ . In Appendix A, this is shown by induction over the graph. For the purpose of convincing the lazy reader, we illustrate the principle by a simple example, taken from [16]. Consider the network of four hidden nodes in Figure 2.8. Let us consider the belief at node 1.

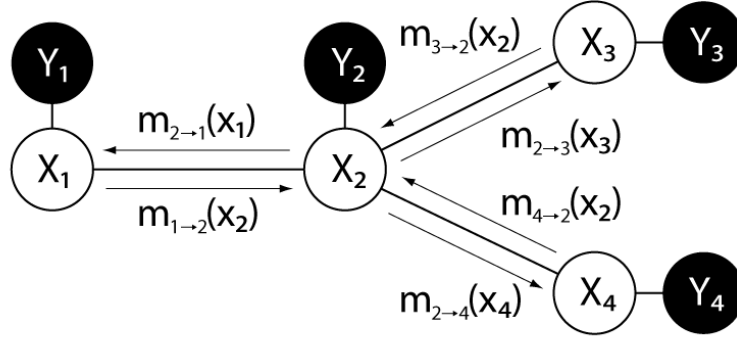
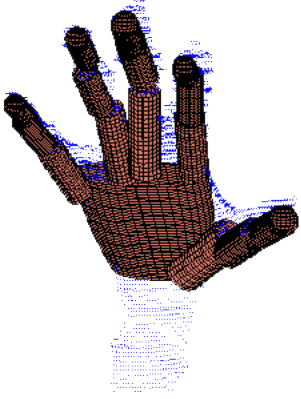


Figure 2.8: Passing of messages in the pairwise MRF from our example.

We will show that it corresponds exactly to the marginal probability  $p_{X_1}$ . Let us assume that we have propagated the messages over the graph, starting from one of the leaves 1, 3 or 4. The belief is then

$$b_1(x_1) = \frac{1}{Z} \Phi_1(x_1) m_{2 \rightarrow 1}(x_1),$$

which, according to (2.8) can be successively developed

$$\begin{aligned} b_1(x_1) &\propto \Phi_1(x_1) \sum_{x_2} \psi_{2,1}(x_2, x_1) \Phi_2(x_2) m_{3 \rightarrow 2}(x_2) m_{4 \rightarrow 2}(x_2) \\ &= \Phi_1(x_1) \sum_{x_2} \Phi_2(x_2) \psi_{2,1}(x_2, x_1) \left( \sum_{x_3} \Phi_3(x_3) \psi_{3,2}(x_3, x_2) \right) \left( \sum_{x_4} \Phi_4(x_4) \psi_{4,2}(x_4, x_2) \right) \\ &= \sum_{x_2} \sum_{x_3} \sum_{x_4} \Phi_1(x_1) \Phi_2(x_2) \psi_{2,1}(x_2, x_1) \Phi_3(x_3) \psi_{3,2}(x_3, x_2) \Phi_4(x_4) \psi_{4,2}(x_4, x_2) \end{aligned}$$

The expression inside the sums can now be recognized as (2.7) and we get that

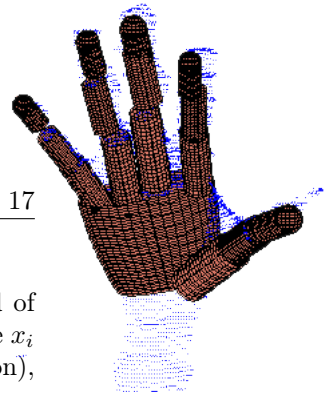
$$b_1(x_1) = \frac{1}{Z} \sum_{x_2} \sum_{x_3} \sum_{x_4} p(x_1, x_2, x_3, x_4) = p_{X_1}(x_1)$$

for some normalization factor  $Z$ .

### 2.5.1 Max-Sum Belief Propagation

In our application we are only interested in maximizing the joint probability (i.e. finding the most probable state configuration). This can be achieved by simply replacing the sum in the message update rule (2.8) with a max,

$$m_{i \rightarrow j}(x_j) = \max_{x_i} \left( \Phi_i(x_i) \psi_{i,j}(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{k \rightarrow i}(x_i) \right). \quad (2.10)$$



The beliefs  $b_i(x_i)$  will be calculated in the same way as before but instead of calculating marginals they will now be measures on how likely a certain state  $x_i$  is in node  $X_i$ . At convergence (which again is guaranteed in our application),  $b_i(x_i)$  becomes the maximum of the joint probability conditioned on  $x_i$  [18],

$$b_i(x_i) = \max_{\bar{x}} p_{X|X_i}(\bar{x}|x_i).$$

A proof similar to the one we given for Belief Propagation is suggested at the end of Appendix A. Because of this, the configurations that maximize the beliefs will coincide with those that maximize the total joint probability.

When iterating through the network, we will successively multiply the message of one node with those of all the previous ones. This will result in very large numbers and possibly numeric overflow. This is often handled by taking the logarithm of the expressions in the algorithm (the evidence- and compatibility function) and using sums instead of products. This is called the Max-Sum algorithm and is the one that we have used. In the description of the algorithm, it can effectively be ignored. In each step we have normalized the messages and evidence by making sure that the maximum value is always zero.

## 2.6 BP on a Redundant Hand Model

Inserting our geometrical hand model into the Belief Propagation framework, we view the phalanges  $X_I$  as stochastic variables in  $\mathbb{R}^6$  (see Figure 2.9). The indices are defined as in Section 2.4 with  $I$  specifying the phalanx (for example  $TD$  for the distal link of the thumb). For the palm node we will use  $X_P$ . The observations  $y_I \sim Y_I$  for each node ( $I$ ) will correspond to the hand point cloud calculated in Section 2.3 and the hidden variables  $X_I$  will be modeled according to Section 2.6.2. The algorithm will run on one frame at the time, generating hypotheses  $x_I$  around the most likely configuration from the previous frame (see Section 2.6.1). We will then propagate through the graph, starting from the leaves (distal phalanges), going down to the root (palm) and then up again. The evidence function  $\Phi_I(x_I)$  is formed by calculating how well a certain hypothesis (state)  $x_I$  of a node ( $I$ ) fits to the relevant area of the point cloud (described in detail in Section 2.6.3). The compatibility function  $\psi$  is used to enforce anatomical constraints to generate a realistic hand configuration. Due to the structure of the algorithm, we are only able to apply constraints between adjacent phalanges. The anatomical constraints consist of proximity  $\psi_{prox}$  and angular constraints  $\psi_{ang}$ . Proximity constraints make sure finger parts stay connected at the joints and angular constraints encapsulates the constraints of the 24 DoF kinematic hand model (Section 2.4). The compatibility function between two hypotheses of two adjacent nodes  $I, J$  is then formed as

$$\psi_{I,J}(x_I, x_J) = \psi_{prox}(x_I, x_J)\psi_{ang}(x_I, x_J).$$

The two constraint functions are described in Sections 2.6.5 and 2.6.6. Messages are updated according to (2.10) and when the propagation is finished, beliefs  $b_I(x_I)$  are calculated as in (2.9). Finally, the best states for each phalanx are calculated by

$$x_I^* = \arg \max_{x_I} b_I(x_I).$$

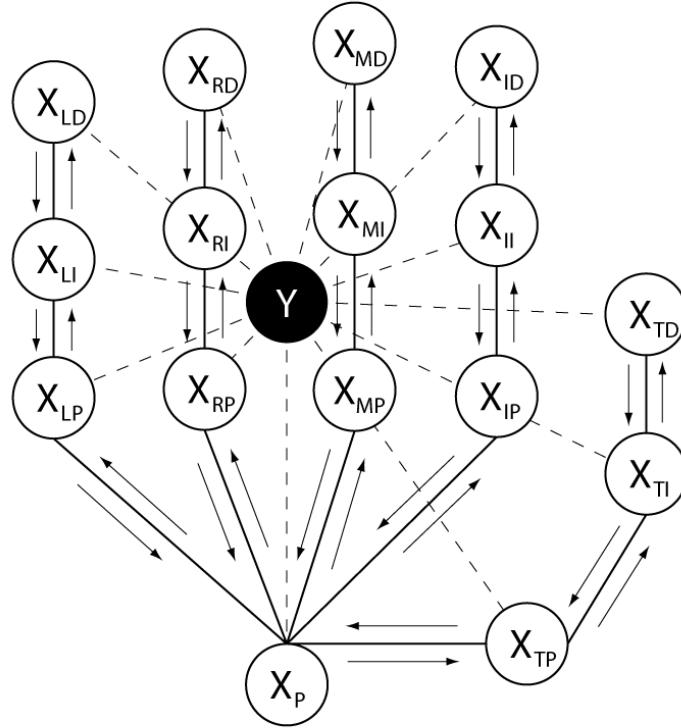


Figure 2.9: The tree graph of our geometrical hand model.

### 2.6.1 Hypothesis Generation

Since the angular model is decoupled from the comparison we do in the BP algorithm we need to be really careful about how we sample. In our kinematic model we have constraints on how fast the angular constraints can change. Since the phalanges move with the phalanges below them in the tree, the combined motion of hand and arm will determine how much individual phalanges in the hand can move between two frames. However, within those limits we do not want to have any bias on how far the angles have moved. In order to distribute our relatively few samples of  $\theta, \phi$  (in the order  $5^2$ ) we form a fixed number of circles separated by an even angle distance on the unit sphere, each centred in the previous point. We then distribute the points on those circles so that the distances between two adjacent points are the same on all circles. The result is illustrated in Figure 2.10.

The change in the variable  $r$ , however, corresponds with the change of the kinematic twist  $t$  of one phalanx with respect to another. Therefore the sampling space of  $r$  will be determined by how fast the hand can twist (since the hand model allows very little twist between most phalanges inside the hand). The sampling is carried out with respect to a sample  $(\theta, \phi)_i$  with direction  $\mathbf{v}_i$ .



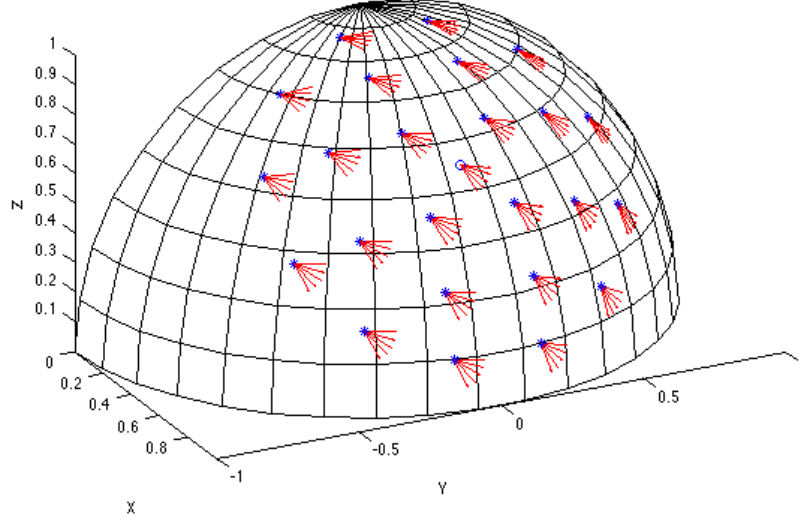
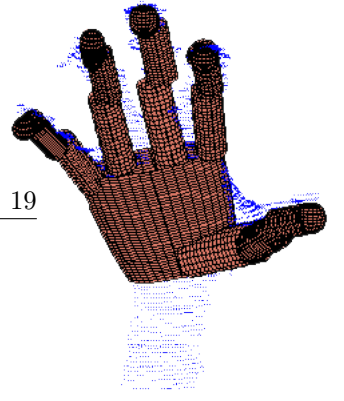


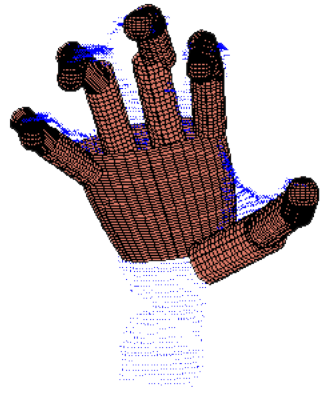
Figure 2.10: The blue stars indicate samples of the direction  $\theta, \phi$ . The blue ring in the middle is the direction from the previous frame and will be drawn now as well. The red arrows represent the samples of  $r$  that we draw for each pair  $\theta, \phi$ .

First we project the normal direction  $\mathbf{n}_0$  of the phalanx from the previous frame onto the normal plane of the direction  $\mathbf{v}_i$ . We then compute the corresponding rotation  $r_0$  around which we sample new hypotheses  $\mathbf{r}$  linearly within a range. The angle sampling procedure is illustrated in Figure 2.10. Hypotheses for the spatial coordinates  $(x, y, z)$  are simply sampled on a grid pattern within a sphere around the current point.

### 2.6.2 Model Generation

The surfaces of the phalanges are approximated with geometric primitives. The finger phalanges are modeled with cylinders of different radiuses and the palm with a truncated ellipsoid. For the distal links  $T_{i,D}$  we also connect a half-sphere to approximate the fingertips. The surfaces of both cylinders and spheres can be represented by ellipsoids using quadratic forms  $\mathbf{x}^T \mathbf{Q} \mathbf{x} = 1$ , with  $\mathbf{Q} = \text{diag}(\frac{1}{r_x^2}, \frac{1}{r_y^2}, \frac{1}{r_z^2})$ . In our case, to get a cylinder along the length of the  $x$ -axis, we set  $\frac{1}{r_x^2} = 0$  and  $r_y = r_z$ . To rotate the ellipsoids to be aligned along the axis defined by our parameters  $\theta$  and  $\phi$  we define a rotation matrix  $\mathbf{R} = \mathbf{R}_z(\theta) \mathbf{R}_y(\phi) \mathbf{R}_x(r)$  with the rotations defined according to the ones in Section 2.4.1. We leave out the rotation  $\mathbf{R}_x(r)$  from this product when dealing with





the phalanges modeled by a cylinder since they are rotationally symmetric. The equation for the surface of the rotated ellipsoid centred around  $\mathbf{x}_0$  is then

$$(\mathbf{x} - \mathbf{x}_0)^T \mathbf{R} \mathbf{Q} \mathbf{R}^T (\mathbf{x} - \mathbf{x}_0) = 1. \quad (2.11)$$

In our images we have VGA resolution, that is  $480 \times 640$  pixels. We want to compute which of these contain projections of a certain phalanx model and, if contained, project a distance to the model. This will allow us to compare the distances to the depth map obtained from the Kinect. But by knowing the position of the phalanx model in 3D space, we can limit the space where we can expect projections. This allows us to look at a smaller patch of coordinates in the image. The Kinect Software is able to translate a image coordinate and a depth value to its corresponding 3D point. This, together with the fact that the image plane is situated at a constant depth  $z = 1$ , is then utilized to translate the image coordinates in the patch to the corresponding points in the 3D image plane. Say that we want to compute the intersection for one of these points  $\mathbf{v}$  in the image plane. We then form parametrized lines  $\mathbf{0} + t\mathbf{v}$  from the camera centre through the point. By inserting the parametrized line into (2.11) and solve for  $t$ , we can compute where the ray captured in the camera intersect the ellipsoid. We define  $\mathbf{A} = \mathbf{R} \mathbf{Q} \mathbf{R}^T$  and observe that  $\mathbf{A}^T = (\mathbf{R}^T)^T \mathbf{Q}^T \mathbf{R}^T = \mathbf{R} \mathbf{Q}^T \mathbf{R}^T = \mathbf{A}$ . We get the equation

$$(t\mathbf{v} - \mathbf{x}_0)^T \mathbf{A} (t\mathbf{v} - \mathbf{x}_0) = 1$$

$$\iff$$

$$t^2 \mathbf{v}^T \mathbf{A} \mathbf{v} - t \mathbf{x}_0^T \mathbf{A} \mathbf{v} - t \mathbf{v}^T \mathbf{A} \mathbf{x}_0 + \mathbf{x}_0^T \mathbf{A} \mathbf{x}_0 - 1 = 0.$$

But since  $\mathbf{v}^T \mathbf{A} \mathbf{x}_0$  is scalar

$$\mathbf{v}^T \mathbf{A} \mathbf{x}_0 = (\mathbf{v}^T \mathbf{A} \mathbf{x}_0)^T = \mathbf{x}_0^T \mathbf{A}^T \mathbf{v} = \mathbf{x}_0^T \mathbf{A} \mathbf{v},$$

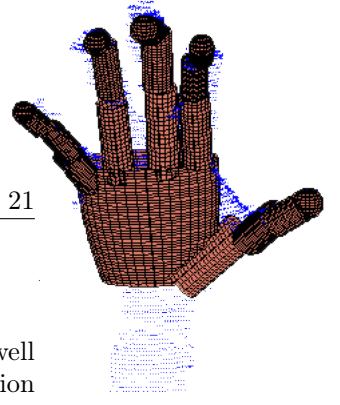
as mentioned above. This gives us the standard quadratic equation

$$t^2 \mathbf{v}^T \mathbf{A} \mathbf{v} - 2t \mathbf{x}_0^T \mathbf{A} \mathbf{v} + \mathbf{x}_0^T \mathbf{A} \mathbf{x}_0 - 1 = 0,$$

where the solutions are given by

$$t = \frac{\mathbf{x}_0^T \mathbf{A} \mathbf{v}}{\mathbf{v}^T \mathbf{A} \mathbf{v}} \pm \sqrt{\left( \frac{\mathbf{x}_0^T \mathbf{A} \mathbf{v}}{\mathbf{v}^T \mathbf{A} \mathbf{v}} \right)^2 - \frac{\mathbf{x}_0^T \mathbf{A} \mathbf{x}_0 - 1}{\mathbf{v}^T \mathbf{A} \mathbf{v}}}.$$

Here, we only care for the minus sign since it will give us the intersection closest to the camera centre. With this solution  $t$ , the distance to the intersection along the z-axis is the third element of the intersection point,  $tv_3$ . This is the value corresponding to the values in the Kinect Depth of Field image. The simplicity of the approach enables us to compute the intersection for all pixels in the patch concurrently, using matrix multiplication. If there are only complex solutions to the equation, we know that there is no intersection. This will be used to determine which pixels in the model should correspond to a pixel in the data. To cut off the ellipsoids we place two planes at the respective endpoints orthogonally to the main direction and make sure that the points lie between those planes.



### 2.6.3 Calculating Evidence

For a certain phalanx and a certain hypothesis we want to evaluate how well it fits with the corresponding phalanx's data points. In the previous section we transformed the generated 3D cylinder to the image plane which gave us a rectangular model patch  $\mathbf{M}$  containing depth values for the generated model and zeros everywhere else. We then extract the same image coordinates from the data depth image to get a data patch  $\mathbf{D}$ . To evaluate the likelihood of a patch, we compare the two patches with a simple distance measure  $\delta$  over all nonzero pixels  $\{(i, j); M_{i,j} \neq 0\}$ . If a pixel at position  $(i, j)$  is nonzero (i.e. has a depth value) in both  $\mathbf{M}$  and  $\mathbf{D}$ , we directly use the depth difference,

$$\delta_{i,j} = M_{i,j} - D_{i,j}.$$

If  $D_{i,j} = 0$  we add the pixel distance to the nearest pixel  $\{(m, n); D_{m,n} \neq 0\}$  and set

$$\delta_{i,j} = \sqrt{(i-m)^2 + (j-n)^2 + w(M_{i,j} - D_{m,n})^2}. \quad (2.12)$$

The weight  $w$  is set to 1 for all phalanges except the palm where we use  $w = 0.5$ . This is because the shape of the human palm is quite complex and can change a lot during different movements, meaning we can expect worse depth fit for the hand using a cutoff ellipsoid than for the fingers using cylinders. We then calculate the likelihood for the hypothesis that generated  $\mathbf{M}$  as

$$\phi(x) = \frac{1}{Z} \exp\left(-\left(\frac{\bar{\delta}}{\sigma}\right)^2\right), \quad (2.13)$$

where  $\bar{\delta}$  is the mean over all considered distances,  $Z$  is used to normalize the likelihood (so that it sums up to 1) and  $\sigma$  is a parameter to specify the accuracy of the depth data, we have found  $\sigma = \sqrt{8}$  to be a good value.

### 2.6.4 Calculating Evidence - Exception for Fingertips

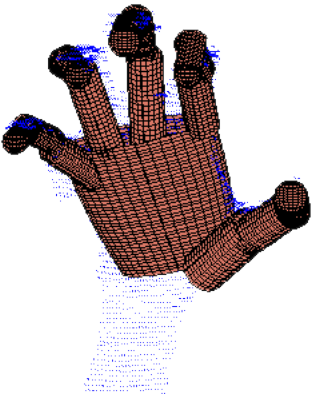
When testing our algorithm we found that it consistently matched poses where the fingers was not matching the data all the way out to the fingertips. To take care of this we introduced a new term  $\epsilon$  to the distance value (2.12) that punishes hypotheses that lie such that there are many data points in front of (in the direction of) the finger tip hypothesis. We do this by generating a slightly longer model for the fingertips and calculate  $\epsilon$  as the proportion of data points that can be seen as inliers (thresholded with 1 centimetre) to the extension part of the tip model. The likelihood (2.13) is then extended to

$$\phi(x) = \frac{1}{Z} \exp\left(-\left(\frac{\bar{\delta}}{\sigma}\right)^2 - \left(\frac{\epsilon}{\sigma_\epsilon}\right)^2\right),$$

where we have used  $\sigma_\epsilon = 0.8$ .

### 2.6.5 Constraints

Since each phalanx is generated individually, constraints are required to ensure that neighbouring phalanges stay connected at the joints, and that their respective orientations result in a valid hand configuration. To make sure phalanges



stay (more or less) connected, we employ proximity constraints, i.e. we penalize configurations of neighbouring phalanges proportionally to the distance between their end points. To ensure valid joint angles, we use angular constraints. As mentioned earlier, the traditional anatomical limits for the free hand are no longer valid in contact with objects, so enlarged ranges has to be used. Note that the constraint network is a tree (with the root at the palm and leaves at the fingertips) obeying the first order Markov property (i.e. the constraints only apply to adjacent phalanges). Hence, constraint enforcement by Belief Propagation will yield a globally optimal configuration. When sending a message from phalanx  $i$  to phalanx  $j$ , a  $N_h \times N_h$  constraint matrix (where  $N_h$  is the number of hypotheses for each phalanx) is computed for all possible combinations of hypotheses for the two phalanges. The matrix entries are the measures on how likely a hypotheses pair  $u_i, u_j$  is to occur, and are defined as

$$\psi(u_i, u_j) = \psi_{prox}(u_i, u_j) \cdot \psi_{ang}(u_i, u_j).$$

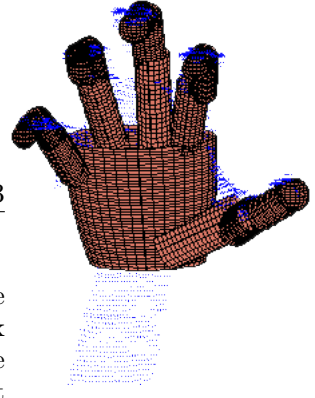
There is uncertainty concerning the placing of the fingers at the palm and the length of the fingers. This is the justification for the loosely coupled model that we use. In our approach, we have emphasized that there is more uncertainty for the length of the phalanges than how the individual endpoints are placed orthogonally to the direction (especially within the fingers). We therefore allow a greater variance of the distance between the endpoints along the mean direction of two phalanges at a joint. To do this, we use a multivariate normal distribution with zero covariance between the variables. Further, we assign the same variance  $\sigma_{prox}^2$  for the axes perpendicular to the mean direction of the phalanges and the variance  $\sigma_{length}^2$  for the mean direction. Say that we have hypotheses of two adjacent phalanges  $L_i$  and  $L_j$  with a common joint that is associated with the respective endpoints  $\mathbf{x}_i$  and  $\mathbf{x}_j$ .  $L_i$  has direction  $\mathbf{v}_i$  and  $L_j$  has direction  $\mathbf{v}_j$  (pointing away from the palm). We form the mean direction  $\mathbf{v}$  by taking  $\mathbf{v}_i + \mathbf{v}_j$  and normalizing the result. The error vector of the endpoints is  $\mathbf{d} = \mathbf{x}_i - \mathbf{x}_j$  and the component for the mean direction are computed by projection  $d_{length} = \mathbf{d}^T \mathbf{v}$ . The rest is put into the factor  $d_{prox}^2 = \|\mathbf{d}\|^2 - d_{length}^2$ . We then define  $L_i$  and  $L_j$ 's compatibility function as

$$\psi_{prox} = \exp\left(-\frac{d_{prox}^2}{\sigma_{prox}^2} - \frac{d_{length}^2}{\sigma_{length}^2}\right).$$

$\sigma_{prox}$  and  $\sigma_{length}$  could be seen as parameters specifying the importance of the observed error, and also the relative weight of the proximity error against the angular constraint errors to be defined next. In the experiments the values are  $\sigma_{prox} = 15$  and  $\sigma_{length} = 50$ . Note again that for all phalanx pairs except the five pairs that involve the palm, the proximity constraints are independent of the twist rotation which we'll take advantage of to calculate proximity constraints for many ( $N_r^2$ ) hypotheses combinations at a time.

### 2.6.6 Calculating Kinematic Angles

In the Belief Propagation algorithm we want to compute the kinematic angles flex, abduction and twist between two phalanges to be able to form a message. This is not straightforward with the angular representation we have chosen and we need to go through some steps in order to do so. Let us consider the case



when we want to compute the kinematic angles of a phalanx  $P_1$  relative to the phalanx  $P_0$  below it in the graph. We let  $\mathbf{R}_1$  and  $\mathbf{R}_0$  denote the rotation matrix of one sample  $\theta$ ,  $\phi$ ,  $r$  of the respective phalanges. These matrices give us the bases  $\mathbf{E}_0 = \mathbf{R}_0\mathbf{E}$  and  $\mathbf{E}_1 = \mathbf{R}_1\mathbf{E}$  by rotation of the canonical base  $\mathbf{E}$ . The first step is to express the rotation of  $P_1$  in the coordinate system  $\mathbf{E}_0$  of  $P_0$ . This is given by

$$\hat{\mathbf{E}} = \mathbf{E}_0^T \mathbf{E}_1.$$

From the base  $\hat{\mathbf{E}}$  in  $\mathbf{E}_0$  we want to reconstruct the angles  $f$  (flexion),  $a$  (abduction) and  $t$  (twist) yielding the corresponding rotation matrix. In order to do so we need to consider what kind of rotation the kinematic angles actually represent. The main difference with our Euler angles is that the initial rotation around the x-axis,  $t$  manipulates the coordinate system for the following rotations  $f$  and  $a$ . Instead of the reference z- and y-axis, the rotations are defined around the rotated axes. Thus the rotation for  $t$ ,  $\mathbf{R}_x(t)$ , will be the same as for  $r$ , described in Section 2.4.1. The rotation for  $f$  on the other hand will be around the y-axis rotated by  $t$ . This corresponds to the rotation  $\mathbf{R}_y(f)$  from Section 2.4.1 carried out in the basis defined by  $\mathbf{R}_x(t)$ . This yields

$$\mathbf{R}'_y(f) = \mathbf{R}_x(t)\mathbf{R}_y(f)\mathbf{R}_x(t)^T.$$

Similarly  $\mathbf{R}_z(a)$  in the new basis is given by

$$\mathbf{R}'_z(a) = \mathbf{R}_x(t)\mathbf{R}_z(a)\mathbf{R}_x(t)^T.$$

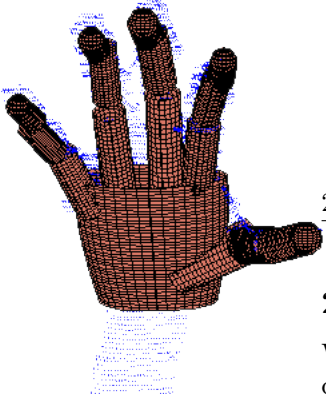
The total equation for the rotation is then

$$\begin{aligned} \mathbf{R}(f, a, t) &= \mathbf{R}'_z(a)\mathbf{R}'_y(f)\mathbf{R}_x(t) \\ &= \mathbf{R}_x(t)\mathbf{R}_z(a)\mathbf{R}_x(t)^T\mathbf{R}_x(t)\mathbf{R}_y(f)\mathbf{R}_x(t)^T\mathbf{R}_x(t) \\ &= \mathbf{R}_x(t)\mathbf{R}_z(a)\mathbf{R}_y(f), \end{aligned}$$

which seems natural, given what the rotations are supposed to do. If we insert the values we get

$$\mathbf{R}(f, a, t) = \begin{bmatrix} \cos(f)\cos(a) & -\sin(a) & \cos(a)\sin(f) \\ \cos(f)\cos(t)\sin(a) + \sin(f)\sin(t) & \cos(a)\cos(t) & \cos(t)\sin(f)\sin(a) - \cos(f)\sin(t) \\ \cos(f)\sin(a)\sin(t) - \cos(t)\sin(f) & \cos(a)\sin(t) & \cos(f)\cos(t) + \sin(f)\sin(a)\sin(t) \end{bmatrix}.$$

To get the values of  $f$ ,  $a$  and  $t$  we compare this matrix element-wise with the values of  $\hat{\mathbf{E}}$ . We've chosen to enforce the constraint  $\hat{E}_{11} > 0$  for valid configurations, i.e. the phalanges can't point in opposite directions. This is not always the case, for example some of the joints between the distal and proximal links can bend slightly more than  $\frac{\pi}{2}$ . To simplify the calculations, we have chosen to ignore this rather rare scenario. Because of this we know that  $|a| < \frac{\pi}{2}$  and we get  $a = -\sin^{-1}(\hat{E}_{12})$ . The same goes for  $f$ , which gives us  $f = \sin^{-1}(\frac{\hat{E}_{13}}{\cos(a)})$ . We choose the sin term because  $\sin^{-1}$  has the desired range  $[-\frac{\pi}{2}, \frac{\pi}{2}]$ . Finally,  $t$  has to be defined on  $(-\pi, \pi]$  and we therefore need to use both of the terms  $\cos(t) = \frac{\hat{E}_{22}}{\cos(a)}$  and  $\sin(t) = \frac{\hat{E}_{32}}{\cos(a)}$ . The angle  $t$  is then given by the four quadrant inverse tangent function through  $t = \text{atan2}(\hat{E}_{32}, \hat{E}_{22})$ .



### 2.6.7 Angular Constraints

What we want to do in the algorithm is to take one hypothesis of a joint and compare it to all the hypotheses of a previous joint. The kinematic angles that are allowed as a valid configuration are to have probability one, while the rest should have little or no probability depending on how far they are from a valid configuration. Similar to [3], we allow the range to be somewhat larger because we could be dealing with hands manipulating objects, which modifies some of the standard allowed ranges. We let  $R_{flex}$ ,  $R_{abd}$  and  $R_{twist}$  denote the sets of valid angles for a particular joint. Apart from the kinematic constraints we have also chosen to incorporate temporal constraints here. As an example we can look at the abduction of a joint. It moves quite slowly and can not change very much between two frames. The valid angles are the intersection of the interval around the previous value and the valid kinematic intervals of each angle. If there is no intersection, the set will simply consist of the point in the kinematic interval closest to the value in the previous frame.

We define the distance from a set  $A$  to a point  $x$  as

$$d_A(x) = \min_{y \in A} |x - y|.$$

Particularly,  $d_A(x) = 0$  for  $x \in A$ . With this notation, our formula for the probability of a configuration  $f, a, t$  can be written as

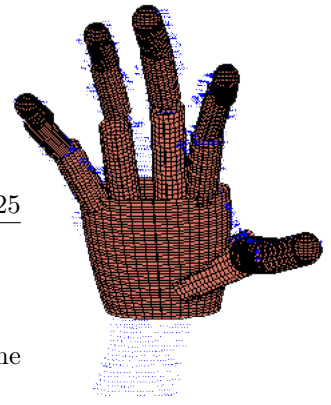
$$\psi_{ang}(f, a, t) = \exp\left(-\left(\frac{d_{R_{flex}}(f)}{\sigma_{flex}}\right)^2\right) \exp\left(-\left(\frac{d_{R_{abd}}(a)}{\sigma_{abd}}\right)^2\right) \exp\left(-\left(\frac{d_{R_{twist}}(t)}{\sigma_{twist}}\right)^2\right),$$

with the denominators signifying how strongly the constraints should be enforced. In our tests we use  $\sigma_{flex} = \frac{\pi}{18}$ ,  $\sigma_{abd} = \frac{\pi}{20}$  and  $\sigma_{twist} = \frac{\pi}{18}$ .

### 2.6.8 Improving Algorithm Efficiency

One positive aspect of the BP algorithm is that the message propagation can often be done in parallel, making use of modern multicore processors. In our application we begin the propagation at each of the five fingers' distal links simultaneously and propagate down to the palm in parallel. We then send messages from the palm and continue the propagation back up to the leaves at all fingers simultaneously.

We also take advantage of the fact that  $\Phi$ ,  $\psi_{prox}$  and  $\psi_{ang}$  are independent of some of the parameters of the phalanx models. The evidence (model fit) function  $\Phi$  is independent of the twist rotation for the cylinder shaped phalanges (all except palm). The same goes for  $\psi_{prox}$  since it only measures the distance between endpoints on the centre lines of the cylinders. Moreover, the angular constraints  $\psi_{ang}$  are totally independent of the 3D position  $(x, y, z)$ . We use this in the algorithm by looping through the different model parameters in a separated fashion. In an outer loop we go through the orientation parameters  $\theta$  and  $\phi$ . Inside the orientation loop we first loop over position parameters to calculate the evidence and proximity constraints and then go through a second loop over the rotation  $r$  to calculate the angular constraints.



### 2.6.9 Algorithm Summary

In the table below we have summarized the Max-Sum algorithm for one frame in the tracking system.

---

#### Algorithm 2 Max-Sum Algorithm on Hand

---

- 1: Simulate hypotheses for each phalanx around the configuration from the previous frame.
  - 2: Start by computing the evidence  $\Phi$  for the distal links.
  - 3: Compute downward messages (with constraints  $\psi$ ) and evidence successively for the links down to the palm.
  - 4: Compute upward messages successively for the links back to the distal links.
  - 5: For each link, compute beliefs from messages and evidence.
  - 6: Pick the hypotheses with the highest beliefs to form the new hand configuration.
- 

## 2.7 Collision Correction

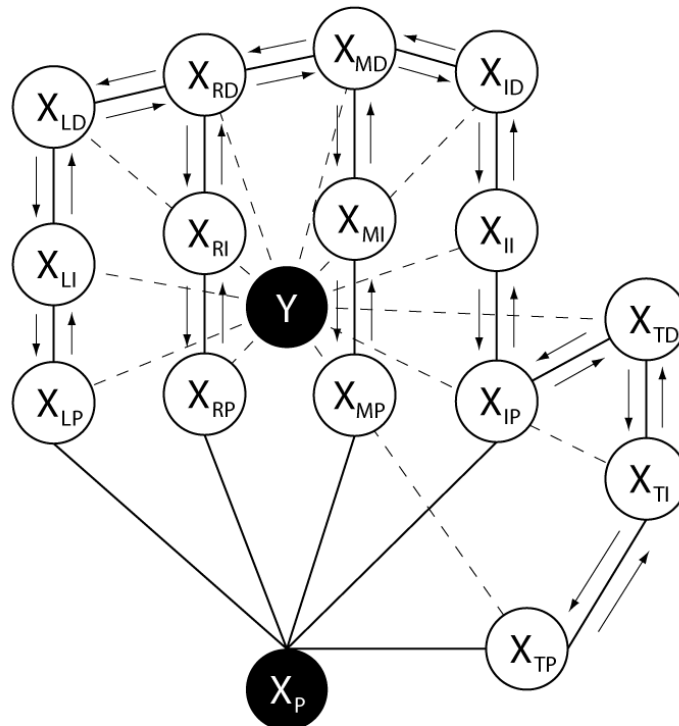
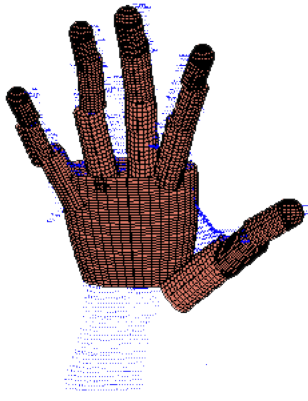


Figure 2.11: Illustration of the message passing in the correction step. This time, the palm is effectively considered evidence.



To take care of an apparent fault in the Belief Propagation, namely the lack of handling of crossing fingers, we propose a post-processing step. After an iteration of the algorithm, we check if there are any crossing distal links in the estimated pose. For the thumb, we check if the tip intersects the index finger's proximal link. If any of the fingers intersect, the post processing is run.

We assume that the palm has found a good fit in the first step, irrespective of the crossing fingers. With this assumption we fix that palm configuration and consider it an observation  $x_p \sim X_p$  of the true pose, similar to  $y$  in Section 2.6. The iteration will then largely be the same as previously but without messages including the palm and with messages between the fingertips reflecting if they are crossing or not. This way, the pairwise MRF of the variables will form a tree graph, see Figure 2.11, and thus have an exact solution via Belief Propagation. The propagation again starts at the leaves, which are now the proximal links. They then propagate to the distal links in parallel. Message passing between the fingertips are then carried out from left to right and then from right to left. Messages are then passed back to the proximal links in parallel and the iteration is complete. In the algorithm, we use the hypotheses drawn in the standard Belief Propagation step, which allows us to reuse for example the model fitting evidence. However, since  $X_p$  is now considered an observed variable with value  $x_p$ , we need to reformulate the evidence function of the proximal links.

### 2.7.1 Modified Evidence Function for Proximal Links

With the observation  $x_P$  of the palm we can formulate the evidence of a hypothesis  $x$  of a proximal link,  $\Phi_p(x)$ , as the product of the compatibility function  $\psi$  and the evidence function  $\Phi$  from Section 2.6.

$$\Phi_p(x) = \psi(x, x_P) \cdot \Phi(x).$$

It is intuitive to reuse the compatibility function for this purpose because the constraints between the static palm and the phalanx should effectively be the same. This palm configuration allows for the same range of endpoints and angles that a palm hypothesis in the standard algorithm would.

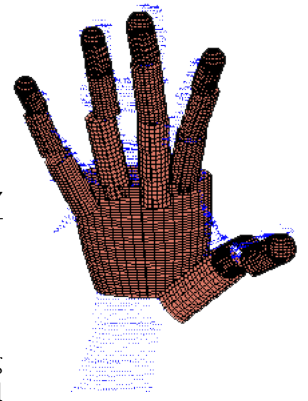
### 2.7.2 Structural Constraints

We also need to define the compatibility function  $\psi_{struct}$  for the message passing between distal links. The function should prohibit any configuration where the links intersect. To check if two phalanges intersect we simply check if the minimum distance between the line segments along the phalanges' centres are smaller than the sum of their respective radius. The function  $\psi_{struct}$  is defined as

$$\psi_{struct}(x_i, x_j) = \begin{cases} 1, & \text{if } x_i \text{ and } x_j \text{ do not cross} \\ 0, & \text{otherwise} \end{cases}.$$

These new evidence and belief functions are taken into account when computing the messages and also at the end of the iteration, for calculating new beliefs.





## 2.8 Tracking System Summary

The table below provides an overview of the different steps in the tracking system. It is worth noting that the skin detection algorithm is actually trained over a larger number of the frames before we initialize the tracking system.

---

### Algorithm 3 Tracking Algorithm

---

- 1: Load the Kinect RGB- and depth image for the frame. Compute the *point cloud*.
  - 2: Remove the plane with the most inliers (hopefully the table).
  - 3: Run the trained skin detection algorithm to remove pixels that do not look like skin.
  - 4: Use the depth segmentation to isolate the connected region recognized as the arm.
  - 5: For these points, compute the point normals.
  - 6: Run the RANSAC algorithm to match two connected cylinders to the upper- and lower arm.
  - 7: Isolate the pixels at the end of the lower arm in the depth image.
  - 8: These points are the evidence for the Max-Sum algorithm described in Algorithm 2, which is now run.
  - 9: If any of the distal links intersect, run the Collision Correction algorithm to part them.
  - 10: Go to the next frame.
- 

## 2.9 Test Setup

Already in an early phase of this thesis, we decided that we wanted the system to work well with very few requirements on the test environment. The algorithm will now work with almost any background and there is no real need for it to be of solid colour. Even if we use light intensity invariant colour spaces for the skin colour detection, it is preferable if the test environment is bright and evenly lit for the skin colour detection to work at its best. Moreover, the Kinect sensor has some serious drawbacks. It will not capture any depth data for objects that are too close. On the other hand we get very low resolution on objects that are far away. The limit for how close an object can be is around 1 metre away from the sensor. The best results are obtained if the test subjects hand stay at around 1.2 - 2 metres away from the sensor during the whole recording. We have found that the optimal test setup for our algorithm is to have the subject seated at a table with the Kinect sensor positioned obliquely from above (see Figure 2.12). The subject should keep the hand of interest above and the other hand below the table at all times.



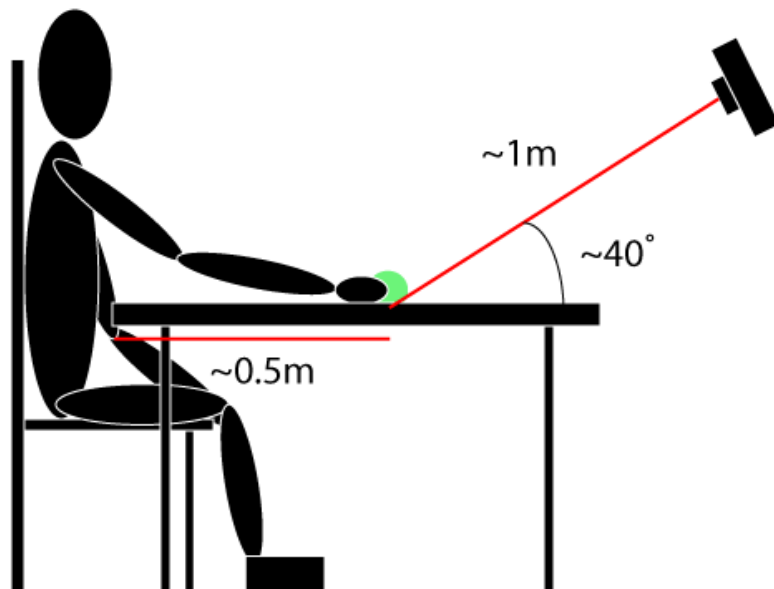
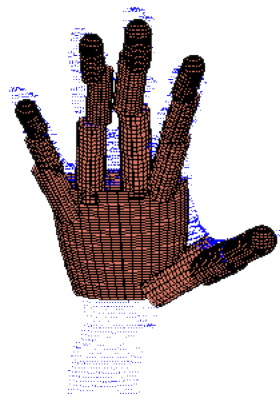
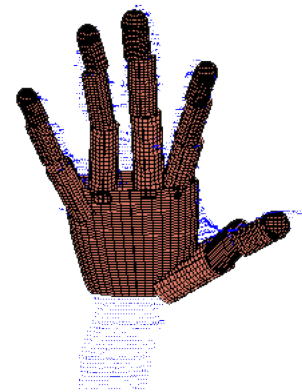


Figure 2.12: Illustration of an optimal test setup



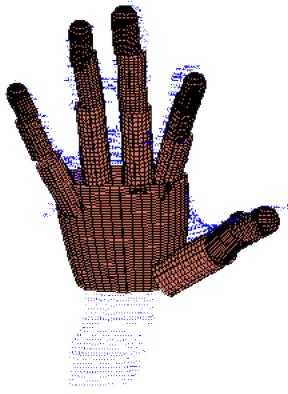
## Chapter 3

# Evaluation and Results

### 3.1 Evaluation Methods

Validating an algorithm like the one described in this report is no easy task. The goal is to track precise movements of individual joints from RGBD video recordings but how can we evaluate the correctness of our results when the true positions and directions of the phalanges are unknown. One approach, used in [2, 3], is to use a computer generated hand and run the algorithm on the synthetic data. A problem with this approach however, is that the synthetic hand data would not look much like the data we get from the Kinect recordings and since it would be generated by a geometric model, probably very similar to the one we use in the algorithm, the results would be biased and not really a good measure of how well this method works in reality. At the same time, one can get a pretty good idea of how well the tracker works by just examining the results visually, frame by frame. It is not the most scientific method and we will not get any measure of the precision of the model fitting, but it does say something about if the tracker has found the, more or less, right pose or not. To do this we both look at the hand model in 3D plotted together with the point cloud of the hand. We also project the best hand hypothesis in each frame to the image plane and colour the model phalanges in the same image as the original RGB photo. We then, for each frame, grade the result with GOOD if it fits 'perfectly' (pose is correct and the model fits well with the points), OK if the overall pose is correct but the fitting might be a bit off, and BAD if the pose is totally wrong, typically when phalanges match with data from the wrong finger. To get some numbers on the model fitting we also calculate the proportion of 3D points that deviates more than 1 centimetre from the point cloud captured by the Kinect.

We will present the results of 2 sequences recorded in an optimal test setup (see Section 2.9). The first sequence consist of mainly abduction movements and the second of mainly flex movements. Both sequences have been run using the same algorithm parameters presented in the table below. For the abduction sequence, we will present two runs, one without the collision correction and one with. Input data for one frame in this sequence is presented in Figure 2.1.



Parameter	Value	Description
$N_{\theta\phi}$	20	Number of direction hypotheses
$N_r$	5	Number of orientation hypotheses
$N_{xyz}$	81	Number of position hypotheses
$I_x = I_y = I_z$	[-18, 18]	Ranges for position hypotheses
$I_\theta = I_\phi = I_r$	$[-2\pi/9, 2\pi/9]$	Ranges for direction/orientation hypotheses

Table 3.1: Parameters used in the test runs.

The numbers in the table above yield an astounding  $(20 \cdot 5 \cdot 81)^{16} \approx 3.43 \cdot 10^{62}$  number of hand configurations to evaluate. Both sequences were run on a MacBook Pro 2.2 Ghz Intel Core i7 with 4 GB RAM taking advantage of 4 physical and 4 virtual cores. Arm detection, arm fitting and hand localization was all done separated from the Belief Propagation beforehand. With the parameters stated above the BP algorithm took about 60 seconds to run for each frame.

## 3.2 Abduction Movements

The first sequence consists of 220 frames ( $\approx 7$  sec) and contains a hand performing 4 full abduction movements. Figure 3.1 shows the proportion of 3D points deviating more than 0.5 centimetres from the model for each frame.

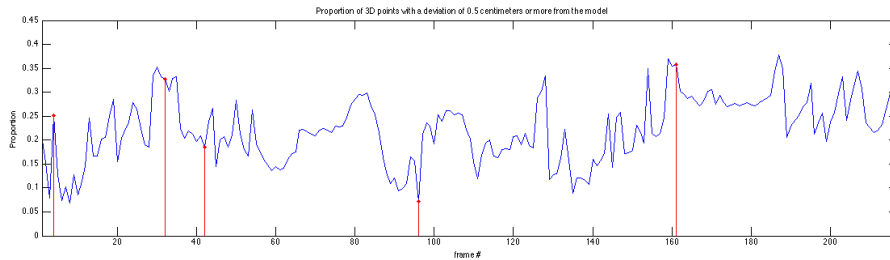
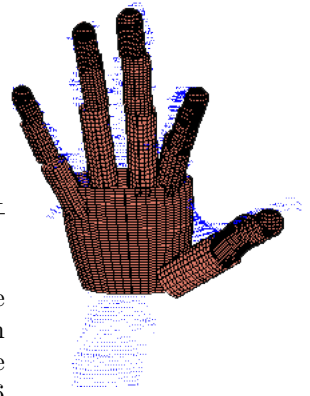


Figure 3.1: Proportion of 3D points deviating more than 0.5 centimetres from the model in the abduction sequence without running the collision correction.

The algorithm successfully tracks the hand and reproduces the correct pose (with some exceptions) for the first three abduction movements. Exceptions are occurring on a few subsequent frames when all fingers lie tight together (i.e. zero abduction). What happens is that fingers cross each other and find matches with the same data points. The tracker does however seem to recover from these bad states pretty fast. For the fourth abduction movement, the thumb get stuck "inside" the hand and stays there for the whole movement. In Figure 3.1 we



can see when the fourth abduction movement starts (around frame 155). We have selected 5 of the frames (marked with red dots in Figure 3.1) for which we have plotted the model together with the corresponding point cloud and the projection of the model to the original photo. These are shown in Figure 3.2-3.6 and they are marked with the grade we gave them in our visual evaluation. Figure 3.2 , 3.4 and 3.5 shows examples of frames graded as GOOD. The poses are correct and the fit looks good.

Figure 3.3 shows frame 32, which was graded an OK frame. The pose is roughly correct with all fingers lying approximately where they should but the middle and little finger do not match the data quite all the way out to the fingertips. The biggest problem however, and the main reason to why there is a peak in Figure 3.1, is that the orientation/direction of the palm is pretty off, which can be seen clearly in the middle plot of Figure 3.3. We believe the problem in this frame is due to the fact that we have been to strict constraining the possibility for fingers to bend slightly backwards. Figure 3.6 shows frame 161, which is a part of the fourth abduction movement and was graded as BAD. The thumb is stuck inside the hand, matching some data points of the index finger.

Classification	BAD	OK	GOOD
Proportion of frames	0.42	0.25	0.33

Table 3.2: Proportion of frames classified as BAD, OK and GOOD respectively for the visual evaluation.

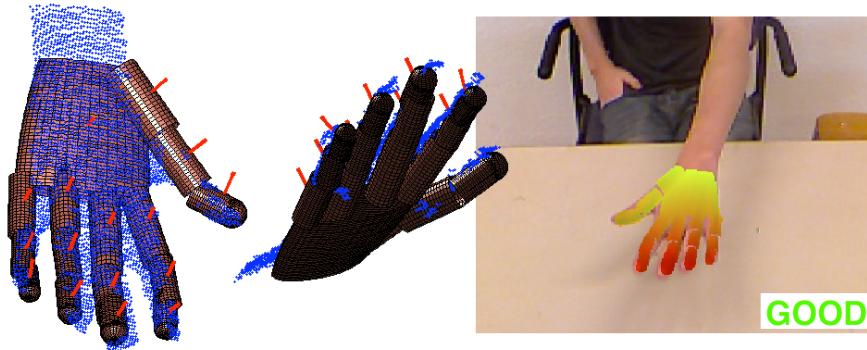


Figure 3.2: 3D plot and image projection for frame 4 of the abduction sequence without running the collision correction.

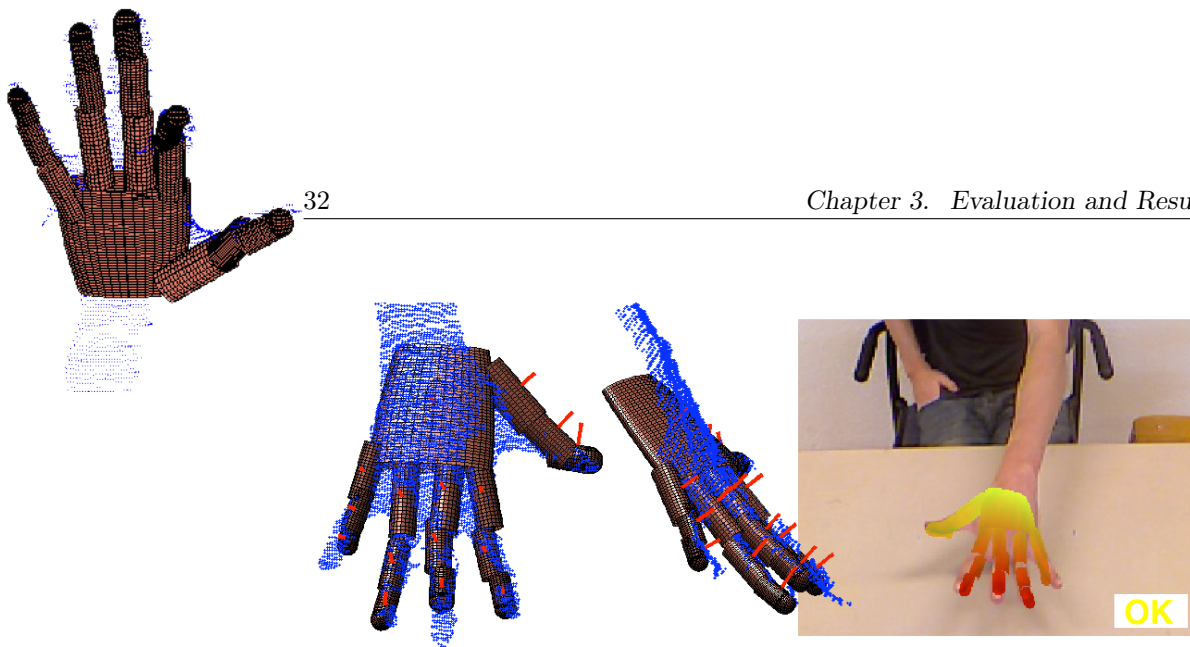


Figure 3.3: 3D plot and image projection for frame 32 of the abduction sequence without running the collision correction.

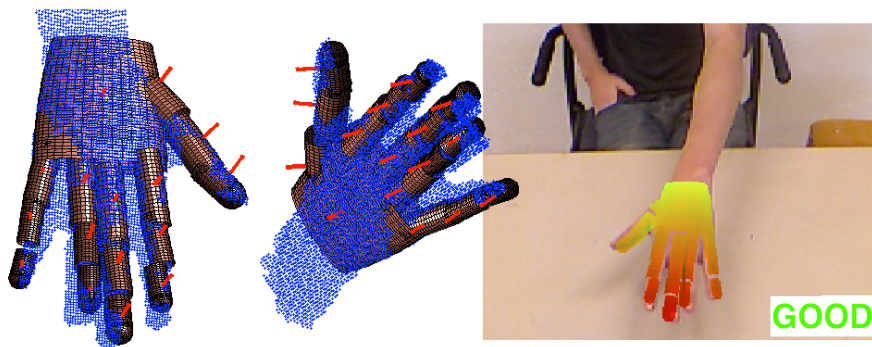


Figure 3.4: 3D plot and image projection for frame 42 of the abduction sequence without running the collision correction.

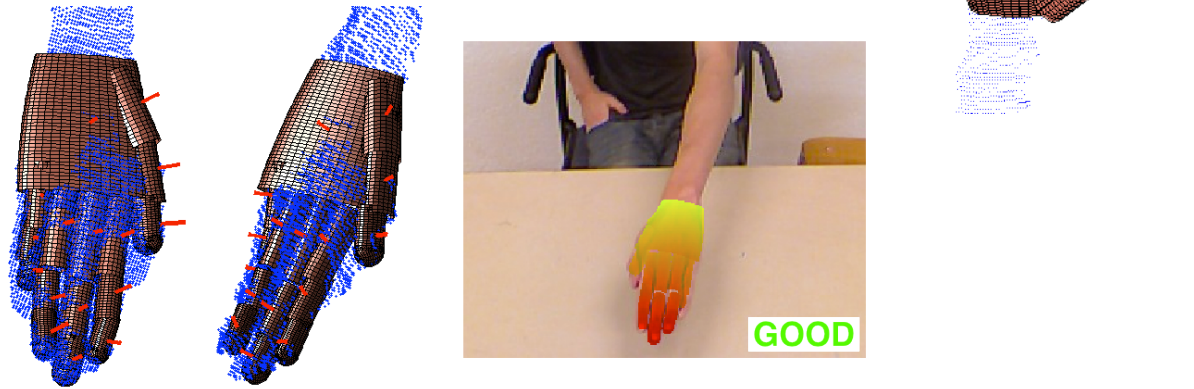


Figure 3.5: 3D plot and image projection for frame 96 of the abduction sequence without running the collision correction.

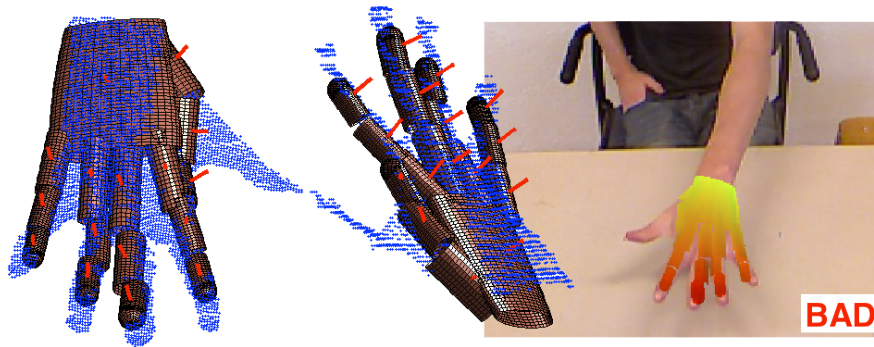


Figure 3.6: 3D plot and image projection for frame 161 of the abduction sequence without running the collision correction.

### 3.3 Abduction Movements with Collision Correction

Classification	BAD	OK	GOOD
Proportion of frames	0.23	0.44	0.34

Table 3.3: Proportion of frames classified as BAD, OK and GOOD respectively for the visual evaluation.



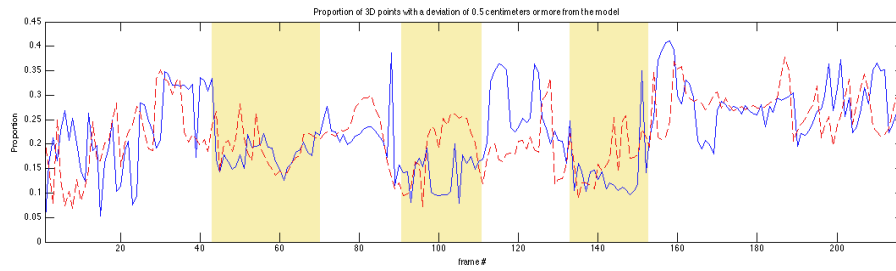
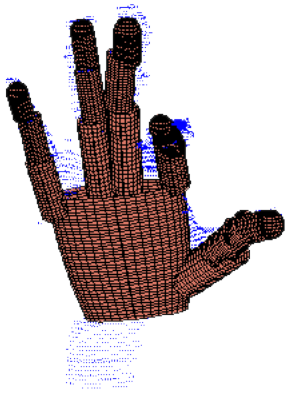


Figure 3.7: Proportion of 3D points deviating more than 0.5 centimetres from the model in the abduction sequence with the collision correction being run when fingers intersect (blue) and without (red). The yellow intervals are where all the fingers lie closely together.

Many frames of the first sequence suffered from problems with intersecting fingers and as we could see, the thumb got totally stuck for the whole of the last movement. We graded over 40 % of the frames as BAD and therefore decided to run the algorithm one more time on the same sequence but this time running the collision correction (described in Section 2.7) for those frames when fingers crossed each other. Figure 3.7 shows the proportion of 3D points deviating more than 0.5 centimetres from the model for each frame and we can see a slight improvement in the intervals where the fingers lie closely packed (yellow areas in Figure 3.7). We can also more easily identify the four abduction movements in-between. When the hand is fully stretched out there are much more points that are not explained by the model than when fingers lie tight together. This is partly due to the webbings (the skin in-) between fingers (especially between the thumb and index finger) that appear when the the hand stretch out. In this sequence, the thumb gets stuck on the index finger during the abduction movement between frames 115 and 130. However, it manages to retain a valid configuration for the fourth movement, contrary to the previous run.

In Table 3.3 we present the result of our visual evaluation. We can see that there are a lot less BAD frames, almost no frames suffered from intersection problems after the collision correction was run. In Figure 3.8 and 3.9 we have plotted two BAD frames from running the sequence without the correction step next to the same frames when running the correction step.

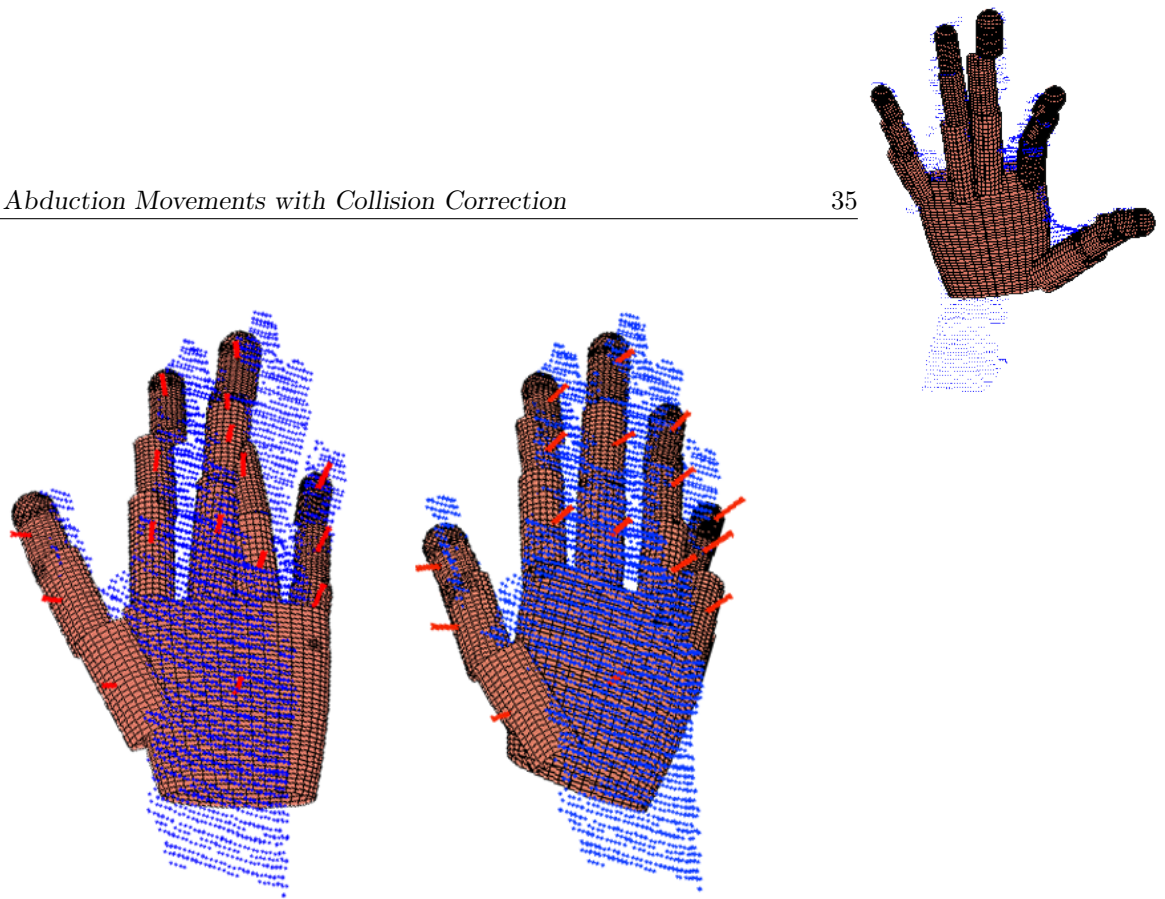


Figure 3.8: 3D plot for frame 43 of the abduction sequence, without and with the collision correction respectively.

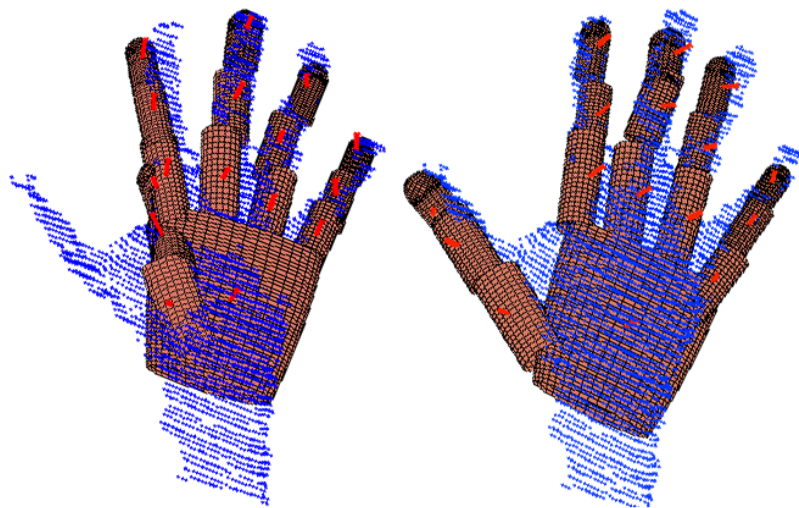
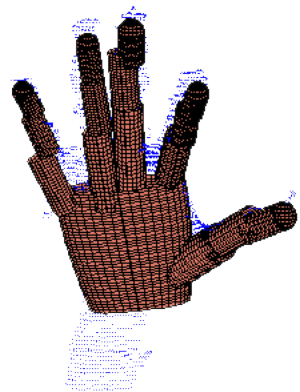


Figure 3.9: 3D plot for Frame 197 of the abduction sequence, without and with the collision correction respectively.





### 3.4 Flex Movements

The second sequence consists of 641 frames ( $\approx 21$  sec) and contains a hand performing flex movements. Figure 3.10 shows the proportion of 3D points deviating more than 0.5 centimetres from the model for each frame.

The algorithm successfully tracks the hand and gives the more or less correct

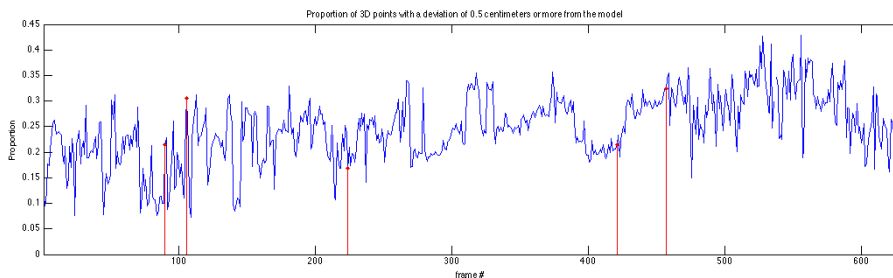


Figure 3.10: Proportion of 3D points deviating more than 0.5 centimetres from the model in the flex sequence.

pose throughout the sequence. The faster changing behaviour in this figure compared to Figure 3.10 is due to the hand is moving around faster in this sequence. This could be resolved by increasing the ranges in the hypothesis generation. Increased ranges would however require an increased number of hypotheses to get as precise matches. Especially the imprecise fits after Frame 500 can be explained by the rapid arm movement during this segment. During a few frames of the segment, after Frame 517, the middle finger positions itself at the ring finger, a similar problem to the one we had in the first sequence. This time, however, the algorithm recovers after only a few frames. At frame 531, the Kinect returned no data points at all, leading the algorithm to ignore the evidence completely. It did not seem to effect the tracking on the following frames notably.

Classification	BAD	OK	GOOD
Proportion of frames	0.06	0.30	0.64

Table 3.4: Proportion of frames classified as BAD, OK and GOOD respectively for the visual evaluation.

This time, we have considered the results to give a much better fit, according to Table 3.4. The fingers are almost always associated with the correct data points, resulting in very few being labeled BAD. We also see the same problems as in the previous sequence with the palm being a bit off in direction, see Figures 3.12 and 3.15. Again, this probably has something to do with angular constraints between the proximal phalanges and the palm. It could also be that the hand has moved too fast in between frames. It is also worth noting that the

resolution was much better in this second sequence than in the first one. The flex sequence contained an average of about 4500 hand pixels per frame, which should be compared to an average of 6000 pixels for the abduction sequence.

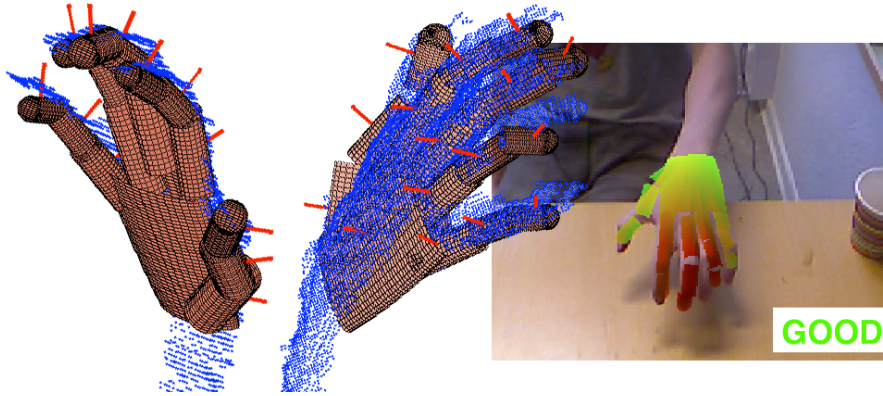
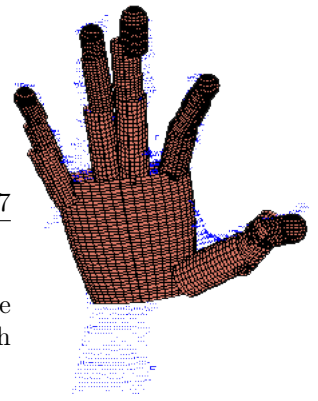


Figure 3.11: 3D plot and image projection for Frame 90 of the flex sequence.

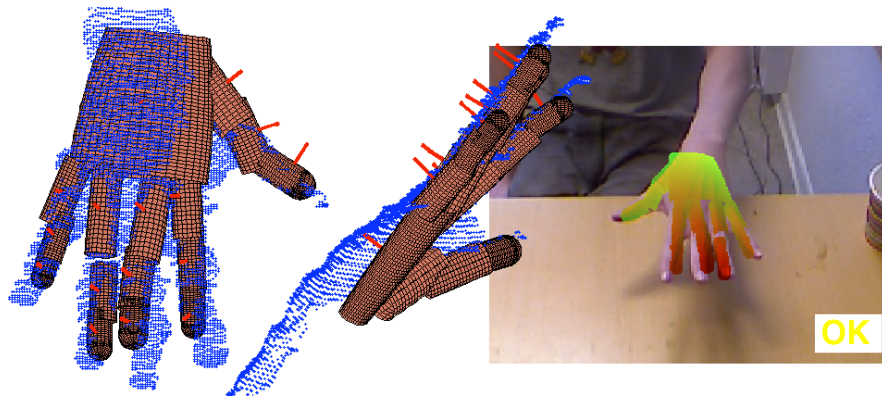


Figure 3.12: 3D plot and image projection for Frame 106 of the flex sequence.

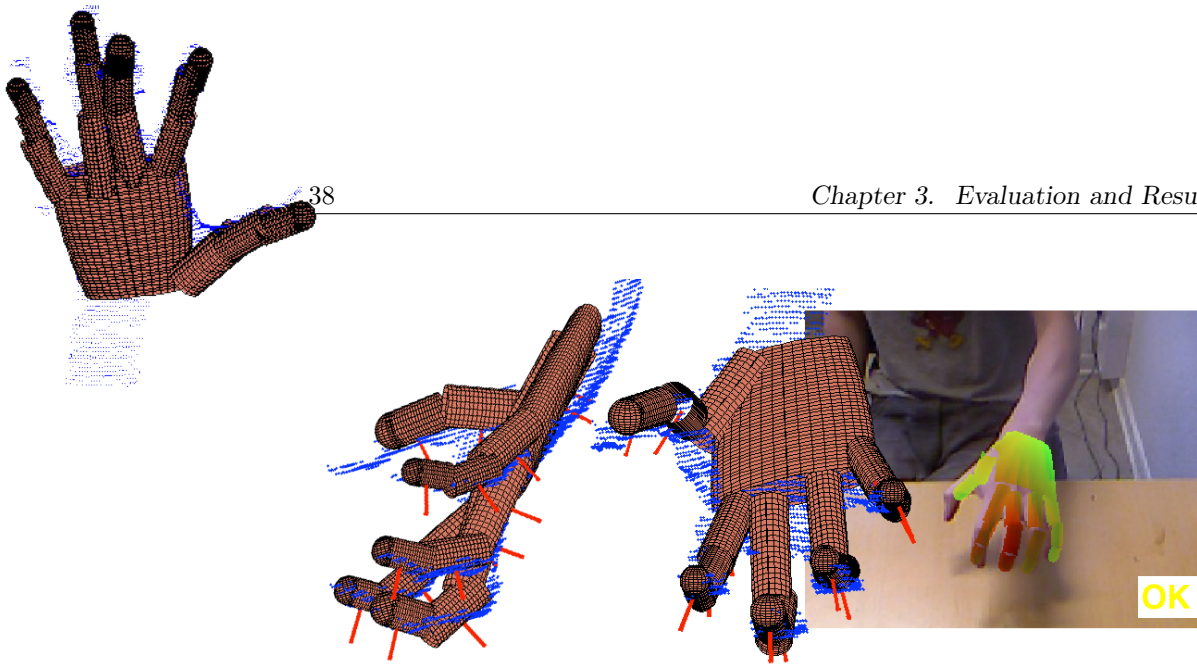


Figure 3.13: 3D plot and image projection for Frame 244 of the flex sequence.

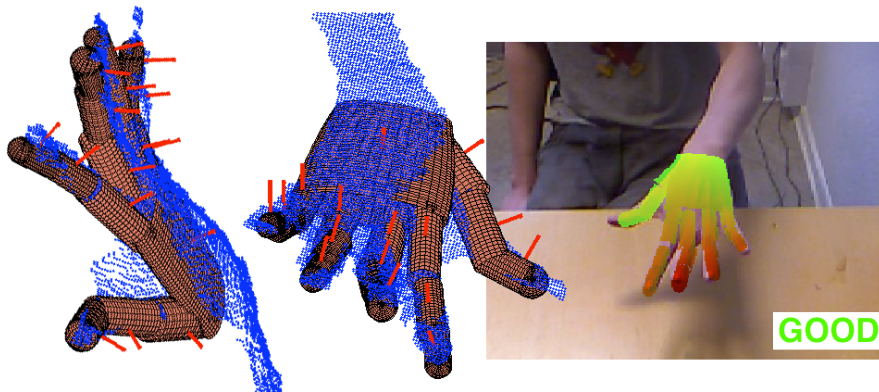


Figure 3.14: 3D plot and image projection for Frame 421 of the flex sequence.

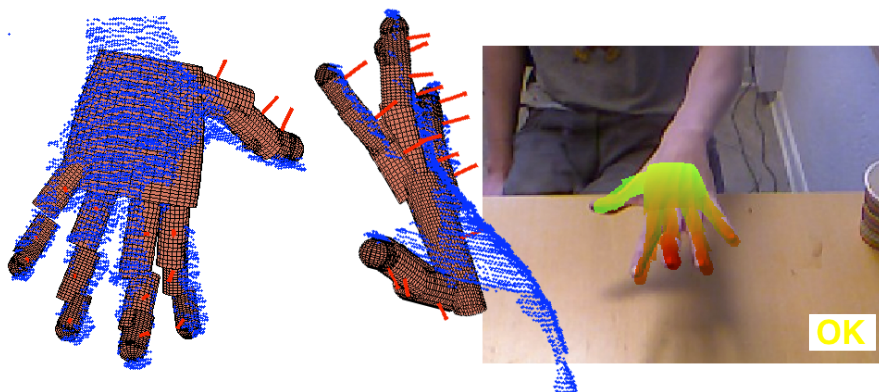
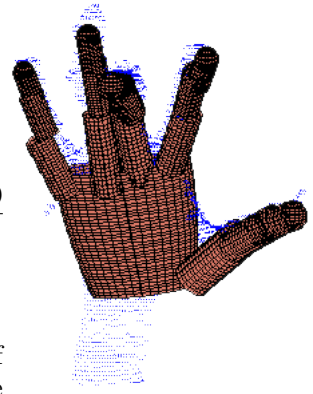


Figure 3.15: 3D plot and image projection for Frame 457 of the flex sequence.



## 3.5 Time Complexity

Below, we have plotted the running time (in seconds per frame) for the Belief Propagation algorithm while varying the number of drawn hypotheses. Figure 3.16 shows the running time against the total number of hypotheses for each phalanx. In Figure 3.17 we have varied the number of drawn hypotheses for the three different kind of hypotheses (position, direction and orientation around direction axis) respectively and plotted the resulting running time in the same figure.

Before running these tests we thought that increasing the number of different kinds of hypotheses would affect the running time differently. This seems not to be the case. Instead all plots below look very similar and the running time seem to grow quadratically with the number of hypotheses for each phalanx. This goes hand in hand with our intuition. In the heaviest step, the algorithm evaluates a total of 30 messages between every hypothesis of two phalanges. This yields a complexity of approximately  $\mathcal{O}(30N^2) = \mathcal{O}(N^2)$ .

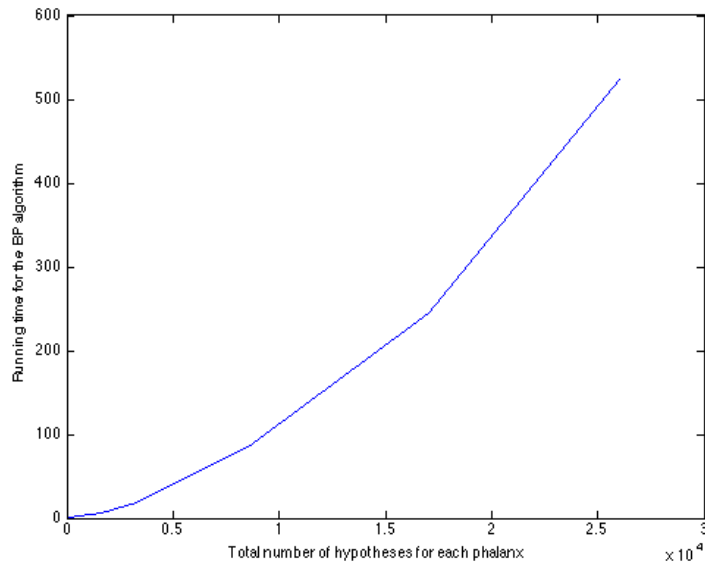
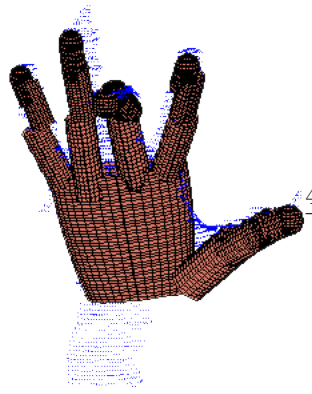


Figure 3.16: Plot of the BP algorithm’s running time (in sec.) against the total number of hypotheses for each phalanx.

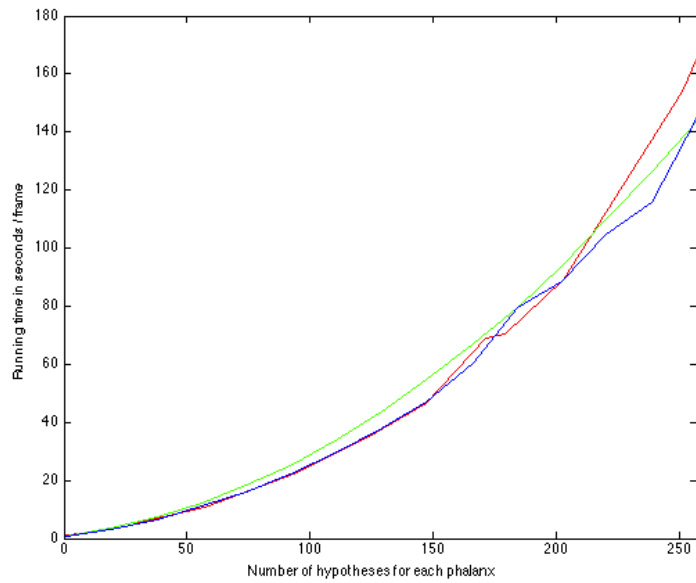
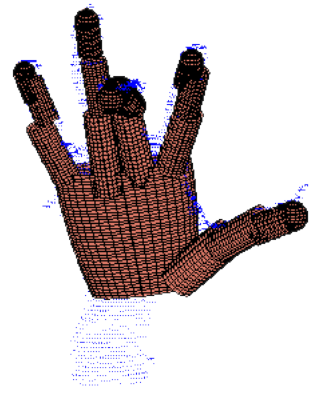


Figure 3.17: Plot of the BP algorithm’s running time (in sec.) against the number of position hypotheses (red), the number of direction hypotheses (green) and the number of hypotheses around the direction axis (blue).



## Chapter 4

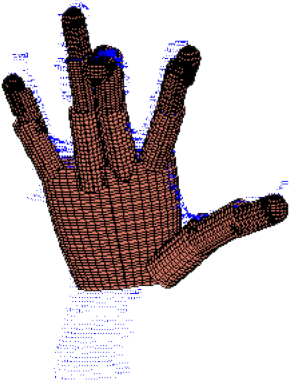
# Conclusions

We have evaluated the algorithm on two sequences, one to test the quality when tracking abduction movements and one for flex movements. One could argue that the test set is too small to come to any conclusions about the performance of the tracker. It is however important to keep in mind that the set contains a total of 858 frames. Even if we can't really tell how it will work over longer sequences, we can say that it will find the correct pose in most situations, with some notable exceptions. If we look at the behaviour of the tracker on a larger scale it finds roughly the correct pose in almost all frames in the test sequences. The collision correction removes most of the frames where the base tracker had problems, i.e. the ones with close fingers. Overall, we came away impressed by how well the Belief Propagation approach works in this particular case. We believe this may be the way forward for developing a fully automated system for medical analysis of precise hand motion.

As we saw when studying the results, the tracker has a tendency to place fingers over each other when the fingers are close together. This is because these structural constraints are not reflected in the basic algorithm. The results from running the collision correction does however seem to take care of this in almost all cases. Another approach opposed to running a separate iteration, would be to introduce loops in the graph and send messages between fingers already in the basic algorithm. With loops in the graph we would however lose the guaranteed convergence and moreover, it would probably be slower since we would need to run more iterations. The collision correction adds some time to the already rather time consuming algorithm but does so only when there is a need for it to run and the fact that it reuses the same hypotheses saves some computation time because it can reuse a lot of calculations, such as the evidence for the different hypotheses.

Even if many of the poses are correct, the fit is often slightly off. This is probably due to a too low number of hypotheses being drawn. With the current implementation and hardware it would be very expensive to refine the sampling with more hypotheses. At the end of the abduction sequence the tracker has some difficulties of finding a good fit due to rapid movement of the hand. A solution to this may be to increase the ranges around the previous parameters, something that would also require an increased number of hypotheses.

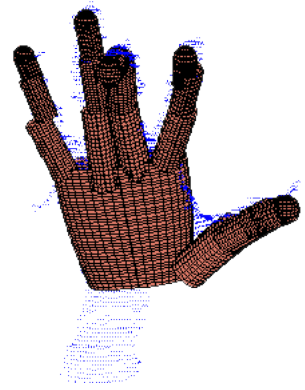




Experimenting with the algorithm, we have found that the tracker is very sensitive to certain parameters. Especially the valid angles for the different joints have shown to be hard to determine. Small changes in the parameters of the "allowed error" does also change the outcome of the algorithm notably. The parameters that we have chosen seem to work well but there is much room for investigation here.

The algorithm works very well given valid previous positions and a large enough amount of data. If the resolution gets too low or parts of the hand get occluded, the tracker encounters difficulties. If the tracker in one frame identifies a false pose, there is a big risk of getting bad poses also for the following frames.

The aim was to develop a fully automated tracker for use in a skilled reaching test setup. In this thesis, we have laid the foundation for such a system. There are still some limitations that need to be overcome to use the system for medical research. Most notably, the fit would have to be improved from what was demonstrated in the test sequences to yield more precise estimates of the kinematic angles. As we will discuss, this can be solved rather easily by doing more iterations or by higher resolution hypotheses sampling. As for now, we also need a manual initialization, as mentioned above, and manually measured proportions of the subject's hand. The need for precise measurements has been reduced from previous implementations by introducing greater variance to the length of the links. Further, it would be practical to speed up the algorithm to be able to run it for more and longer sequences. An implementation in a c-based language instead of MATLAB would probably solve the speed issues. *Hamer et al.* have implemented a very similar algorithm in c++/Cg, which they run in  $\approx 6.2$  sec/frame [3].



## Chapter 5

# Discussion

### 5.1 Initial Guess

As for now, we have initiated the tracking using a manually specified starting position and orientation of the hand. Naturally, we would like to have a fully automated tracker that could find a rough but good starting guess without knowing anything about the configuration from the previous frame. A rough hand pose recognizer like this could also be used to get the tracker back on track when it loses the correct pose completely. This could probably be implemented using the same BP algorithm working over much larger ranges, with successive iterations over finer search spaces to refine the pose.

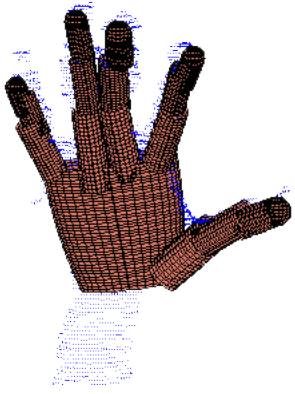
Another approach, which we have tested without success, is to take the result from our arm pose estimate to find a position and direction of the palm. The palm could then be fitted to a plane to find an orientation. The problem for us seems to be finding good starting guesses for the fingers, something that could be solved by the approach above. It may also be possible to do some kind of database search over common poses.

For the lengths of the phalanges, we have introduced a greater allowed variance. Our idea is that the tracking of a hand could be initialized with the same lengths for all subjects and then gradually refined. Depending on how far apart the algorithm estimates the phalanges to be, the lengths can be adjusted in the first part of the sequence. Simultaneously, the allowed variance can be made smaller as we come to a decision on how long the phalanges are.

### 5.2 Using Multiple Kinect Sensors

As mentioned several times earlier, the depth images captured from the Kinect sensor are quite low-resolution. We have also found that its not very reliable, depth data can disappear from parts of the image from frame to frame, giving our algorithm a really hard time to fit a model. We believe that both these problems could be taken care of using more than one Kinect sensor. Our algorithms could very easily be extended to accommodate more sensors. One would





need to calibrate the camera system (i.e. computing the Kinect sensors relative position and orientation) and instead of computing the mean projection error in one camera, take the mean in both. The only possible problem we can think of is that the structured dot patterns might interfere with each other, something that seems to be solvable [19].

Another thing to keep in mind is that the current version of the Kinect has been out since 2010. There is speculation about a new version being released in the near future. This will most probably have a higher resolution. Especially a larger amount of pixels in the depth image would benefit our algorithm. Added resolution would also have very little impact on the speed of the algorithm. The algorithm could also easily be made to work with any camera producing a depth of field image.

### 5.3 Handling Self-Occlusion and Object-Occlusion

One of the goals was to be able to track hands interacting with simple objects. Due to lack of time we haven't however had the time to implement and test this. As for now our algorithm will not work very good when hand points are occluded by some object or even when parts of the hand itself occlude other parts of the hand. *Hamer et. al* [3] suggest a very easy solution to this, namely to calculate the evidence dependent on the fraction  $\alpha$  of unoccluded pixels within the patch in question. Our evidence function (2.13) would then become

$$\phi(x) = \frac{1}{Z} \exp \left( -\alpha \left( \frac{\bar{\delta}}{\sigma} \right)^2 \right),$$

and thus they yield the same results for when no pixels are occluded, and make all hypotheses as likely when a phalanx is totally occluded. To label occluded pixels one could just look for data within the patch, that is substantially closer to the camera than predicted by the current model hypothesis. One could also extract the same patch from the photo to label object pixels as those closer to the camera (in depth) that are not skin coloured. Much of the current self-occlusion problems would however get solved by just using multiple Kinect sensors.

### 5.4 Smarter Hypothesis Generation

In order to speed up the algorithm or increase the number of hypotheses, the hypotheses generation could be improved by studying the depth data. For example, if there is no data in the depth image corresponding to a certain 3D point, it is probably a bad hypothesis for the position of a phalanx. It would also be possible to calculate better ranges for  $\theta$  and  $\phi$  hypotheses based on the depth data and the fact that the corresponding 3D points are quite closely situated.

Another improvement could be to keep track of the larger motion of the hand and adjust the parameter ranges to take the current velocity of both position and direction into account.

# Bibliography

- [1] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly, “Vision-based hand pose estimation: A review,” *Computer Vision and Image Understanding*, vol. 108, no. 1-2, pp. 52–73, Oct. 2007.
- [2] I. Oikonomidis, N. Kyriazis, and A. Argyros, “Efficient model-based 3D tracking of hand articulations using Kinect,” *Proceedings of the British Machine Vision Conference 2011*, pp. 101.1–101.11, 2011.
- [3] H. Hamer and K Schindler, “Tracking a hand manipulating an object,” *Vision, 2009 IEEE*, no. Iccv, 2009.
- [4] T. Palmér, M. Tamtè, P. Halje, O. Enqvist, and P. Petersson, “A system for automated tracking of motor components in neurophysiological research,” *Journal of Neuroscience Methods*, 2012.
- [5] I. Q. Whishaw and S. M. Pellis, “The structure of skilled forelimb reaching in the rat: a proximally driven movement with a single distal rotatory component.,” *Behav Brain Res*, vol. 41, no. 1, pp. 49–59, 1990.
- [6] A. Klein, L.-A. R. Sacrey, I. Q. Whishaw, and S. B. Dunnett, “The use of rodent skilled reaching as a translational model for investigating brain damage and disease.,” *Neuroscience and biobehavioral reviews*, vol. 36, no. 3, pp. 1030–42, Mar. 2012.
- [7] Y. Qian, G. Lei, F. X. Castellanos, H. Forsberg, and R. D. Heijtz, “Deficits in fine motor skills in a genetic animal model of ADHD.,” *Behavioral and brain functions : BBF*, vol. 6, Jan. 2010.
- [8] J. Doan, I. Whishaw, S. Pellis, O. Suchowersky, N. de Bruin, and L. Brown, “Challenging context affects standing reach kinematics among parkinson’s disease patients.,” *Behav Brain Res*, vol. 214, no. 1, pp. 135–41, 2010.
- [9] V. Castaneda and N. Navab. (Oct. 2012). Time-of-Flight and Kinect Imaging, [Online]. Available: [http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors\\_LabCourse\\_Kinect.pdf](http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors_LabCourse_Kinect.pdf).
- [10] C Prema and D Manimegalai, “Survey on Skin Tone Detection using Color Spaces,” *International Journal of Applied Information Systems*, vol. 2, no. 2, pp. 18–26, 2012.
- [11] V. Vezhnevets, V Sazonov, and A Andreeva, “A survey on pixel-based skin color detection techniques,” *Proc. Graphicon*, 2003.
- [12] W Skarbek, A. Koschan, and Z Veroffentlichung, “Colour image segmentation-a survey,” no. October, 1994.

- 
- [13] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, “Comparison of surface normal estimation methods for range sensing applications,” *2009 IEEE International Conference on Robotics and Automation*, pp. 3206–3211, May 2009.
  - [14] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
  - [15] J. Pearl, A. Science, and L. Angeles, “Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach,” *AAAI-82 Proceedings.*, pp. 133–136, 1982.
  - [16] J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Understanding Belief Propagation and its Generalizations,” *Intelligence*, 2001.
  - [17] A. Gelfand, M. Fuentes, P. Guttorp, and P. Diggle, *Handbook of Spatial Statistics*, ser. Chapman & Hall/CRC Handbooks of Modern Statistical Methods. Taylor & Francis, 2010, pp. 194–195, ISBN: 9781420072877.
  - [18] Y. Weiss, “Belief propagation and revision in networks with loops,” no. 1616, 1997.
  - [19] K. Berger, K. Ruhl, and C Brümmer, “Markerless motion capture using multiple color-depth sensors,” *Vision, Modeling and Visualization*, 2011.

## Appendix A

# Belief Propagation Proof

Belief Propagation is really just a clever way of reorganizing the sums in the marginals derived from (2.7). We will consider the tree graph  $T = (V, E)$ , where  $V$  are the nodes and  $E$  the edges. We let  $X_v$  denote the variable corresponding to a node  $v \in V$ . We want to verify that the belief computed for a node variable  $X_a$  is the marginal  $p_{X_a}(x_a)$ . To do this we place the node  $X_a$  at the top of the tree and consider it the root. This can be done for all nodes, also the leaves. The adjacent nodes of  $a$  are then roots of the subtrees springing out from those nodes. We will refer to the subtree with root  $r$  as  $T^r = (V^r, E^r)$ . We will

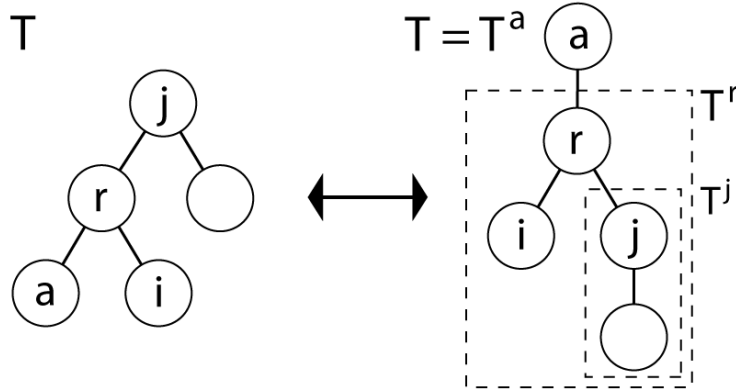


Figure A.1: A tree can always be arranged so that any node can be considered the root.

pursue an induction based proof, starting at the leaves and iterating upwards toward the root. The message from one of the neighbours  $X_j$ ,  $(j, r) \in E^r$  to the subtree root is given by

$$m_{j \rightarrow r}(x_r) = \sum_{x_j} \left( \psi_{j,r}(x_j, x_r) \Phi_j(x_j) \prod_{k \in N(j) \setminus r} m_{k \rightarrow j}(x_j) \right). \quad (\text{A.1})$$

We now look at the subtree with root  $j$  below  $r$ ,  $T^j$ . By (2.9) the belief for the node variable  $X_j$  in this subtree is given by

$$b_j^j(x_j) \propto \Phi_j(x_j) \prod_{k \in N(j) \setminus r} m_{k \rightarrow j}(x_j). \quad (\text{A.2})$$

Inserting this into (A.1) gives us

$$m_{j \rightarrow r}(x_r) \propto \sum_{x_j} \psi_{j,r}(x_j, x_r) b_j^j(x_j).$$

We now consider the induction approach. Assume that the belief of the root in the subgraphs really are the correct marginals  $p_{X_j}^j$ . This clearly holds for the leaves, since the subgraph contains only one node and according to (2.7) the marginal (joint probability in this case) is simply equal to  $\Phi$ ; the same as the belief (A.2). Hence the base step is proven. We proceed with the induction step. We consider the subtree  $T^r$  at any height (might be the whole graph  $T^a$ ) in the tree with root  $r$ . Assume that the neighbours of  $r$  in  $T^r$  have correct beliefs in their respective subgraphs. The belief for the root  $X_r$  in the subtree is then

$$b_r^r(x_r) \propto \Phi_r(x_r) \prod_{j \in N(r)} \sum_{x_j} \psi_{j,r}(x_j, x_r) p_{X_j}^j(x_j), \quad (\text{A.3})$$

where  $p_{X_j}^j(x_j)$  can be developed according to (2.7). The first sum should be understood as the sum over all configurations of states in  $T^j$  with  $x_j$  fixed,

$$p_{X_j}^j(x_j) \propto \sum_{(x_k)_{k \in V^j \setminus j}} \left( \prod_{\substack{(m,n) \in E^j, \\ m > n}} \psi_{m,n}(x_m, x_n) \cdot \prod_{i \in V^j} \Phi_i(x_i) \right).$$

Inserting this into (A.3) gives us

$$\begin{aligned} b_r^r(x_r) &\propto \Phi_r(x_r) \prod_{j \in N(r)} \sum_{x_j} \psi_{j,r}(x_j, x_r) \sum_{(x_k)_{k \in V^j \setminus j}} \left( \prod_{\substack{(m,n) \in E^j, \\ m > n}} \psi_{m,n}(x_m, x_n) \cdot \prod_{i \in V^j} \Phi_i(x_i) \right) \\ &= \Phi_r(x_r) \prod_{j \in N(r)} \sum_{(x_k)_{k \in V^j}} \psi_{j,r}(x_j, x_r) \left( \prod_{\substack{(m,n) \in E^j, \\ m > n}} \psi_{m,n}(x_m, x_n) \cdot \prod_{i \in V^j} \Phi_i(x_i) \right). \end{aligned}$$

Here the sums of the configurations for each subgraph are multiplied together. One configuration (element) in one subgraph (sum) is multiplied with another configuration in another subgraph. This is equivalent to summing over the possible configurations for all subgraphs,

$$b_r^r(x_r) \propto \Phi_r(x_r) \sum_{(x_k)_{k \in V^r \setminus r}} \prod_{j \in N(r)} \psi_{j,r}(x_j, x_r) \left( \prod_{\substack{(m,n) \in E^r, \\ m, n \neq r, \\ m > n}} \psi_{m,n}(x_m, x_n) \cdot \prod_{i \in V^r \setminus r} \Phi_i(x_i) \right)$$

$$= \sum_{(x_k)_{k \in V^r \setminus r}} \left( \prod_{\substack{(m,n) \in E^r, \\ m > n}} \psi_{m,n}(x_m, x_n) \cdot \prod_{i \in V^r} \Phi_i(x_i) \right) = \sum_{(x_k)_{k \in V^r \setminus r}} p^r(\bar{x}).$$

The last equality is given by the definition of the joint probability on the pairwise MRF (2.7). The resulting expression is exactly equal to the marginal  $p_{X_r}^r$  by definition. This concludes the induction step and thus the proof, we have shown that  $b_a = p_{X_a}$ .  $\square$

The proof for the Max-Product algorithm in Section 2.5.1 is exactly the same with the sums exchanged for maxima and the marginals exchanged for the maximum of the joint probability given one of the states.

Master's Theses in Mathematical Sciences 2012:E42  
ISSN 1404-6342  
LUTFMA-3236-2012  
Mathematics  
Centre for Mathematical Sciences  
Lund University  
Box 118, SE-221 00 Lund, Sweden  
<http://www.maths.lth.se/>