Preface

This report documents the work performed by project group CVG10-1024 during the 9th and 10th semesters of Computer Vision and Graphics at Aalborg University, Denmark. The work was mainly carried out during a stay at the University of Otago, New Zealand.

The bibliography and appendices are found at the back of this report. Enclosed with the report is a CD containing animations of the test and learning sequences used, and the archived pose estimations. Furthermore the CD contains an article written on the basis of this project. Literature is referenced using brackets containing an index number such as [1].

Tables, figures and equations are numbered sequentially in each chapter throughout the report and referred to as e.g. figure 1.1, table 4.1 and equation 6.2. When referencing to a numbered list "§" is used, e.g §3 is a reference to the third item of the list.

Constants and variables in equations are denoted using italic letters e.g. x while vectors are denoted with small bold letters e.g. $\mathbf{a}$. When nothing else is mentioned, coordinate systems are right hand systems. In the report "." is used as decimal point so the number $2,050.52$ is read two thousand and fifty point five two.

The report is written using **LaTeX** and **BibTeX**, and is set with 10 point Times. All figures are made using Matlab, Visio 2000, CorelDraw or Paint Shop Pro.

We would like to thank Brendan McCane, Geoff Wyvill, Sui-Ling Ming-Wong and the rest of the Graphics Lab at the University of Otago for making the exchange possible, their guidance and providing a friendly atmosphere.

Aalborg University
June 10th, 2002

Michael Vittrup

Mads K. D. Sørensen

Contents

I Analysis **1**

1 Introduction **3**
- 1.1 Definitions and Terms 4
- 1.2 The Watching Window 5
- 1.3 Initiating Problem 8

2 Previous Work **9**
- 2.1 Appearance-Based Approach 9
- 2.2 Model-Based Approach 10
- 2.3 Initial Experiments 11

3 Problem Definition **15**
- 3.1 Intentions 15
- 3.2 System Overview 16
- 3.3 The Hand Model 17
- 3.4 Synthetic Input 17
- 3.5 Outline of Report 19

4 The Human Hand **21**
- 4.1 Medical Definitions 21
- 4.2 Constraints 21
- 4.3 Modelling 25
- 4.4 Summary 26

II Search Algorithms **29**

5 Numerical Optimisation **31**
- 5.1 Introduction 31
- 5.2 Cost Function 32
- 5.3 Extremes 33
- 5.4 Search Strategies 34
- 5.5 Search Optimisations 35
- 5.6 Evaluation of the Algorithms 36
- 5.7 Summary 38

6 Spider Optimisation **39**
6.1 The Snake Principle 39
6.2 The Spider 41
6.3 Model Fitting 44
6.4 Changing Tree Topology 52
6.5 Modifying the Node Structure 53
6.6 Splitting the Estimation 58
6.7 Summary 61

7 Downhill Simplex **63**
7.1 Algorithm 63
7.2 Constraints and Downhill Simplex 66
7.3 General Behaviour 70
7.4 Summary 76

8 Simulated Annealing **79**
8.1 Analogy 79
8.2 Theory 80
8.3 Downhill Simplex and Simulated Annealing 81
8.4 Parameter Estimation 81
8.5 Summary 87

9 Comparison of Algorithms **89**
9.1 Performance 89
9.2 Speed Discussion 92
9.3 Perspective 93
9.4 Summary 94

III Closing 95

10 Alternative Descriptors **97**
10.1 Shape Descriptors 97
10.2 Theory 98
10.3 Function Sequence 99
10.4 Results 99
10.5 Summary 101

11 Conclusion **103**
11.1 Summary 103
11.2 Concluding Remarks 105

12 Process Analysis **107**
12.1 The Project 107
12.2 Our Plans 107
12.3 Chronological Description 108
12.4 Final Remarks 109

IV Appendix **115**

A Workbench **117**

B Gestures and Learning Sequences **119**

C Wilcoxon Signed-Rank Test **125**

D Error Examples **129**

E Test Results **131**

Part I

Analysis

Introduction

Previous Work

Problem Definition

The Human Hand

Chapter 1

Introduction

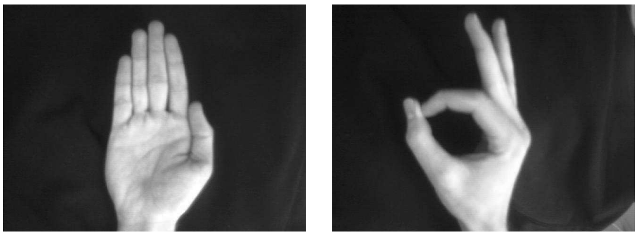
> "Artificial Intelligence: The art of making computers that behave like the ones in movies."
> - *Bill Bulko*

Within the last 10 years the computer has found its way into virtually everybody's life. Whether at work or at home, people spend countless hours in front of the computer screen working, playing games, chatting, browsing the Internet etc. These days, interaction with the computer is mostly done by keyboard and mouse - devices designed decades ago. This communication with the computer, restricted and unnatural to the human, has to be learned. Communicating on the computer's terms can be a source of frustration, often resulting in human creativity being lost.

Much of today's research focuses on developing new ways to interact with computers, aiming at making the interaction more natural. This research area is commonly known as *Human Computer Interaction* (HCI), and covers a wide range of areas within the technical as well as humanistic sciences.

As the speed of computers has dramatically increased and cameras have become accessible to everybody, the computationally intensive research area of computer vision has become a growing part of HCI. Within the computer vision area of HCI, the vast majority of research has been on gesture and body language recognition (see Moeslund and Granum [2] for a comprehensive survey).

Humans use gestures to communicate non verbally in every day life. A common example is a nod with the head to signal agreement. Often humans communicate feelings and emotions in this way rather than through speech. As such, communication with gestures comes naturally to humans.



(a) Stop gesture (b) OK gesture

Figure 1.1: **Two well know hand gestures.**

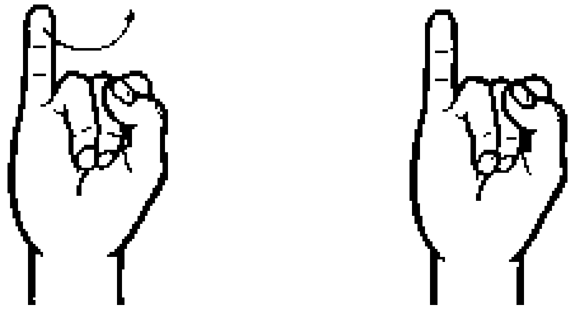
The hands especially are used to gesticulate different meanings, two well known examples of hand gestures being the *Stop* and *OK* signs depicted in figure 1.1 next page. These gestures are well known across country and language borders[1].

One aim of recognizing gestures could be to replace the mouse as a way to interact with the computer. Combining this with e.g. speech recognition instead of a keyboard, communication with the computer would be more on human terms than vice versa.

Once we can see the computers and interfaces as conduits and not restrictions, the phrase "learn computers" will hopefully never be spoken again.

1.1 Definitions and Terms

In the scientific literature dealing with gesture recognition, the word gesture is treated with some ambiguity. In general the literature defines two kinds of gestures: *Dynamic* and *static*. A dynamic gesture must be viewed over time to be understood, such a gesture is shown in figure 1.2(a). Taking one snapshot as the hand is moved from the start position to the end position does not provide information that this is the letter “j” being shown using sign language. As opposed to the dynamic gesture, a static gesture as the sign “i” shown in figure 1.2(b) can be recognized from just one snapshot: Taking into consideration only the configuration of the hand, the gesture can be understood.



(a) Sign letter ’j’ (b) Sign letter ’i’

Figure 1.2: **Examples of gestures from the sign language, (a) dynamic gesture, (b) static gesture. Taken from [3].**

Another term often used is *pose*. In general computer vision terms this word is often interpretated as meaning the position and orientation of an object. This report will be focused on hand poses, with the meaning of the word specified a little. Throughout this report the word pose, unless otherwise stated, is defined as:

Pose: The position, orientation *and* configuration of an object.

When referring to hand poses, the *configuration* means how the individual finger joints are bent.

As such, one can regard static hand gestures as a subset of possible hand poses. Note that a hand pose can be any of the gestures shown in earlier figures, but a pose does not necessarily have a specific meaning to an observer, contrary to gestures. A *gesture* in this report is defined as:

---

[1] Although in certain cultures or in specific situations, the “OK”-sign on figure 1.1(b) is interpreted as a “zero” or a “nothing”.

Gesture: A pose which can be interpreted as having a defined meaning to the observer (although sometimes ambiguous).

Thus, the famous “V”-sign symbolising victory as shown by Sir Winston Churchill, a fist, an open hand, sign letters etc. are all gestures.

An abbreviation used in the scientific literature and this report is *DoF*, referring to Degrees of Freedom. This is a single number specifying how many parameters are free to vary in the model. If only the position of an object in space is changeable, the object has three DoF representing its (X, Y, Z) coordinates, respectively. To be able to also rotate the object another three DoF (one for each axis) would be added, and so forth.

1.2 The Watching Window

As already indicated in the above, this project deals with hand pose estimation, something of utmost importance in an application called “The Watching Window” ([4]), a system currently under development at the University of Otago, New Zealand.

1.2.1 Description

The aim of The Watching Window, named so because it “watches” the user, is to create an inexpensive environment in which a user can experience and interact with three dimensional models. An overview of the system is provided in figure 1.3.

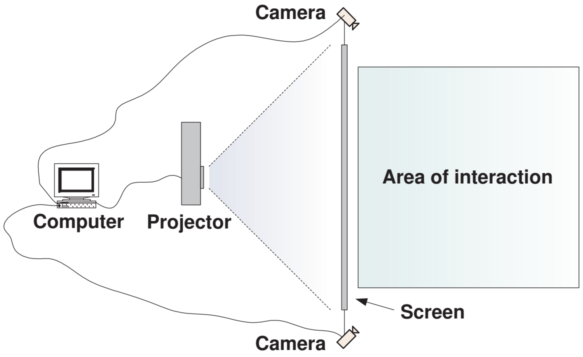
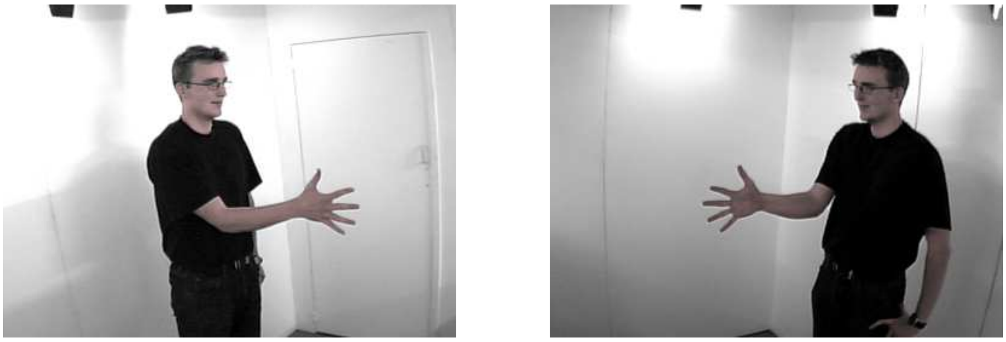


Figure 1.3: **The fundamental setup of The Watching Window.**

In this system, stereo computer vision is used to track the movement of the user interacting with the 3D model. The two cameras provide input images of which an example is given in figures 1.4(a) and 1.4(b) on the following page. In turn these input images are segmented using an advanced background subtraction algorithm, resulting in a silhouette of the user. Using heuristics and simple contour analysis the system estimates the position of the head and hand in the two images, marked with crosses in figures 1.4(c) and 1.4(d).

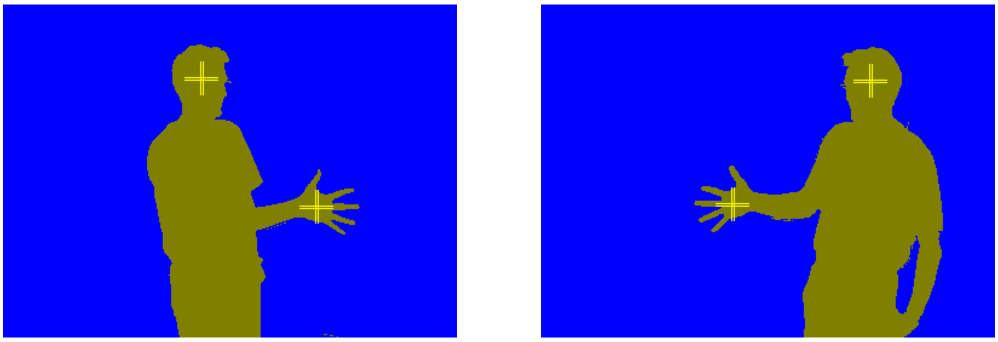
Triangulation is applied to compute 3D positions of the head and hand, based on the two 2D positions of each. The head position is used to determine the viewpoint of the user and from this present a correctly rendered 3D model on the screen, and tracking the hand position in space allows for simple interaction.

The illusion of a 3D model freely floating in space is at this stage created using the same technique as seen in some movies during the 1970's. By presenting slightly different images to each eye, using red/green glasses, the user is tricked into believing that the model is in front of the screen (or behind), rather than "on" it. More sophisticated presentation techniques are in the melting pot, though.



(a) Left camera image

(b) Right camera image



(c) Left segmented image

(d) Right segmented image

Figure 1.4: **Figures (a) and (b) show an example of the input from the cameras, and the segmented images are shown in (c) and (d). The crosses indicate the estimated location of head and hand.**

1.2.2 Existing Demonstrations

At the Innovation Conference in Christchurch[2], New Zealand, March 2002, The Watching Window was for the first time presented to the public by Kee-Kim Heng and the authors of this report. For this conference, a few simple applications were written:

The Desert

To demonstrate head tracking, an application showing a window frame and an outdoor desert scene was implemented; as seen from inside a room.

When the user steps closer to the screen, looking down, the ground and windowsill will be visible. If the user ducks his or her head and looks up, more of the sky will become visible; stepping from side to side reveals other parts of the landscape and window frame.

---

[2] http://www.innovate.org.nz/

Figure 1.5: **Looking out the window at a desert scene.**

The Ringing Bells

A simple application demonstrating basic interaction with the virtual world is a 3D model of three bells of different sizes positioned "in front" of the screen as depicted in figure 1.6. The hand is tracked and if one of the bells is "touched", a corresponding ringing tone to the size of the bell sounds, and the "touched" bell swings. A simple melody can be composed this way. Again, it is possible to duck and look up inside the bells, or step from side to side to see the scene from a different perspective.

Figure 1.6: **Touch the bells and compose a simple melody.**

Asteroids

Furthermore, the user has a chance of "boldly going where no-one has gone before" when playing the arcade game "Asteroids" (not to be confused with the old classic), illustrated in figure 1.7. The head is tracked, and by using more or less athletic movements the goal is to avoid the asteroids floating against the player while trying to collect the crystals.

1.2.3 Long Range Goal

The long-range goal for The Watching Window is to develop a console which allows for interactive sculpturing and modelling: A user views a virtual lump of clay floating freely in space in front of the screen. By applying pressure with fingers on various places on this lump, the artist will be able to model a sculpture. Eventually, visualisation and interaction with objects as diverse as machine parts, pottery, clothing and biological structures is desired.

Hence the ultimate goal is to let artists interact freely with a 3D model, without the use of restrictive gloves or wires. At this point the user can only interact with one entire hand

Figure 1.7: **Avoid the rocks and collect the crystals in the game "Asteroids".**

(the position in space, only); however, artists and designers use considerably more than "just a hand" to sculpt. Both hands and all fingers are used intensively when modelling, thus the complete pose of each hand must be deduced at an "acceptable" frame rate; depending on the specific application.

1.3 Initiating Problem

While distinguishing between the two hands is a project in itself, this project deals with pose estimation, using the existing Watching Window setup. Using the existing setup involves certain limitations and demands to be fulfilled, topics that will be discussed later. An initiating problem for this project can be stated as:

> .. creating a system capable of estimating the pose of both hands in real-time using the stereo vision setup of The Watching Window.

To solve this problem it is important to consult the literature on approaches already taken to the problem of pose estimation. Such a survey may give leads to which approaches can be taken and just as important give an indication of whether or not pose estimation of the hand is feasible. After the survey, the aim of this project can be more tightly defined.

This is done in the following two chapters - chapter 2 presents a survey on previous research in this field and based on this, the aim of this project is specified in chapter 3.

Chapter 2

Previous Work

> "Sir Isaac Newton once said, 'If I am considered a great man, it is only because I have stood on the shoulders of other great men.'
> ... in computing we mostly stand on each other's feet."
> - *Dr. Richard Hamming*

An extensive amount of work has been done in the area af human pose estimation using vision. While speech recognition and synthesis were picked up as research areas for HCI since the early days of computing, computer vision has gained momentum during the last decade due to faster hardware. Most of the research has taken place only within the last decade, and the literature available on the topic is extensive. Most of the literature surveyed seems to concentrate on gesture recognition, that is, a limited number of predefined hand poses.

A great number of surveys/taxonomies have been published, including Moeslund and Granum [2], Gavrila [5], Huang and Pavlovic [6], Pavlovic et al. [7], Cohen [8]. These try to cover the research field as a whole and give a good introduction to the major problems and obstacles to be conquered. In this chapter, emphasis will be on applications concerning hand gestures / pose estimation, and applications estimating body pose will not be presented. One should keep in mind, though, that many of the same principles apply to body pose as well as hand pose estimation.

A brief survey will be given here in order to present different approaches considered and the current stage of development with respect to hand pose estimation.

The survey by Pavlovic et al. [7] divides gesture/pose estimation into two main approaches: *Appearance-Based* and *Model-Based*; these will be discussed in the next sections, with emphasis on the model-based approach as it seems to be the most feasible technique with respect to The Watching Window.

2.1 Appearance-Based Approach

The appearance-based approach is based on the apperance of the hand in the image. The "raw" camera data is preprocessed and used as input to a classification routine, basically a function measuring the likeliness distance from the input data to a set of predefined possible gestures. The input is classified as a member of the class to which its distance is smallest, using some classification routine - Bayes Classifier, K-nearest, neural networks, template matching etc.

Yonghua [9] presented a gesture recognition system for human vehicle interaction. The hand is segmented from the background using a neural network based on human skin colours, and using descriptors such as moments and signatures, the system successfully uses template matching to classify a set of predefined gestures.

A sophisticated example is the use of image eigenvectors by Birk et al. [10]. In this work gestures from the international sign language alphabet are recognised using Principal Component Analysis (PCA). The system consists of off- and on-line parts. The off-line part consists of training the system with numerous images of each gesture. From these learning images features are extracted using the PCA technique. Only the best features, yielding largest differences between gestures, are selected to form a basis for the on-line part. In the online part of the system, the image to be analyzed is mapped to the feature space extracted in the off-line part. Using a statistical classification algorithm the image is classified, and a specific gesture is reported as the result.

These methods are typical examples of appearance-based approaches, in which the system has no knowledge of what object it is trying to recognize. The above methods can just as well be applied to recognize different human faces or classify fruits or bolts, basically by applying other sets of training images. These methods as well as other methods belonging to the appearance-based approach are applicable when the goal is to recognize hand gestures, seen from a more or less fixed viewpoint.

However, if the goal is to determine the pose of the hand, these approaches generally come out short. Consider using the PCA method for pose estimation. To do this, one should obtain several images of each hand pose seen from all possible angles, to be able to train the system. Considering the complexity of the hand, this represents a hard if not an impossible task.

2.2 Model-Based Approach

As opposed to appearance-based recognition, the 3D model-based approach assumes prior knowledge of the object for which pose is to be estimated. In the case of estimating the pose of a hand, a model of the hand is used in the process. These models range from simple skeleton abstractions to full sophisticated 3D models of the hand.

One example is the use of active contours (snakes) to track a hand on a desktop [11, p.275]. Snakes employ the use of deformable 2D templates and furthermore predicts the next pose by use of eg. a Kalman filter. However, the method has only been used to recover the position of a hand in an image, and for simple gesture classification.

The majority of applications applying 3D models use the *analysis-by-synthesis* approach to estimate the hand pose. The model is synthesized at a certain abstraction level and compared to the input data. The various abstraction levels for comparing model and image data are: Silhouettes, edges, sticks, joints, blobs, depth, texture, motion and contours [2].

Although the first articles on analysis by synthesis were published in the early 1980s, the technique really gained momentum in the middle of the 90s as the accelerating speed of hardware made the technique more feasible.

Rehg and Kanade [12] presented a complete hand gesture recognition system named *DigitEyes*. This system uses a three dimensional cylindrical kinematic model of the human hand utilizing 27 DoF. Fingertips and links were chosen as the matching features. Both mono- and stereovision are used, where the fingertips are extracted using edge-based analysis. This system uses dedicated hardware for the image processing part and assumes uniform background. Furthermore the problems of scaling and occlusions are not considered.

An application not requiring special hardware was presented by Ouhaddi and Horain [13]. Using biomechanical constraints they simplify the hand to 21 DoF and compares surface overlapping to contour distance, as well as different optimization methods. However, no global rotation or scale factor are considered in their work.

Davis and Shah [14] also presented a solution where no special hardware is required. Based on a silhouette image of the hand, heuristics are applied in order to determine the finger orientation *perpendicular* to the camera axis. Once the finger regions are identified, straight homogeneous generalized cylinders are fitted to the outermost phalangeal of each finger and thumb. By fitting cylinders to the fingers and thumb, trajectories in 3D rather than 2D can be found. The authors show how their system successfully recognizes the transformation of the hand in all three dimensions. Also an example of tracking abduction/adduction is given using the Star Trek character Spock's "Live Long and Prosper" gesture. Constraints to the system are the inablility to rotate the hand and to flex fingers.

Kameda [15] proposes two matching algorithms to estimate the pose of the hand. Both uses the silhouette as the base of the matching, evaluating the overlapped region. The first algorithm, *Order Independent Strategy*, has the ability of partial parallel processing while the second, *Order Dependent Strategy*, according to the author gives a good estimated pose similar to the original pose. Again monovision as well as a good deal of constraints are expected. The forearm axis must be perpendicular to the camera direction, and the distance between the camera and forearm is assumed to be known.

Stenger et al. [16] uses truncated quadrics and an Unscented Kalman Filter as the basis for a 7 DoF hand model, representing global transformation and thumb extension, to simulate a "point-and-click interface". For an arbitrary number of cameras, the profiles of the model are computed and compared to images from a video sequence. The model is deformed to minimize the geometric error between the projections of the model and edges detected in the images. Their algorithm is easily extendible to a distributed system, and using a single viewpoint it runs at 3 frames per second on a Celeron 433 MHz computer. A serious constraint is the fact that the forearm apparently must be more or less perpendicular to the camera direction.

Wesseling [17], who did a feasibility study with respect to pose estimation in The Watching Window environment in 2001, basically tried to locate the finger tips in an image sequence and match these tips to the correct fingers. Unfortunately, the system has too many fundamental constraints, uses monovision, and *again* the forearm is required to be perpendicular to the camera axis. Furthermore no global rotation is taken into account and the system is not stable enough to be used in a commercial appliciation. Nevertheless, his approach is interesting and intuitive.

2.2.1 Summary

Generally, all of the surveyed techniques seem to require the forearm to be more or less perpendicular to the camera axis. This consequently means that the user is spatially and rotationally restricted to an awkwardly unacceptable small field of activity. As an example, one cannot point in the direction of the camera because of this limitation.

Removing this constraint is of utmost importance if pose estimation is to be commercially applied.

2.3 Initial Experiments

The literature survey was a source of inspiration to us, and some unsuccessful attempts were initially made to apply a voxel representation technique and inverse kinematics to analytically solve the problem of pose estimation as discussed in the following sections.

2.3.1 Volume Intersection

Instead of using only 2D images, as in many of the methods discussed, people have used multiple viewpoint vision to reconstruct 3D voxel representations of the object which are often subject to further processing. Slabaugh et al. [18] gives a survey of some techniques applied in this field.

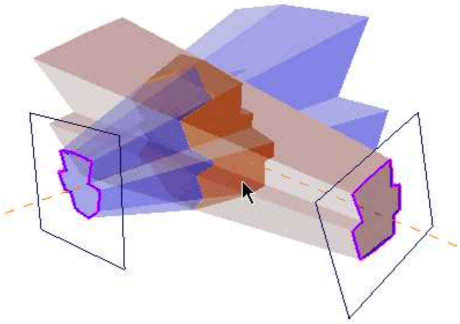


Figure 2.1: **The Volume Intersection Approach (picture from [19]).**

Figure 2.1 shows the basic principle of volume intersection. A series of spatial regions is obtained by "carving" a cone, based on the camera focal points (as convergence point) and the image captured by the cameras. The volume ("visual hull"), shared by these cones more or less approximates the unknown object.

A specific example of hand analysis using volume intersection is presented by Ueda et al. [20]. Figure 2.2(d) shows their resulting voxel model, obtained from the silhouettes in figures 2.2(a)-2.2(c). This appearance-based voxel model was in their article subject to further model-based analysis, where an articulated 3D model was deformed to fit inside the voxel representation.

(a) Front view (b) Top view (c) Side view (d) Voxel model

Figure 2.2: **The voxel representation in figure 2.2(d) is obtained using the three silhouettes in figure 2.2(a) to 2.2(c).**

With promising and visually appealing results like these in mind (although there were major problems with slow and wrong pose estimation in [20]), the volume carving was with some effort initially implemented, to allow for volume intersection experiments. In practice, a cube of $40 \times 40 \times 40$ voxels was used to represent the hand; each voxel was set according to whether its pixel projection was set or not set in both camera frames.

It became clear, however, that stereo vision did *not* provide enough information to carve a satisfactory spatial representation of the hand.

Figure 2.3 visualises why this approach is not feasible using only two cameras. The hand is seen from above, and the voxels in the interference area will be set as false positives ("ghosts")

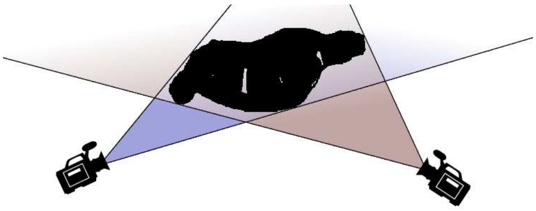


Figure 2.3: **Volume intersection problem; notice the ghost voxels in front of the hand.**

as their projections indeed *are* set in both camera image frames. The ghost voxels behind the hand (seen from the cameras) were expected, but the false positives in front of the object came as a rash surprise. A third camera (as used in [20]) will solve this problem - the reader of this report is located at the position of this required third camera, and indeed it is easy to see which false positives should be removed from the voxel representation.

This method was dropped as The Watching Window uses stereo vision only, and no immediate solution could be found to the arised problems.

2.3.2 Inverse Kinematics

Another approach discussed and investigated further was the use of kinematics. Kinematics are largely used in robot control to describe the robots arm. Not unlike a human arm, a robot arm consists of a series of links, comprised in what is known as a chain, connected by joints. Say the arm has to reach out to grab an object. To move the robot arm to the object one can manually specify all joint angles, termed *forward kinematics* (FK) since these joint angles determine the shape of the arm. If the robot is to work in a highly controlled environment, where the object will always be at a certain position, it may be sufficient to provide the joint angles to the control program. A great improvement is to employ *inverse kinematics* (IK) instead. Staying with above example IK can be described as: *Given the position of the object, which joint angles are suitable to move the arm to this position?* Using IK to estimate feasible joint angles makes the robot more versatile and able to work in a more unconstrained environment.

FK is straightforward because a set of joint angles specifies exactly one point in space. Going the other way is harder as there in general exists more than one solution. To intuitively verify this fact, consider how many different ways a person is able to reach out for a cup. The number of solutions literally explodes, the more joints the person is allowed to use.

For an excellent dicussion on Inverse Kinematics and various solutions, see Welman [21].

Only the paper by Wu and Huang [22] seems to apply IK to the problem of hand pose estimation. In their work the pose estimation is split into two parts; estimation of the palm and estimation of finger positions. Each finger is modeled as a kinematic chain and its pose estimated by IK. Using image processing the position of the fingertips are identified in the image, and used as input to the IK algorithm. However, as the fingers are highly articulate multiple pose solutions are obtained for each finger. Hence other techniques to choose one of these solutions must be applied. Furthermore as the position of the fingertips are used as input to the IK algorithm, positions of all fingertips must be obtained, implying that occlusion is not allowed to occur.

Further studies in this area revealed that only less articulate human pose problems have been solved using IK. An example is due to Tang et al. [23] where the pose of the arm is estimated. But even in this small case where only the shoulder and elbow joint are to be estimated, magnetic tracking is needed to determine the exact positions of shoulder, elbow and hand in space.

Another problem is the fact that in order to apply IK, the joints must be located in the images. This is relatively straight forward if the hand is perpendicular to the camera axes (see [17]), but as soon as the hand is freely rotated and translated in space, occlusions will occur. Also, some protrusions can be located in each image, but it is not trivial to decide which protrusions from one image correspond to which protrusions from the other, if found, at all.

This was attempted as well, but as other of the surveyed approaches seemed more feasible, use of inverse kinematics was rejected.

Chapter 3

Problem Definition

> "Ok, wait a moment here. Hold on. There.
> Now you see the change was made in real time."
> \- *Frame showperson*

The literature survey in chapter 2 revealed that although people have presented solutions to the problem of pose recognition, using a variety of different techniques, one common characteristic is shared: Restricting constraints.

To the best of our knowledge, constraints apply to all solutions proposed so far in this field. The solutions may not run at an acceptable frame rate, they may not take global translation or rotation into account, they only deal with a single hand, they may be an attempt of applying only monocular vision, they may require special hardware, the hand may be synthesized to a too simple abstraction level, they may have problems with occlusions and they may simply not be stable or precise enough for commercial "real-life" applications.

The solution for The Watching Window has to deal with global translation and rotation; a combination that is avoided in most other solutions presented so far. The user should be able to use both hands, and as The Watching Window uses a stereo approach, this project deals with pose estimation using stereo vision. No special hardware can be expected as a fundamental aim of The Watching Window project is to create a cheap, affordable virtual reality environment. Furthermore, the estimation has to deal with a sufficiently complex hand to be usable to artists sculpturing a virtual lump of clay. It has to respond to user interaction within a short time limit, and as occlusions are common, the software should be able to deal with them as well. It goes without saying that the stability and precision are of high concern, too.

In other words: Constraints are undesirable. What is needed for The Watching Window project, is

...an efficient, stable, flexible, stereo-vision based fast pose estimation algorithm which allows for the use of both hands.

3.1 Intentions

It is not possible to solve all problems in one go - Rome was not built in a day. Furthermore, as stereo vision applied to pose estimation is a relatively untried approach, this project can be looked upon as a feasibility study; merely exploring some of the possibilities and analyzing what is possible and what is not. New methods and some well-known algorithms from the world of monocular vision will be presented and applied to this specific case and some test results will be compared and discussed.

Taking these statements into consideration, a more realistic project goal is formulated: What is needed for the Watching Window project *at this stage of development*, is

> ...an investigation of whether pose estimation is achievable, taking the requirements and setup of The Watching Window into consideration, and gain insight to some approaches which may or may not lead to the ultimate goal, i.e. to let designers interact freely with a 3D model, *without restrictive gloves or wires.*

In this feasibility study, the speed of the pose estimation algorithm is considered less important. A real-time solution is not pursued at this point.

When experimenting with approaches to solve problems in this field, the primary goal is a solution that works; the speed is of secondary importance. Algorithm simplifications and optimizations are possible *after* a feasible, but most likely slow, solution has been found.

3.2 System Overview

As the study of literature showed an analysis-by-synthesis approach to be most feasible and promising, this approach is chosen. A flowchart of the approach pursued is shown in figure 3.1.

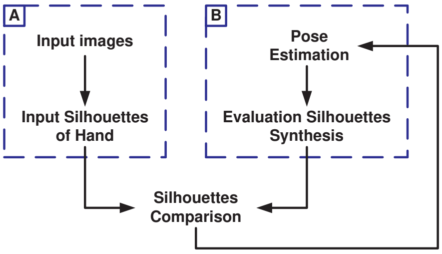


Figure 3.1: **Flowchart of the approach. Input images are acquired from The Watching Windows cameras (box A). The hand silhouettes are obtained using simple image processing techniques. Using a deformable 3D model of the hand, evaluation silhouettes are generated. The input and evaluation silhouettes from (A) and (B) are compared, and based on this result a new and hopefully more feasible pose is estimated. This iterative improvement is repeated until a sufficient correspondence between silhouettes is reached, or a certain number of evaluations have been carried out.**

The input to the system is images of the user, one from each camera as shown in figures 1.4(a) and 1.4(b) on page 6. Using contour analysis, the silhouettes of the hand are acquired from each of theses images, henceforward know as the *input silhouettes.* This all happens inside box A in figure 3.1.

The second part of the system is enclosed by box B. A pose estimation of the hand is proposed by the system, and synthetic silhouettes reflecting this pose as seen from the cameras are generated. These are henceforward known as the *evaluation silhouettes.*

These evaluation silhouettes are compared to the input silhouettes and a measurement of their correspondence level is derived. This is fed back to the pose estimator, which estimates a new and hopefully more feasible pose. Evaluation silhouettes of this new pose estimation are synthesized, and yet again compared to the input silhouettes (in box **A**).

This feed-back loop is repeated until the correspondence between the input- and evaluation silhouettes is acceptable, new pose estimations do not improve the correspondence, or a predetermined number of cycles is completed.

Once the algorithm returns, the well-known pose of the evaluation model hopefully reflects the correct pose of the real hand. The correspondence measure is a single value where a low value theoretically indicates high pose correspondence and vice versa.

Thus the pose estimation algorithm becomes a matter of numerical optimization where the aim is to minimize the correspondence measurement between the input silhouette and the evaluation silhouette, rendered using the hand model presented in the next section.

3.3 The Hand Model

The approach described requires a 3D model of the human hand, but most biological structures are overwhelmingly complex; the hand being no exception. The human wrist alone has eight bones, and the rest of the hand consists of 19 more - a relatively large percentage of the total 206 bones of the skeleton ([24]). On top of this, soft tissues and the interactions among ligaments, tendons, muscles, nerves and cartilage influence the movement.

Thus a simplification is necessary to reduce the complexity of the hand, reflected as the number of DoFs. Conversely the number of DoF cannot be reduced too much without affecting the precision of the pose estimation required for an artist to use the system. An indepth discussion on these topics is presented in chapter 4, page 21, resulting in the model pictured in figure 3.2. Even at this relatively coarse abstraction level the model is still dauntingly complex as shall become clear throughout this report. Therefore, in order to ease this feasibility study, a fundamental precondition is assumed - the use of synthetic input images.

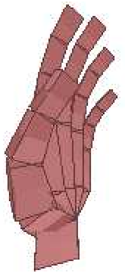


Figure 3.2: **The hand model used throughout this project, shown in a neutral position.**

3.4 Synthetic Input

As a main difference to most other research on this topic, this project uses synthetic generated input silhouettes. Instead of using pictures of a real human hand, the same 3D model as is used for the model fitting is used to render the input silhouettes. In effect, box A in figure 3.1 is replaced by a module rendering synthetic input silhouettes as theoretically seen from the two cameras, see figure 3.3. The reason for this priority is that if it is not possible to achieve

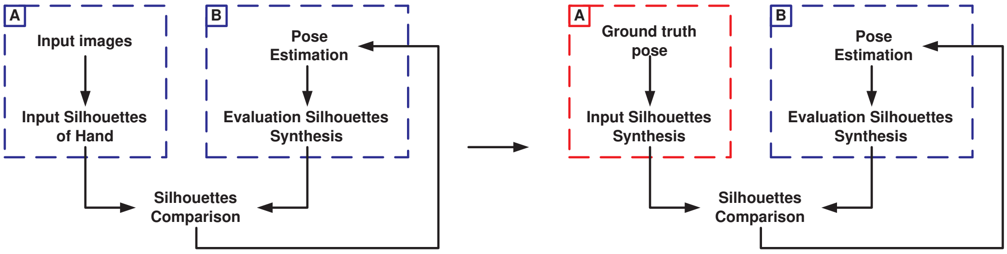


Figure 3.3: **Modification of the flow chart. Instead of using real images and applying image processing techniques to deduce the input silhouettes, synthetic input silhouettes are utilized in this project.**

acceptable performance on synthetic images showing a single hand, it certainly is not possible on real-life images either.

As discussed in chapter 2, scientists worldwide have struggled to achieve acceptable pose estimations under constrained conditions. As this project deals with the removals of these constraints, it was chosen to go with the synthetic solution as this clearly lightens the task.

The main benefits of using synthetic input data are:

1. Easy evaluation of fitting. If images of a real hand are used, little is know of the actual pose of the hand. In the vast majority of research on this topic, the success of fitting is visually evaluated, which must be regarded as a subjective evaluation. By using synthetic input silhouettes, the ground truth pose of the hand is known, thus allowing for an objective evaluation of the success of the fitting.

2. Test scenarios. Applying synthetic input silhouettes gives the capability of easy creation of new test scenarios, as well as repeating old ones. Minor well-defined changes can be applied to investigate how the algorithms perform when facing these changes.

3. Preprocessing of input images is not necessary. Using real images, image processing techniques must be applied to generate input silhouettes. By using synthetic input silhouettes this step is redundant and no image artifacts influence the matching process.

4. An optimal result is known to exist. As the input silhouettes are rendered using the same 3D model used for the evaluation silhouettes, an evaluation silhouette which matches the input silhouette perfectly is known to exist. If real images were used, the 3D model must be customized to the hand which pose is to be estimated. As any 3D model is a more or less rough approximation of a real hand, this customization will never be perfect. As a result, it is not known whether pose estimation problems are caused by model limitations/constraints, segmentation or other more fundamental problems.

5. No camera calibration. By generating synthetic input silhouettes, the parameters of the cameras are known beforehand. This also opens up for the testing of different camera settings if needed, including placement and focal length.

By utilizing synthetic input silhouettes, the focus of this project is laid on pose estimation, model deformation and silhouette comparison.

At this point it should be stressed that it is *not* an optimal solution. To get a realistic idea of how the algorithms behave they should be presented for real-life data, with the associated problems - especially segmentation and differences between the model and a real human hand

(§3). What is possible using synthetic data may not be possible using real-life data, and vice versa.

An environment is developed to test different camera configurations, various pose estimation algorithms, etc. This environment, referred to as the "Workbench", is briefly presented in appendix A.

3.5 Outline of Report

The outline for the rest of this report is as follows:

In chapter 4 the anatomy of a human hand is discussed, constraints are applied and finally a 3D model of the hand is presented in details.

In part II, starting page 31, numerical optimization is discussed and three algorithms are investigated further in chapter 6, 7 and 8, respectively. Chapter 9 concludes on the numerical optimization part, by presenting a series of comparative tests between the three algorithms. Different things are of interest and therefore analyzed in the three approaches, thus the three chapters do not share same basic composition and different plots are presented.

In part III of the report, possible further refinements are discussed, and the results presented in the report are concluded upon.

Chapter 4

The Human Hand

> "No man should marry until he has studied anatomy
> and dissected at least one woman."
> - *Honore de Balzac*

The decision of using a 3D model based approach demands a certain adfunderstanding of the anatomy of the human hand. In this chapter the structure of the hand is analyzed, certain constraints to reduce DoF are discussed and finally a 3D model for further use in this report is presented.

As the surveyed literature uses different ways to define bones, joints and the DoF for each joint, it is decided to apply the definitions from Teo [25]. The reason for choosing this reference over the medical literature is that the hand is synthesized to a complex enough abstraction level to give an artist the basic freedom desired, yet not as overwhelmingly complex as presented in the medical literature. The hand is one of the most complex bone structures in the human body, and due to this fact numerous studies on the anatomical structure of the hand have been conducted. For more in-depth medical information, see [24] for an excellent pedagogical introduction to the human skeletal system.

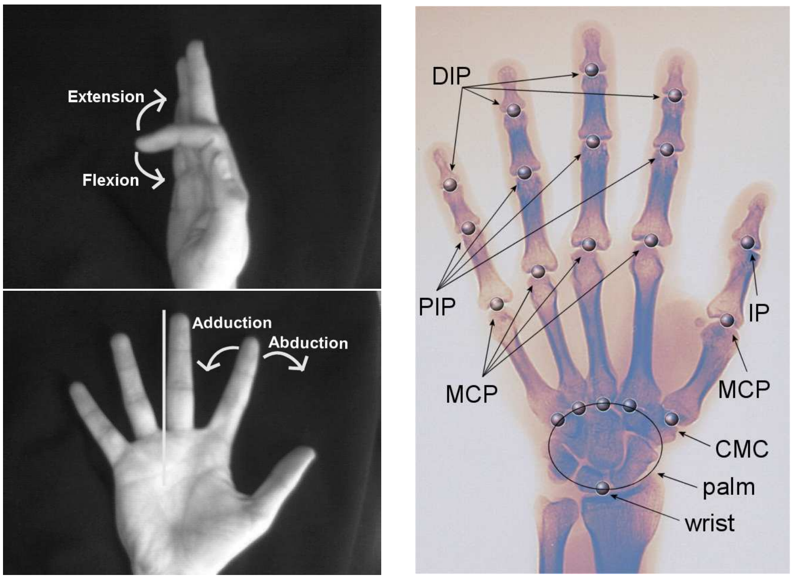
4.1 Medical Definitions

The human hand is built up from components named wrist bones, palm bones and digit bones, and at this stage it is important to introduce the basic medical terms used when discussing finger movement so as to avoid confusion, see figure 4.1(a). *Flexion* and *extension* refer to the movement of a finger respectively towards and away from the palm, whereas *abduction* and *adduction* refer to the movement of a finger away from or closer to the plane dividing the middle and ring finger (perpendicular to the palm).

Figure 4.1(b) shows an x-ray of a hand and joint abbreviations. As can be seen, the palm consists of numerous bones, allowing for a wide variety of movements; thereby being very hard to model. For this reason the palm itself is treated as a rigid object, as this approximation decreases complexity significantly.

4.2 Constraints

From a medical point of view, the human hand is a dauntingly complex structure, whereas a 3D modeller roughly can adapt the hand using 27 DoF. However; the human hand, although



(a) Medical terms (b) X-ray of a human hand.

Figure 4.1: **(a) Medical terms for basic finger movements, the Flexion/Extension and Abduction/Adduction. (b) X-ray of a human hand, with matching joint abbreviations. Notice especially the complexity of the palm.**

articulate, is also statically and dynamically constrained. By examining and applying these constraints the DoF can be decreased, which later will turn out to be an advantage.

Constraints can be split up into *static constraints*, dealing with absolute limits for the movement, and *dynamic constraints* which deal with inter- and intra-finger dependencies. These are discussed in the following subsections.

Theoretically, probabilities for gesture and movement sequences can be derived, but because this is highly application dependent and not within the scope of this project, it is not investigated any further.

4.2.1 Static Constraints

Static constraints can be split up into anatomical constraints, referring to the extreme limits of each joint and application-related constraints, representing feasible constraints with respect to The Watching Window console.

Anatomical Constraints

Henceforward, the abbreviations introduced in figure 4.1(b) are used to identify a specific joint - as an example, the *MCP* joint refers to the first knuckle of a finger.

Because of the fact that the thumb is a more complex mechanism than each of the other four fingers, it is treated separately in the following. Each of the four fingers consists of three joints. Starting with the joint closest to the palm, they are named:

- *MCP*, the *meta-carpophalangeal* joint. Two DoF, a flexion-extension movement of 90 degrees and an abduction/adduction range of approximately 60 degrees.

- *PIP*, the *proximal interphalangeal* joint.One DoF, with a flexion-extension movement of 100 degrees.
- *DIP*, the *distal interphalangeal* joint. One DoF, and with a flexion-extension range of approximately 60 degrees.

This yields four DoF for each finger, resulting in 16 DoF for four fingers. The thumb, too, has three joints but an extra DoF applies as indicated here:

- *CMC*, the *carpometacarpal* joint. Two DoF, a flexion-extension movement of 45 degrees and an abduction/adduction range of as much as 120 degrees.
- *MCP*, the *metacarpal-phalangeal* joint. Two DoF, with a flexion-extension range of 50 degrees and an abduction/adduction range of approximately 30 degrees.
- *IP*, the *interphalangeal* joint. One DoF, with a flexion-extension movement of 85 degrees.

This yields five DoF on top of the 16 DoF from the fingers, in total summing up to 21 DoF.

In table 4.1 the DoF and static joint ranges of the simplified hand are summarized. Also the spatial rotation and translation of the hand, discussed in next subsection, must be taken into consideration, hereby requiring an additional six DoF. This yields a total of 27 DoF to be initially considered.

Hand element	Joint	DoF	Flexion/extension (θ)	Abductuion/adduction (δ)
Finger	DIP	1	60	-
“	PIP	1	100	-
“	MCP	2	90	30
Thumb	IP	1	85	-
“	MCP	2	50	30
“	CMC	2	120	45

Table 4.1: **DoF and joint limits of the fingers and thumb (from Teo [25]).**

Application-Related Constraints

An artist using The Watching Window is expected to face the screen, this seems feasible and acceptable. This allows for the following constraints with respect to wrist rotation: Around the x-axis, the rotation is constrained to lie within [0; 180] degrees, rotation around the y-axis must lie within $[-90;\ 90]$ degrees and z-rotation is constrained to $[-180;\ 180]$ degrees.

To estimate the position of the hand, The Watching Window uses triangulation. Because the position is only estimated within a certain accuracy, this element of uncertainty is complied with by *not* fixing the position of the model to this position, but allowing translation along each of the x-, y- and z-axes. This spatial translation is constrained to lie within a margin of ± 5 centimetres from the last estimated position of the hand[1]. These are the only values that are not angles; yet they can for simplicity be looked upon as simply three additional dimensions.

These constraints allow the user to work within a hemisphere in front of the screen, as seen in figure 4.2.

---

[1] Some overlapping *must* occur from frame to frame, though, as described later

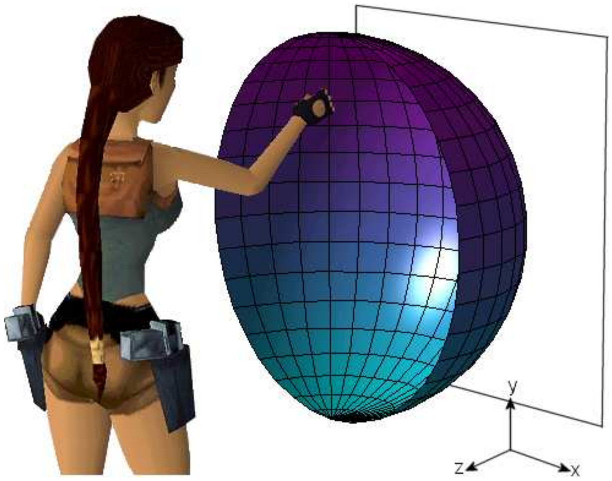


Figure 4.2: **An artist or designer using The Watching Window is expected to face the screen, and is allowed motion within the hemisphere shown in front of the screen.**

4.2.2 Dynamic Constraints

In spite of the fact that the hand itself has many DoF in its simplified form (table 4.1), many of these degrees are interdependent. This is caused by inter-joint dependency constraints (internal biomechanical muscular interconnections of the hand). As an example, the DIP and PIP of a finger cannot be flexed or extended independently unless an external force is applied. These constraints are called *intra-finger constraints*.

A large number of dynamic constraints can be expressed and modelled using relatively simple equalities or inequalities. Other constraints such as *inter-finger* constraints are difficult to express in closed form, these still need further investigation and will not be discussed any further as they are not within the scope of this project.

MCP Abduction and Adduction

The level of abduction/adduction (δ) of a finger varies with the flexion and extension (θ) of that finger. If the MCP joint is fully extended, the range of δ is maximized, whereas if the joint is fully flexed, the range is minimized. This relationship can be expressed using formula 4.1 (Vardy [26]):

$$\delta_{MCP} = \frac{\theta_{MCP} \times (\theta_{min} - \delta_{max})}{90^{o}} + \delta_{max} \tag{4.1}$$

The constant θ_{min} is the flexion angle where maximum range of δ is observed (stretched finger), δ_{max} is the maximum abduction/adduction angle, observed at the angle of θ_{min}. For the model, $\theta_{min} = 0^{o}$ and $\delta_{max} = 15^{o}$, hereby simplifying equation 4.1 to:

$$\delta_{MCP} = \frac{-\theta_{MCP} \times 15^{o}}{90^{o}} + 15^{o} \tag{4.2}$$

DIP and PIP

An intuitive constraint for the index, middle, ring and little fingers is the fact that bending the PIP joint causes the DIP joint to be bended as well. This relationship can be approximated to ([26]):

$$\theta_{DIP} = \frac{\theta_{PIP}}{3} \tag{4.3}$$

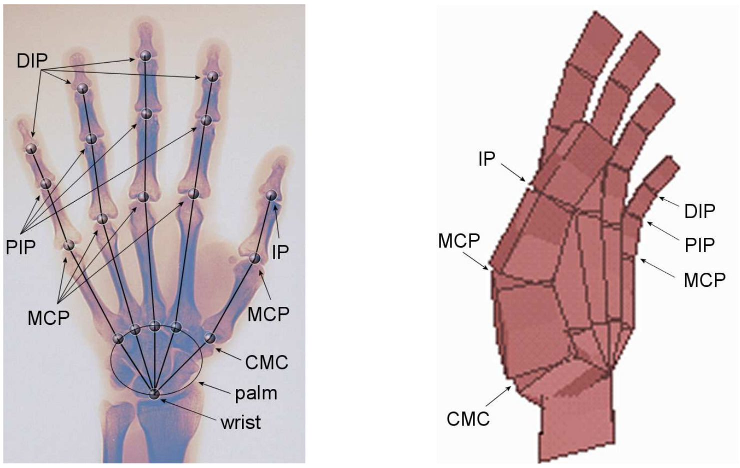
This simple formula decreases the DoF from 27 to 23, one for each finger. Thus, the hand and all feasible poses can be represented using a vector with 23 entries, an entry for each DoF.

4.3 Modelling

For The Watching Windows project, Kee-Kim Heng from the University of Otago, New Zealand, built a highly flexible model of a left hand which seems feasible to use (figure 4.3(b)).

The model is based on OpenGL[2] which uses a hierarchical representation of objects, very suitable for hand representation. Some adjustments to the specific case have to be performed; yet it significantly lightens the burden of modelling the hand from scratch. Other hand models, more or less anatomically credible, are available from the "3D Cafe"[3]. In appendix B, a series of predefined typical gestures is shown.

The hand model is based on a hierarchical structure, and figure 4.3(a) visualises this hierarchical composition of a real human hand. It can be considered as being built up from separate objects (bones) connected by rotational joints; the rotation of each joint is propagated to the end point of each fingertip.



(a) Tree structure of human hand (b) Hand model

Figure 4.3: **Figure (a) visualises the hierarchical structure of the human hand, figure (b) shows the hand model used.**

The root of the tree is the palm (treated as a single rigid object), this palm is rotated around the wrist joint. For such a hierarchy, rotating the wrist joint is propagated to the palm, the MCP, the PIP and finally the DIP of each finger, each joint attributing to the final spatial end position of each fingertip.

Because no deformable objects need to be represented (such as skin or muscles), the polygonal representation of the model seems well-suited for this specific task, in spite of the fact that cracks appear in the surface of the object. These cracks are of no importance to

---

[2] http://www.opengl.org

[3] http://www.3dcafe.com/asp/anatomy.asp

the overall performance of the system at this phase. Firstly, they will appear in the input silhouettes, as well as in the generated evaluation silhouettes; secondly because only a few pixels are affected.

4.3.1 Vector Representations

For the internal software representation of the hand a pose vector **s** containing the 23 DoF of the hand is introduced, thereby allowing for easy definition of a pose. The entries are summarised in table 4.2, with matching description.

Element Index (i)	Description
0,1,2	Global translation
3,4,5	Global rotation
6,7	CMC, thumb
8,9	MCP, thumb
10,11	MCP, index finger
12,13	MCP, middle finger
14,15	MCP, ring finger
16,17	MCP, little finger
18	IP, thumb
19	PIP, index finger
20	PIP, middle finger
21	PIP, ring finger
22	PIP, little finger

Table 4.2: **Internal state vector representation of the hand. The table summaries what each index in the state vector s represents. The hand pose can be fully specified using these 23 variables.**

Hence forward, when referring to a specific joint the symbol χ will be used - the pose vector $\mathbf{s} = [\chi_0 \ldots \chi_{22}]$ and the value of the ith joint is referred to as χ_i.

Following this notation, χ_i^{min} represents the minimum value of this joint, while χ_i^{max} represents the maximum value. In effect any value for the ith joint must satisfy the inequality $\chi_i^{min} \leq \chi_i \leq \chi_i^{max}$ to be valid. The entire set of inequalities to be satisfied is represented by Ω.

In figure 4.4 on the facing page, χ^{min} and χ^{max} are visualised for all joints, thus Ω is the grey area of the figure.

4.4 Summary

In this chapter some basic terms and definitions for the hand were presented. Whereas the medical literature have very complex models of the hand, for the 3D modeller a 27 DoF hand is often sufficient. Such a 27 DoF model was initially considered, but by applying a simple relationship between the DIP and PIP joints of the fingers, the model was reduced to 23 DoF.

Static constraints based on the anatomical constraints of the hand as well as constraints related to the Watching Window was discussed, together with anatomical dynamic constraints.

A 3D hand model based on OpenGL was chosen as the model for this project. A pose is easily created by providing the model with a pose vector, **s**. This pose vector and its contents

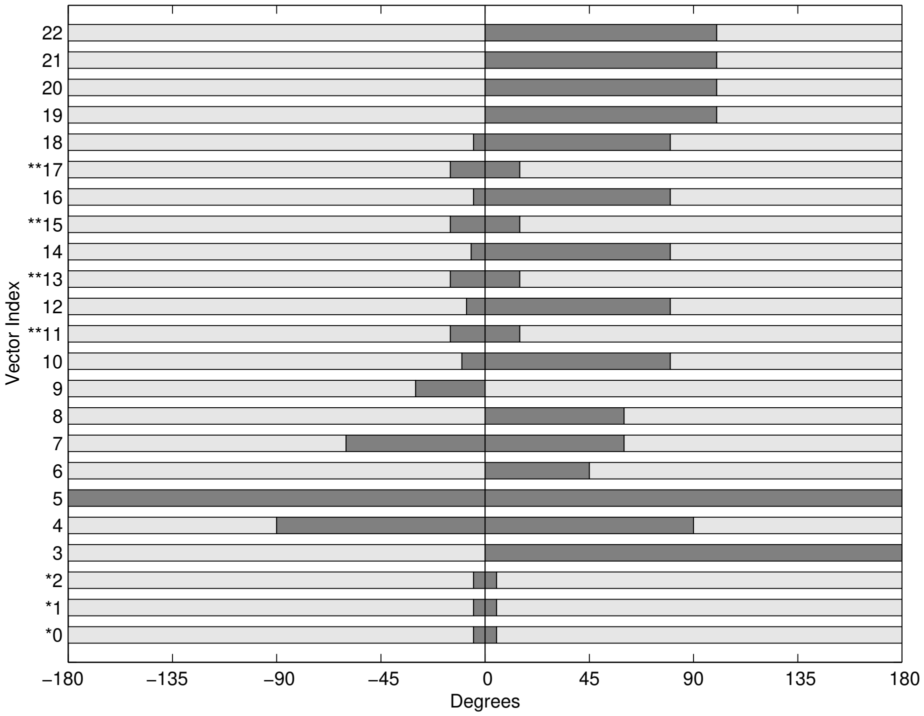


Figure 4.4: **Constraints for each vector index, showing χ_i^{min} and χ_i^{max} for $i = 0 \ldots 22$. *) Translation entries: These are *not* measured in degrees but in centimetres. **) MCP δ limits: Constraint limits are dynamically updated, see formula 4.1 on page 24.**

were also defined. Furthermore the static and dynamic constraints discussed in the analysis of the hand were implemented as a set of inequalities to be satisfied, this set was denoted Ω.

The pose vector $\mathbf{s}$, the set of inequalities Ω and the model of the hand all play an important role in the next part, Search Algorithms - how to deform the model to minimise the mismatch between the input silhouettes and evaluation silhouettes - will be discussed; starting with an optimisation overview followed by in-depth discussions of three algorithms and a final comparison.

Part II

Search Algorithms

Numerical Optimisation

Spider Optimisation

Downhill Simplex

Simulated Annealing

Comparison of Algorithms

Chapter 5

Numerical Optimisation

> "This is NP-complete, which, if you don't recognise that,
> it's something we in computer science use to frighten small children."
> - *Prof. Eric Cooper*

As described in section 3.2, two silhouettes must be compared - the input silhouette and the evaluation silhouette. A cost function representing their difference can easily be computed, this will be discussed later. By gradually deforming the model, using this cost function as a guideline, the silhouette difference can be minimised, thereby indirectly matching the estimated hand pose to the real hand pose.

In this project the cost function is subject to numerical optimisation, using three different algorithms.

5.1 Introduction

One way of looking at the cost function, denoted $f(\mathbf{s})$, is to think in geographic terms as hills, valleys and canyons - hills being high function values in a neighbourhood; valleys as low values and canyons representing relatively low, connected values of $f(\mathbf{s})$. One should keep in mind, though, that the topography is n-dimensional and may be of an unimaginable complexity, thus not as simple as described here. Throughout this report the terms "cost function", "error function" and "energy function" are used interchangeably.

Figure 5.1 show a 2D example of an error plot which has a correct match at $x = z = 180^o$, yielding a low error. Changing either of the parameters x or z from this position results in a higher cost.

In the specific case of fitting a hand model with 23 DoF, this surface should be extended from $\Re^2$ to $\Re^{23}$; clearly impossible to visualise.

The purpose of a numerical optimisation algorithm is to detect the optimum of a cost function $f(\mathbf{s})$ ($\mathbf{s}$ a vector in n dimensions, describing the current state), by searching a subspace of the n-dimensional vector space $\Re^n$ with respect to some user-defined criteria. Without any constraints Ω is $\Re^n$.

An optimisation algorithm is usually a minimisation or maximisation algorithm, the purpose being to minimise or maximise $f(\mathbf{s})$. Only minimisation is considered in this report: If the objective is to maximise and not minimise $f(\mathbf{s})$, this can trivially be accomplished by setting $f(\mathbf{s}) = -f(\mathbf{s})$.

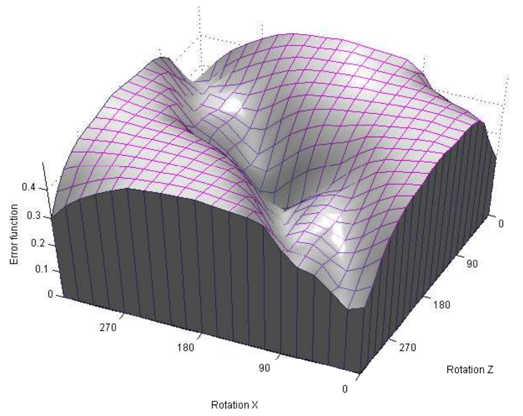


Figure 5.1: **An error function based on the parameters x and z, visualised in $\Re^2$. The correct match is at x = z = 180° where a valley is located. Also notice the valley at x = z = 0°, this is due to a good although not perfect match.**

Starting from an initial state $\mathbf{s}_{init}$, the minimum energy (function value) of the energy function is found iteratively by evaluating $f(\mathbf{s})$ and pertubing $\mathbf{s}$ to decrease the energy function towards its closest minimum value.

Multidimensional optimisation has been subject to intensive research, thus an overwhelming amount of information can be found when searching the Internet for this topic. The methods to be discussed in this chapter have been chosen because of either their intuitive simplicity or their use in the scientific literature surveyed with respect to the specific problem of model registration, especially Ouhaddi and Horain [13] and Kameda [15]. They by no means cover the field but are intended to be samples and a brief survey of some available methods.

Furthermore, because of the fact that multidimensional strategies are of most interest to the specific problem, mono dimensional strategies such as the Golden Section Search and Parabolic Interpolation will not be mentioned. For further information and implementation details on those, see Press et al. [27, chap. 10].

5.2 Cost Function

To evaluate the difference between the input and evaluation silhouettes, a cost function must be defined. The cost function is viewed upon as an error measure; the lower the error is, the better match between silhouettes.

Ouhaddi and Horain [13] compared two different error measures with respect to model fitting:

1. The non-overlapping surface between the two silhouettes.

2. The integral of distances between the two contours.

Their work documents that the non overlapping surface approach gives better results than using distances to contours, which seemed to appear very sensitive to model initialisation and to image noise. Taking the results of their article into consideration, it is initially decided to go with their surface matching strategy. Literature survey shows that Kameda [15] also based his matching strategy on surface overlapping, initially defining his error function as

$$f(\mathbf{s}) = \sum_{\mathbf{i}\in D} S(\mathbf{i}) \oplus M(\mathbf{i}) \tag{5.1}$$

where $\oplus$ is an *exclusive or*. M is the image of the evaluation silhouette (based on the pose vector $\mathbf{s}$), and S is the input silhouette image. The images are binary, of same size, and D represents possible positions in the images.

This energy function is decompositioned into two separate energy functions, the "inner pressure condition" ($f_{in}(\mathbf{s})$) and the "outer pressure condition" ($f_{out}(\mathbf{s})$):

$$f_{in}(\mathbf{s}) = \sum_{\mathbf{i}\in D} S(\mathbf{i}) \cdot (1 - M(\mathbf{i})) \tag{5.2a}$$

$$f_{out}(\mathbf{s}) = \sum_{\mathbf{i}\in D} M(\mathbf{i}) \cdot (1 - S(\mathbf{i})) \tag{5.2b}$$

$$f(\mathbf{s}) = f_{in}(\mathbf{s}) + f_{out}(\mathbf{s}) \tag{5.2c}$$

Minimising $f_{out}(\mathbf{s})$ prevents the model (M) from going out of the silhouette region (S), minimising $f_{in}(\mathbf{s})$ prevents the model from going into the silhouette region.

This energy function does not take scaling into account. To do so would involve implementing normalisation by replacing equation 5.2c with equation 5.3:

$$f(\mathbf{s}) = \frac{f_{in}(\mathbf{s}) + f_{out}(\mathbf{s})}{\sum_{\mathbf{i}\in D}(S(\mathbf{i}) + M(\mathbf{i}))} \tag{5.3}$$

In this project the degree of matching of left and right silhouettes are combined into one cost function. The left input silhouette is matched to the left evaluation silhouette, as is the right input to the right evaluation. The values of these two are then added and divided by 2 so the final cost function returns a number between 0 and 1, with 1 as the highest cost possible. This value is the *pixel error*.

Figure 5.1 on the preceding page visualises a typical behaviour of this pixel error. Several separate valleys and hills can be seen; having the search stuck in a wrong minimum (as an example, the valley at $x = z = 0$ in the figure) and not being able to escape, is a problem facing almost all search algorithms if $f(\mathbf{s})$ is sufficiently complex. This is briefly discussed in the next chapter.

5.3 Extremes

Having $\mathbf{s}$ stuck in a local minimum represents a very difficult problem to overcome. Only allowing the search algorithm to walk downhill, known as greedy search, often causes it to be trapped in a wrong valley; unable to escape even from extremely disadvantageous situations - the algorithm does not "know" there is a more favourable solution somewhere else in the error landscape. Finding a global extreme is, in general, a very difficult problem to solve, and two solutions are proposed in Press et al. [27, chap. 10] to overcome the problem, although they are only work-around solutions and do not solve the fundamental problem:

- Using a series of (uniformly distributed) starting positions $\mathbf{s}_0..\mathbf{s}_x$, all within Ω, hoping that $f(\mathbf{s})$ ends up in different valleys. The most extreme valley is reported back as a result of the search. This solution requires the entire optimisation processes to be restarted several times and thus is only feasible to apply when few evaluations of a relatively simple cost function are expected. Furthermore, if the function has many small minima, this technique may be of little use - too many restarts may be necessary to reach an "acceptable" solution ([28]).

- When a solution is found, take a finite step away from it and see if the algorithm returns to the same or a better valley. The disadvantage is that one does not know the size of the finite step or the dimensions to which it should be applied - if it is too small, or wrong dimensions are chosen, the routine may be likely to return to the same local minimum. If it is too large, it may just find another, maybe less feasible, local minimum.

In spite of the fact that non-greedy algorithms (section 5.5.2) can to some extent avoid the worst problems of being trapped too early, the valley may simply be too "deep" for it to escape.

5.4 Search Strategies

This section covers some basic ideas and strategies that can be applied when searching for the minimal energy in $\Re^n$. Two simple methods are presented and more sophisticated strategies to speed up the search process are shortly presented.

5.4.1 Grid Search

The global minimum can always be identified, by evaluating the energy of $f(\mathbf{s})$ in the (discrete) parameter space Ω (Mendivii et al. [29]). However, evaluating the cost function in entire Ω is prohibitively expensive for higher dimensional problems. In this project, 23 DoF yields a 23 dimensional search space. If each dimension is evaluated using only 10 uniformly distributed states in its discrete space, this strategy requires as much as 10^{23} evaluations - a clearly impossible task for the computers of today. Furthermore, one should keep in mind that Ω must be finite and the borders well defined as this method requires knowledge of boundaries for each dimension. Another problem facing this method is that the precise minimum is rarely found because of the discrete nature of the search, as illustrated in figure 5.2.

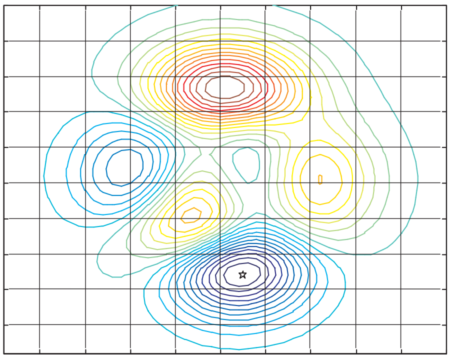


Figure 5.2: **Grid search of a function. Notice that the function is not evaluated at the position of the global minimum (marked with a star); merely close to it.**

In this figure, red colours indicate hills and blue colours valleys. Notice how the position of the global minimum is not evaluated in this case. This can, to some extent, be complied with by performing a higher resolution search when the best candidates have been found using a low-resolution grid.

5.4.2 Random Search

A simple probabilistic method is Random Search - simply pick as many random points as possible, evenly or "intelligently" distributed within Ω, and report the most extreme value encountered as the result, as shown in figure 5.3. This method is acceptable in many cases

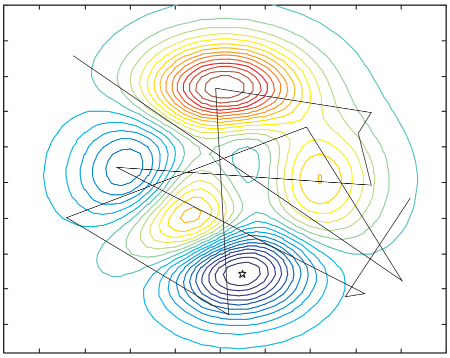


Figure 5.3: **Random search of a function. The global minimum is marked with a star.**

but in other cases it is clearly inefficient as discussed by Mendivii et al. [29, chap. 0.2.6]: If a goal has been established and the chance of hitting the goal is p for one iteration, the probability of NOT hitting the goal in m iterations is $(1-p)^m$. Thus the probability of success is

$$s = 1 - (1-p)^m \tag{5.4}$$

Solving equation 5.4 for m yields

$$m = \frac{log(1-s)}{log(1-p)} \tag{5.5}$$

If a success rate of 99% is required when searching for 1 point among 10^{23} possibilities, m becomes

$$m = \frac{log(1-0,99)}{log(1-10^{-23})} \approx 4.6 \cdot 10^{23} \tag{5.6}$$

In other words, equation 5.6 states that more than $4.6 \cdot 10^{23}$ iterations, or 4.6 as many points as in the discrete search space, must be considered - a clearly infeasible situation.

It should be stressed, however, that in other situations Random Search has much better statistics.

5.5 Search Optimisations

More advanced techniques can be applied to speed up the search process. These techniques do not represent specific search strategies; merely philosophical ideas which can be implemented in various ways.

5.5.1 Derivatives

Instead of blindly searching the search space Ω using grid or random search, partial derivatives ($n = 1$) or vector gradients (in the multidimensional case) of $f(\mathbf{s})$ can be used to speed up the search process by guiding the direction of search in Ω using some kind of "intelligent reasoning" (Press et al. [27, chap. 10]).

The idea is relatively simple: Everything else being equal, the shortest path to the local minimum must be in the direction of the steepest descent from the current state $\mathbf{s}$ - this requires, however, that the derivatives/gradients can be easily calculated without too many time consuming computations, if at all.

5.5.2 Greedy vs. Non-greedy Search

A greedy algorithm, also known as iterative improvement, is a deterministic method, characterised by always taking the direct way to the nearest minimum from the point of evaluation, often using the derivatives: From point $\mathbf{s}_{init}$ go immediately downhill as far as possible. If the algorithm is run twice starting from the same initial starting vector $\mathbf{s}_{init}$, the same sequence of steps occurs (Mendivii et al. [29]).

An example of a non-greedy strategy is random search as mentioned earlier, but also "Simulated Annealing" is in this category. This relatively new approach which introduces stochastic variables and probabilities in the search process to allow the algorithm to perform "uphill" steps from time to time; this somewhat advanced method will be discussed later in chapter 8.

5.5.3 Tabu Search

Tabu search can be used in modifications of greedy search strategies. Instead of greedy walking downhill, one can use probability to allow uphill or just random perturbations, as in Simulated Annealing. This strategy helps avoid entrapment in local minima but may cause the routine to loop between several states. However, keeping a list of recently visited locations and avoiding visiting a valley more than once can, to some extent, eliminate this problem. This method thus requires gathering information of the error surface and using this information to develop adaptive rules to guide the search process (Mendivii et al. [29], Y.C.Ho [30]).

5.5.4 Constrained Search

Engineers and economists are often concerned with constrained optimisation. As an example, the living cost for a person must be a non-negative number, temperature must be above 0 Kelvin and in this specific case, the individual joint angles of a hand must lie within an interval (chapter 4.2). Constrained search introduces pros and cons; usually it can be used to minimise the search space Ω; however, the chances of getting stuck in a local minimum often increases as a search may ooze down a canyon, just to hit a constraining wall. The exact implementation of constraints depends on the search strategy chosen and other factors; this will be discussed later and a rather intuitive example will be given in section 7.2.

5.6 Evaluation of the Algorithms

In the following three multidimensional optimisation algorithms are presented, tested and compared. When referring to the "performance" of an algorithm, it does not refer to the speed of the algorithm; merely how small a pixel- or *pose* error it manages to obtain:

5.6.1 Pose Error

The cost function defined in 5.2 is used in the optimisation algorithms, as it can easily be computed from the input and evaluation silhouettes.

Because this value only indicates how the silhouettes match, and not how well the hand pose is estimated, a *pose error* function is required when evaluating the search algorithms. As an example, a rotation of the wrist 180^o around the forearm yields a relatively good silhouette match but is certainly not acceptable with respect to pose estimation evaluation.

An intuitive solution is to measure the spatial Euclidean distance from joints on the well-defined input model to matching joints of the evaluation model. Because of the fact that the exact applications of The Watching Window console are not yet known, it is at this stage chosen to include *all* joints of the hand in the computation of this evaluation measurement:

$$p(\mathbf{s}) = \sum_{i=0}^{22} \sqrt{((x_i^{input} - x_i^{eval})^2 + (y_i^{input} - y_i^{eval})^2 + (z_i^{input} - z_i^{eval})^2)} \tag{5.7}$$

In equation 5.7, $p(\mathbf{s})$ is the pose error for the estimated pose vector $\mathbf{s}$ in centimetres, i represents every joint of the hand, x_i^{input} the spatial x-position for joint i of the input model, x_i^{eval} the x-position for joint i of the evaluation model and so on.

One should keep in mind, though, that the value of P still shows some ambiguity. As an example, if the wrist is rotated 180^o around the z-axis, a fist will give a smaller pose error than an open hand in the same situation.

Another fact to consider is that the result of too persistent optimisation with respect to pixel error occasionally causes the pose error $p(\mathbf{s})$ to *increase*. The reason for this curious fact may be that once a wrong valley is found, the pixel error can be decreased to a certain local minimum, but the estimated pose actually diverges more and more from the “real” pose, which lies in another valley somewhere in R^{23}.

To give an impression of the order of magnitude for the pixel error $f(\mathbf{s})$ as well as the pose error $p(\mathbf{s})$, a series of gestures are presented in appendix D, together with their pixel- and pose error. Looking at the examples also reveals that $f(\mathbf{s})$ and $p(\mathbf{s})$ do not necessarily show linear dependence.

Based on a subjective evaluation, a pose error of less than 100 centimetres in the following is regarded as satisfactory.

5.6.2 Learning- and Test Sequences

Because it is crucial not to optimise the algorithms to a single animation, twelve different sequences are designed. These sequences obviously do not cover the whole spectrum of possible movements; yet they give an indication of the feasibility of the optimisation algorithms. These sequences are named *learning sequences.* In particular sequences 7-12 serve the purpose of being subject to manual algorithm optimisations.

As it shall become clear later, sequences 1-6 constitute no serious challenge to the algorithms and are therefore of minor interest. The difference from sequences 1-6 and 7-12 is mainly in the wrist movement; explained in appendix B on page 119, where all sequences are presented. Furthermore a CD is enclosed with the report, containing animations of both learning- and test sequences. It is strongly advised to view these animations in order to get a feel for the problem domain.

After having optimised the algorithms, the search routines are compared using a new set of sequences, the *test sequences.* These sequences have never been presented to the algorithms before, hereby making sure that the results obtained are general for a variety of sequences and not just the learning set.

These test sequences are based on the fixed-wrist learning sequences 1-6, but with wrist rotation and translation applied. This is done because they will basically constitute completely new sequences due to the fact that wrist movement plays a major role - another thing that shall become clear later.

5.6.3 Statistics

Due to the speed of the algorithms, it is only possible to present the algorithms to a limited number of sequences. To add objectivity to the evaluation, statistics are desired and will be based on the Wilcoxon Signed Rank Test (appendix C), as this test can be used in small-sample cases and seems suitable to at least give a statistical indication of the performance of the routines.

If nothing else is stated, the Wilcoxon test is based on a confidence interval of 95%.

5.7 Summary

In this chapter a brief description of the principle of numerical optimisation has been given. The cost function to be used in the numerical optimisation was developed based on the degree of matching between evaluation and input silhouettes. The cost function presented combines the matching in the left and right image into one. The result of the cost function, a number between 0 and 1 also referred to as the "pixel error", provides a measure of how good the input silhouettes match the evaluation silhouettes. A low pixel error does not necessarily equal a good pose estimation due to the ambiguity of the silhouette. Thus another measure named "pose error" was introduced to provide a measure for the quality of the pose estimation.

Furthermore, examples of two optimisation algorithms were given together with a brief description of possible improvements. Many other optimisation algorithms exist, and in the next three chapters three of these will be presented. The first algorithm is completely new to the problem of high dimensional optimisation while the other two are established algorithms in this field. $Id : spider_algorithm.tex, v1.422002/05/2722 : 49 : 24purpleExp$

Chapter 6

Spider Optimisation

> “Snakes, why’d it have to be snakes?”
> - *Indiana Jones*

In recent years much attention has been given to active contours, also known as *snakes*, as a means of tracking objects. The concept of snakes was first presented by Kass et al. [31] in 1988 as a new approach for shape analysis in image processing. A snake is an elastic contour which is attracted to features detected in an image, while trying to stick to preferred configurations. This is done by minimizing an energy function for the snake. The minimization principle used in a snake algorithm is quite different to other algorithms used for modelfitting.

Recently a generalization of the snake principle, named a *Spider*, was presented by McCane [32]. As will become clear in the following chapter this opens up for applying this optimization principle to the model fitting problem, which to our knowledge has not been attempted before.

This chapter presents the basic principles of the snake on which the Spider is based, putting emphasis on the optimization part. Afterwards the generalization of the snake to the Spider and its use for model fitting is presented.

Finally four different implementations of the Spider are presented, and their pros and cons are discussed. Each implementation is tested using the learning sequences in appendix B on page 119.

6.1 The Snake Principle

In image processing it is a well known fact that segmented objects in an image cannot be expected to be smooth and continuous. For instance, trying to detect the cells in figure 6.1 on the next page can be a hard task using traditional techniques as e.g. the Hough Transform or template matching, because of poor segmentation which may blur the features or cause false positive or negative pixels.

In its descrete version the snake countour is represented by a spline with n control points in the image, which can be denoted as in (6.1).

$$\begin{aligned} V &= [v_1, v_2, \dots, v_n] \\ v_i &= [x_i, y_i] \end{aligned} \tag{6.1}$$

The position of each of these control points is governed by an energy function which is a combination of an external and an internal energy value as seen in equation 6.2.

$$E(v_i) = E_{ext}(v_i) + E_{int}(v_i) \tag{6.2}$$

The external energy (E_{ext}) is used to attract the snake to image features while the internal energy (E_{int}) attempts to maintain certain preferred configurations, e.g. one may wish to maintain a certain length between control points or favour if the control points form a circle. As such E_{int} represents the prior knowledge of the object which is to be recognised. The fundamental idea is to obtain a minimum sum of these energies, done by recursively evaluating each point in the contour.

A popular intuitive comparison of the snake is to that of a rubber band. Imagine stretching a rubber band to a big circle with your hands. The internal energy of the hand can be made smaller by reducing the circumference of the rubber band, but the external energy supplied from the hands prevents such a reduction. Now place this big circle around a coffee cup, and let go of the rubber band. By letting go, the external energy abruptly disappears and the internal energy is free to shrink the circumference of the rubber band. Unfortunately (for the rubber band) as it shrinks it eventually meets resistance from the coffee cup, providing a new external energy to the system. Again to reduce the internal energy a reduction of the circumference is attempted but dismissed as the external energy becomes too big.

The principle is shown in figures 6.1(a)-6.1(c) for locating cells in an image. Figure 6.1(a) shows the snake just being initialised (letting go of the rubber band), figure 6.1(b) shows the internal energy working to minimise the circumference of the snake and finally figure 6.1(c) shows how the external and internal energies combine to a minimum energy for the system (rubber band meets coffee cup).

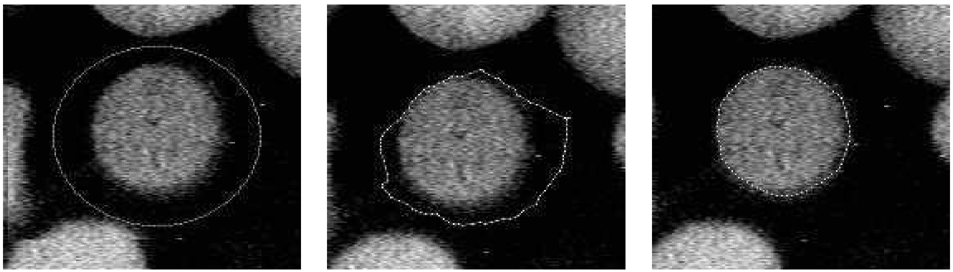

(a) Snake initialised (b) Snake adjusting (c) Final state

Figure 6.1: **Detecting cells using a snake. In its initial state the snake loosely surrounds the cell (a). As the snake tries to minimise the sum of its external and internal energies, the snake approaches the circumference of the cell (b). As a minimum is reached the snake contour fits closely to the cell (c). Pictures from [33].**

The way to accomplish the fitting in figure 6.1 is by minimising the overall energy of the snake (eq. 6.3).

$$E(V) = \sum_{v_i \in V} E(v_i) = \sum_{v_i \in V} (E_{ext}(v_i) + E_{int}(v_i)) \tag{6.3}$$

The Viterbi algorithm [34], a dynamic programming approach, can be applied to minimise equation (6.3). The internal energy of a control point is a property of its neighbour control points yielding a total internal energy of:

$$E_{int}(V) = E_{int}(v_1, v_2) + E_{int}(v_2, v_3) + \ldots + E_{int}(v_{n-1}, v_n) \tag{6.4}$$

The Viterbi algorithm now defines intermediate variables S_i:

$$\begin{aligned}
S_1(v_1) &= E_{ext}(v_1) + \min_{v_2}[S_2(v_2) + E_{int}(v_1, v_2)] && \text{(6.5a)}\\
S_2(v_2) &= E_{ext}(v_2) + \min_{v_3}[S_3(v_3) + E_{int}(v_2, v_3)] && \text{(6.5b)}\\
&\vdots\\
S_{n-1}(v_{n-1}) &= E_{ext}(v_{n-1}) + \min_{v_n}[S_n(v_n) + E_{int}(v_{n-1}, v_n)] && \text{(6.5c)}\\
S_n(v_n) &= E_{ext}(v_n) && \text{(6.5d)}
\end{aligned}$$

Using the equations defined in equations 6.5a - 6.5d, finding the minimum energy of the whole snake is equal to minimising $S_1(v_1)$:

$$\min(E) = \min_{v_1}(S(v_1)) \tag{6.6}$$

As the value of S_1 depends on S_2 which again depends on S_3 and so on, the algorithm is initialised by calculating $S_n(v_n)$ where only external energies are considered. Instead of evaluating $S_n(v_n)$ for all possible values of v_n a grid of neighbour positions around the current position of the control point is used as shown in figure 6.2.

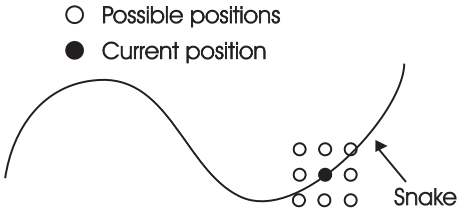


Figure 6.2: **Possible positions of v_i. The black dot is the current position of this control point. The circles around this dot are possible new positions for this control points, depending on the total energy at these positions.**

Subsequently S_{n-1} is computed. For each v_{n-1}, the possible links to v_n are evaluated and the one with lowest energy is chosen as the position of v_n. This is repeated for all nodes until the algorithm reaches and calculates the v_1 energies. The v_1 position yielding lowest energy is chosen as the best position for v_1. Once this position is determined, the algorithm back steps. The v_2 position which links to the chosen v_1 position is chosen as the best position for v_2. This continues through all control points of the snake, in turn giving the control points of the snake contour, V, with the lowest energy. As the algorithm only examines a grid around each control point the above is repeated until the total energy of the snake converges. The guarantee of convergence is one of the appealing features of this optimisation scheme.

6.2 The Spider

In a conventional snake each node (control point) has a maximum of one parent and one child; basically represented as a linked list. However for the Viterbi algorithm to be applied as an optimisation algorithm for the fitting of the hand, the algorithm must be able to handle multiple children. Fortunately a generalisation of the snake algorithm applicable to tree structures has been presented by McCane [32]. Due to the resemblance of the trees structure with a spider's many legs, the generalisation was aptly named the *Spider*. The

Spider generalises the snake to allow each node to have multiple children, from which a tree structure can be obtained. By constraining each node to only having one predecessor, each path from the root node to a leaf node of the tree can be looked upon as one snake.

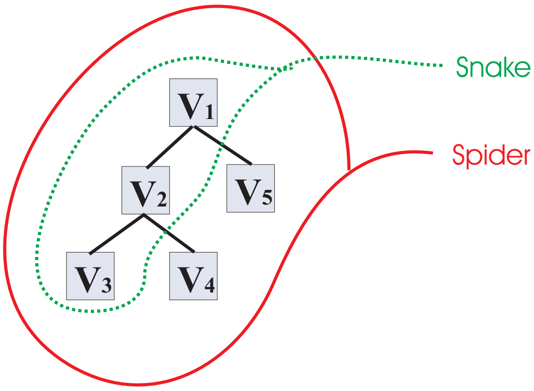


Figure 6.3: **The Spider tree. Each branch from the root node $\mathbf{V_1}$ to leaf node can be regarded as a snake, while the whole tree constitutes the structure of a Spider.**

An illustration of the Spider principle is shown in figure 6.3, the only major difference between the Spider and the snake algorithm lies in the evaluation of the intermediate variables. When calculating the node energy, all node energies of all children have to be evaluated; thus the the intermediate variables all take the form of equation 6.7:

$$S(v_i) = E_{ext}(v_i) + \sum_{v_c \in C(v_i)} min_{v_c}(S(v_c) + E_{int}(v_i, v_c)), \quad (i = 1 \ldots n) \tag{6.7}$$

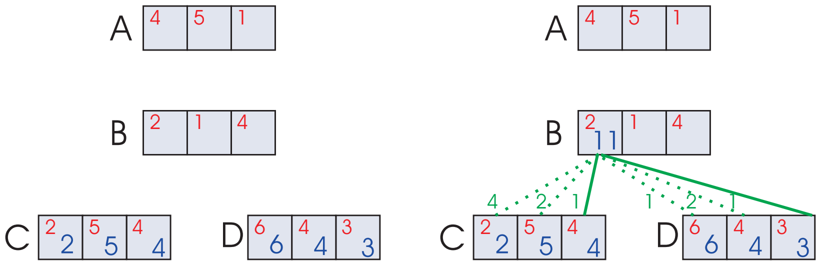
As can be seen the generalisation is applied without penalties in complexity. The cost of adding a node is the same for the Spider algorithm as for the snake.

6.2.1 Spider Example

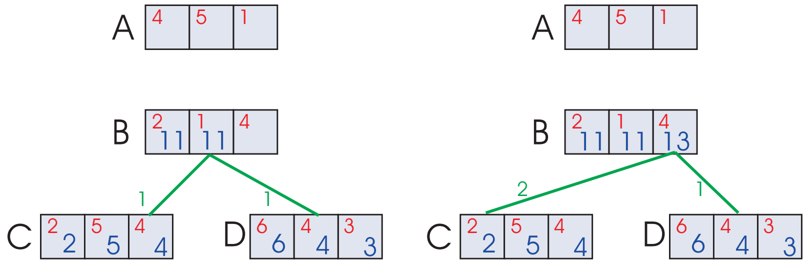
To better understand the principle of the Spider algorithm an example of a single iteration is shown in figure 6.4. Figure 6.4(a) shows the initial state of the Spider. **A** is the root node of the snake, and **B** has two children, **C** and **D**. The red number in the top left corner of each cell shows the external energy of each state. The three cells, referred to as neighbours, constitute a node. Notice that each neighbour represents a possible position of the node. If for instance a node represents the angle of a joint, each cell represents a possible angle for this joint. The computation of the internal and external energies of a cell is based on this position.

In theory a node can contain any number of neighbours, but for simplicity this is shown with only three. Also due to computational efficiency, in most implementations of snakes/Spiders the number of neighbours are kept small. As **C** and **D** do not have any children, the total energy (blue number in bottom right corner) of each neighbour is set to the same as their respective external energy.

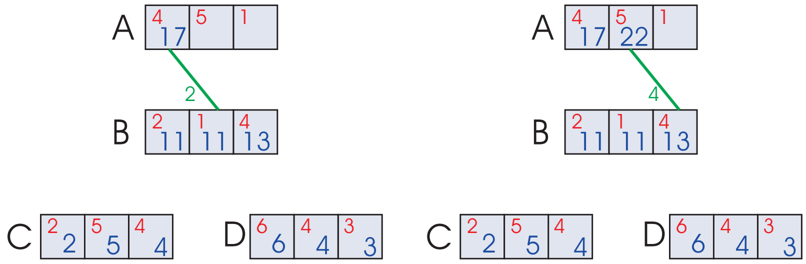
The first step, figure 6.4(b), is to evaluate the energy of the first neighbour in node **B**. As the aim is to minimise the total energy, the connections to node **C** are first evaluated. The first neighbour in node **C**, $\mathbf{C}_1$, has a total energy of 2 and the internal energy $E_{int}(\mathbf{C_1}, \mathbf{B_1})$ is 4, which adds up to 6. The internal energy is shown as the green number besides the green



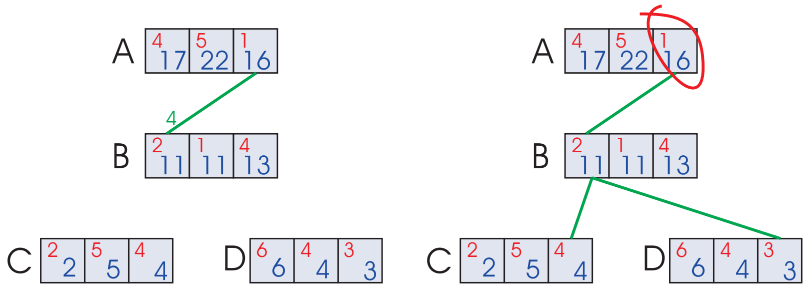
(a) Initial state (b) Step 1



(c) Step 2 (d) Step 3



(e) Step 4 (f) Step 5



(g) Step 6 (h) Step 7

Internal energy External energy Total energy

Figure 6.4: Example of the Spider algorithm

line connecting the neighbours. The connection to the second neighbour in **C** adds up to 7 as the total energy of $\mathbf{C}_2$ is 5 and the internal $E_{int}(\mathbf{C_2}, \mathbf{B_2})$ energy is 2. The connection to the last neighbour, $\mathbf{C}_3$, (solid green line) adds up to 5 and thus this is the lowest energy for any of the neighbours in **C**. This connection is chosen as the best connection between $\mathbf{B}_1$ and **C**.

In a similar fashion all connections from $\mathbf{B}_1$ to **D** are evaluated, and the smallest energy found to be the connection to $\mathbf{D}_3$. The energy of the two connections $\mathbf{B}_1$**:**$\mathbf{C}_3$ and $\mathbf{B}_1$**:**$\mathbf{D}_3$ are then added together with the external energy of $\mathbf{B}_1$ to give a total energy of 11 for this node.

In step 2 to 6, shown in figures 6.4(c) to 6.4(g), the total energy of $\mathbf{B}_2$, $\mathbf{B}_3$ and all neighbours in **A** are calculated in a similar way. Having calculated energies for each of the neighbours in the root node (**A**), the neighbour with the lowest energy is $\mathbf{A}_3$ (encircled in red) in figure 6.4(h). Having the lowest energy, this neighbour is chosen as the best position for the root node **A** and forms the basis for the best configuration of the entire Spider.

The algorithm then backtracks through the tree to find out which neighbour in each node comprises the best configuration. In figure 6.4(e) the connection $\mathbf{A}_3$**:**$\mathbf{B}_1$ provides the lowest energy, and as a result $\mathbf{B}_1$ is also a part of the solution. Finally figure 6.4(c) shows that the children $\mathbf{C}_3$ and $\mathbf{D}_3$ provide the lowest energies for $\mathbf{B}_1$ and thus these are also chosen as parts of the solution. The overall lowest energy for the Spider is then $V = [A_3 \; B_1 \; C_3 \; D_3]$.

The above represents one iteration, and multiple iterations are performed until a satisfactory low energy level is reached, where the Spider converges or a predefined number of iterations has been performed.

6.3 Model Fitting

In this section the Spider algorithm is applied to the problem of this project, model fitting. Initially a tree structure is chosen where each dimension χ of the hand model is represented as a node. This tree structure is shown in figure 6.5, the numbers referring to joints in table 4.2 on page 26.

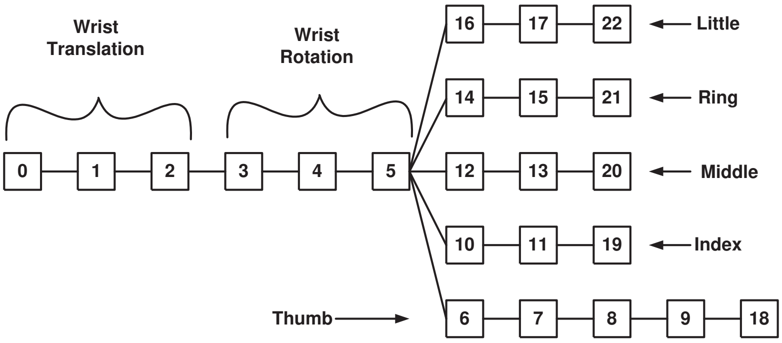


Figure 6.5: **Tree structure of the Spider. The root node is node 0. The algorithm starts by estimating possible positions of node 9 and 18, hereafter node 8 and 9 and so on. The order of estimation is thumb, index, middle, ring, little and finally wrist.**

6.3.1 Node Structure

Each node is chosen to consist of three neighbours, for which suitable position values must be determined. These position values are determined for each iteration depending on the current

χ values. To choose the χ values for node i to be examined, the following scheme is applied:

```
          if $(\chi_i + \Delta_i) > \chi_i^{max}$
             $node_i.neighbour[0] = \chi_i;$
             $node_i.neighbour[1] = \chi_i - \Delta_i;$
             $node_i.neighbour[2] = \chi_i^{max};$
        else
             if $(\chi_i - \Delta_i) < \chi_i^{min}$
                $node_i.neighbour[0] = \chi_i;$
                $node_i.neighbour[1] = \chi_i + \Delta_i;$
                $node_i.neighbour[2] = \chi_i^{min};$
           else
                $node_i.neighbour[0] = \chi_i;$
                $node_i.neighbour[1] = \chi_i + \Delta_i;$
                $node_i.neighbour[2] = \chi_i - \Delta_i;$
```

Here Δ_i is the distance between the position values of $node_i$ one wishes to employ.

For instance, by choosing Δ_{18} to be 5, the angle value of the IP joint of the thumb is tested in steps of 5 degrees. If the current angle of the joint, χ_{18}, is 17 the Spider will test χ_{18} values of 12, 17 and 22 in this iteration.

An advantage compared to the optimisation algorithms presented later is the ability to manually control each joint value to be tested. Here the hand constraints, Ω, are employed when choosing the values for each neighbour of a cell as shown in the above pseudo code. For instance if the current value of the joint is so close to the maximum allowed value, χ_i^{max}, that an addition of Δ_i will exceed the constraint limit, χ_{max} is chosen as the possible value instead of $\chi_i + \Delta_i$, basically a simple truncation. The lower constraints are imposed in a similar way. This way the Spider is allowed to search to the edge of the possible values for the joint, but will never violate the joint constraints Ω.

6.3.2 Step Size

From the above, it should be fairly obvious that the choice of Δ values, also referred to as step size, is of great importance as the choice of these values determine how precise one can expect the pose of the hand to be estimated and how fast movement is allowed. By setting $\Delta_{18} = 5$ this joint angle cannot be estimated with a precision of more than 5 degrees. On the other hand, setting $\Delta_{18} = 0.1$ enhances the precision of the estimate but introduces other problems. For one, the number of iterations required to obtain a minimum energy rises. Secondly, such a small variation will make the algorithm prone to getting caught in a local minima.

In choosing Δ values a consideration to be made is what effect the individual χ values have on the silhouette. Here we argue that small changes in the translation or rotation of the wrist, $(\chi_0 \dots \chi_5)$ generally cause a bigger change in the silhouette than changes in any of the outermost joints. Due to this the Δ values chosen for the wrist rotation/translation nodes should be smaller than that of the remaining nodes.

The choice of Δ values is also dependent on the frame rate and the maximal joint velocities one can expect from the user, as this influences the maximal change in pose from frame to frame.

A logical approach to choosing the Δ values is to gradually decrease Δ for each iteration, thus gradually refining the search of the Spider. Different schemes can be applied, which will be discussed in more depth in section 8.4. Initial tests show that the decreasing Δ indeed is better than keeping it fixed.

In this case a linear decrease of Δ is chosen: $\Delta_i' = \Delta_i - \rho_i$.

The initial Δ values are heuristically based on the sequences described in appendix B on page 119, and listed in table 6.1.

Initial Δ	Value
Δ_{0-2}	1
Δ_{3-5}	7
Δ_{6-22}	10

Table 6.1: **The initial Δ values for the Spider nodes.**

6.3.3 Energies

An external as well as an internal energy for the Spider must be specified, but first the original Spider algorithm from equation 6.7 is modified so the external energy depends on both the parent and child node:

$$S(v_i) = \sum_{v_c \in C(v_i)} min_{v_c}(S(v_c) + E_{int}(v_i, v_c) + E_{ext}(v_i, v_c)), \quad (i = 1 \dots n) \tag{6.8}$$

The cost function derived in section 5.2 on page 32 is used to calculate the external energy.

$$f(\mathbf{s}) = \frac{f_{in}(\mathbf{s}) + f_{out}(\mathbf{s})}{\sum_{\mathbf{i} \in D}(S(\mathbf{i}) + M(\mathbf{i}))} \tag{6.9}$$

To calculate the cost function, a pose vector $\mathbf{s}$ is needed, but due to the dynamic nature of the Spider algorithm, according to equation 6.8, only two values in $\mathbf{s}$ (v_i and v_c) are evaluated at any given time, hence the other values of $\mathbf{s}$ have to be supplied. To accomplish this, a copy of the initial pose vector $\mathbf{s}_{init}$ is used, denoted $\mathbf{s}'$.

The values v_i and v_c are inserted into $\mathbf{s}'$ at the their corresponding indexes before evaluation; $\mathbf{s}'[i] = v_i$ and $\mathbf{s}'[i] = v_c$, therefore external energy becomes:

$$E(v_i, v_c) = f(\mathbf{s}') \tag{6.10}$$

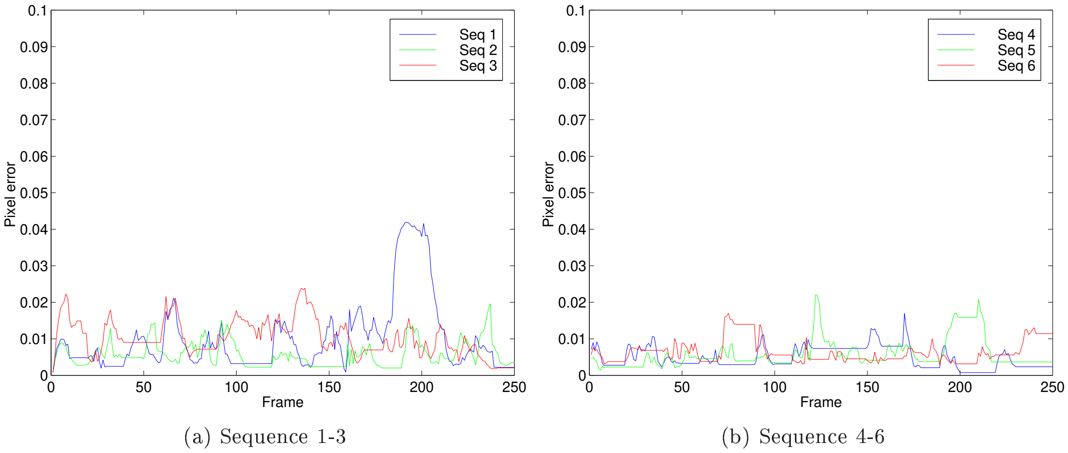
At this point there is no desire to favour certain hand poses or in other ways imply inter joint constraints, and hence the internal energy E_{int} is set to zero in this study.

6.3.4 Results - Initial Tests

In this first implementation the Spider is allowed 10 iterations per frame. Ideally the Spider should be allowed to iterate until convergence, but because the Δ values are linearly decreased for each iteration a fixed number of iterations must be provided to calculate the amount of decrease. The delta value is decreased linearly with 20% for every second iteration. So Δ_1 is 1.0 for the two first iterations, and is then decreased to 0.8 for the following two and so on. Eventually $\Delta_1 = 0.2$ for the last two iterations

The initial implementation is tested on the learning sequences and the results based on pixel error are presented. The complete results in terms of both pixel and pose estimation error are shown in appendix E on page 131.

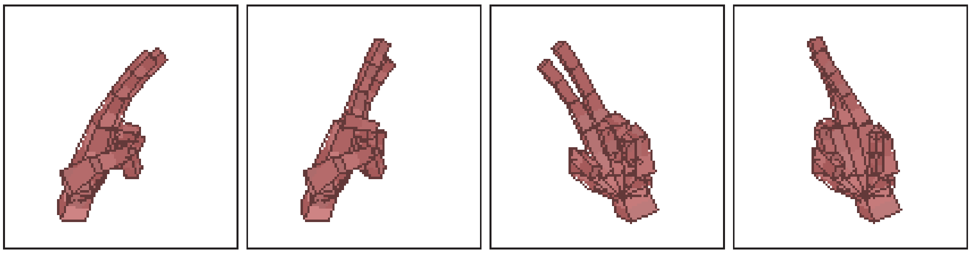
As this is the first implementation applying the Spider algorithm to the problem of model fitting, these initial results indicate if this approach is at all feasible. The pixel error after the final iteration in each frame for learning sequences 1-3 and 4-6, where the position and



(a) Sequence 1-3 (b) Sequence 4-6

Figure 6.6: **Result for test sequence 1-3 and 4-6. Notice the peak in the pixel error of sequence 1 singling out this sequence from the others.**

rotation of the wrist are fixed, is shown in figure 6.6. As can be seen the algorithm generally performs with a pixel error below 0.02, a remarkable low error. However learning sequence 1 (blue graph in figure 6.6(a)) shows an error peak standing out from the general impression. This error peaks at frame 193 with a pixel error of approximately 0.045. Captures of this frame are shown in figure 6.7, showing that the algorithm is trapped in a local minimum.



(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure 6.7: **Frame 193 of test sequence 1. (a) and (c) show the input images to the silhouette fitting, while (b) and (d) show the estimated poses. Comparing (c) and (d) it is clear that the index and ring fingers have switched positions in the estimated pose causing a big pose error.**

When visually comparing figure 6.7(a) and figure 6.7(b), notice how similar the silhouettes are. However when comparing figures 6.7(c) and 6.7(d), it becomes clear that to estimate the correct pose the algorithm would have to extend the index finger while flexing the ring finger.

Extending the index finger is unlikely, as this imposes a larger pixel error from the left silhouette comparison, which in turn cancels out the lower pixel error obtained in the right silhouette comparison.

Likewise it is difficult to flex the ring finger because of the larger error in the left comparison cancelling out the right comparison. And as flexing the ring finger is not an option, the algorithm instead chooses to “hide” the ring finger in the silhouette of the middle finger as less pixel error is obtained by doing so.

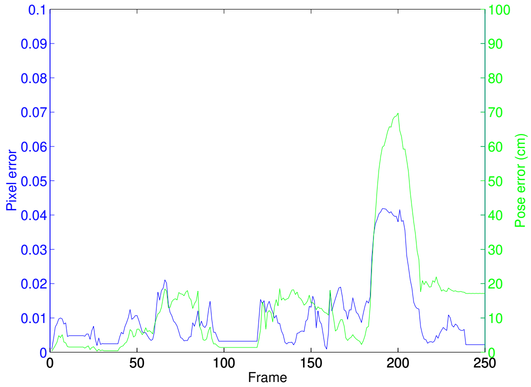


Figure 6.8: **Pixel and pose error for sequence 1. Both pose and pixel error begins to rise, but while the pixel error stabilises around frame 190 the pose error continues to increase until frame 200.**

Figure 6.8 shows the relationship between the pixel and pose error in sequence 1. In general the pixel and pose error follows the same pattern as mentioned earlier in the text with the peak in the 190-200 frame section standing out. Note that both pose and pixel error begin their rise at around frame 178, but the pixel error stabilises at around frame 193 while pose error continues to rise until frame 200. By reviewing the animation sequence, the cause of this difference is found in frame 184 and 185 as shown in figure 6.9.

The pose error reaches a local minimum at frame 178 with very little difference between input and estimated silhouettes. In frame 180 a morphing from the "OK" gesture to the "Victory" gesture begins. In this morphing the index finger begins to extend while the little and ring fingers begin to flex. When reaching frame 183 the index and ring fingers in the estimated pose obtain the same space of the left estimated silhouette. In the left input of frame 184 the index finger continues towards full extension while the ring finger flexes.

When comparing the left input frame (LI-184) to the estimated pose of frame 183 (LE-183) which is the start estimate pose for frame 184, there are two ways for the algorithm to manipulate the silhouette to match the input. One is to flex the ring finger and extend the index finger, the other is to do the opposite. Both ways will synthesise similar silhouettes for the left view, so the decision must be based on the right silhouette.

Now compare the estimated pose from frame 183(RE-183) to the input silhouette of frame 184 (RI-184). Note how the ring finger has flexed in the input image and makes a clean "V", and how the distal tip of the ring finger is in this area in the estimated frame. Obviously comparing the two silhouettes will give a pixel error, but both flexion and extension are possibilities for the ring finger as both will give a better pixel error. Unfortunately the pixel error of extending the ring finger to lie in the same silhouette as the middle finger is less than flexing the ring finger.

Although the results of learning sequences 1-6 indicate that using the Spider algorithm is feasible for this minimisation problem, the results of sequences 7-12 as seen in figure 6.10 show that there is still room for improvement if the wrist is not fixed. As can be seen the pixel errors of these sequences are much higher than the previous sequences in which the wrist position and rotation were fixed. Visually comparing the estimated sequences to the correct input sequences, it is clear that these big pixel errors mainly occur because of wrongly estimated wrist position and rotation. An example of this is from the very beginning of sequence 7. When observing the development of pose and pixel error as shown in figure 6.11, these begin

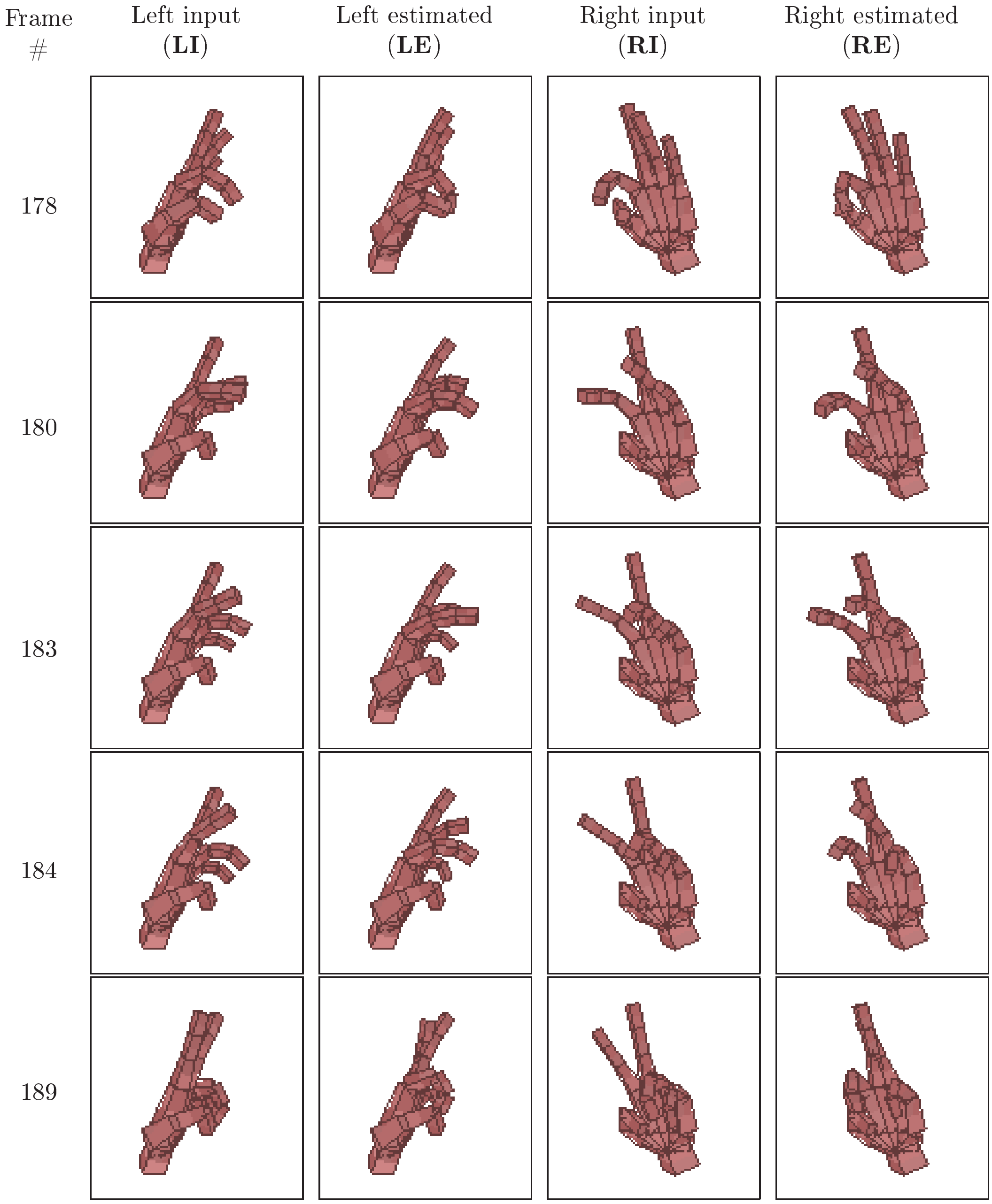
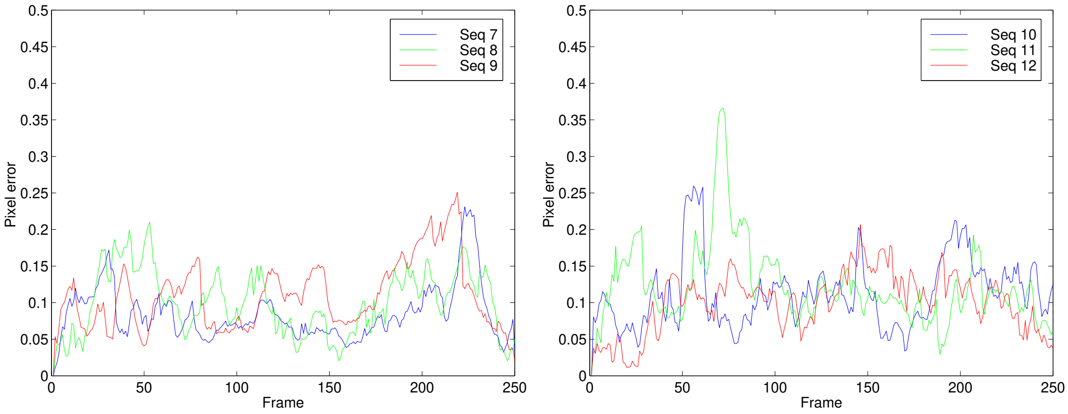


Figure 6.9: **Example of estimation failure. In frame 178 the estimation of the "OK" gesture is spot on, and the input begins to morph towards the "V" gesture. Notice how the index finger begins to extend while the little and ring fingers begin to flex. The wrong estimation is due to the events in frame 183 and 184, where the index and ring fingers cross in the left input image. Instead of continuing its extension, the index finger begins to flex while the ring finger begins to extend once more. As discussed in figure 6.7, once this pose is obtained the algorithm cannot recover to estimate the correct pose.**



(a) Sequence 7-9

(b) Sequence 10-12

Figure 6.10: **Result for test sequence 7-9 (slow) and 10-12 (fast). In general the introduction of wrist translation and rotation presents big problems for the algorithm. The slow sequences (a) show smaller fluctuation in the pixel error estimate than the fast sequences (b).**

to rise immediately from frame 0.

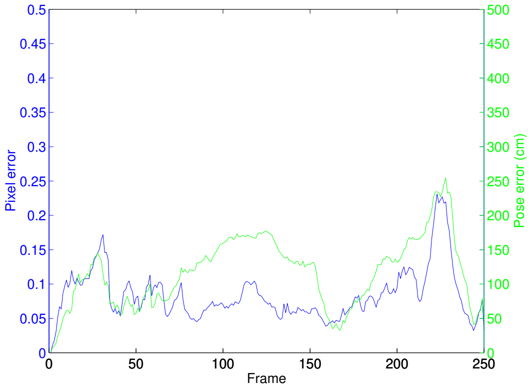


Figure 6.11: **Pixel and pose error for sequence 7.**

In figure 6.12 the input and the pose estimation are shown for frame 0 to 20 in steps of five frames. In this sequence it is noticeable how the wrist position and especially rotation is estimated wrong. Every sequence from 7 to 12 shows this similar behaviour, which suggests that more emphasis must be laid on the estimation of the wrist position and rotation if the algorithm is to work.

One reason why this initial Spider algorithm may fail, is due to the topology of the Spider tree. As the root node of the Spider tree is node 0, as shown in figure 6.5 on page 44, the nature of the Viterbi algorithm implies that nodes $0 \ldots 5$, representing wrist position and rotation, are the last to be evaluated. As wrist transformation imposes relatively big changes in the silhouette as opposed to finger joints, estimating these last may not be optimal.

Furthermore, as the final estimated pose from the previous frame is used as the start pose for the estimation of the current frame, the error is propagated as the sequence progresses.

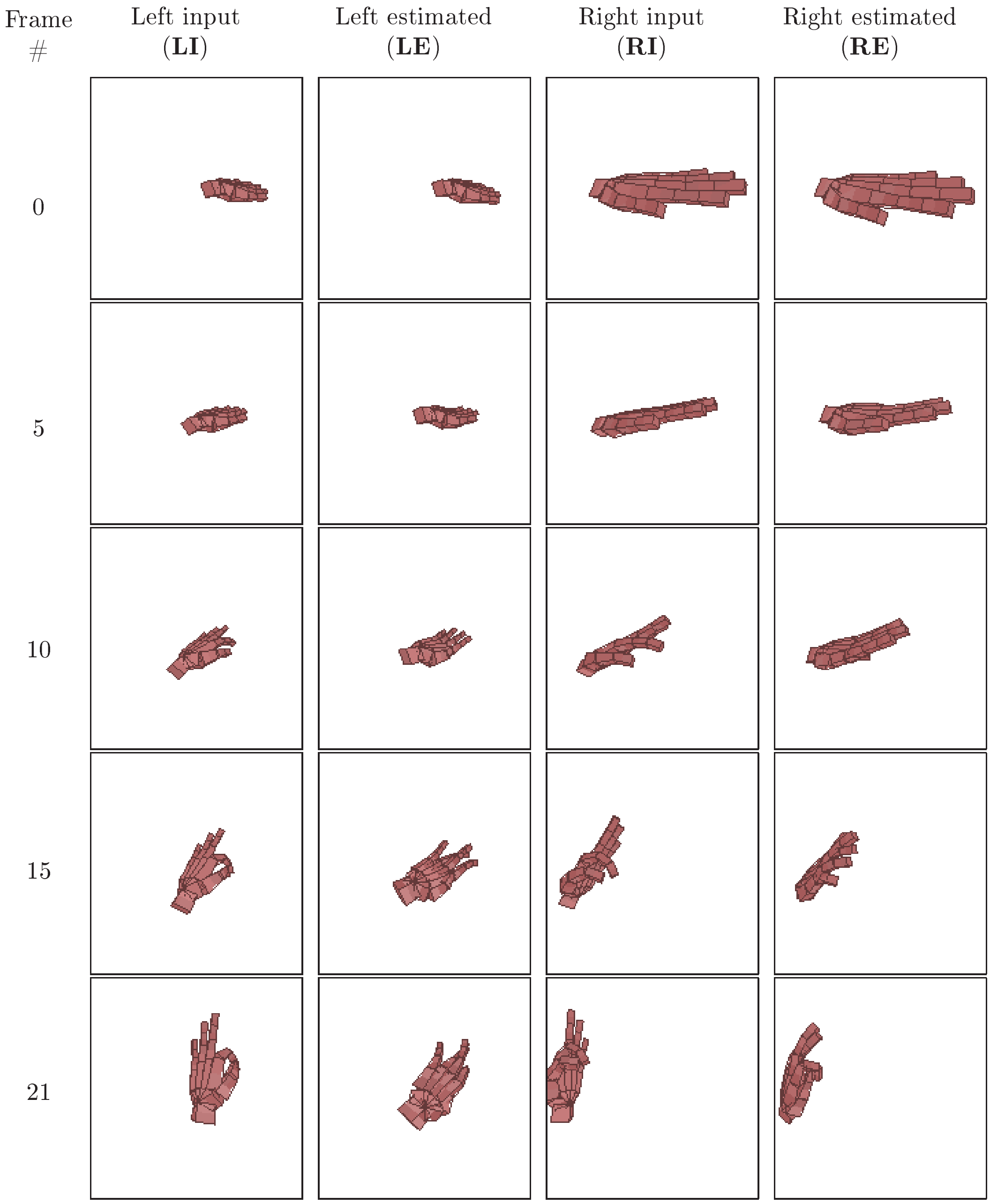


Figure 6.12: **Example of estimation failure in sequence with wrist movement. In frame 0 and 5 the pose is correct, but as the sequence progresses from here the estimation begins to fail. Notice especially how the wrist rotation is estimated wrong.**

A last point to be made is concerning the minimum of the pixel errors depicted in figure 6.10. From just looking at the graphs one may get the impression that the algorithm manages to obtain a better fit in periods of each sequence. This is indeed true but keep in mind that much of this improvement is due to the input images gradually approaching the wrongly estimated pose, allowing for the Spider algorithm to catch up again. This explains why sequences 10-12 show fast fluctuations while sequence 7-9 show more soft fluctuations.

As a general conclusion to the initial tests, the Spider algorithm is feasible and in fact very capable of handling the optimisation problem as long as the wrist is fixed. When introducing wrist movement the Spider algorithm is prone to errors. It is only expected as more DoFs are changing and thus have to be estimated.

As discussed earlier in the text, choosing good Δ values is important to the algorithm. Varying these values may give better results for the sequences where wrist movement occurs, but instead of spending valuable time optimising these values, modifications that emphasise estimation of wrist transformation are examined.

In the following this initial implementation is referred to as the original Spider while any modified version is referred to as a modified Spider.

6.4 Changing Tree Topology

The first modification considered is relatively simple. As discussed in the previous section a possible problem with the tree topology presented in figure 6.5 is the sequence in which the nodes are evaluated and hence the χ values estimated. Due to the recursive behaviour of the Viterbi algorithm, this causes the values of nodes $5 \ldots 0$ to be estimated last. A simple way to make the estimation of these nodes the very first is by choosing another root node.

The tree topology is modified by choosing node 22 as root node and making node 4 the first child of node 5. This way nodes $0 \ldots 5$ are estimated first instead of last. The modified tree structure is shown in figure 6.13

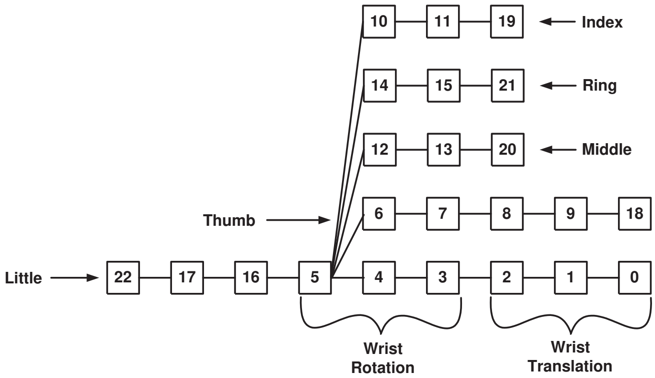


Figure 6.13: **Restructured Spider tree. By making node 22 the root node, another order of estimation is obtained. The new order of estimation is; wrist-, thumb-, middle-, ring-, index- and finally little finger. By estimating the wrist position and rotation first more emphasis is laid on this.**

Results

As the only thing changed from the initial tests is the root node, 10 iterations for each frame are allowed yet again. Also, the same decrease in Δ values is chosen. The complete results are found in appendix E on page 133.

Furthermore as the original Spider shows very good results in the learning sequences where the wrist is fixed, the following concentrate on sequences 7-12 due to the significant problems the original Spider has with these.

When reviewing the test results in figure 6.14 on the following page the general impression is that a little advantage is gained from switching the topology. With the exception of sequence 8 (figure 6.14(b)), the modified Spider manages to reduce or keep the same pixel error as the original. A highlight is sequence 10, figure 6.14(d), where the modified Spider manages to eliminate the spike that occurs for the original at frame 50 to 60. This is also observed in sequence 8 for frames 25 to 60, but here a new spike occurs around frame 190 instead.

In table 6.2 the results of the modified Spider compared to the original are shown.

	Pose			Pixel		
Sequence	Original	Modified	Difference	Original	Modified	Difference
7	117	135	−18	0.0843	0.0881	−0.0038
8	117	136	−19	0.1000	0.1075	−0.0075
9	140	127	+13	0.1092	0.1024	+0.0068
10	141	109	+32	0.1123	0.0972	+0.0151
11	157	142	+15	0.1223	0.1112	+0.0111
12	128	116	+12	0.1006	0.0867	+0.0139

Table 6.2: **Comparison of errors for the original and modified Spider topology. A (+) in the difference section indicates that an improvement is obtained using the modified Spider, while a (−) indicates that the original comes out best..**

Using the Wilcoxon test no statistical significant difference from the original to the modified Spider can be proven, neither for pixel nor pose error.

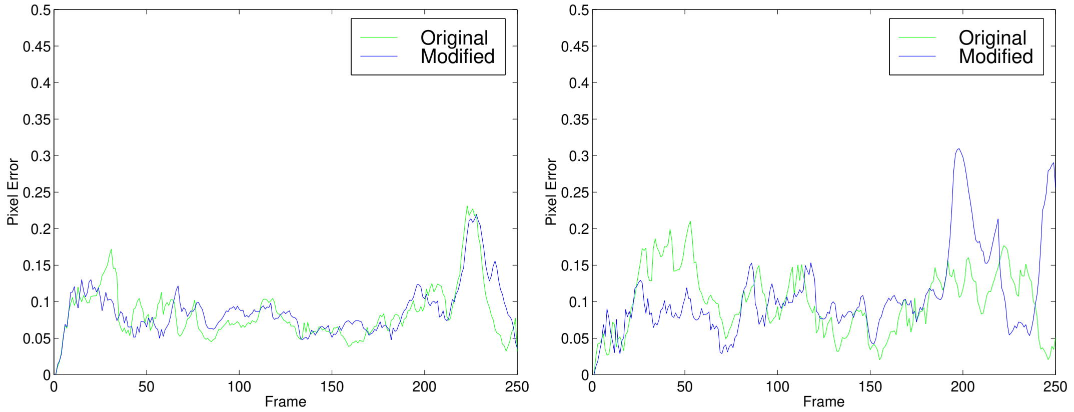
In general conclusion to the test results, modifying the Spider topology showed no statistical improvement in the overall estimation, and the errors are still too big to produce a satisfying pose estimate of the hand.

6.5 Modifying the Node Structure

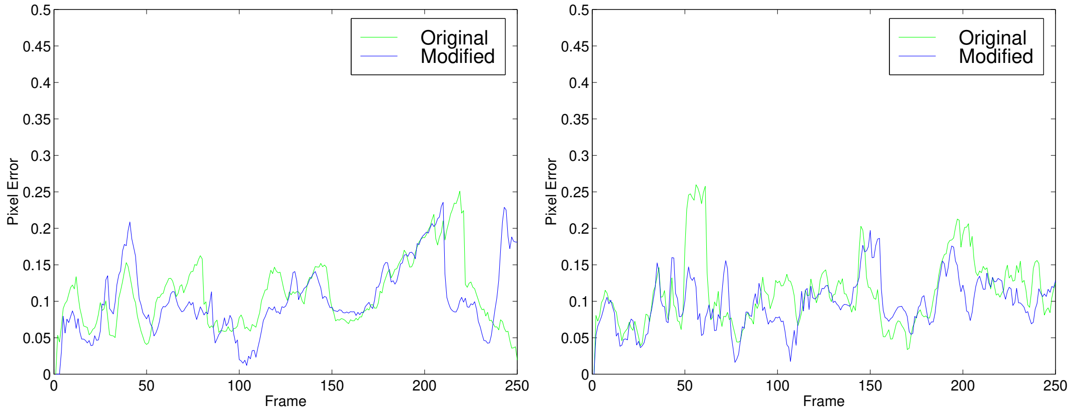
Another approach to emphasise the estimation of the wrist position and rotation is modifying the structure of the Spider nodes. The initial implementation uses only one dimensional nodes, where each node represents one χ value. The traditional snake used in image processing uses two dimensional nodes as shown in figure 6.2 on page 41, allowing for different combinations of the x and y position for the same node. As wrist rotation consists of three χ values, 3D nodes are introduced.

Three Dimensional Node Structure

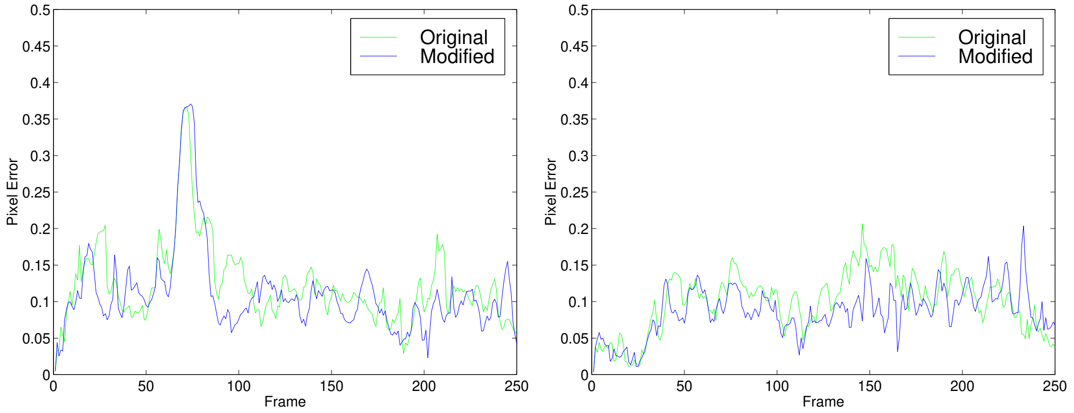
The concept of extending the Spider nodes to handle three dimensions is quite simple. Three 1D nodes each consisting of three possible values are combined into one node where all possible combinations of the three 1D nodes are represented as shown in figure 6.15.



(a) Sequence 7 (b) Sequence 8



(c) Sequence 9 (d) Sequence 10



(e) Sequence 11 (f) Sequence 12

Figure 6.14: **Comparison in pixel error between the original Spider and the modified Spider tree. No statistical significant changes are observed although the modified Spider seems to cut down some spikes.**

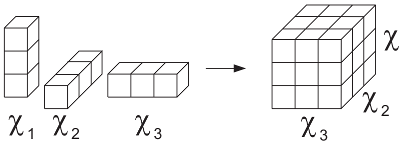
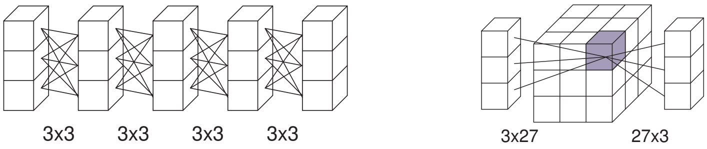


Figure 6.15: **Extension of the node from 1D to 3D. By combining the three 1D nodes into one 3D node more emphasis on the coherence between the χ values is obtained.**

By creating a 3D node, more emphasis on the coherence between the χ values is obtained.

In the one dimensional case the energies of each possible position for χ_2 is calculated for each possible position of χ_3. Afterwards the energies at possible χ_1 positions are calculated for each possible χ_2 position. As such, χ_1 and χ_3 are only indirectly joined by χ_2.

Using a 3D node, the three values are directly joined, and thus more emphasis on their relationship is obtained. This emphasis is made at the expense of more computational power, as shown in figure 6.16.



(a) 1D nodes (b) 3D node

Figure 6.16: **Computational overhead example. One iteration of five 1D nodes requires $4 \times 3 \times 3 = 36$ evaluations (a). Replacing the three middle nodes with one 3D node increases the number of evaluations to $2 \times 3 \times 27 = 162$ for one iteration (b).**

In this example a tree with five 1D nodes is compared to a tree where the three middle nodes are replaced by a 3D node. In figure 6.16(a) the computation between the first and second node requires 3×3 evaluations, when assuming three neighbours. As four connections exist the total comes to $3 \times 3 \times 4 = 36$ evaluations. In figure 6.16(b) each cell in the cube must be evaluated three times with the previous 1D node and three times with the subsequent node. As this cube consists of $3^3 = 27$ cells, the total number of evaluations comes to $2 \times 3 \times 27 = 162$.

In this modification the 3D Spider nodes are introduced to replace the three nodes for wrist position and the three nodes for wrist rotation respectively. The new tree structure is shown in figure 6.17.

Results

Because of the computational overhead, the 3D Spider is only allowed 5 iterations for each frame. Instead of decreasing the initial Δ values with 20% for every second iteration, this is done for each iteration. The test results can be found in appendix E on page 134.

The general impression when looking at figure 6.18 is that some improvements in the estimation have occurred. Visually the pixel estimation in all slow sequences have improved, most notably in sequences 8-9, figures 6.18(b) and 6.18(c). By using 3D nodes the modified algorithm manages to reduce the pixel error or at least maintain the level of error, causing the spikes which occurred using the original algorithm, to be levelled.

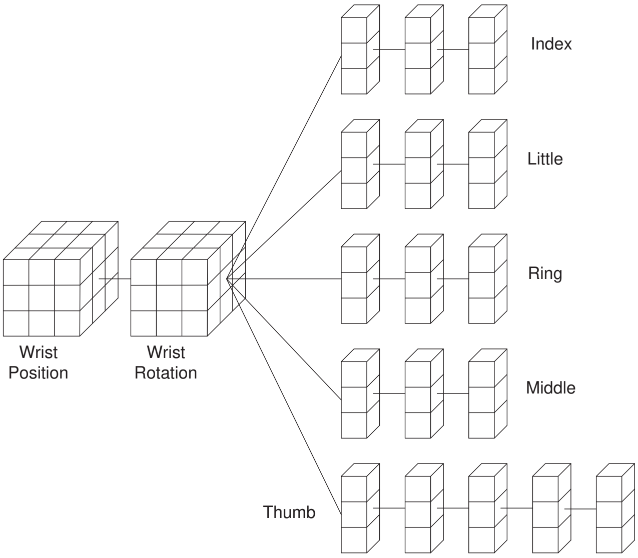


Figure 6.17: **Modified Spider tree structure. The six 1D nodes representing wrist position and rotation are replaced by two 3D nodes.**

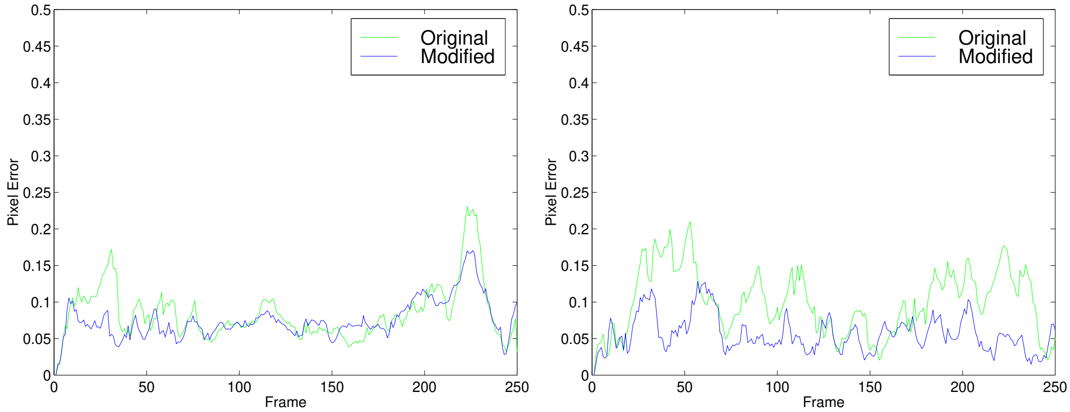
However, little improvement is noticed in the estimation of the fast sequences and in the case of both 6.18(e) and 6.18(f) the pixel error has actually deteriorated with the use of 3D nodes. The results indicate that at present the choice of Δ values is optimised towards the slow sequences. Looking at table 6.3 it is clear that the modified algorithm improves or at least maintains previous pixel errors for the slow sequences. Note, that even though sequence 7 actually improves in pixel errors, the average pose error is worse. Conversely the pixel error in the fast sequences have not improved.

	Pose error mean			Pixel error mean		
Sequence	Original	Modified	Difference	Original	Modified	Difference
7	117	120	-3	0.0843	0.0755	$+0.0085$
8	117	54	$+63$	0.1000	0.0563	$+0.0437$
9	140	90	$+50$	0.1092	0.0811	$+0.0281$
10	141	142	-1	0.1123	0.1045	$+0.0078$
11	157	175	-18	0.1223	0.1385	-0.0162
12	128	136	-8	0.1006	0.1069	-0.0063

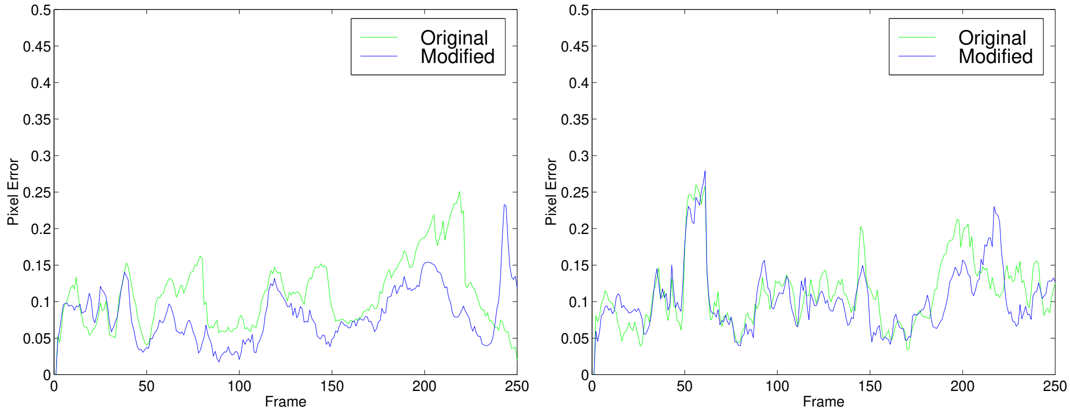
Table 6.3: **Comparison of errors for the original and 3D Spider topology. A (+) in the difference section indicates that an improvement is obtained using the modified Spider, while a (−) indicates that the original comes out best. Notice the high reduction in error, both pixel and pose wise, for sequences 8 and 9.**

Although sequences 8 and 9 show big improvements in both pixel and pose error, the negative results of the other sequences cause the Wilcoxon test to accept its NULL-hypothesis at a confidence level of 95%, implying that statistical speaking no significant change has occurred.

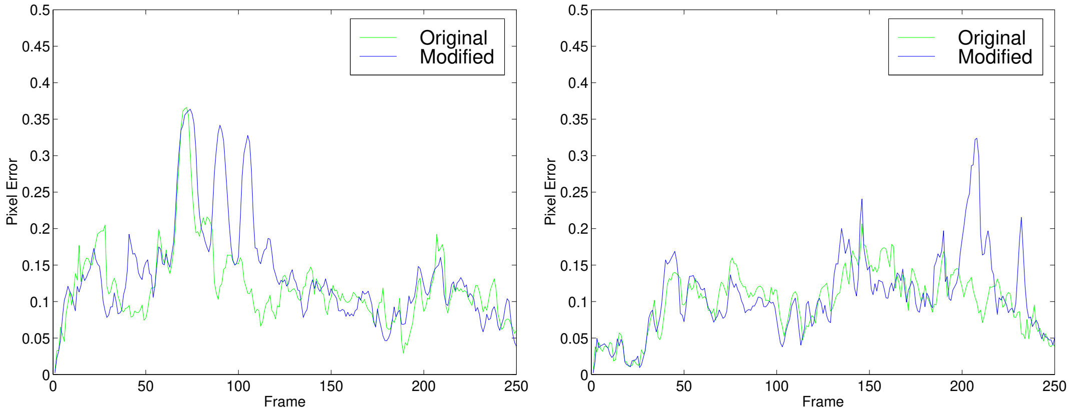
The results are nevertheless promising as the pixel errors of estimating the pose in sequence



(a) Sequence 7 (b) Sequence 8



(c) Sequence 9 (d) Sequence 10



(e) Sequence 11 (f) Sequence 12

Figure 6.18: **Comparison in pixel error between the original Spider and using 3D nodes to represent wrist position and rotation. A definite improvement is noticed, in the slow sequences (a)-(c), while the pixel error in the fast sequences are unchanged or worse than those obtained with the original Spider.**

8 are approaching an acceptable level. It is important to note that a low pixel error does not necessarily result in a low pose error. Sequence 7 is a good example of this. Compare the results of sequence 7 and sequence 9 in table 6.3. Even though sequence 7 is estimated with an average pixel error of 0.0755, which is 0.0056 less than the average 0.0811 of sequence 9, the average pose error is 120 cm, 30 cm more than the average of 90 cm in sequence 9.

To conclude the test of the Spider using 3D nodes, the algorithm showed considerable improvements in the slow sequences, but not in the fast sequences. Due to this, no statistical significance between the original and 3D Spider can be proved. One possible cause may be that the choice of Δ values suits the slow sequences better than the fast sequences.

6.6 Splitting the Estimation

As a last approach to emphasising wrist position and rotation, a splitting of the estimation into two parts is considered. The first part only tries to optimise $\chi_0 \dots \chi_5$, the values for the wrist, while the second part of the estimation considers $\chi_0 \dots \chi_{22}$ values. The flow is shown in figure 6.19.

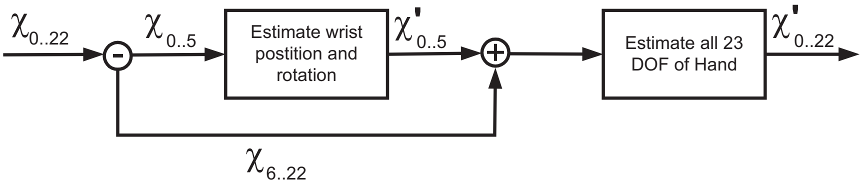


Figure 6.19: **Splitting the estimation. The Spider estimation is split into two where the first part only estimates $\chi_0 \dots \chi_5$. These new $\chi'_0 \dots \chi'_5$ values are then merged with $\chi_6 \dots \chi_{22}$ and used in an estimation of all χ values.**

Part I - Wrist Estimation

As described above, the first part aims at estimating the wrist rotation and position, more specifically $\chi_0 \dots \chi_5$. Two approaches for synthesising the evaluation silhouettes can be considered:

- Continue using the full model of the hand for the synthesis.
- Use only a model of the palm of the hand for the synthesis.

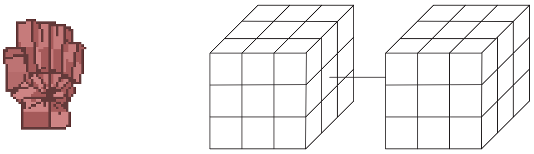
If the full model is used a decision must be made concerning the values $\chi_6 \dots \chi_{22}$. Although these are not to be evaluated they must still be provided in the synthesis process. A reasonable choice is the pose estimate from the previous frame. Kameda [15] avoided the use of these χ values and was able to obtain reasonable results by using only a model of the palm. It is chosen to go with this option[1].

In effect the same model of the hand is used, but synthesis of fingers is prevented. An example of the palm is shown in figure 6.20(a).

The tree structure used for this first part is shown in figure 6.20(b). As use of 3D nodes yields the best results so far, these are employed in the Spider tree for the first part.

---

[1] Note that Kameda did not take wrist transformation into account.

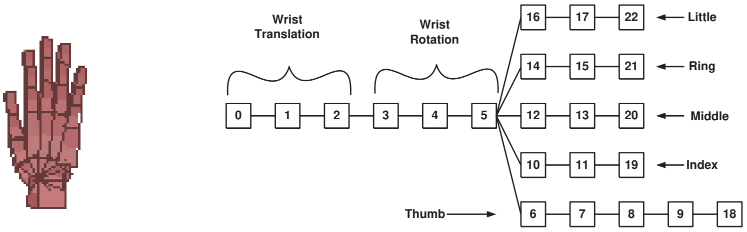


(a) Palm model. (b) The spider tree.

Figure 6.20: **Silhouette and tree used for the wrist estimation part.**

Part II - Whole Hand Estimation

Although using 3D nodes yields better pose estimation, the original Spider tree is chosen for the estimation of the whole hand. This is done because the first part of this new algorithm already employs 3D nodes from which a reasonable estimation of the wrist position and rotation is expected, and the original Spider shows good results on sequences where the wrist transformation is known. The model of the hand from the previous test is of course used for this part.



(a) Hand model. (b) The spider tree.

Figure 6.21: **Silhouette and tree used for the whole hand estimation.**

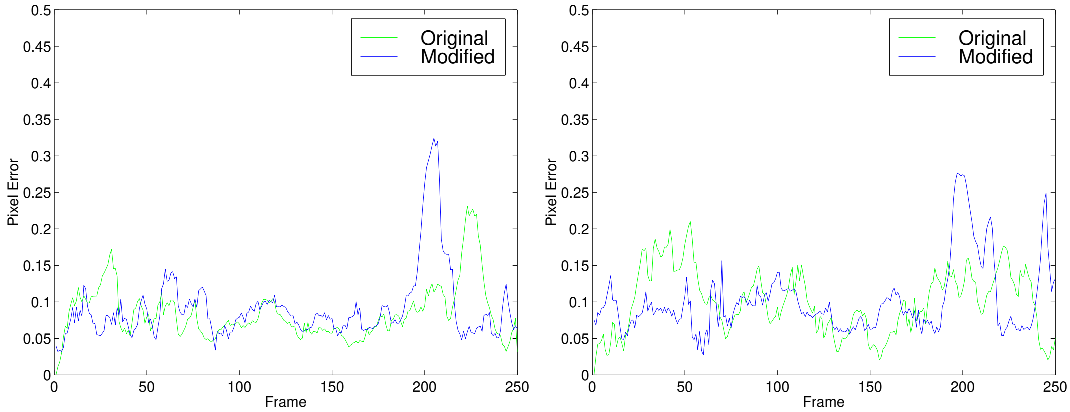
6.6.1 Results 3rd Modification

In this test five iterations of wrist estimation are performed followed by ten iterations estimating the whole hand. Complete results can be found in appendix E on page 135.

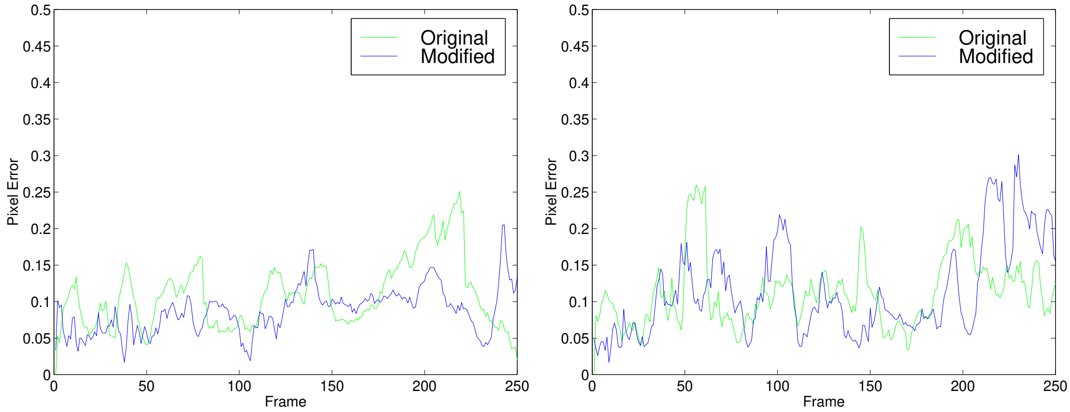
The test presented in figure 6.22 reveals ambiguous results. The two part Spider is able to reduce some of the spikes in pixel error which the original produced. However, the algorithm generally loses track at some point in the second half of each sequence. This results in new spikes, at points where the original algorithm performs better.

Again it is in the slow sequences the modified Spider performs best. Table 6.4 compares the pose and pixel error of the modified Spider with the original Spider. Here it becomes obvious that the modified Spider does not offer any significant advantage over the original Spider when comparing pixel and pose error over 250 frames.

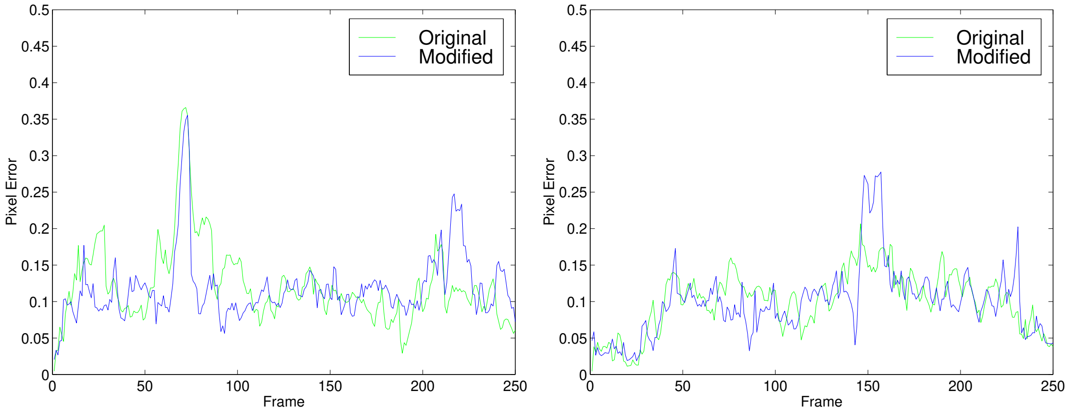
Once again statistically speaking, no difference in estimation error between the original and modified can be proven using the Wilcoxon signed rank test at a confidence interval of 95%. The estimation of sequence 9 separates from the pack, showing an improvement in pixel error of 0.0219.



(a) Sequence 7 (b) Sequence 8



(c) Sequence 9 (d) Sequence 10



(e) Sequence 11 (f) Sequence 12

Figure 6.22: **Comparison in pose error between the original Spider and splitting the estimation in two. As was the case with the use of 3D nodes a general improvement is observed. Notice especially the long series frames in which sequences 7, 8 and 11 are estimated with previously unseen low pose errors.**

	Pose error mean			Pixel error mean		
Sequence	Original	Modified	Difference	Original	Modified	Difference
7	117	115	+2	0.0843	0.0913	−0.0070
8	117	125	−8	0.1000	0.1013	−0.0013
9	140	119	+21	0.1092	0.0873	+0.0219
10	141	149	−8	0.1123	0.1139	−0.0016
11	157	156	+1	0.1223	0.1181	+0.0042
12	128	127	+1	0.1006	0.0983	+0.0023

Table 6.4: **Comparison of errors for the original and splitting the estimation. A (+) in the difference section indicates that an improvement is obtained using the modified Spider, while a (−) indicates that the original comes out best.**

6.7 Summary

In this chapter the principle of optimisation using snakes, and the generalisation from snakes to Spiders was presented. An initial Spider structure to be used for model fitting of the hand was presented and tested using the learning sequences. The algorithm proved very capable of estimating pose in sequences where wrist transformation is known. The pose estimation error in such a case was very low.

Problems arose when the wrist transformation had to be estimated as well. In such cases the algorithm failed to estimate a usable pose of the hand over time. The conclusion was that more emphasis should be on the estimation of wrist transformation. Three modifications to the original Spider was presented and tested on the same learning sequences; change of topology, introducing 3D nodes, and splitting the estimation in two parts. The modifications varied in success, as can be seen in table 6.5. All the modifications improved the estimation for some sequences while performing worse on others.

	Pose mean				Pixel mean			
Seq.	Org	New Top.	3D	Two part	Org	New Top.	3D	Two part
7	117	−18	−3	+2	0.0843	−0.0038	+0.0085	−0.0070
8	117	−19	+63	−8	0.1000	−0.0075	+0.0437	−0.0013
9	140	+13	+50	+21	0.1092	+0.0068	+0.0281	+0.0219
10	141	+32	−1	−8	0.1123	+0.0151	+0.0078	−0.0016
11	157	+15	−18	+1	0.1223	+0.0111	−0.0161	+0.0042
12	128	+12	−8	+1	0.1006	+0.0139	−0.0063	+0.0023

Table 6.5: **Comparison of errors for the original and the different modifications. A (+) in the modification section indicates that an improvement was obtained.**

For each modification a Wilcoxon signed rank test was performed comparing the original Spider with the modification, with a confidence interval of 95%. As all of these tests accepted the NULL hypothesis (both samples equal); statistically speaking no improvement was gained from any of the above modifications.

The Wilcoxon test compares the mean estimation error in the sequences and this cannot be regarded as an optimal comparison in this case, as errors are propagated from frame to frame.

Also this chapter has only touched on the effect of changing and gradually decreasing the

Δ values, and this should also be subject to further investigation.

Although, statistically speaking, none of the above modifications can be proven to make a definite improvement in pixel and pose estimation. Using 3D nodes for wrist translation and rotation showed improvements in the estimation of sequence 8 and 9.

Based on this the 3D Spider implementation is chosen to be compared with the Downhill Simplex algorithm and its modification, the Downhill Simplex/Simulated Annealing-hybrid which will be presented in the following two chapters.

Chapter 7

Downhill Simplex

> "If you find yourself in a hole, the first thing to do is stop diggin'."
>
> - *Texas Bix Bender*

In the last chapter, a new method was presented and applied to the problem of model fitting. In order to get an idea of how it performs compared to already well-known methods in the field of model matching, the method proposed by Ouhaddi and Horain [13] is discussed and implemented as it seems a successful attempt to solve the problem of model matching; although their solution does not allow for wrist rotation. Among other things, this constraint will be removed in this chapter.

Ouhaddi and Horain [13] showed the effectiveness of Downhill Simplex (DS) over the "Levenberg-Marquardt" and "Powell's" multidimensional optimisation strategies with respect to the specific problem of model matching. In the light of their work it does seem feasible to pick DS over Levenberg-Marquardt or Powell's method.

Many multidimensional optimisation methods are fundamentally based on one-dimensional optimisation strategies, each dimension being optimised more or less independent of other dimensions. The DS method uses an alternative strategy, described briefly in this chapter.

DS is due to Nelder and Mead [35]; our implementation is derived from Press et al. [27] with modifications to suit the specific circumstances. It should be stressed at this point that numerous people have rated this book as doubtful ([36, 37]). After critical examination and fixes, though, it seems to contain a reasonable implementation of the Downhill Simplex algorithm.

7.1 Algorithm

In short, a series of points - referred to as vertices - is strategically distributed in R^n, each vertex representing a possible solution. The vertices are moved around in the search space according to some rules, and the geometrical figure they constitute - referred to as a simplex - slowly contracts around the lowest energy vertex.

A simplex is the most elementary geometrical figure that can be formed in $\Re^n$ and has $n+1$ vertices (each vertex is a n-dimensional vector representing a point in $\Re^n$), so the simplex consists of a series of states, $\mathbf{s_0} \ldots \mathbf{s_n}$. On a plane (two dimensions, $\Re^2$), a simplex is a triangle. In 3D space, $\Re^3$, a tetrahedron, in higher dimensions a hyper tetrahedron. If any vertex of a simplex is considered as the origin, the n other vertices define vector directions spanning entire $\Re^n$. It should be stressed that the vectors spanning $\Re^n$ may or may not be of equal length, thus a simplex is not necessarily regular.

7.1.1 Initialisation

Only one vertex is specified, $\mathbf{s_{init}}$, the other n vertices can be found using equation 7.1 where $\mathbf{e_m}$ is the m'th unit vector and λ_m specifies the length scale of this vector direction.

$$\mathbf{s_m} \quad = \quad \mathbf{s_{init}} + \lambda_m \mathbf{e}_m \quad \text{for} \quad m = 1 \ldots n \tag{7.1}$$

A simple choice for $\mathbf{s_{init}}$ in this case is the pose estimate vector from last frame; the initial simplex is built up from this vertex and each axis of $\Re^n$ is added. The simplex size should, according to Ouhaddi and Horain [13], be maximised within $\Re^n$, though still within the constraints discussed later.

7.1.2 Transformations

A series of transformations of the simplex is now performed, each transformation generating a new simplex based on its previous geometry and the cost function evaluations at every vertex. Most steps simply move the highest-energy vertex through the opposite simplex face to a lower-energy vertex, these steps are called reflections. Other transformations are expansions, contractions and multiple contractions ("shrinkage"). Each step is visualised in figure 7.1, for clarity reasons only shown in $\Re^2$, and explained in the following.

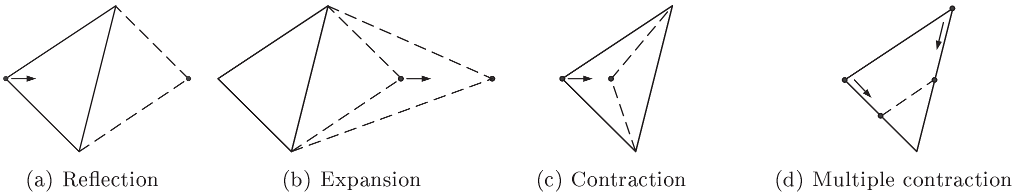

(a) Reflection (b) Expansion (c) Contraction (d) Multiple contraction

Figure 7.1: **DS geometrical transformations in $\Re^2$.**

1. Reflection - Figure 7.1(a). A new vertex is found by a reflection of highest-energy vertex through the opposite simplex hyper plane, replacing the worst vertex with the new one.

2. Expansion - Figure 7.1(b). If the energy evaluated at the new point found in step 1 equals or is lower than the old vertex, the step was in the right direction and another step is made in the same direction to speed up the search process. This transformation increases the hyper volume of the simplex.

3. Contraction - Figure 7.1(c). A contraction along a hyper line from the vertex through the opposite simplex side, thereby trying an intermediate point, is performed if the energy at the reflected vertex (step 1) equals or is worse than the function evaluated at the second-worst vertex.

4. Multiple Contraction - Figure 7.1(d). The simplex is shrunk if neither of the two previous moves improve the situation. In a multiple contraction, all higher energy vertices are moved towards the lowest-energy vertex, thus shrinking the hyper volume of the simplex and contracting around the best vertex.

Figure 7.2 shows a flowchart for the decision process. As can be seen, a reflection will be performed as a first step. The decision process from this point tries some of the predefined transformations, hopefully improving the situation. If the situation cannot be improved, the simplex shrinks around the best vertex.

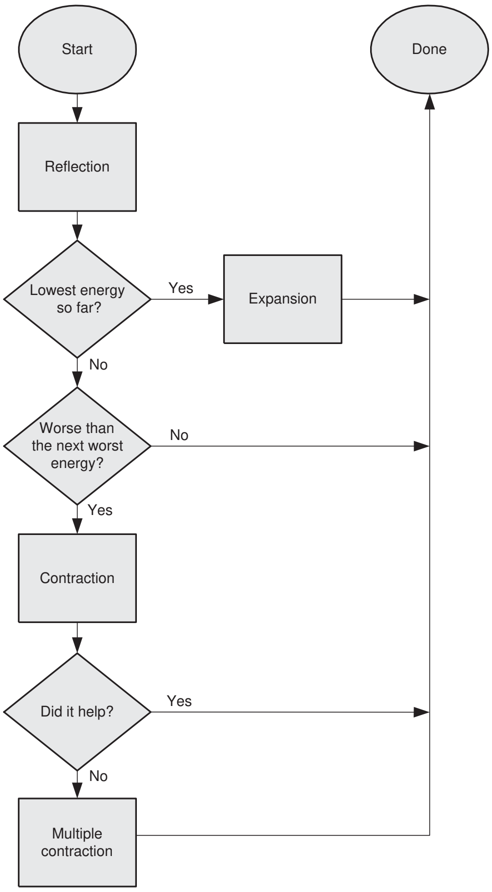


Figure 7.2: **Flowchart illustrating the decision process for a single DS iteration. A reflection will *always* be performed as a first step, regardless of the situation; a multidimensional contraction only if the situation seems hopeless.**

Press et al. [27, chap. 10] states that an appropriate sequence of these steps will always converge to at least a local minimum energy vertex of the specified objective function. However this is only partly true as Kaczmarczyk [38] apparently found specific cases where it did *not* converge in higher dimensions scenarios.

It is at this point important to stress that the number of iterations does not equal the number of evaluations, or the number of transformations:

- Iterations - A number describing how many times the decision process in figure 7.2 is called. Thus, each iteration can involve from one to three transformations.
- Transformations - Any of the geometrical transformations depicted in figure 7.1. Each transformation requires from one to n evaluations.

- Evaluations - How many times the cost function is evaluated.

These indeed are dependent, but they describe different aspects of the DS search on different abstraction levels.

7.1.3 2D Example

Figure 7.3 shows a simple example in $\Re^2$ of how the initial purple simplex is replaced by the green simplex after nine transformations.

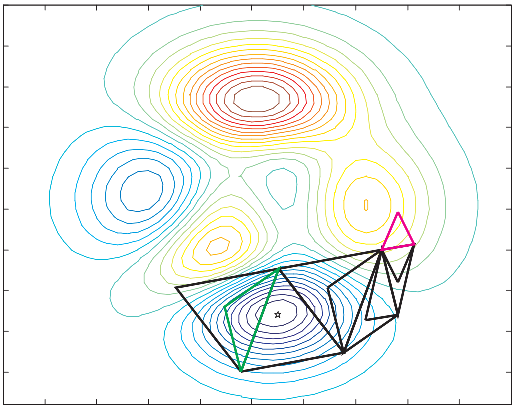


Figure 7.3: **Simplex transformations in $\Re^2$ for a simple example. The purple initial simplex is after nine iterations transformed to the green simplex, clearly closer to the global minimum (marked using a star).**

The transformations are in this example, in order: Reflection, expansion, reflection, expansion, reflection, expansion, reflection, reflection, contraction. More steps will contract the simplex around the global minimum, marked with a star, until a specified floating point tolerance threshold is reached or the maximum number of iterations are performed.

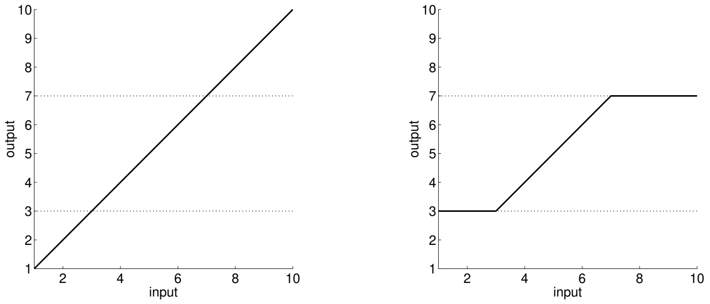
7.2 Constraints and Downhill Simplex

As described in section 6.3.1 on page 44, the feasible search space can be very clearly defined using the Spider algorithm, but the simplex presented in the last section is free to move anywhere within $\Re^n$, thus in principle requiring the flexibility of the hand to be infinite. As this clearly is not the case, this problem must be solved by introducing constraints limiting the search space from entire $\Re^n$ to a feasible subspace Ω of $\Re^n$, by restricting each dimension i to values within χ_i^{min} and χ_i^{max}. This assures that each parameter lies in the individually defined feasible interval.

Constrained search can be implemented in several ways, some more intuitive than others. Some methods for constraining the search are presented and analysed in this section and a method is chosen based upon the analysis.

7.2.1 Truncation

Perhaps the easiest method to implement constraints is simply to truncate values outside Ω. That is, if an element χ_i in the pose vector $\mathbf{s}$ at any time goes below its feasible minimum value χ_i^{min} or above its maximum value χ_i^{max}, it is in the evaluation function simply set to the bounding value, hereby leaving the simplex unaltered. Figure 7.4 shows the principle of truncation.



(a) Without truncation

(b) With truncation

Figure 7.4: **Principle of truncation, (a) is without truncation and (b) shows how χ_i is truncated to lie within the interval $[\chi_\mathbf{i}^\mathbf{min};\ \chi_\mathbf{i}^\mathbf{max}]$**

In figure 7.4(a), no constraints are implemented. In figure 7.4(b) χ_i^{low} is arbitrarily set to 3 and χ_i^{max} to 7. So, if the optimisation algorithm suggests a value of 10 for χ_i, this value is truncated to 7 before evaluating the energy function.

$$\chi_i' = \begin{cases} \chi_i^{min} & \text{for } \chi_i < \chi_i^{min}, \\ \chi_i^{max} & \text{for } \chi_i > \chi_i^{min}, \\ \chi_i & \text{for } \chi_i^{min} < \chi_i < \chi_i^{max}. \end{cases} \tag{7.2}$$

This approach seems reasonable but involves some problems: If χ_i is truncated to χ_i^{min} or χ_i^{max} the algorithm may get stuck because the error function does not give any information as to which direction is favourable.

A variation of the simple truncation method discussed can be implemented by using e.g. some kind of Sigmoid function (known from Neural Networks) to normalise χ_i from $]-\infty;\ \infty[$ to $[\chi_i^{low};\ \chi_i^{high}]$ prior to evaluation.

Equation 7.3 shows the definition of the basic Sigmoid function and figure 7.5 shows an example plot based on this formula[1]. Compare to figure 7.4(b).

$$\chi_i' = \frac{1}{1 + e^{-\chi_i}} \tag{7.3}$$

This method does not face the same risk of getting stuck, as the distance from the suggested unfeasible solution to Ω is somewhat taken into account. However, it requires many iterations to reach χ_i^{max} or χ_i^{min} for χ_i because of the relatively slow convergence to the limits.

7.2.2 Simplex Restriction

Another method of implementing constraints is proposed in Ouhaddi and Horain [13]: The simplex transformations are geometrically bounded inside Ω. If at any time a vertex is moved outside the constraint domain Ω as a result of a simplex transformation, it is replaced by

---

[1] Figure 7.5 is *not* obtained directly from formula 7.3, it merely visualises its behaviour and the characteristic "S"-shape.

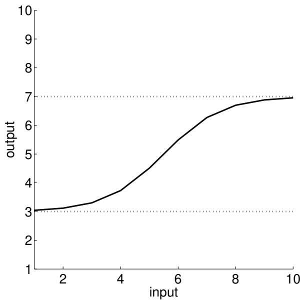


Figure 7.5: **A plot visualising behaviour of a Sigmoid-based function. Notice the slow convergence to the limits χ_i^{low} and χ_i^{max}.**

a new vertex at the intersection of the line connecting the old and the new vertex, and the violated constraint hyper plane. Figure 7.6 shows the principle of this method.

This method assures that the constraining hyper plane is not exceeded: A simplex is defined in $\Re^2$ as the triangle $(\mathbf{A}, \mathbf{B}, \mathbf{C})$. Vertex **A** has highest energy, thus DS will reflect this point to the a new point, **D**. If this point lies outside the constraint hyper plane it will be projected onto the hyper plane (in this case **E**). The new simplex now consists of $(\mathbf{B}, \mathbf{C}, \mathbf{E})$ instead of $(\mathbf{B}, \mathbf{C}, \mathbf{D})$.

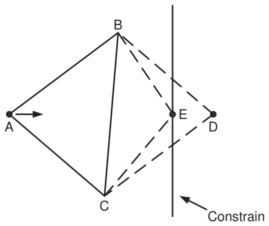


Figure 7.6: **Simplex geometrically restricted inside Ω. Vertex D is replaced by vertex E in the simplex.**

A disadvantage to this approach is that it directly violates the four basic simplex transformations explained in section 7, thereby affecting the optimisation procedure. An example of a doubtful situation is shown in figure 7.7.

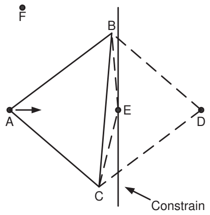


Figure 7.7: **Simplex restriction problem, notice the elongation of the simplex $(\mathbf{B}, \mathbf{C}, \mathbf{E})$ which causes difficulties reaching, as an example, point F.**

One can see that the new simplex, $(\mathbf{B}, \mathbf{C}, \mathbf{E})$, is relatively elongated. The algorithm will hence forward have problems reaching any points (as an example, the vertex **F**) "inside" the domain Ω; the simplex will for a long time after this transformation merely "crawl along" the hyper plane bounding Ω.

7.2.3 Cost Function Penalty

As none of the previous techniques for implementing constraints satisfies us, another approach is considered: Simply make any perturbation outside the constraint domain prohibitively expensive. This can be achieved by, as an example, adding a relatively large number as penalty to the cost function if any element in the vertex is *not* within Ω.

Figure 7.8 shows an example of an arbitrary cost function in R^2, $f(x, y)$, visualised for $x \in [0;\ 24]$ and $y \in [0;\ 24]$, without constraints.

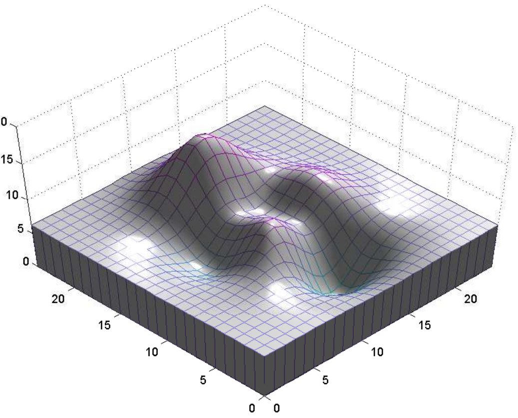


Figure 7.8: **The surface of a simple 2D cost function without constraints. Nothing stops the simplex for searching outside Ω.**

The simplex is free to explore the entire landscape (also outside the visualised range), hopefully finding the global minimum.

Figure 7.9 visualises the effect of applying a cost function penalty. The search space is not physically restricted to Ω ($x < 20$ and $y < 20$), yet the simplex stays inside Ω because of the fact that it is prohibitively expensive to climb the "cliffs" bordering Ω. Notice that the penalty is increased as the rate of offence increases.

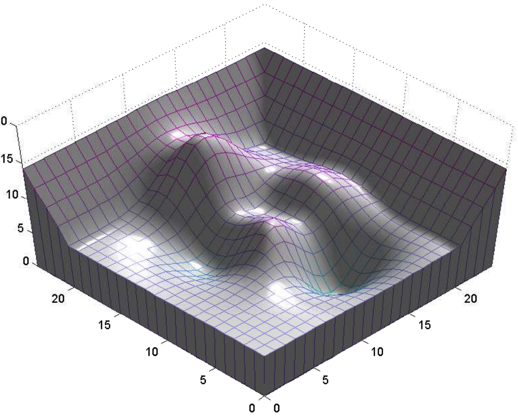


Figure 7.9: **The surface of a simple 2D cost function with constraints for $x \geq 20$ and $y \geq 20$. The penalty for searching outside the constraints indirectly causes the routine to search only within Ω. The penalty is in this example linear dependent of the rate of offence.**

This method is relatively simple to implement, especially when several factors are internally dependent. Furthermore, whereas dimension truncation and simplex restriction work on individual dimensions or merely a subset of dimensions, this method of cost function penalty can work on not only an individual dimension or a subset of dimensions but also entire configurations and therefore widely different matching techniques can be applied, as shall become clear in chapter 10.

In Mendivii et al. [29] the penalty approach is discussed in-depth, together with other methods of constraining search algorithms.

7.3 General Behaviour

After the theory behind the DS optimisation method has been described, tests are carried out in order to get an impression of its efficiency.

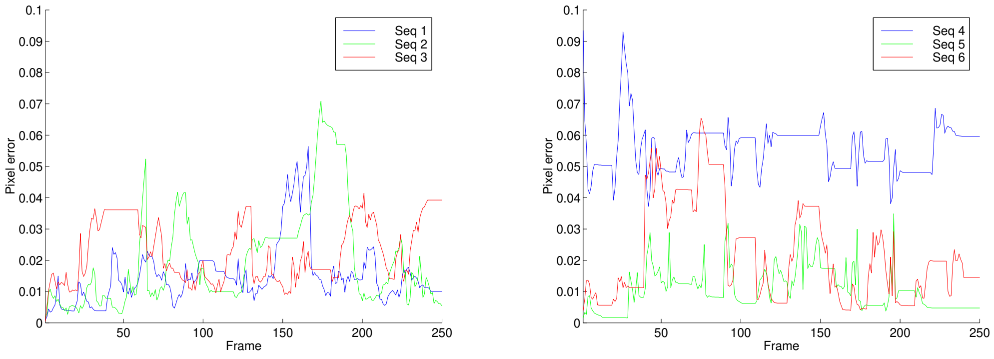
A brief analysis of the convergence process is presented as well, and the question of algorithm restart will be shortly discussed.

7.3.1 Initial Tests

Because DS already *has* been used for hand pose estimation with more or less fixed wrist by Ouhaddi and Horain [13], it seems natural to see how well it performs in this specific case, if the wrist indeed is fixed.

Fixed Wrist

Figure 7.10 shows pixel error for learning sequences 1 to 6; that is, tests using a fixed wrist - compare to figure 6.6 on page 47.



(a) Sequence 1-3

(b) Sequence 4-6

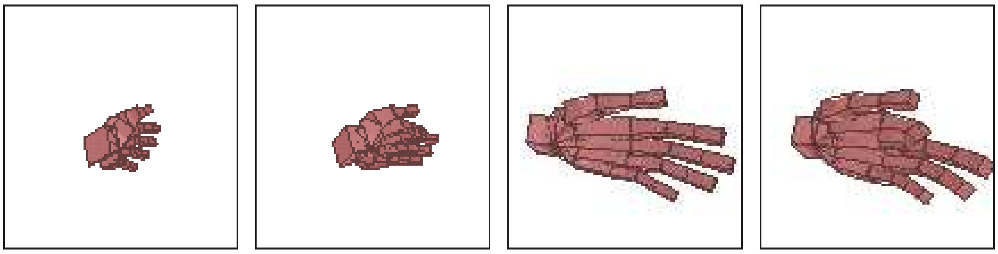
Figure 7.10: **Pixel error for learning sequences 1-6. Notice the flat plateaus caused by no morphing and no wrist movement; DS takes the same series of steps from $\mathbf{s}_{\mathbf{init}}$ and ends at the same wrong vertex.**

The conspicuous flat plateaus arise when no morphing takes place and can be explained by the fact that the simplex starts each frame maximised within Ω, see formula 7.1 on page 64. It takes the same series of steps from $\mathbf{s}_{\mathbf{init}}$ in the same order, and ends at the same (wrong) position in $\Re^{23}$ - due to its deterministic nature. Generally, these sequences have a pixel error of less than 0.06, reflected in acceptable pose errors of less than 100 centimetres in general; see appendix E on page 136.

Sequence 4 clearly behaves differently from the others. The reason for this sequence having an average error of 2-3 times the other sequences may be found in the fact that the hand is displayed from a somewhat awkward angle. This can cause the pose estimation to fail as the finger movement to some extent will be hidden inside the hand silhouette. The error peaks at frame 26; figure 7.11 shows the input pose and estimated pose for this frame.

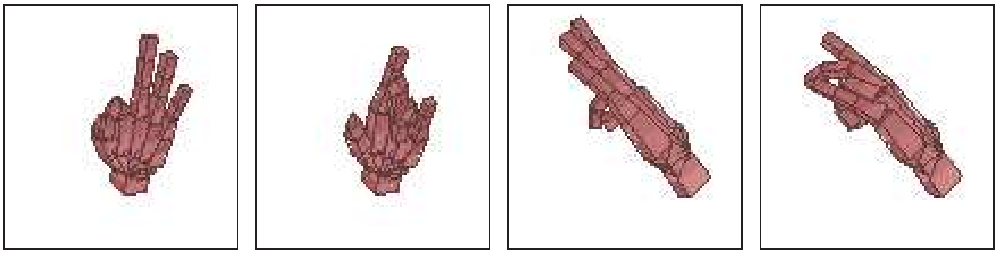
Another interesting example is sequence 2, the green plot in figure 7.10(a) on the facing page, which has a general pixel error of around 0.02 (with a few peaks as exceptions), but increases to a plateaus around frame 140 and stays there until it yet again increases and peaks at frame 174 to a pixel error of more than 0.07, resulting in a pose error of almost 76.

The error is relatively high until around frame 190 where over a few frames it drops to a pixel error just below 0.02.



(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure 7.11: **Frame 26 of learning sequence 4. The awkward point of view may cause the pose estimation to fail as the finger movement to some extent will be hidden inside the hand silhouette. This frame has a pixel error of 0.0935 and a pose error of almost 100 cm.**



(a) Left input (b) Left estimate (c) Right input (d) Right estimate

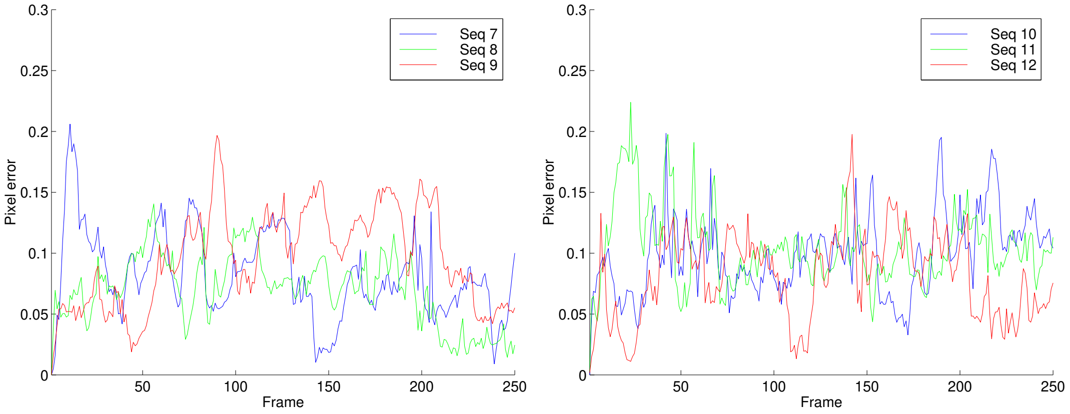
Figure 7.12: **Result from frame 174 of sequence 2. Looking at the left figures, it is clear that the wrong fingers have been chosen for flexion/extension, thereby causing the error.**

Figure 7.12 clearly shows that wrong fingers have been flexed/extended. This misconfiguration is caused by the fact that around frame 150, the index extends instead of the middle finger (thus the flat plateau). This error propagates through the next frames, thereby forcing the hand into an increasingly wrong configuration over time.

The obtained results correspond to the results already obtained by Ouhaddi and Horain [13]; although they used real-life captured videos. An important part of this feasibility study is to examine the problems arising when the position and rotation of the wrist are *not* known, therefore the same apparently effective method is applied to these more complex cases.

Moving Wrist

Figure 7.13 shows the pixel error for learning sequences 7 to 12, for slow (7-9) and fast (10-12) wrist transformation respectively; appendix E on page 137 again shows the corresponding pose plots. Notice the scaling difference from sequences 1-6 to these sequences.

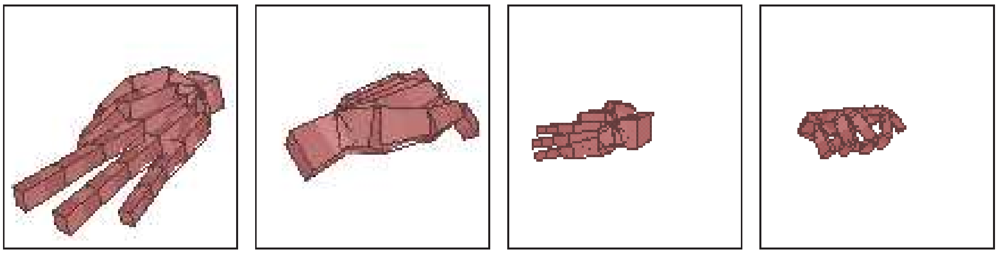


(a) Sequence 7-9 pixel error (b) Sequence 10-12 pixel error

Figure 7.13: **Pixel error for learning sequence 7-12. The flat plateaus from figure 7.10 do not appear because of the wrist being rotated and translated, thereby forcing DS *not* to perform the transformations in the same order over a series of frames.**

The flat plateaus do not appear in these sequences because the hand is never static over a series of frames. However, because of wrist transformation, pixel errors of more than 0.1 are not unlikely to occur, reflected in unacceptable pose errors of up to 300 cm.

Often the cause of the huge pose errors can be traced back to the wrist rotated 180^o around one or two of the axes. As an example, sequence 10 at frame 207 has a pose error of as much as 314 cm, caused by a pixel error of 0.1113. Figure 7.14 shows that this is due to wrong wrist rotation with respect to the y-axis, thereby causing the algorithm to fail.



(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure 7.14: **Result from frame 207 of learning sequence 10, showing how the hand orientation is estimated approximately 180^o wrong. This frame has a pixel error of 0.0895 and a pose error of more than 326.**

Thus it must be concluded that the wrist has exceptionally strong influence on the matching process, just as in the spider algorithm. Theoretically, if the position and rotation of the wrist can be determined *before* fingers are matched the situation will more or less be the same as presented in sequence 1-6, thereby significantly lightening the pose estimation task.

7.3.2 Convergence Process

Because DS in $\Re^{23}$ is very complex, and because at least an impression of the general DS behaviour is desirable, the convergence process is shown for an early frame of sequence 12, see figure 7.15. Convergence graphs for other frames in other sequences have the same general behaviour.

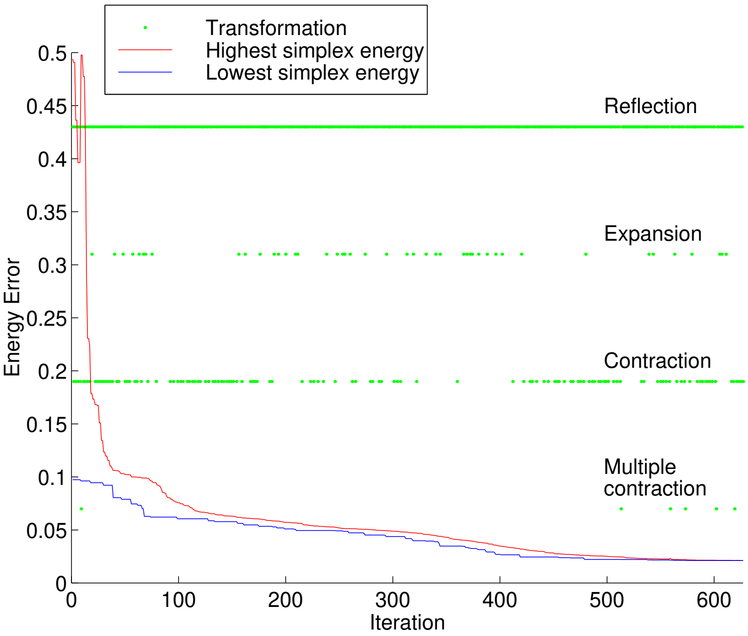


Figure 7.15: **Convergence process. Notice the stepwise decrease of the blue graph compared to the early drastic decrease of the red graph.**

In this figure, the green dots (although overlapping) show which of the four fundamental simplex transformations (figure 7.1 on page 64) is performed as it convergences, the red graph shows the energy function evaluated at the highest-energy vertex in the simplex and the blue graph shows the energy at the lowest-energy vertex.

Generally, the blue graph decreases in steps, and an early drastic reduction of the red curve indicates that high-energy vertices quickly are discarded in favour of better vertices closer to the “correct” vertex. A corresponding figure for the pose error shows for this frame the same general behaviour, although high frequency fluctuations are present.

The fluctuation of the red graph at early iterations may be due to reflections and contractions of the maximised simplex. Because of the simplex size, a great chance of a high-energy vertex being reflected to another high-energy point exists. The energy of this new vertex is worse than the energy evaluated at the second-highest vertex, thereby resulting in a contraction. This contraction may simply not be aggressive enough to move the simplex sufficiently close to the “correct” solution. This leaves it at a high-energy position until the next reflection, where this vertex may be reflected to a new, hopefully more feasible position.

The probability of which transformations are performed for this frame is shown in table 7.1, although not uniformly distributed over time.

This distribution indicates that the simplex is “walking” Ω as well as shrinking from spanning entire Ω to a subset of it, just as expected. The Euclidean distance in $\Re^{23}$ from the lowest to the highest energy vertex is reduced, thereby minimising the error difference until a specified lower threshold is reached (at 627 iterations for this frame). Also notice that most multiple contractions are not performed until the simplex is shrunk to a certain size.

Transformation	Occurrences	Probability
Reflection	450	71.77
Expansion	45	7.18
Contraction	126	20.10
Multiple contraction	6	0.01
Total	627	100.00

Table 7.1: Transformation probability. The distribution indicates that the simplex is “walking” the search space and contracting over time.

Because performance is more important than speed in this experimental phase, the optimisation process is not stopped until DS is stuck in a local minimum. When the difference of the error function from the highest energy vertex to the lowest energy vertex reaches a threshold slightly higher than the internal floating point precision limit. This indicates convergence to a minimum, so the algorithm is stopped. The error threshold is specified to be 100ϵ so as not to be confused by roundoff errors; ϵ being the smallest representable positive number.

As one can see, the DS algorithm *could* be stopped earlier as the last iterations do not contribute anything worth mentioning to the convergence process due to the insignificant simplex size, but this is not considered any further as speed is of secondary importance.

7.3.3 Transformations

The number of transformations performed is sampled from each of the 12 sequences, each dataset yielding a normal distributed curve. Figure 7.16 shows the mean μ and standard deviation, σ, for the number of transformations for each sequence.

Convergence usually happens within the first approximately 500 transformations for this specific problem in $\Re^{23}$, although sequences with fixed wrist generally seem to require slightly less iterations and have smaller standard deviation.

In order to deal with theoretically potentially non-converging situations, as found by Kaczmarczyk [38], an upper limit of $1,000$ evaluations is specified. This limit is based on figure 7.16 on the next page and the analysis in last subsection, and seems reasonable because in the few cases this limit is reached, the simplex size is insignificant, thus more iterations are not likely to improve the match significantly.

7.3.4 Restarts

Because speed is of secondary importance, the number of iterations does not need to be decreased at this stage; on the contrary, it will be *increased* in this subsection; in order to analyse how restarts will influence the matching process.

Intuitively, it pays off to restart the DS algorithm once a local minimum is found, using the so far best vertex as $\mathbf{s_{init}}$ in formula 7.1 on page 64 when computing the initial simplex size

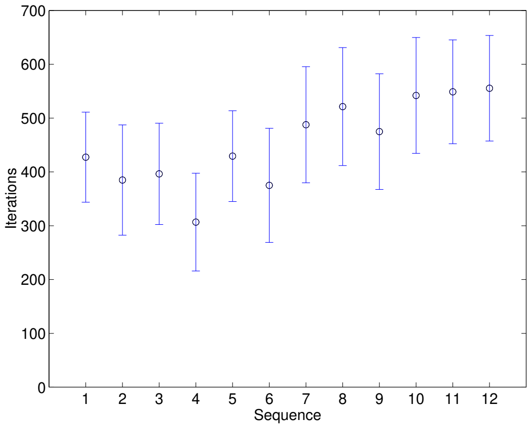
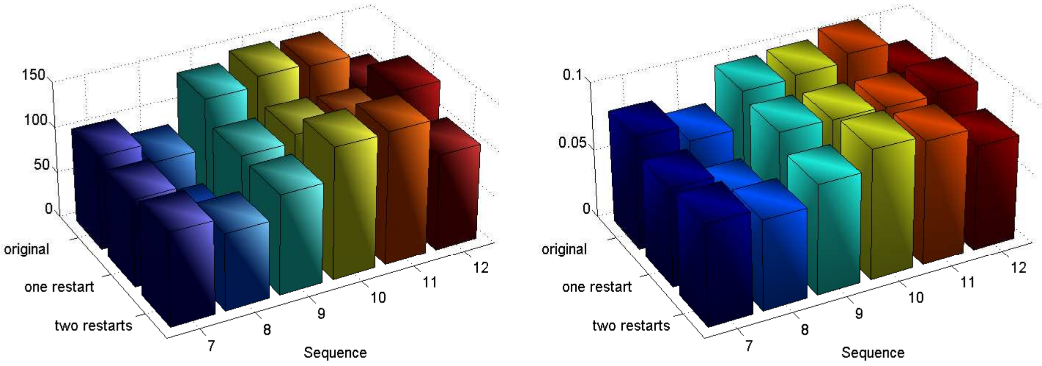


Figure 7.16: Average number of transformations (μ) and standard deviation (σ) required to reach convergence for each sequence.

and position for the next restart[2]. Press et al. [27] advises a single restart but theoretically and intuitively, more restarts result in a better fit, as every restart most likely improves the match for the specific frame - or at least, does not worsen it.

Each bar in figure 7.17 shows the mean error for each sequence (7 to 12) as a function of the number of DS restarts: (a) shows pose error, (b) pixel error. As can be seen, a single restart seems to pay off, but limited improvement with respect to average error is obtained from multiple restarts - at times it even worsens the situation, especially with respect to pose error.



(a) Pose error

(b) Pixel error

Figure 7.17: **Average pose error (a) and pixel error (b) for sequences 7 to 12 as a function of the number of DS restarts.**

A Wilcoxon Signed Rank Test with a confidence interval of 95% is carried out to test if one or two restarts statistically improve the mean errors, compared to no restart at all - see table 7.2 on the following page.

The Rank Test reveals that the difference in pose as well as pixel error is not significantly different from zero for a single restart. For two restarts, there is no statistical difference in

---

[2] The very first DS call each frame is based on the best vertex from previous frame

	Pose error mean			Pixel error mean		
Sequence	Original	One restart	Difference	Original	One restarts	Difference
7	103	95	+8	0.0816	0.0712	+0.0104
8	81	54	+27	0.0697	0.0541	+0.0156
9	136	111	+25	0.0962	0.0911	+0.0051
10	146	119	+27	0.0984	0.0902	+0.0082
11	143	116	+27	0.1045	0.0892	+0.0153
12	104	137	−33	0.0804	0.0907	−0.0103
	Pose error mean			Pixel error mean		
Sequence	Original	Two restarts	Difference	Original	Two restarts	Difference
7	103	102	+1	0.0816	0.0738	+0.0078
8	81	84	−3	0.0697	0.0657	+0.0040
9	136	109	+27	0.0962	0.0799	+0.0163
10	146	143	+3	0.0984	0.0943	+0.0041
11	143	145	−2	0.1045	0.0897	+0.0147
12	104	105	−1	0.0804	0.0761	+0.0043

Table 7.2: **Comparison of errors for the original DS and one as well as two restarts. A (+) in the difference section indicates that an improvement is obtained using a restart, while a (−) indicates that just a single DS call comes out best. Notice that all sequences but number 12 gain from being restarted.**

the pose error, but there *is* a significant improvement in mean pixel error, though.

Leaving the statistics out of account and just looking at table 7.2, it is believed that using a single restart indeed improves the situation with respect to pose error - which is the ultimate goal. Two or more restarts may cause DS to dig too deep into a wrong valley. This certainly improves the pixel error but at the expense of poorer pose error.

Therefore, in spite of the statistical test, it is chosen to go with *one* restart of the algorithm. In figure 7.4 on page 78 a comparison between not restarting the DS algorithm and one restart is shown.

Using one restart increases the average number of evaluations to 1246, not including simplex initialisation. In general the pixel estimation error improves when making one restart. The gain is especially noticed in sequences 7, 8 and 11. Unfortunately one restart may also lead to higher pixel errors in some frames as seen in sequence 8 around the 200 frame mark. Both in sequence 10 and 12 there is an increase in pixel error around the 150 frame mark, and comparing to table 7.2 the only reason that sequence 10 actually shows improvements in its mean error is due to the good recovery at the 200 frame mark and ability to maintain a relative low pixel error in the last fifty frames.

Figures showing pixel- and pose error for the restarted learning sequences 7-12 (the blue plots in figure 7.4 on page 78) can be found in appendix E on page 138.

7.4 Summary

In this chapter the theory of the Downhill Simplex algorithm was presented. Due to the structure of the algorithm, joint limits constraints had to be implemented in another way than was the case with the spider. Different methods were presented and discussed, and the choice to employ a cost function penalty as the solution was chosen.

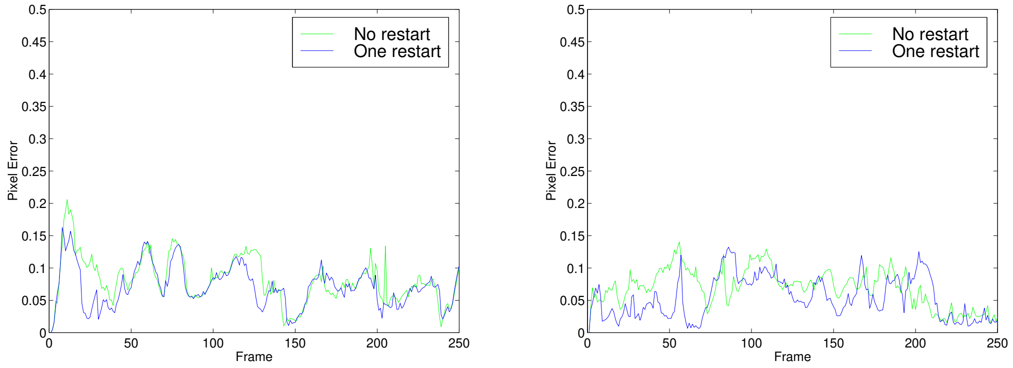
The tests using the learning sequences showed a similar pattern to the spider. Sequences

1-6 were estimated with relatively low errors, but the introduction of wrist transformation also caused problems for DS, once again indicating that the main problem lies in the wrist movement.

Also the DS algorithm was modified to employ restarts for each frame. Several articles advised this, and intuitively this seemed like a good idea. Statistically no significant difference could be proved, the exception being the pixel error using two restarts.

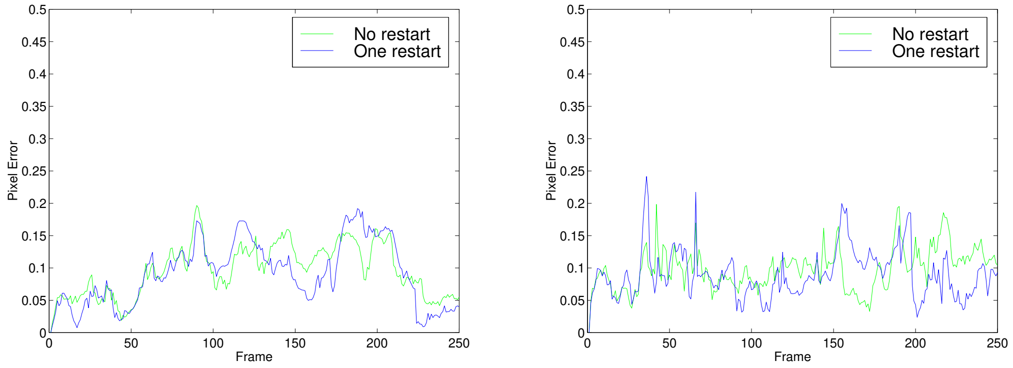
In this case it was decided to disregard the statistical result and go with one restart on the basis of the average improvement noticed in pose error, although this could not be statistically proved.

Before the comparative tests, one more algorithm is introduced in the next chapter; the combination of Downhill Simplex and Simulated Annealing.



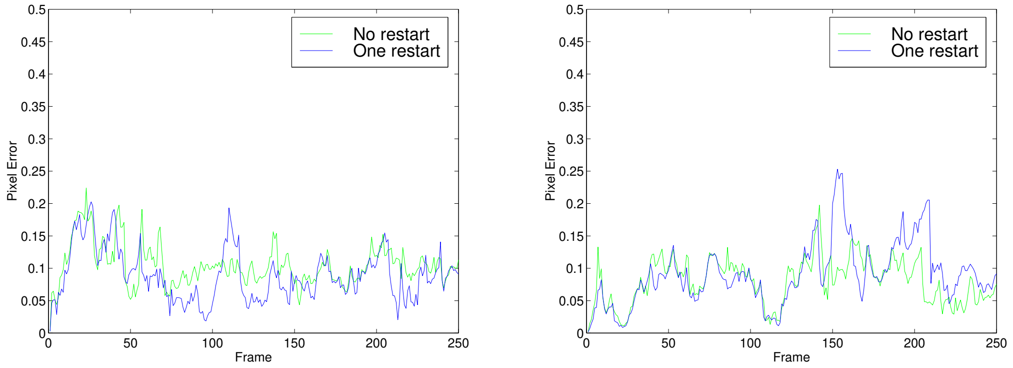
(a) Sequence 7

(b) Sequence 8



(c) Sequence 9

(d) Sequence 10



(e) Sequence 11

(f) Sequence 12

Figure 7.18: **Final pixel error results for learning sequences 7-12. The original DS is plotted using green, the restart is plotted using blue.**

Chapter 8

Simulated Annealing

> "Software is about getting things working.
> It is not about pretending to be a math guy."
> - *Robert Rodgers*

In recent years more attention has been paid to stochastic optimisation methods as a means of avoiding being trapped in local minimum at the expense of more evaluations. Although the theory of *Simulated Annealing* (SA) was first published in 1953 by Metropolis et al. [39], not much attention was paid to it until the mid 80's when it was computationally feasible to experiment with relatively complex optimisation problems. SA has attracted significant attention as a mean of solving large-scale optimisation problems and has among other successfully solved the famous *Travelling Salesman Problem* (TSP) with an arbitrary number of cities to be visited by a salesman travelling the shortest path. Furthermore the method has been used in the design process of Integrated Circuits (IC) with arrangement of a large amount of circuit element to be placed on a tiny silicon surface with minimal electrical interference ([27, chap. 10.9]). SA has also been used in problems such as (instruction) scheduling, automated program flaw finding, image compression, neural network training, priority assignment and production line optimisation, according to Internet search hits.

Because SA algorithms are based on probability they are not guaranteed to find the global minimum within a fixed number of iterations; yet they are not likely to get stuck in the first local minimum encountered as the greedy methods are.

8.1 Analogy

The name, *Simulated Annealing*, indicates that the method has an analogy with thermodynamics - especially attempts to simulate the way metals cool and anneal, or the way liquids freeze and crystallise. At high temperatures the individual molecules move freely among another; as temperature slowly decreases thermal mobility is gradually lost. The liquid (or metal) becomes progressively more organised into a solid and the molecules are able to form a pure, highly ordered crystal that can be billions of times the size of the individual atoms ([27]). This state is known as the minimal energy state of the system.

The SA strategy can solve continuous space problems ("simple" function optimisations) as well as state configuration space problems (TSP and IC design). When solving continuous space problems the algorithm "dances" the surface of Ω, with the amount of movement and virility controlled by a temperature T. As T is decreased using some schedule from the starting temperature T_{init}, so is the excitement of the dance and the process is slowly "crystallised" into a minimum energy state - hopefully the lowest-energy point of the energy function. If searching a state configuration space, the SA process likewise dances the state space, again slowly crystallising to an optimal state configuration.

8.2 Theory

The Boltzmann probability distribution,

$$P(X_t = E) \approx \frac{1}{Z} e^{\frac{-E}{kT}} \tag{8.1}$$

determines the probability of finding a system, such as a molecule or a group of molecules, with a certain energy E at a given temperature T. Z is a normalisation constant, called the partition function, and can be ignored ([29]). k is Boltzmann's constant[1], which relates temperature to energy. However, because k is merely a constant scaling of T, it is ignored in the following: Whenever referring to the temperature T, it is an indirect reference to kT.

Instead of always taking a downhill step as the only option, as known from the greedy algorithms, a simulated thermodynamic system goes downhill if this is an option, but *sometimes* take a step uphill, allowing it to escape from a local minimum. This general strategy has come to be known as the *Metropolis Algorithm*, named so because Metropolis et al. [39] were the first to apply this strategy to numerical optimisation problems back in the early 50's.

The probability of going uphill (p) can be computed using equation 8.2, where E_{old} is the energy of the old point and E_{new} is energy of a new point; $\Delta E = E_{old} - E_{new}$. Notice that if $\Delta E < 0$ the probability of taking the step downhill from E_{old} to E_{new} is arbitrarily set to $p = 1$.

$$p = e^{\frac{-(E_{old} - E_{new})}{T}} = e^{\frac{-\Delta E}{T}} \tag{8.2}$$

Thus, the probability of accepting the new point is p.

Because of the fact that a SA algorithm may encounter a good fit and afterwards takes a step away from it in favour of a higher-energy state, it may never return to the best vertex found during the annealing process. Therefore the best fit encountered should be kept in memory and preferably be subject to a greedy search. The reason for the latter is that the algorithm is allowed to escape a valley before finding the local minimum of that valley, thus a greedy search makes sure the local minimum in fact is found.

The following elements must be provided to apply the Metropolis Algorithm to numerical problems:

1. The objective energy function to be minimised.
2. The system state; either a configuration (state configuration problem) or a vertex/vertices in Ω (continuous function optimisation).
3. A control parameter T and a cooling schedule.
4. A generator of random chances in the configuration (state configuration problem) or a procedure for taking a step from point $\mathbf{s}$ to $\mathbf{s} + \Delta\mathbf{s}$ (continuous function optimisation).

These points add further issues to be addressed on top of the ones already discussed, especially with respect to §3 and §4 and are subject to further investigation in section 8.4. Press et al. [27, chap. 10.9] and [40] suggest several methods; however, the common denominator is that it should be "determined by experiment".

---

[1] $k \approx 1.381 \cdot 10^{-23} JK^{-1}$

8.3 Downhill Simplex and Simulated Annealing

As already mentioned, SA itself is not a specific algorithm; rather a philosophical idea that can be implemented in various ways and can be applied to various already existing methods.

Press et al. [27, chap. 10.9] proposes a combination of SA based on DS which seems like a promising hybrid, henceforward known as DS/SA. Their implementation is adopted in this project; with modifications to suit the specific needs.

The Metropolis Algorithm is basically implemented by modifying the basic DS method by *adding* a logarithmically distributed random variable based on T to the energy value E associated with every vertex of the simplex, and *subtracting* a similar random value from the energy value of any new vertex tried as a replacement point. Using this method, a downhill step is always accepted, but uphill steps occur occasionally. For $T \to 0$ the influence of the stochastic element will be infinitesimal and the algorithm will reduce to the basic greedy DS algorithm for $T = 0$, thus converging to a local minimum.

In practice, random values r are computed using formula 8.3 and simply adding or subtracting r to/from E.

$$r = T \log(rand(0..1)) \tag{8.3}$$

Using the log function insures that most r values are small, but occasionally big.

As this DS/SA hybrid is based on the DS algorithm, the general steps are the same (the flow chart in figure 7.2 on page 65). The initial simplex size is maximised as in the basic DS method, and computed using the same formula, equation 7.1 on page 64. Also the constraints are applied using the same principle as in the DS method, described in details in section 7.2.3 on page 69.

The differences lie in the choice of temperatures and cooling schedule, and the values of these parameters have to be chosen by experiment.

8.4 Parameter Estimation

The parameters influencing the outcome of the matching process are extremely difficult to optimise manually. Examining one or two parameters at a time in the “optimal parameter space” causes new problems: The techniques discussed can in theory be applied to this new optimisation problem of “optimising an optimisation algorithm”, like a dog chasing its own tail.

Furthermore, the heuristically estimated “optimal solution” will be found with respect to the problem of matching two rendered OpenGL models in a specific predefined scenario, thus most likely causing problems and requiring time-consuming re-estimation when applied to real-world input data.

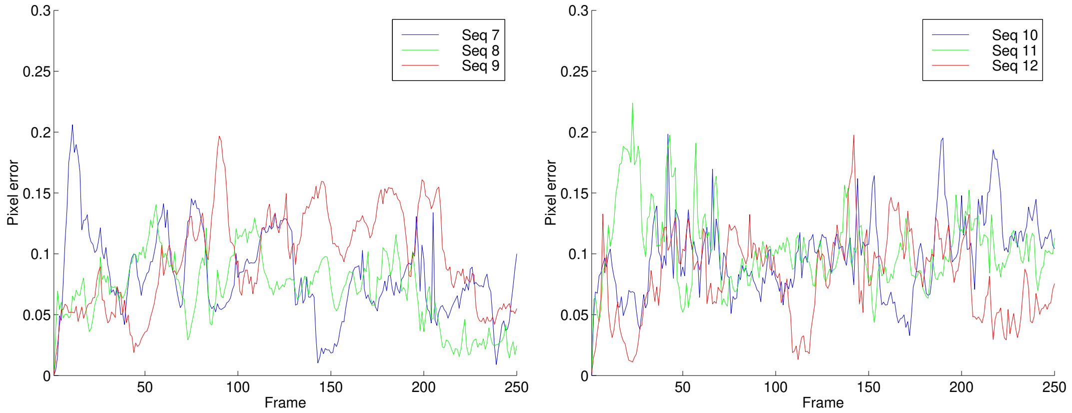
A fundamental source of uncertainty is the fact that the stochastic influence requires each test to be repeated several times in order to derive useful statistics. Because of logistical reasons this cannot be accomplished, thus the output data presented cannot be fully relied upon with respect to statistics and tendencies.

Nevertheless, a brief survey of the apparent influence of the individual parameters is desired.

8.4.1 Initial Tests

Because the implementation is derived from [27], with certain modifications, an initial test is carried out in order to get at least an idea of its feasibility. In this test $1,000$ evaluations are allowed and the temperature schedule is linear, starting at $T_{init} = 0.04$ and ending at $T_{end} = 0.00$, distributed over four temperature intervals.

As the fixed wrist situations can be dealt with using the Spider algorithm or pure DS, this chapter focuses on situations without fixed wrist as they are of most interest and constitute serious problems to the already presented methods.



(a) Sequence 7-9 pixel error

(b) Sequence 10-12 pixel error

Figure 8.1: **Pixel error for learning sequences 7-12 using DS/SA. The results are based on 1,000 evaluations; starting temperature T_{init} 0.04, end temperature T_{end} 0; this temperature decrease linearly distributed on four temperature steps.**

These initial tests indicate that this routine *may* be better than the already presented algorithms, as it seems to perform roughly as well on sequences 7-12, although the parameters were chosen heuristically. The pixel- and corresponding pose errors can be seen in the figures in appendix E on page 139.

In the following sections the algorithm parameters will be discussed with the difficult sequences 7-12 in mind; but in order to base this discussion on facts and not suppositions, the convergence process will be described in the next subsection.

8.4.2 Convergence

The main difference between the original DS and this DS/SA hybrid is the random values used to fluctuate the vertex energies. This indirectly simulates an unstable system slowly cooling down to a (hopefully) global minimum.

Comparing figure 7.15 on page 73 and figure 8.2 on the next page, this principle is visually demonstrated. The figures show the convergence process for the same frame (early in learning sequence 12); the green dots show which transformation is performed, the red graph shows the energy function evaluated at the worst vertex of the simplex and the blue graph the energy evaluated at the best vertex. The number of transformations allowed for the annealing process is specified to be 627, the same number of transformations as required by DS to reach convergence for that frame; thereby allowing for directly comparison with figure 7.15 on page 73.

The vertical dotted lines in figure 8.2 on the next page indicate where the temperature is decreased, linearly from $T_{init} = 0.06$ in the left quarter over 0.04 and 0.02 to $T_{end} = 0.00$ in the right quarter. This schedule is arbitrarily chosen.

The same general behaviour in the first iterations can be observed for figure 7.15 on page 73 and figure 8.2 as the high energy vertices quickly are discarded in favour of lower energy vertices.

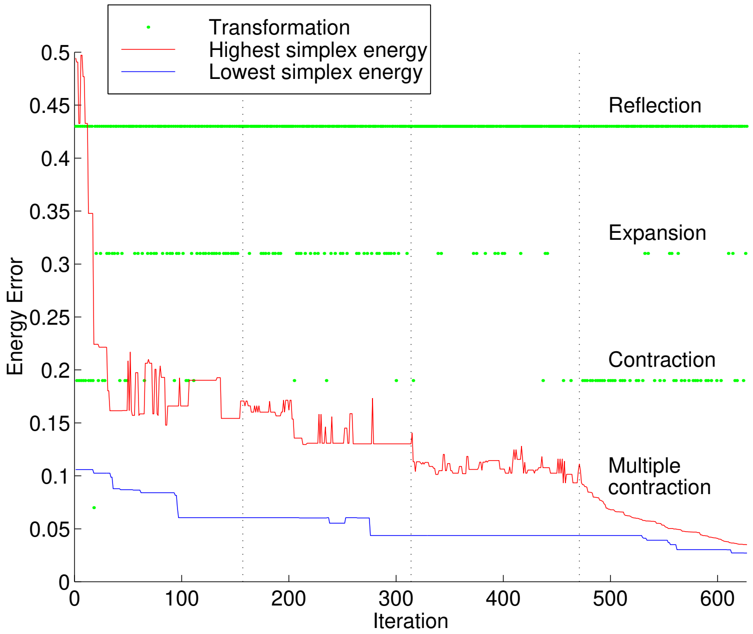


Figure 8.2: **Convergence process for DS/SA. Compare this figure to figure 7.15 on page 73 and notice the fluctuations; theoretically allowing the algorithm to escape from local minima. Also notice the flat plateaus on each interval, indicating some sort of equilibrium.**

A notable plateau can be seen for each temperature step - the annealing process indirectly reaches equilibrium, except the last step ($T = 0.00$) where the final greedy search is performed. 627 transformations clearly are not enough distributed on four temperature intervals for this problem - the simplex size is not shrunk enough, thereby no multiple contractions are performed and the routine returns a doubtful minima.

Similar comparing table 7.1 on page 74 and table 8.1 notice that the ratio of expansions and contractions has been reversed. Instead of performing relatively many contractions, instead the DS/SA performs more expansions, once again indicating that the DS/SA is not allowed to settle.

Transformation	Occurrences	Probability
Reflection	464	74.00
Expansion	95	15.15
Contraction	67	10.69
Multiple contraction	1	0.00
	627	100.00

Table 8.1: **Transformation probability; the distribution indicates that the simplex is not given enough time to contract.**

One can see that for this specific temperature schedule, DS/SA requires more transformations than DS to reach convergence. Many transformations are spent at high temperatures, not contributing significantly to the convergence process. Lowering T_{init} indeed results in better match for *this* frame, but setting it too low in cases with high initial energy error increases the possibility of the algorithm getting stuck in the bad local minimum in which it started.

Therefore temperature is discussed in next subsection, followed by a discussion on transformations.

8.4.3 Temperature

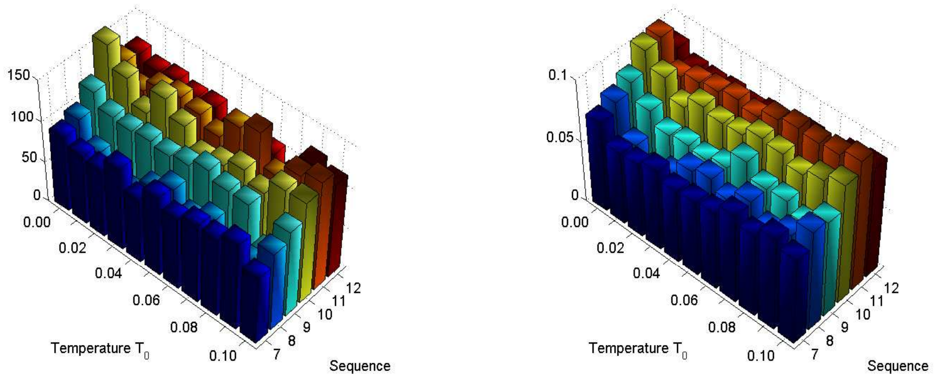
The values specified for T_{init} and T_{end} are crucial for the efficiency of any SA algorithm. If T is too high, SA randomly dances the search space being unable to settle in any valley. This more or less reduces it to a random search (section 5.4.2, page 35). On the other hand, if T is too low, it may not be possible for SA to escape any bad local minima, thereby undermining the fundamental idea of SA and more or less reducing it to a traditional greedy search (section 5.5.2, page 5.5.2).

In other words, should the algorithm stay somewhere in the neighbourhood or seek out new territories to explore?

To test the influence of the temperatures chosen, the number of evaluations is raised to $1,000$ (exclusive simplex initialisation), evenly distributed on four different temperature intervals. A relatively aggressive cooling schedule is suggested by Mendivii et al. [29] for use in practice, but Press et al. [27] and other sources advise a slower and much more conservative scheduling.

This depends on the specific implementation and application, so the cooling schedule is for this initial test specified to be $T' = T - \delta$ (linear). T' is updated after 250 iterations at each temperature interval and δ is computed to satisfy a specific initialisation temperature of T_{init} and an end temperature T_{end} of 0.00. The reason for choosing $T_{end} = 0.00$ is the fact that this results in a final greedy search each frame, thereby refining the match.

For more in-depth information on SA temperature schedules, see Cohn and Fielding [41].



(a) Pose error

(b) Pixel error

Figure 8.3: **Mean pose- and pixel error for different T_{init} and a linear cooling schedule from T_{init} to T_{end} (set to 0).**

In figure 8.3 each bar shows the mean error for sequences 7 to 12, respectively, computed using different values as T_{init}. Each sequence is run only once, resulting in ambiguous test outputs. Comparing this figure to figure 7.17 on page 75 reveals that there seems to be no significant improvement from the original DS to the DS/SA hybrid.

The reasons for these obscure and somewhat unexpected results may be one of the following:

1. Each sequence is run only once; statistical analysis requires more data.
2. Ineffective schedule chosen; linear scheduling may not be effective for this problem.
3. Wrong values specified for T_{init} and/or T_{end}; too high temperature results in more or less random output.
4. Not enough iterations performed; the annealing process requires time to settle down ([29]).
5. ... or Simulated Annealing may simply be counterproductive in this specific case.

The items in the list ought to be investigated one at a time, but instead of spending valuable time on §1 or adjusting the parameters with respect to §2 and §3, we choose to go for §4 as figure 8.2 shows that DS/SA requires many iterations to anneal in order to reach an acceptable level of convergence.

8.4.4 Transformations

Whereas the basic DS method is stopped when a certain tolerance level is reached for ΔE, a fixed number of transformations must be specified in the DS/SA case as the uphill moves can cause ΔE never (or at least not within an acceptable time limit) to reach zero (if $T > 0$).

The more transformations allowed, the better the algorithm is expected to perform because of its stochastic nature. If there is just the slightest possibility of something happening (finding the correct global minimum), it *will* happen if given enough time; clearly expressed by Mendivii et al. [29]:

"[...] by careful control of the temperature and by allowing the system to come to equilibrium at each temperature, the process finds the global minima of the energy function".

In practice, though, it is not as simple as described here as it may take an infinite (or unacceptable) amount of time to find the global minimum, and the "*...careful control of the temperature...*" is more difficult than it sounds.

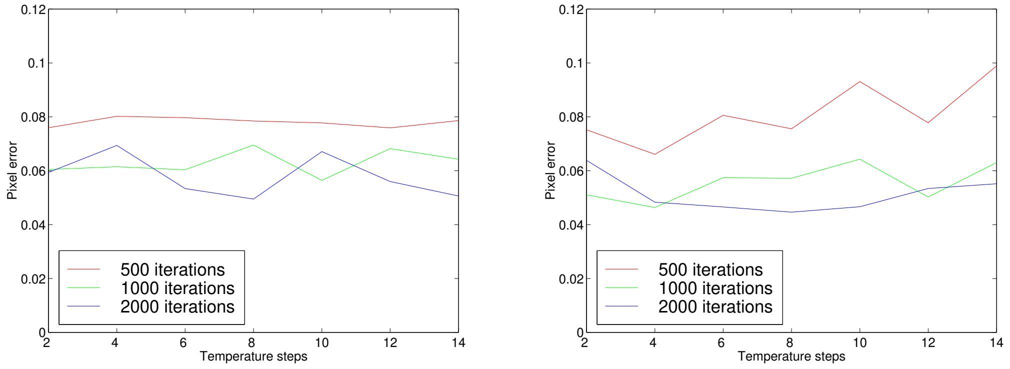
Not only does the number of evaluations influence the matching process; so does the specified number of temperature intervals, as the above quotation indicates. Simple tests are carried out to determine the relationship between the allowed number of evaluations, and the specified number of temperature intervals, see figure 8.4 on the next page.

The figure is statistically dubious due to the test sequences; yet some general tendencies are vaguely intimated. The colours represent the number of evaluations allowed[2], the x-axis how many temperature intervals are used, and the y-axis the resulting pixel error. In general, more transformations results in a better match - just as intuitively expected.

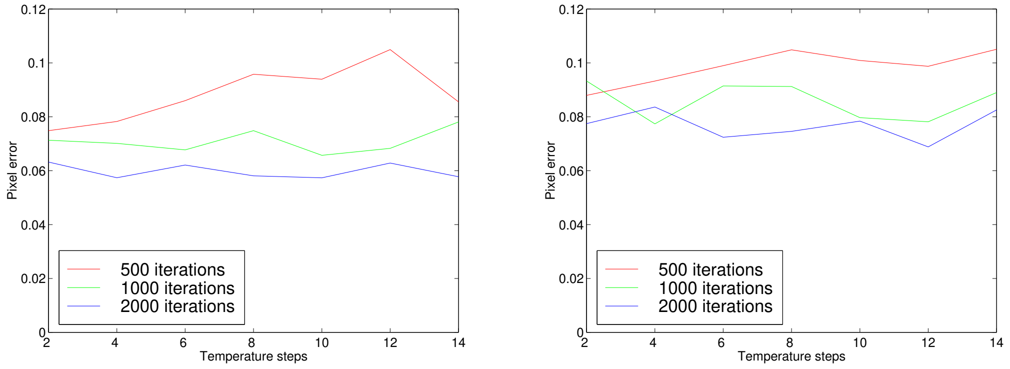
A Wilcoxon Signed Rank Test (confidence interval 95%) indeed indicates significant improvement from 500 to 1,000 transformations, as well as from 1,000 to 2,000 iterations; although the absolute improvement in mean is statistically more significant from 500 to 1,000 transformations than from 1,000 to 2,000 transformations.

---

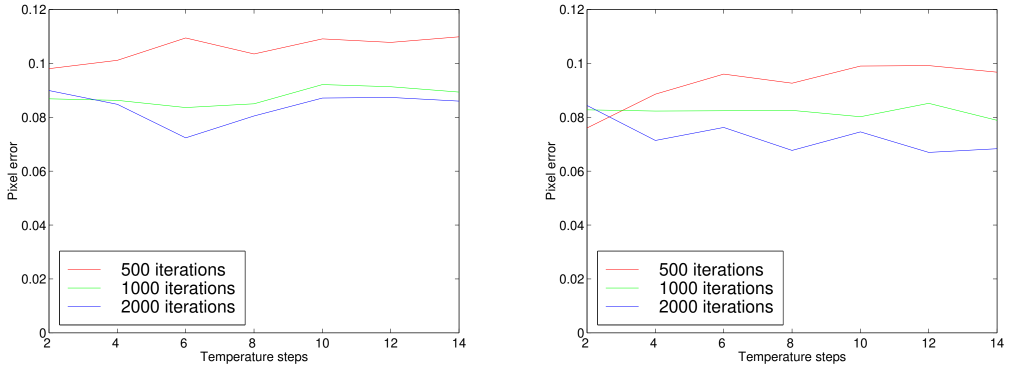
[2] Due to the specific implementation, the actual number of evaluations is slightly higher



(a) Sequence 7 (b) Sequence 8



(c) Sequence 9 (d) Sequence 10



(e) Sequence 11 (f) Sequence 12

Figure 8.4: **Coherence between number of allowed transformations and number of temperature steps, for sequences 7-12.**

Oddly enough, the specified number of temperature intervals does not seem to cause much of a difference. The reason may be that a completely wrong schedule is applied (linear), thereby causing results without the strong characteristics hoped for - an optimal combination of temperature steps and number of transformations, for the starting temperature $T_{init} = 0.02$ found in section 8.4.3.

The best of the combinations tested seems to be 8 intervals at 2,000 transformations, which under the circumstances is chosen as the "optimal" parameters for the DS/SA algorithm, in spite of its apparent inefficiency, compared to other DS/SA parameter combinations at 2,000 transformations.

	Pose error mean			Pixel error mean		
Sequence	Original	Optimised	Difference	Original	Optimised	Difference
7	89	68	+21	0.0615	0.0495	+0.0391
8	46	53	−7	0.0464	0.0446	+0.0290
9	105	46	+59	0.0701	0.0415	+0.0364
10	114	94	+20	0.0774	0.0745	+0.0663
11	130	120	+10	0.0863	0.0805	+0.0058
12	129	110	+19	0.0823	0.0677	−0.0146

Table 8.2: **Comparison of errors for the original DS/SA and the optimised. A (+) in the difference section indicates that an improvement is obtained, while a (−) indicates the opposite.**

Applying a Wilcoxon Signed Rank Test to the differences in table 8.2 with a confidence interval of 95% reveals statistical significance with respect to pose error mean as well as pixel error mean; thus it must be concluded that improvement *has* been obtained.

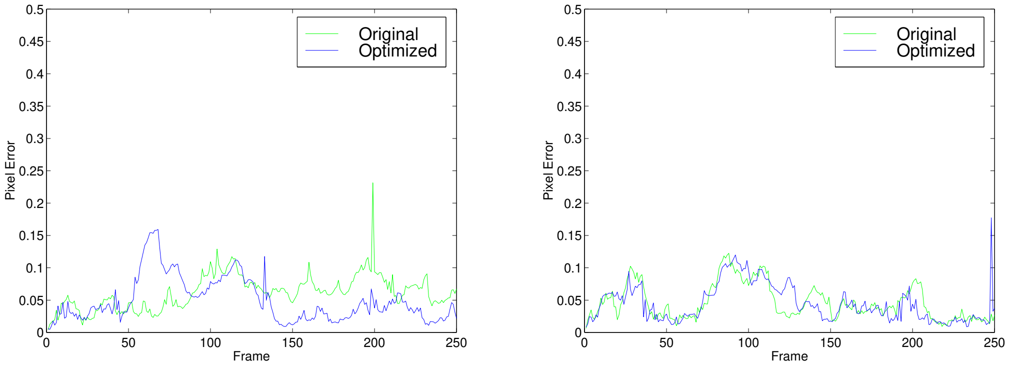
Figure 8.5 on the following page shows the pixel error for the original as well as the optimised DS/SA, and test results for sequences 7-12, showing pixel- and pose error, can be found in appendix E on page 140.

Testing the optimised DS/SA on sequences 1-6 reveals that the pure DS actually performs better. The reasons may be that the parameters of DS/SA are adjusted to sequences with non-fixed wrist, and that there is nothing to optimise that the DS has not already done very well. The conclusion is that the random fluctuations in the easy cases deteriorate more than they help.

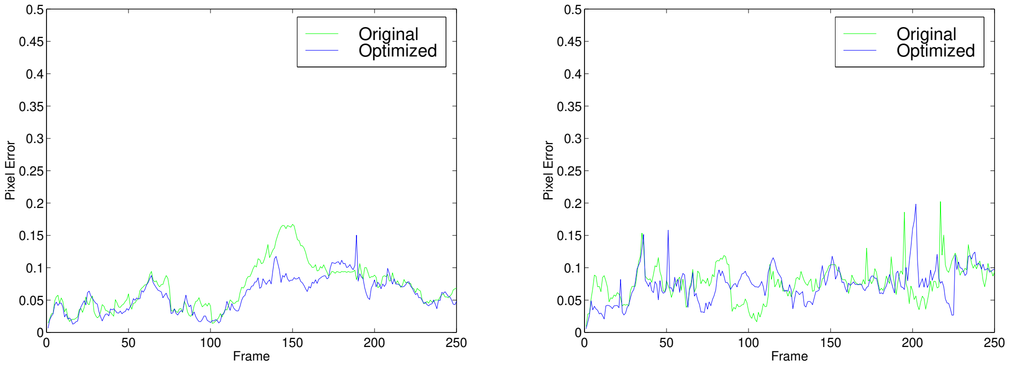
8.5 Summary

In this chapter the theory of Simulated Annealing was presented, and a combination of Downhill Simplex and Simulated Annealing was shortly discussed. Because SA requires many parameters to be estimated correctly in order to work as expected, the temperature was initially discussed, followed by a discussion on the allowed number of transformations.

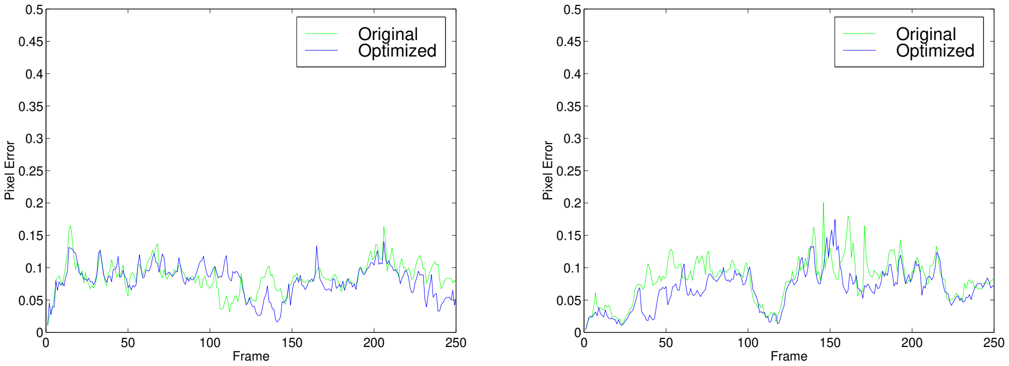
A wide dynamic range was tested with respect to starting temperature and temperature intervals, without showing the unambiguous tendencies hoped for. The only parameter which indeed yielded a statistical significant difference was the number of iterations; more iterations resulted in better fit. One reason may be a completely wrong schedule, but unfortunately it is necessary to go with the parameters found.



(a) Sequence 7 (b) Sequence 8



(c) Sequence 9 (d) Sequence 10



(e) Sequence 11 (f) Sequence 12

Figure 8.5: **Final results for learning sequences 7-12. The original DS/SA is plotted using green, the optimised is plotted using blue.**

Chapter 9

Comparison of Algorithms

> "Don't worry! As long as you hit that wire with the connecting hook at precisely 88 miles per hour the instant the lightning strikes the tower .. everything will be fine."
> *- Doctor Emmet Brown, "Back to the Future"*

The Spider, the Downhill Simplex (DS) and the Downhill Simplex/Simulated Annealing (DS/SA) algorithms are compared in this chapter, using the test sequences described in section 5.6.2; these sequences have never been presented to the algorithms before. The speed of the individual algorithms is discussed as well, and finally a summary concludes this chapter. As both the Spider and DS algorithm performed very well on learning sequences with fixed wrist all of these test sequences employ moving wrist.

9.1 Performance

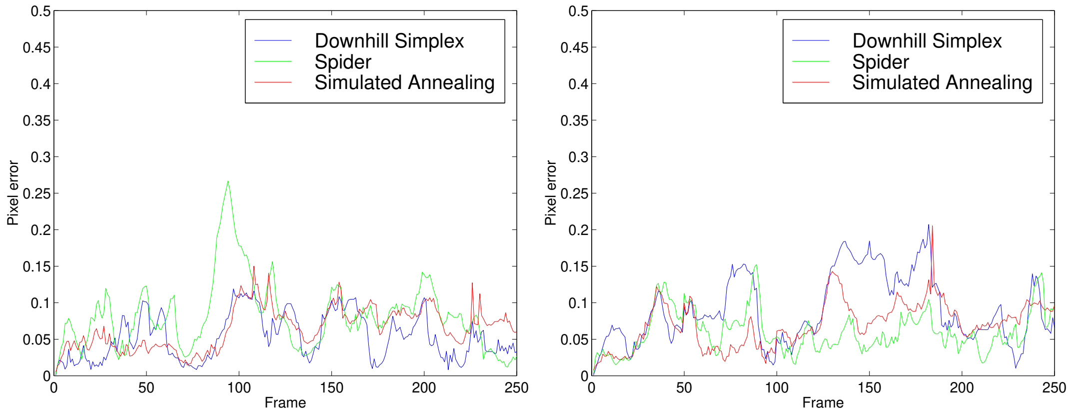
In figure 9.1 the pixel error of the three algorithms is plotted for each of the test sequences. In appendix E on page 141 the equivalent pose errors are shown.

As can be seen it is once again a motley set of results. In sequence 1 it is obvious that the Spider makes the worst estimation, while DS and SA compete for first place with a vague tendency of DS being best.

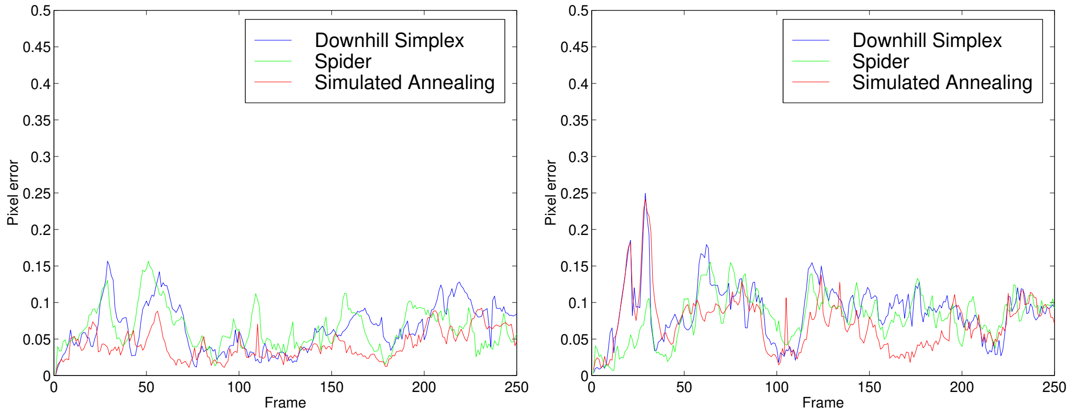
In sequence 2 the tables have turned and now the Spider algorithm is performing best, with the DS performing worst. Around frame 120, it seems that DS and DS/SA are caught in same local minimum, but DS/SA manages to escape a few frames later due to its stochastic nature.

The familiar resemblance for DS and DS/SA is also very striking in many other of the sequences, especially in sequence 4 where they yet again seem to hit the same local minimum, this time early in the sequence. It is clear that the Spider uses a different technique as it manages to escape these minima.

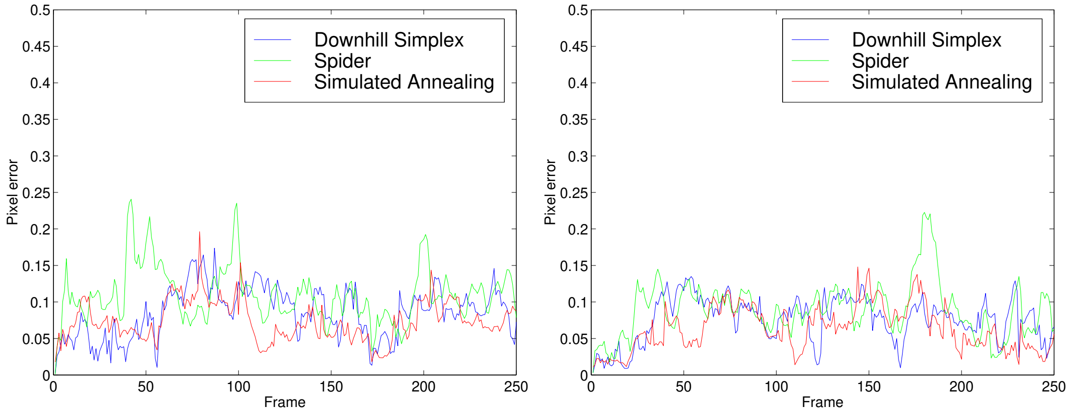
It is common for all sequences that the three algorithms vary in performance from sequence to sequence. However, the DS/SA algorithm in general performs better than the two other algorithms, as it never has the highest average pixel error, as can be seen in table 9.1. In this table the mean errors for the three algorithms are listed. Once again it is noticeable how pose and pixel error do not correspond, but the perhaps most noticable thing is the pose error achieved by the Spider.



(a) Sequence 1 (b) Sequence 2



(c) Sequence 3 (d) Sequence 4



(e) Sequence 5 (f) Sequence 6

Figure 9.1: **Comparison in pixel error between the three algorithms. Looking at these results, it is difficult to appoint one algorithm over an other; yet DS/SA has lowest average error.**

Test	Pose mean			Pixel mean		
Sequence	Spider	DS	DS/SA	Spider	DS	DS/SA
1	76	64	104	0.0825	0.0528	0.0650
2	48	115	113	0.0605	0.0980	0.0706
3	63	73	39	0.0648	0.0652	0.0420
4	104	126	99	0.0809	0.0891	0.0726
5	131	107	91	0.1098	0.0856	0.0739
6	100	114	76	0.0896	0.0760	0.0632

Table 9.1: **Comparison of errors between the three algorithms. Notice how well the Spider estimates the hand pose in the slow sequences (1-3), but has problems with the fast sequences (4-6); forming a contrast to the DS/SA results, which performs best on the fast sequences.**

In the slow sequences the Spider shows consistency in performing with relatively low pose errors compared to the two other algorithms. DS/SA has an amazingly low average pose error for sequence 3, but this may be regarded as an outlier as it seems to have problems with sequences 1 and 2.

Another appreciable observation is the fact that the Spider has a pixel error of 0.0825 in sequence 1, much more than the DS/SA pixel error of 0.0650; yet it returns a mean pose error of only 76, compared to the DS/SA pose error of 104 centimeters. This once again shows that low pose errors are not necessarily caused by low pixel errors.

When it comes to the fast sequences there is no doubt that DS/SA is the most efficient algorithm, yielding lowest errors both pixel- and pose wise. The fact that DS/SA performs best on fast and not slow sequences is surprising and indicates that there is room for improvement with respect to the temperature schedule - if DS/SA performs well on the fast sequences, it *should* intuitively perform better on slow sequences.

Also the fact that the Spider performs unacceptably on the fast sequences but not slow sequences, in contrast to DS/SA, is worth mentioning. As stated earlier in chapter 6 on page 39, the choice of Δ values is important for the performance of the Spider, and the Spider results indicate that the Δ values chosen seem to be aimed at slow sequences.

Taking the specific problem of estimating hand poses into account, none of the discussed algorithms are able to consistently make a satisfying continuous pose estimation over time, though. There may be many reasons for this; one certainly being the ambiguity of the silhouette, creating deep local minima from which the optimization algorithms are unable to escape.

In order to supplement the observations with statistical analysis, a Wilcoxon Signed Rank Test is carried out, see table 9.2.

The observation that DS/SA in general performs better than the other two algorithms is in part supported by the statistical test. The table shows the results of testing the three algorithms against each other. Notice that the H_0 hypothesis is rejected when comparing pixel errors between the Spider and DS/SA, but also between the DS and DS/SA. In other words, the DS/SA algorithm performs significantly better than both the Spider and DS when it comes to minimizing the pixel error. However; this is not the case with regards to the pose error, where no performance difference betweeen either of the algorithms can be proven.

Another interesting fact is that there is no significant difference between the well-known DS and the proposed Spider algorithm, when looking at pixel as well as pose. A closer look at W for the pixel error between Spider and DS reveals a -1, indicating that the two algorithms take turns in performing best.

Wilcoxon signed rank						
Test	Spider vs DS		Spider vs DS/SA		DS vs DS/SA	
Sequence	Pose	Pixel	Pose	Pixel	Pose	Pixel
1	$+2$	$+5$	-4	$+3$	-6	-2
2	-6	-6	-6	-2	$+1$	$+6$
3	-1	$+1$	$+3$	$+4$	$+4$	$+5$
4	-4	-2	$+1$	$+1$	$+3$	$+4$
5	$+5$	$+4$	$+5$	$+6$	$+2$	$+1$
6	-3	-3	$+2$	$+5$	$+5$	$+3$
W	-7	-1	$+1$	$+17$	$+9$	$+17$
Accept/Reject H_0	A	A	A	R	A	R

Table 9.2: **The Wilcoxon Signed Rank Test. The signed rank results for comparison between the algorithms. The Accept/Reject is based on a 95% confidence interval. When comparing pixel errors the tests show statistical difference between the performance of the Spider and SA, and also between the DS and SA; both in favour of the SA algorithm. The pose error, however, shows ambiguity.**

It is only to be expected that the Spider does not perform statistically better seen over all sequences as its performance on the fast sequences did not live up to the standards set in the slow sequences.

9.2 Speed Discussion

Emphasis so far has been on how well the algorithms are able to match the silhouettes and how good a pose estimation can be obtained. In this section a short comparison of the speed of the algorithms is given.

In terms of speed the obvious measure to use is the number of evaluations per frame, as the computation of the cost function accounts for almost all use of processor power. To translate the concept of an evaluation, tests show that approximately 50 evaluations per second can be made using a P3-550MHz featuring a Matrox Millenium graphics card. As the graphics card is used in the synthesis of evaluation silhouettes, the speed of the card is of great importance to the overall speed of the algorithms.

The Spider algorithm employing two 3D nodes is the most demanding of the three algorithms. As discussed shortly in section 6.5 on page 53, employing 3D nodes causes a significant computational overhead. Matters are worsened by using a 3D node in the junction between the wrist and finger parameters. Each of the five iterations of the Spider requires 1257 evaluations, bringing the total evaluations required for one frame up to 6285.

Because of the implementation of the DS and DS/SA, the exact number of evaluations per frame vary according to the circumstances. A reasonable measurement is the number of estimations for each of the six test sequences, which have been recorded. On average, DS using one restart consumes 1246 evaluations per frame, while the DS/SA consumes a mean of 2076 evaluations.

DS is clearly the fastest of the three algorithms, almost twice as fast as DS/SA and five times faster than the Spider algorithm, but none of the above algorithms can be expected to perform in real time with the computer power available at the present time.

All algorithms are relatively computationally demanding, and taking the computational burden into account, the best suggestion must be the DS/SA. Although the pure DS is twice as fast, DS/SA seems to perform more consistently when looking at both fast and slow sequences; this computational burden will be discussed in the next section.

9.3 Perspective

We do not believe it to be immediately possible to estimate hand poses for interaction purposes, based entirely on "flat" silhouette information and analysis-by-synthesis, mainly due to two points.

- First, the speed of the algorithms. Realistically a number in the area of 2,000 evaluations per frame is the minium required for an algorithm to solve for a decent pose. With the 50 evaluations per second which can be performed using an average computer of 2002, evaluating 2,000 silhouettes takes approximately 40 seconds (in our non-optimized implementation).

  If a frame rate of 10 fps is desired, this requires the hardware to run 400 times faster than today. If the processing power indeed doubles every 18 months (Moore's Law, Newsweek [42]), this will theoretically not be possible until 2015.

- Secondly, the silhouette of the hand contains a lot of ambiguity, even with the use of stereo vision. It becomes clear when reviewing some of the sequences used in this project that even we as humans are not always able to recognize the pose of a hand based solely on the silhouettes. With ambiguity comes the problem of local minima, which none of the presented algorithms can solve in an intelligent way.

If hand pose estimation based on analysis-by-synthesis is to be feasible, other means of comparing the input images of the hand and the model must be sought. An alternative descriptor must fulfil at least two vital conditions; the descriptor must be easily deducted from the input images as well as from the model, and secondly the descriptors must be easily comparable.

Even if such a descriptor is found it is unlikely that it in itself is enough to solve the pose problem. With the hand being such an articulate object, the descriptor will most likely be ambiguous, and the optimization algorithm will once again face the challenge of avoiding a great many local minima.

Taking these facts into consideration, an approach other than general analysis-by-synthesis is strongly encouraged; the algorithms presented simply require too much processing power if silhouettes are used, but may very well be the basis for other more intelligent algorithms in the future.

If analysis-by-synthesis is to be used, these items should be subject to further investigation:

- The algorithm should be able to combine several easily computed descriptors; quite possibly shadow- and color information could be taken into consideration, and not just flat silhouettes as in this project. By using only *one* descriptor it will be prone to ambiguity.
- Do not rely solely on history by using last estimated pose as the best initial guess for the current frame! Errors will be accumulated, and when the algorithm loses track, it will have *serious* problems getting back on the rails.
- Estimating wrist translation and rotation using an appearance-based approach *prior* to fine adjusting the fingers will give invaluable clues to the search algorithm.

9.4 Summary

In this chapter the three algorithms were compared using six new test sequences. The Wilcoxon test results showed that DS/SA statistically performs better on average than the two other algorithms, judged by the pixel errors.

This was not the case for pose error however, once again stressing that pose and pixel do not go hand in hand. The Spider algorithm performed impressively on the slow sequences but did not perform very well when facing the fast sequences.

Of the three algorithms evaluated only DS has been used by others ([13]). In light of this it is interesting to notice that the Spider as well as the DS/SA algorithm are able to compete with the DS. In the case of DS/SA this was partly expected as the implementation is a hybrid of the DS and SA. The introduction of both the Spider and DS/SA algorithms are important contributions to the field of analysis-by-synthesis as both are able to compete with an established method.

A section on the speed of the algorithms led to the conclusion that the DS/SA algorithm would be the best choice for an optimisation algorithm, even though it is computationally infeasible at present.

At the present stage the errors occuring in the estimation are simply too high to be used for the Watching Window console, and none of the algorithms are able to perform consistently over time with a non-fixed wrist - it may very well be that the applied cost function simply is not enough to always estimate the correct hand pose. Therefore, a few suggestions to other researchers, based on our own experience, were briefly presented.

This valuable realization, and the fact that the algorithms presented require too much processing power to be feasible within the next decade, are used in the next chapter where alternative cost functions based on descriptors other than surface overlapping are presented, hopefully serving as inspiration to other researchers in this field.

Part III

Closing

Alternative Descriptors

Conclusion

Process Analysis

Chapter 10

Alternative Descriptors

> "God hates me. This I know. I saw it in the shape of my Quaker Instant Oatmeal this morning. The tracks and valleys left by the spoon were unusual and probably demonic."
> - *Robert Rodgers*

From the previous chapter it is clear that none of the optimisation algorithms perform consistently. One reason for this may be the ambiguity of the silhouettes; different poses can create almost the same silhouettes.

The cost function used throughout this report is solely based on silhouettes. In this chapter some alternative cost functions are considered and tested. In the tests the basic DS method is used although both the Spider and DS/SA algorithms in general perform better. DS is chosen because it is the fastest algorithm of the three and the experience from the previous chapter tells that if something improves the DS it most likely will improve the other algorithms as well.

10.1 Shape Descriptors

In chapter 4.2 the basic biodynamic constraints of the hand were introduced, and in section 7.2 on page 66 these constraints were implemented to narrow the search space considerably.

In addition simple image analysis can intuitively be used to guide the search process as well, instead of relying entirely on the surface overlapping discussed in chapter 5.2 on page 32.

Great many descriptors can be applied to an image, but the following basic shape analysis techniques are considered to guide the simplex in this project, due to their inexpensive computation costs. As the surface overlapping energy function requires running through each pixel anyway, these pixel-based methods require little extra overhead:

1. Area
2. Circumference
3. Width+Height; size of bounding rectangle (hereby indirectly examining the aspect ratio ("elongation", [43]))
4. Geometric centre; city block distance

5. Centre of mass ($\overline{x}$ and $\overline{y}$); city block distance
6. Compactness; defined as $(4\pi \cdot area)/circumference^2$ ([43])
7. Eccentricity; defined as $((\mu_{02} - \mu_{20})^2 + 4\mu_{11})/area$ ([43])
8. Moments up to fourth order: $\mu_{11}, \mu_{20}, \mu_{02}, \mu_{30}, \mu_{03}, \mu_{22}, \mu_{40}, \mu_{04}$; thereby examining skewness and curtosis. μ_{pq} is defined as $\sum_x \sum_y (x - \overline{x})^p (y - \overline{y})^q f(x, y)$ (Gonzales and Woods [1]).

In addition to the above, more sophisticated descriptors such as signatures, convex areas, curvature and shape variances to name a few, could be applied. For more information on all the mentioned descriptors as well as others, see [1, 43].

10.2 Theory

The first technique considered is to combine a penalty value from some of the contour descriptors and add it to the energy function, in the same way dynamic constraints were applied for DS. Initial tests show that DS does not gain from using this combination, in fact the algorithm performs worse.

This may be traced to the fact that the descriptors not only differ in order of magnitude, but even after normalisation, this method only yields a new static complex landscape in $\Re^n$, this time most likely with even more local minima.

Instead another approach using the contour descriptors as separate supplementing cost functions is considered. As an example, consider the example in figure 10.1.

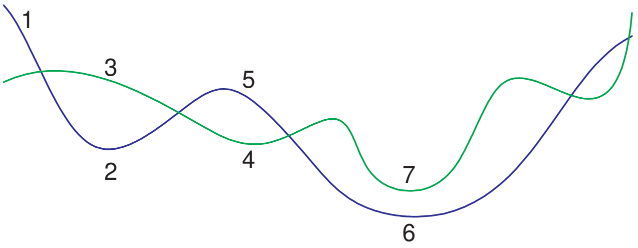


Figure 10.1: **The two lines represent two different cost functions to the same problem. Using these functions alternatively may improve the search algorithm.**

Two different cost functions (as an example, surface overlapping (blue) and circumference (green)), henceforward known as the original and alternative functions, yield the same global minimum but hopefully prevailing different local minima. They can therefore be said to be *semi-dependent*.

If a DS greedy search starts with a vertex at (1) and uses the original cost function, it will reach the local minimum at (2). Now, using this vertex as a starting point for a new search based on the alternative energy function (3) will cause the simplex to reach a new local minimum at (4). Using the original function once again with this point as initial vertex (5) will result in the simplex finding the global minimum at (6).

Thus, letting the original and alternative cost functions supplement each other when a local minimum is reached can in theory guide the search to the one global minima which is common to the different energy functions. This method has to the best of our knowledge not been applied to this kind of problem before.

The method is not flawless, however: If the hill of the original cost function (solid line) in figure 10.1 on the facing page close to (5) is displaced to the right, the vertex will simply be trapped in two different local minima alternately (minimum (2) and (4)). Also, the energy functions chosen may be too dependent, yielding the same local minima.

This may be solved by introducing a third, fourth or even fifth evaluation function; this however involves new problems as it is very difficult to find the optimal sequence of cost functions to apply to the problem - in this case, with nine different energy functions ($\mathbf{S}_1$ - $\mathbf{S}_8$ and surface overlapping), empirical try-and-compare methods must be applied.

It is not possible to analytically compute the number of shared minima, so in order to obtain at least some idea of the feasibility of this technique, a systematic method is described in the next section, hopefully yielding which alternative functions are preferred.

10.3 Function Sequence

The cost function throughout this report has been purely based on surface overlapping. To examine if an alternative cost functions based on one of the shape descriptors can be applied to assist the surface overlapping in finding the global minimum, a general sequence scheme is designed.

To test the effect of the descriptors each descriptor is tested on learning sequences 7-12. In each frame three DS calls (two restarts) are made, as usual using the so far best vertex as $\mathbf{s}_{init}$ for each restart[1]:

1. The initial call uses the surface overlapping (the original cost function), thereby tuning the previous pose estimation to the new input silhouette. In many cases, only this DS call is required to obtain a satisfactory model fit.

2. The second call uses an alternative energy function based on the descriptor being tested. If a good fit is already obtained from the first call, using the alternative energy function will hopefully only result in minor changes. If DS did not manage to obtain a "satisfactory" fit during the first call, this second call will hopefully act as a catalyst and force the search to abandon the local minimum in which it is stuck.

3. The third call once again use the original cost function, hereby fine adjusting the match. Because of the "roughness" of the alternative energy functions, this call is necessary in order to return a feasible pose.

Using this general scheme allows for systematic, individual testing of each of the alternative cost functions and how it performs as complementing energy function to the original surface overlapping function.

10.4 Results

Table 10.1 and table 10.2 show the results from the alternative cost functions applied to sequences 7-12, for pixel- and pose error, respectively.

As an example, the results from the "Area"-function in table 10.1 on the next page are obtained from an initial DS call with a cost function based on the original surface overlapping

---

[1] The very first DS call is based on the best found vertex from last frame

	Learning sequence						Wilcoxon
Alternative function	7	8	9	10	11	12	test
Surface	*0.0755*	*0.0646*	*0.0799*	*0.0943*	*0.0902*	*0.0714*	0
Area	0.0628	0.0522	0.0672	0.0716	0.0843	0.0562	+
Circumference	0.0674	0.0714	0.0636	0.0842	0.0824	0.0672	+
Width+height	0.0516	0.0506	0.0563	0.0869	0.0844	0.0791	−
Geometric centre	0.0604	0.0620	0.0662	0.0746	0.0781	0.0815	+
Centre of mass	0.0523	0.0450	0.0761	0.0819	0.0800	0.0701	+
Compactness	0.0655	0.0845	0.0828	0.0929	0.0878	0.0803	−
Eccentricity	0.0647	0.0720	0.0843	0.0716	0.0905	0.0807	−
μ_{11}	0.0513	0.0679	0.0536	0.0740	0.0756	0.0734	−
μ_{20}	0.0630	0.0634	0.0699	0.0660	0.0738	0.0769	+
μ_{02}	0.0535	0.0606	0.0614	0.0735	0.0690	0.0609	+
μ_{30}	0.0526	0.0702	0.0787	0.0789	0.0874	0.0669	−
μ_{03}	0.0680	0.0691	0.0677	0.0789	0.0790	0.0800	−
μ_{22}	0.0445	0.0687	0.0705	0.0718	0.0891	0.0714	−
μ_{40}	0.0806	0.0657	0.0544	0.0782	0.0789	0.0744	−
μ_{04}	0.0642	0.0574	0.0590	0.0690	0.0780	0.0769	+

Table 10.1: **Pixel error for learning sequences 7-12, utilising a range of different alternative energy functions. The *surface*-line serves as a standard of reference. The Wilcoxon Signed Rank Test uses a ’+’ to indicate improvement and ’-’ to indicate deterioration. Four of the alternative functions result in a statistical significant improvement. Table 10.2 is the corresponding pose error table.**

	Learning sequence						Wilcoxon
Alternative function	7	8	9	10	11	12	test
Surface	*109.8*	*87.8*	*109.3*	*143.5*	*145.6*	*89.7*	0
Area	96.8	54.5	95.8	90.9	119.7	64.8	+
Circumference	100.9	104.5	85.6	119.9	105.2	88.4	−
Width+height	68.0	64.5	66.8	150.8	125.6	133.3	−
Geometric centre	89.6	80.8	83.9	101.9	126.9	125.6	−
Centre of mass	80.2	68.5	130.3	113.3	124.8	105.6	−
Compactness	91.3	143.3	112.6	144.0	106.0	121.3	−
Eccentricity	107.8	119.7	140.4	110.5	152.1	136.0	−
μ_{11}	71.8	87.8	63.0	100.5	92.5	113.6	−
μ_{20}	109.2	102.5	92.8	96.8	101.7	128.9	−
μ_{02}	77.0	86.1	105.9	101.1	86.7	78.5	+
μ_{30}	68.5	88.1	122.3	99.4	121.7	83.7	−
μ_{03}	99.2	95.7	90.8	106.0	88.8	110.5	−
μ_{22}	45.6	100.7	101.5	93.4	126.7	107.9	−
μ_{40}	146.5	96.3	68.7	112.5	102.1	114.4	−
μ_{04}	109.1	68.7	80.7	82.5	96.9	116.0	−

Table 10.2: **Pose error for learning sequences 7-12, utilising a range of different alternative energy functions, see table 10.1 for the corresponding pixel errors.**

(item §1 in the list), followed by an alternative area-based function (§2), finishing with yet another energy function based on the original surface overlapping (§3).

A standard of reference is provided, by initially using the original surface overlapping as "alternative" energy function, simply by calling DS three times with surface overlapping as basis (as done in section 7.3.4).

The alternative energy functions do not converge faster or slower than the original surface overlapping function; therefore all of the results in table 10.1 on the facing page and table 10.2 on the preceding page are obtained from approximately the same number of DS iterations, thereby allowing for directly comparison of the individual energy functions.

According to the Wilcoxon Signed Rank Test (confidence interval 95%), seven of the suggested alternative energy functions imply statistical significant improvement with respect to pixel error, with central moment 02 yielding the best absolute improvement in terms of pixel error. This may be because it shares fewest local minima with the surface overlapping function, or simply because it is a "lucky" choice in this specific case for R^{23}.

Looking at the pose error results, table 10.2, only two alternative cost functions result in statistical improvement - area and central moment 02; this time it is area which yields best absolute improvement.

Some of the alternative functions yield improvement with respect to pixel error but not pose error. This *again* indirectly indicates that the simplex has been digging too deep into wrong local minima, making it more and more difficult for it to escape - thus, in order to improve pose error, the pixel error obtained from the cost function should not be relied upon 100%, thereby again indicating that the cost function needs to be improved.

10.5 Summary

In this chapter, our idea of using alternative energy functions supplementing each other is shortly presented. Learning sequences 7-12 were subjects to a new optimisation philosophy, and results presented. One half of the suggested alternative energy functions yielded statistical improvement with respect to pixel error; thereby indicating that this idea deserves more attention in the future from other researchers who will hopefully be able to confirm or deny its feasibility.

Chapter 11

Conclusion

“My business cards have said "engineer" for several years,
but that’s mostly to impress women.”
- *Scott Adams*

In this chapter the contents of this report is summed up and a final conlusion on the project is drawn.

11.1 Summary

During resent years the area of Human Computer Interaction has received a lot of attention. In computer vision focus has been on recognising and interpreting hand gestures and body language. The aim of this project was to conduct a feasibility study and to investigate different possibilities for estimating hand poses to be used in the “Watching Window” application under development at the University of Otago, New Zealand.

A literature review showed that using computer vision for the purpose of hand pose estimation is still in an early stage of development. The bone structure of the hand makes it extremely articulate and thus hard to analyse by computer vison techniques. Nevertheless, numerous attempts have been made but all have required some constraints, where the most common is that no rotation and only insignificant translation of the wrist are allowed.

Based on the literature review it became clear that a complete solution to the problem could not be expected. Instead the aim of the project became to investigate whether pose estimation is achievable taking into account the setup of the Watching Window.

To accomplish this it was chosen to employ a method within the category of *analysis-by-synthesis*, where synthetic silhouettes are matched to input silhouettes from the cameras. Because the aim was to investigate different algorithms to perform this match, synthetic input images were rendered as this insured that problems could be traced back to the algorithms and not more fundamental problems.

In order to render synthetic silhouettes a model of the hand was needed. This led to an investigation of the structure of the hand and joint constraints. Based on this investigation, an existing model of the hand was chosen and changed to suit the specific needs.

The actual fitting was accomplished using numerical optimisation techniques, and thus a brief introduction was given on this subject. A cost function was needed and this was based on the overlapping of input and synthetic silhouettes; the more overlapping the lower value returned. Furthermore an error measure to evaluate the precision of the pose was defined, named “pose error”.

Three numerical optimisation algorithms were chosen:

- Spider Algorithm.
- Downhill Simplex (DS) Algorithm.
- Downhill Simplex combined with Simulated Annealing (DS/SA) algorithm.

From the literature only the DS was known to have been used in this context, while the Spider and DS/SA algorithms are new, to the best of our knowledge.

The Spider algorithm, based on a generalisation of the snake principle, was introduced first. An initial implementation was tested on a series of predefined learning sequences, each containing morphing between a number of different gestures. The Spider algorithm proved very capable of estimating the hand pose in the sequences with fixed wrist. Introducing wrist transformation in the sequences caused the Spider algorithm to produce big errors in the pose estimation. Three modifications to the initial implementation was proposed to remedy these big errors; changing the Spider tree topology, introducing 3D nodes and splitting the estimation in two parts, estimating only the wrist in the first and the whole hand in the second. All three modifications were tested and compared to the initial implementation. The implementation using 3D nodes showed some improvements in the pose estimation of the learning sequences. Based on these tests the 3D Spider algorithm was chosen to represent the Spider from that chapter.

The second optimisation algorithm presented was the Downhill Simplex algorithm. The principle of the algorithm was presented and an intuitive example given. As opposed to the Spider algorithm, DS had to be constrained to the dynamic range of the hand using a different technique. Several methods were discussed before a cost penalty function was chosen. Testing the algorithm on the learning sequences showed results similar to that of the Spider. In sequences where the wrist was fixed, very low pose and pixel errors were obtained, but in sequences where the wrist was free to move the errors were unacceptably high. Trying to improve the results, several restarts in each frame were tested. When the DS converges, a new simplex created from the lowest energy point was made. Both one and two restarts were tested as well, and one restart proved to be most effective and thus selected for the comparative test.

The last algorithm considered was Downhill Simplex in combination with Simulated Annealing, a hybrid of the Downhill Simplex. Where DS was a deterministic algorithm always seeking a lower function energy, DS/SA sometimes allows for a new energy to be higher. This way DS/SA is theoretically able to avoid getting trapped in local minima unlike the DS. DS/SA requires several parameters which had to be determined, and the first test using heuristically chosen parameters provided results comparable with those of the Spider and DS. Further investigation into the choice of starting temperature and number of allowed transformations and temperature intervals were conducted, but unfortunately there was no time to experiment with different cooling schedules.

To draw a conclusion on the three presented algorithms, comparative tests were conducted, where the algorithms were compared using six new test sequences, all with moving wrist, as acceptable results were already obtained in sequences with fixed wrist.

The test showed that the DS/SA algorithm on average does perform better than the other two algorithms when looking at pixel errors. This was backed up by the Wilcoxon Signed Rank test. When looking at pose errors, though, no significant difference between the algorithms could be proven.

When comparing speed between the three algorithms DS is the fastest algorithm; twice as fast as DS/SA and five times as fast as the Spider. However if a choice has to be made taking speed into consideration the choice should be DS/SA. This algorithm performs good on the overall, while both the Spider and DS perform best on the slow sequences.

The overall conclusion on the three algorithms is that none of them are feasible as a stand alone solution for estimation of the hand pose, they all fail at some point during the sequences. This said, there is still plenty of work to be done before DS/SA and the Spider algorithm can be rejected as possible solutions. Another conclusion to be made is regarding the use of silhouettes for the matching between input and evaluation. The varying results of the three algorithms indicate that silhouettes might not be the optimal descriptor. The hand is highly articulate and the silhouettes contain ambiguity, which despite the use of stereo vision cannot always be solved.

This realisation led to the investigation of other possible descriptors than the silhouette. A combination of the silhouette cost function and a cost function based on another descriptor was presented and implemented using the DS. A number of alternative descriptors were tested and the results were promising.

11.2 Concluding Remarks

The aim of this project has been to conduct an investigation into whether or not pose estimation is achievable using the current setup of the Watching Window.

To fulfil this we proposed several possible solutions based on analysis-by-synthesis. What sets these solutions apart from previously attempts is the combination of stereo vision and wrist movement. Furthermore using the Spider, the DS/SA and shape analysis algorithms for this purpose have to the best of our knowledge never been attempted before.

Currently none of the suggested solutions are able to perform pose estimation at the level desired for the Watching Window, not to mention performing in real time. It is our belief that this goal cannot be reached solely by the use of silhouette matching and history, due to the ambiguity of the silhouettes. Other methods should be considered, such as wrist estimation and maybe shadow analysis, to complement the pose estimation algorithm. We have proposed several other descriptors that may prove usefull in this context, but this must be subject to further investigation.

In this report we have focused on trying to estimate the hand pose in sequences where wrist movement occurs, one of the perhaps most difficult problems in pose estimation at present time. This proved very hard as the factor of wrist movement is very important for the estimation, underlined by the fact that in most reviewed litterature the authors actually constrain their research to be on sequences where no wrist movement occurs. We felt that research in this area would be more fruitful as methods for estimation of the hand pose with fixed wrist already exists.

This being said, both the Spider and DS/SA algorithms are able to perform with very low errors when fixing the wrist movement, being at least equal to the in this field already used DS algorithm. Seen from this perspective we feel that the Spider and DS/SA algorithms are important contributions to analysis-by-synthesis field; perhaps the use of these algorithms could prove very usefull in solving less difficult problems than searching a 23D search space.

Chapter 12

Process Analysis

> "Why do I get the feeling that someday
> I'll be describing this to a psychiatrist?"
> - *Lisa Simpson*

In this section a short description of the work process related to this project is given.

12.1 The Project

For both of us, this project has been somewhat different to previous projects - from 1st to 8th semester we have primarily dealt with problems which, to some extent, already had a solution - or a solution was almost provided beforehand.

This time around it was different, no immediate solution to the problem was given. Many attempts to capture the pose of a hand have been made, but none of these have solved the problem without requiring a series of constraints to be fulfilled.

However with the relative success of our previous projects we came into this project with the, in retrospect, naive attitude that this problem would be solved within a few months. We even had further plans for the system after having solved the problem of pose estimation.

12.2 Our Plans

Although this project has covered both 9th and 10th semester, real work on the project did not start until arrival in New Zealand, late November; primarily because the project was not decided on until mid October and some time was used on getting everything ready to go overseas. The time from September until departure for New Zealand was spent on acquiring knowledge within the use of C++ and OpenGL, with which neither of us had a lot of experience.

Upon arrival, a temporary timetable was made roughly resembling the one in table 12.1.

We soon realized that it might be a difficult topic, and as we in previous projects had focused on image processing (segmentation, etc.), we decided to work on synthetic images this time, as to focus on the pose estimation algorithms to avoid the associated problems with image segmentation. The philosophy was that if the system does not work on synthetic images, it certainly does not work on real images, either.

Month	To do...
December	Work on the Workbench and look on own ideas to solve the problem.
January	Continue working on own ideas.
February	Implement optimisation algorithm and obtain hand pose from this.
March	Switch to real-world input images.
April	Test the system.
May	Finish project, write up report.

Table 12.1: **The temporary timetable as of late November 2001.**

With the pose estimation complete and (of course) working, we *should* be able to move and optimise the system to real images, hereby showing the efficiency of our algorithms and how well they perform on real-life captured movie sequences.

It did unfortunately not turn out this way - we faced problems upon problems, hereby seriously undermining our initial time table.

12.3 Chronological Description

The Workbench alone took two weeks to implement (OpenGL is designed for image synthesis, *not* analysis, we found out), and after this we began working on our own ideas.

The main idea was from early start to design some methods to reduce the search space for a multidimensional optimisation algorithm. We chose to look at Volume Intersection and Inverse Kinematics. Early January it became clear that nothing could be gained from Volume Intersection in this case without too much effort, thus this idea was rejected. Instead work on the Downhill Simplex optimisation algorithm began, trying to estimate the pose of the hand using analysis-by-synthesis.

By this time we began to realize how many problems hand pose estimation involves. The work with Inverse Kinematics also turned out to be a dead end; even locating the wrist turned out to be overwhelming difficult under the circumstances, so by the end of January our focus in the project was shifted. Instead of just using Downhill Simplex as a optimisation algorithm we decided to look at alternatives, the Simulated Annealing and Spider algorithm - the latter suggested by our Otago supervisor, Brendan McCane.

Throughout February and March we worked on these three optimisation algorithms and their modifications. During this time it was decided to drop the initial idea of using real images as input data. Mainly because there were so many opportunities with these three algorithms that exploring only some of these would take up the rest of our time.

Furthermore, it is very difficult to generalise using only six learning sequences and on top of that having to face random fluctuations.

Because the algorithms did not work in real time, actually taking up to 30 seconds per frame to process, for once we were waiting for the computers and not the other way around. This left us with other problems too, especially with respect to the parameter estimation of the algorithms. There are endless number of parameter combinations which could be tried with the algorithms, so in the end we found ourselves manually optimising optimisation algorithms; spending valuable time on this (in our eyes) foolish but necessary task.

12.4 Final Remarks

In retrospect we have gained a lot of experience from this project. As this was the first project in which we conducted actual research on our own, we gained insight into how difficult and at times frustrating research can be. On several occasions hard and fun work on an idea turned out to be a dead end, and suddenly not fun at all. Taking these kind of "beatings" have been a new experience for us and if nothing else taught us valuable lessons.

Secondly, when reading and reviewing articles one should be critical to the text, not taking everything for granted. Even if an article shows a solution to a problem, one cannot expect to obtain the results obtained by the authors.

Thirdly, when the workload is distributed on only two persons, instead of the usual five or six people, it naturally takes more time to get things done, but we did not really realize this fact until we found ourselves in the situation.

Finally, coming to another country and working with other people has been a great positive experience, both on a personal and professional level. Being so far from home ground implied more responsibility on our shoulders, as e-mail correspondence with our main supervisor in Denmark, Claus B. Madsen, was more difficult than expected - taking the nature of the work into consideration; it is *very* difficult to describe technical problems in an e-mail. Luckily, staff from the CS department at the University of Otago were very friendly and willing to assist us if we had questions.

Bibliography

[1] Rafael C. Gonzales and Richard E. Woods. *Digital Image Procesing, Third edition.* Addison Wesley Publishing Company, 1993. ISBN 0-201-50803-6.

[2] T. Moeslund and E. Granum. A survey of computer vision-based human motion capture, 2001. URL `http://citeseer.nj.nec.com/moeslund01survey.html`.

[3] International signlanguage alphabet, 2002. URL `http://www.deafhome.com/tegnsprog/haandalfabetet/`.

[4] Alister West and Keekim Heng. The watching window: A reactive real-time model-viewer. In *Image and Vision Computing New Zealand*, pages 25–30, 2001.

[5] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding: CVIU*, 73(1):82–98, 1999. URL `http://citeseer.nj.nec.com/gavrila99visual.html`.

[6] T. Huang and V. Pavlovic. Hand gesture modeling, analysis, and synthesis, 1995. URL `citeseer.nj.nec.com/huang95hand.html`.

[7] Vladimir Pavlovic, Rajeev Sharma, and Thomas S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997. URL `http://citeseer.nj.nec.com/pavlovic97visual.html`.

[8] Charles Cohen. A brief overview of gesture recognition, 1999. URL `http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/COHEN/gesture_overview.html`.

[9] Dong Guo Yonghua. Vision-based hand gesture recognition for human-vehicle interaction. URL `citeseer.nj.nec.com/467750.html`.

[10] Henrik Birk, Thomas B. Moeslund, and Claus B. Madsen. Real-time recognition of hand alphabet gestures using principal component analysis. *Proceedings of the 10th Scandinavian Conference on Image Analysis*, 1997. URL `http://www.cvmt.auc.dk/~tbm/publications/`.

[11] Andrew Blake and Michael Isard. *Active Contours.* Springer, 1998. ISBN 3-540-76217-5.

[12] James M. Rehg and Takeo Kanade. Visual tracking of high DOF articulated structures: an application to human hand tracking. In *ECCV (2)*, pages 35–46, 1994. URL `http://citeseer.nj.nec.com/rehg94visual.html`.

[13] Hocine Ouhaddi and Patrick Horain. Hand tracking by 3d model registration. 1999. URL `http://citeseer.nj.nec.com/234649.html`.

[14] James Davis and Mubarak Shah. Toward 3-d gesture recognition. URL `http://citeseer.nj.nec.com/20415.html`.

[15] Yoshinari Kameda. Human pose and motion estimation with a three dimensional articulated object model, 1999.

[16] B. Stenger, P.R.S.Mendonca, and R.Cipolla. Model-based 3d tracking of an articulated hand.

[17] Hugo Wesseling. A hands-on approach.

[18] Greg Slabaugh, Bruce Culbertson, Tom Malzbender, and Ron Schafer. A survey of methods for volumetric scene reconstruction from photographs. URL `http://www.hpl.hp.com/research/cp/cmsl/publications/ibg/VolReconSurvey.pdf`.

[19] Visual hull. URL `http://www.polito.it/~bottino/VisualHull/index.html`.

[20] Etsuko Ueda, Yoshio Matsumoto, Masakazy Imai, and Tsukasa Ogasaware. Hand pose estimation using multi-viewpoint silhouette images. pages 1989–1996, October 2001.

[21] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, 1993.

[22] Ying Wu and Thomas S. Huang. Capturing articulated human hand motion: A divide-and-conquer approach. *In Proc. IEEE Int'l Conference on Computer Vision*, pages 606–611, 1999.

[23] Wen Tang, Marx Cavazza, Dale Mountain, and Rae Earnshaw. A constrained inverse kinematics technique for real-time motion caputure animation. *The Visual Computer*, 15:413–425, 1999.

[24] The skeletal system, 2002. URL `http://emuseum.mankato.msus.edu/biology/humananatomy/skeletal/skeletalsystem.html`.

[25] Chor Guan Teo. A hybrid procedural/knowledge-based approach to the animation of human hand grasping, 1994.

[26] A. Vardy. Articulated human hand model with inter-joint dependency constraints, 1998. URL `http://www.scs.carleton.ca/~avardy/hand/hand_doc.pdf`.

[27] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, Second edition.* Cambridge University Press, 1992. ISBN 0-521-43108-5.

[28] Numerical non-linear inversion techniques. URL `http://citeseer.nj.nec.com/huang95hand.html`.

[29] Franklin Mendivii, R. Shonkwiler, and M. C. Spruill. Optimization by stochastic methods, 1999. URL `http://www.math.gatech.edu/~shenk/soc.pdf`.

[30] Y.C.Ho. Class notes #13 - capsule description of the tools of computational intelligence, 2001.

[31] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[32] Brendan McCane. Snakes and spiders, 2000.

[33] Active contours (snakes) for boundary delineation. URL `http://markschulze.net/snakes/`.

[34] Amir A. Amini, Terry E. Weymouth, and Ramesh C. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, 1990.

[35] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[36] Why not use numerical recipes?, . URL `http://math.jpl.nasa.gov/nr/`.

[37] Distressing rumors, and how we handle them, . URL `http://www.nr.com/bug-rebutt.html`.

[38] Grzegorz Kaczmarczyk. Downhill simplex for many (∼20) dimensions. URL `http://paula.univ.gda.pl/dokgrk/simplex.html`.

[39] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, pages 1087–1092, 1953.

[40] Combinatorial optimization problems, state spaces and local search. URL `http://www.seas.gwu.edu/∼simhaweb/cs151/lectures/module10/module10.html`.

[41] Harry Cohn and Mark Fielding. Simulated annealing: Searching for an optimal temperature schedule. URL `http://epubs.siam.org/sam-bin/dbq/article/32968`.

[42] Newsweek, march 25, 2002.

[43] Class notes #10 - shape analysis & measurement, 2001. URL `http://hebb.cis.uoguelph.ca/∼mwirth/Teaching/CIS6320/Lecture10.pdf`.

Part IV

Appendix

Workbench

Defined Gestures and Learning Sequences

Wilcoxon Signed-Rank Test

Error Examples

Test Results

Appendix A

Workbench

> "When the final design seems too simple for the amount of work you've put in, then you know you're done."
> - *Brady Clark*

Because many different ideas are tried throughout this report, a common workbench is built in order to simplify and ease the implementation of the more sophisticated parts. This utility is named the "WorkBench" because of its versatility. All high-speed code throughout this report was written using the Microsoft Visual C++® 6.0 compiler. This compiler was chosen because all code so far in the Watching Window project was written in this environment.

Desktop

Figure A.1 shows a screenshot of the WorkBench in action.

Five windows are opened when starting up the WorkBench: The two uppermost windows represent the stereo input to the system. The two middle windows show the output of the matching process (again stereo), and the bottom window is a console displaying debug information and generel data output.

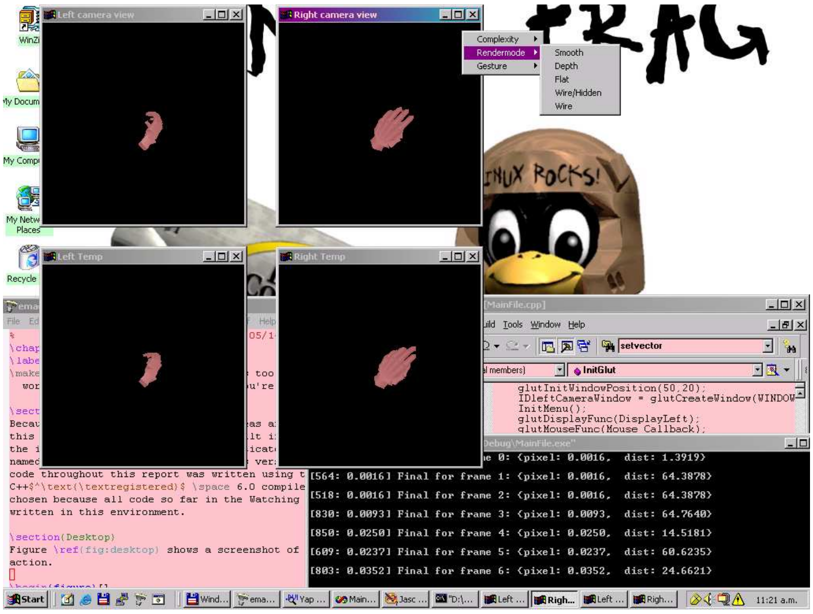
Structure

Fundamentally, the WorkBench is based upon three object classes and the mainfile. The three classes are: A hand model, a camera and a "playground" class.

The classes are handled by the mainfile, which also handles windows, menus and system callbacks. Two hand objects are created; one representing the input to the system (the input silhouette), and one for output- and evaluation purposes. Furthermore, four camera objects are created - two for the input cameras and two for the "output" cameras. Finally, the playground object is created, allowing for encapsulated experiments.

The hand model class handles everything with respect to the hand model; rendering in OpenGL, handling dynamic constraints and setting and retrieving the position vector for the hand.

The camera class handles a model of the camera instantiated with a position in space, roll, pitch, yaw and focal length. Based on these parameters the object provides functions



```
code throughout this report was written using t
C++$^\text{\textregistered}$ \space 6.0 compile
chosen because all code so far in the Watching
written in this environment.

\section{Desktop}
Figure \ref{fig:desktop} shows a screenshot of
action.
```

```
glutInitWindowPosition(50,20);
IDleftCameraWindow = glutCreateWindow(WINDOW
InitMenu();
glutDisplayFunc(DisplayLeft);
glutMouseFunc(Mouse_Callback);
```

```
e 0: <pixel: 0.0016,  dist: 1.3919>
[564: 0.0016] Final for frame 1: <pixel: 0.0016,  dist: 64.3878>
[518: 0.0016] Final for frame 2: <pixel: 0.0016,  dist: 64.3878>
[830: 0.0093] Final for frame 3: <pixel: 0.0093,  dist: 64.7640>
[850: 0.0250] Final for frame 4: <pixel: 0.0250,  dist: 14.5181>
[609: 0.0237] Final for frame 5: <pixel: 0.0237,  dist: 60.6235>
[803: 0.0352] Final for frame 6: <pixel: 0.0352,  dist: 24.6621>
```

Figure A.1: **Desktop shot**

to position the viewpoint correct in OpenGL, and convert between the camera and world coordinate system.

Finally, the "playground" class provides the playground for development. This is the only primary class whose contents vary for each optimization algorithm. The playground class handles a hand model object to render the evaluation silhouettes and also provides functions for estimation of the pixel and pose error.

Appendix B

Defined Gestures and Learning Sequences

> "There is no difference between Time and any of the three dimensions of Space, except that our consciousness moves along it."
> - *H.G.Wells*

To optimise the optimisation algorithms, twelve learning sequences are designed. Each sequence is 250 frames long and can basically be defined as a series of gestures, morphing from gesture to gesture over time. To test how speed as well as wrist translation influence the optimisation process, the 12 sequences are divided into four groups, as shown in table B.1:

	Slow animation	Fast animation
Fixed wrist	Sequences 1, 2, 3	Sequences 4, 5, 6
Moving wrist	Sequences 7, 8, 9	Sequences 10, 11, 12

Table B.1: **Split-up of sequences.**

Fixed vs. Moving Wrist

"Fixed wrist" means that the wrist is spatially fixed with respect to translation and rotation, although different from sequence to sequence. "Moving wrist" implies transformation of the wrist, implemented by using sine functions, with variables chosen to achieve a "nice, smooth, natural motion" of the hand. Translation limits are ±5 centimetres from origo along all axes, rotation is restricted to $x \in [0^o;\ 180^o]$, $y \in [-90^o;\ 90^o]$, $z \in [-180^o;\ 180^o]$.

Speed

To test how the algorithms perform under different conditions with respect to speed, six sequences (table B.1) are based on "slow animation", six on "fast animation".

The terms "slow" and "fast" are somewhat vague, as the frame rate influences the relative movement from frame to frame.

Fingers

Slow animation means that the individual finger joint angles each frame change at a rate of approximately $\pm 5^{o}$, fast animation means approximately $\pm 10^{o}$ per frame when morphing to a new gesture. In order to avoid integer angles and constant morphing speed, an "uncertainty" of $\pm 10\%$ is implemented, thus slow animation should be interpreted as rotation speed in the range $[-4.5;\ 5.5]$ degrees per frame, fast animation in the range $[-9.0;\ 11.0]$ degrees each frame.

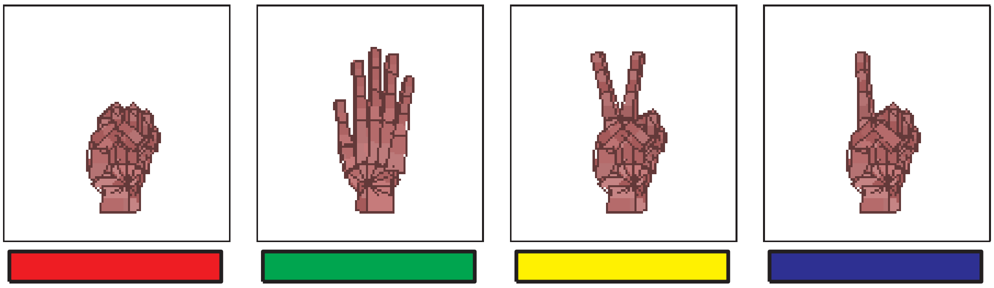
At 10 frames per second, the fast sequences imply morphing from any gesture to any gesture within less than one second. In the slow sequences, two seconds are required to perform the same task.

Wrist

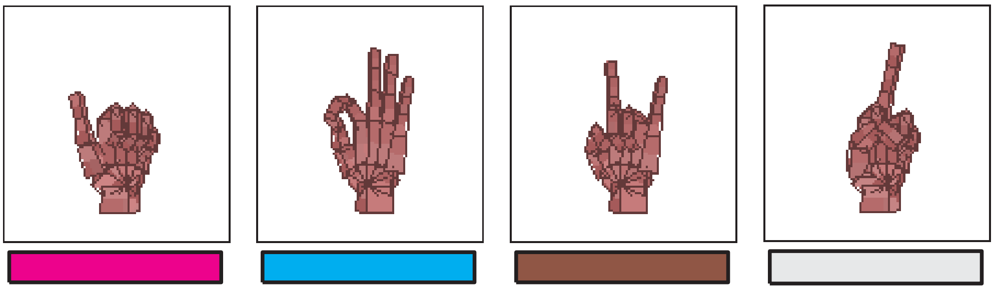
Translation of the wrist is based on a sine movement with an amplitude of ± 5 centimetres. The angular velocity specified involves movement from one extreme to the other within $1\frac{1}{2}$ seconds for the slow animation, $\frac{3}{4}$ seconds for the fast animation (10 fps).

Wrist rotation is, as wrist translation, based on a sine movement. Rotation from one extreme to the other around the x- and y-axes is performed within less than 6 seconds (180^{o}), and less than 12 seconds for the z-rotation (full 360^{o} rotation).

Gestures



(a) Fist (b) Open hand (c) "Victory" (d) Pointing finger



(e) Thumb (f) "OK" (g) "Homie" (h) "Birdie"

Figure B.1: **The eight gestures defined to create different learning sequences, and their associated colour codes.**

Eight different gestures are defined for the learning sequences, as shown in figure B.1. The gestures are chosen in pairs with some similarities. The "Victory" gesture for instance has some similarities with the "Homie" gesture. Likewise for the pointing finger and the "birdie" gesture. Each gesture has a colour associated with it to simplify sketching of the sequences.

Learning Sequences

The course of the twelve learning sequences is shown in figure B.2. All sequences start with the open hand gesture as seen in figure B.1(b) in frame 0. Thus sequence 1 starts out by morphing from the open hand to the “OK” gesture for the first 20 frames. After a couple of frames with no morphing, indicated by the white space, the sequence then continues to morph from the “OK” gesture to the “Homie” gesture, and so on.

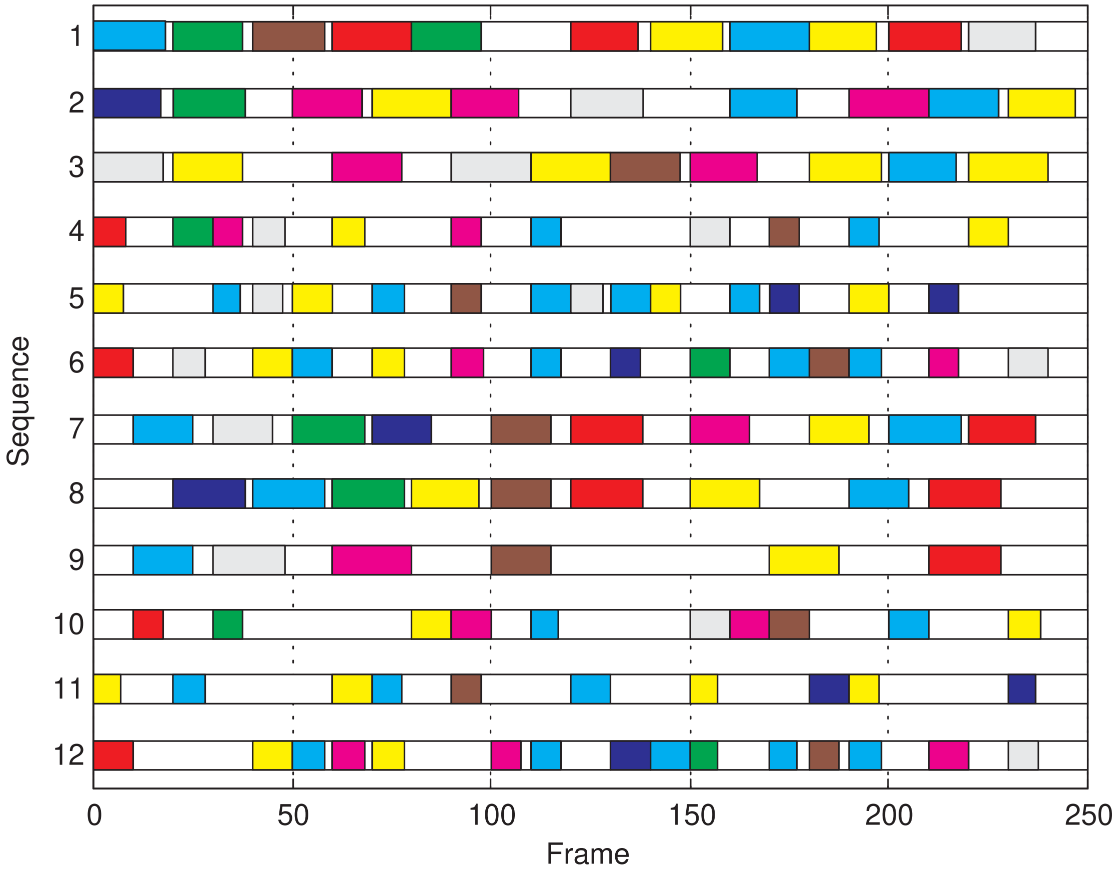


Figure B.2: **Sequences used for optimising the performance of the optimisation algorithms.**

Examples of Sequences

Two examples of the learning sequences are given here. In figure B.3 on the following page snapshots for every 7th frame of learning sequence 1 are given. Note that the images have been cropped, causing the hand to appear larger than in reality. This could be done because of the fixed wrist. In figure B.4 the images are not cropped giving the true aspect ratio between area taken up by the hand, and the area of the image. These snapshots are from learning sequence 9.

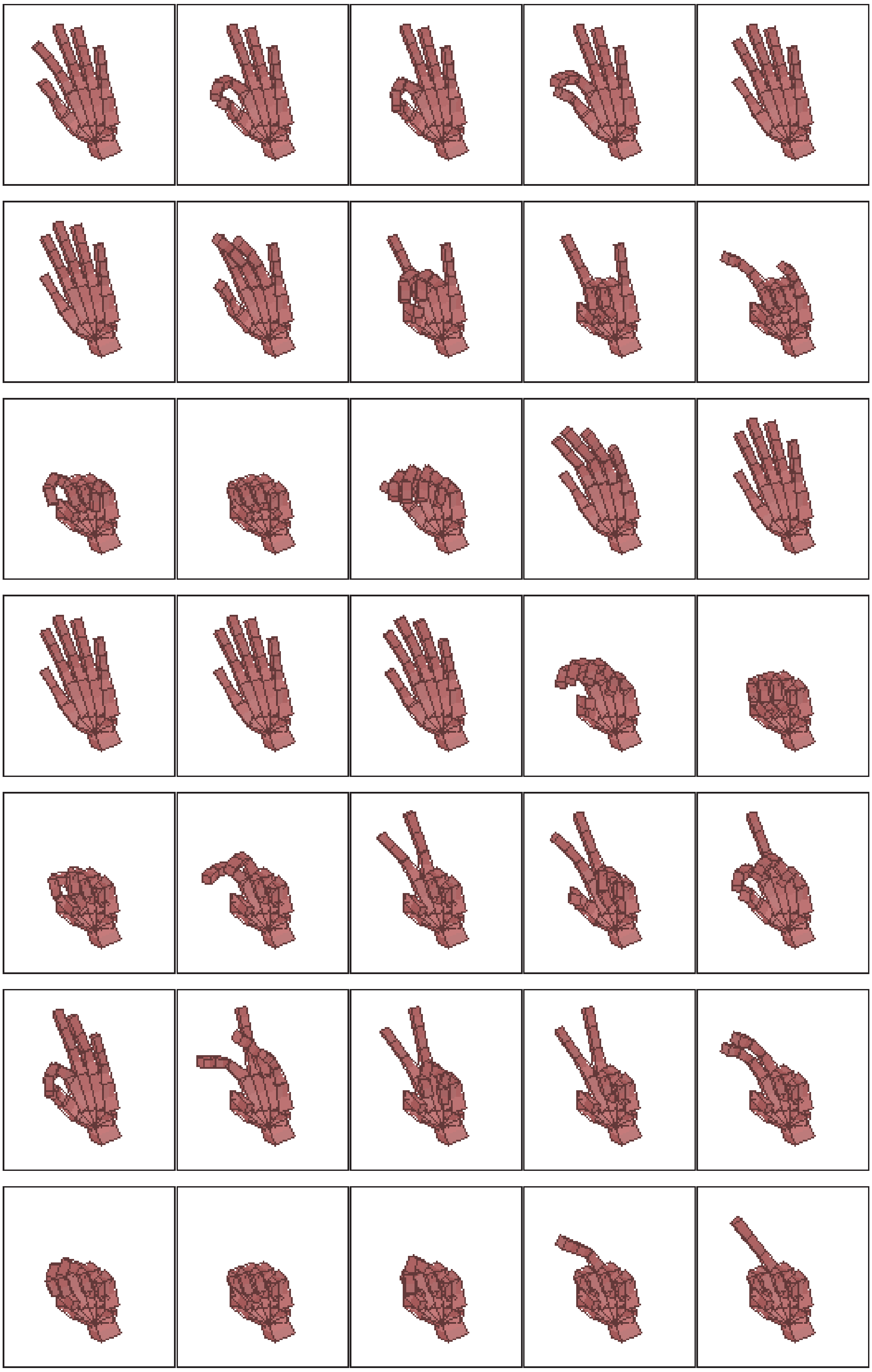


Figure B.3: **Snapshots of every 7th frame of learning sequence 1. Notice that the images have been cropped, causing the hand to appear larger than it is in the input images.**

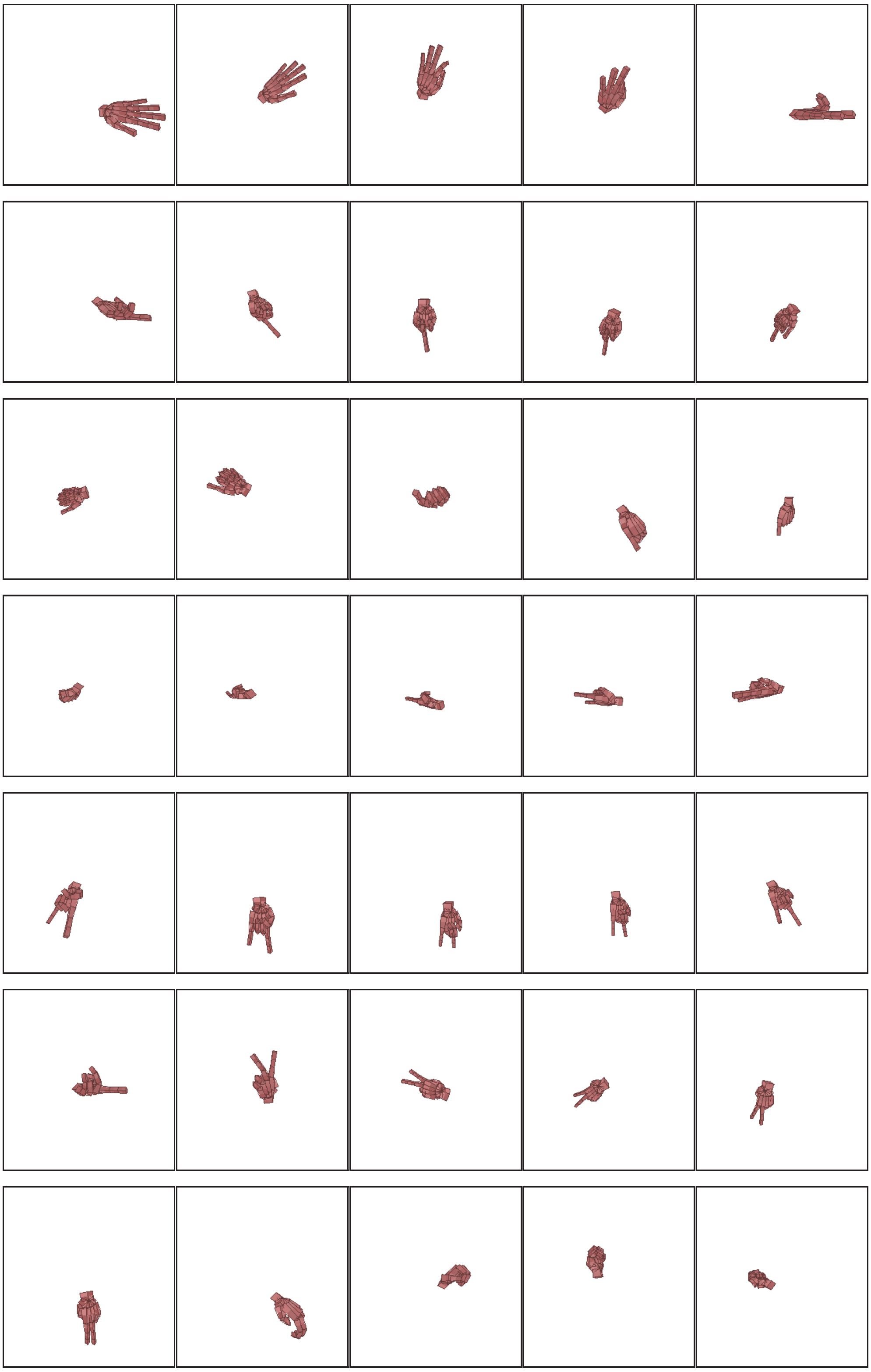


Figure B.4: **Snapshots of every 7th frame of learning sequence 9. Notice that the images are displayed as they appear in the input images.**

Appendix C

Wilcoxon Signed-Rank Test

> "Statistics; a tool which can be used to prove
> that the moon is made of cheese."
> \- *Scott A. King*

To test whether or not a modification to the optimisation algorithms has had any effect a paired non-parametric test for significance is necessary to determine if the modification has had positive or negative influence on the outcome.

The Wilcoxon Signed-Rank Test can be used for this kind of statistical significance testing and will shortly be described in this appendix.

Consider as an example, a medical experiment - N patients are subjects to a new medical treatment. Their well-being is recorded before and after treatment, and it is desirable to evaluate if the treatment had the desired effect (usually improving the health of the subjects). However, some patients are in better condition than others before the treatment, thus it is not statistically feasible to apply the same yardstick to everybody.

The same principle applies to the tests in this report. To examine if a modification has positive or negative influence on the general performance of an algorithm, the state of the individual sequences must be compared before and after the modification, hereby taking the magnitude of their performance into account.

Preconditions

To return to the medical example, suppose the subjects in table C.1 gives the test results in column X_1 and X_2 as result before and after treatment, respectively. Their differences, $X_1 - X_2$, and absolute values of these differences, $\|X_1 - X_2\|$, are calculated and the subjects are sorted according to these values.

The subjects are assigned a rank and a "signed" rank, as can be seen in the rightmost two columns. If $X_1 - X_2$ equals 0, the subject it is eliminated as it provides no useful information. In this case, it causes the test statistics to be based on 5 instead of six samples ($N = 5$). If values of $X_1 - X_2$ from different subjects are equal, they are assigned an average rank. As an example, subject 3 and 4 in table C.1 share ranks of 2 and 3, therefore they are both assigned a common rank of $(2 + 3)/2 = 2.5$. Afterwards this rank is given a sign according to the sign of $X_1 - X_2$ to provide the signed rank. The sum of the signed ranks is then calculated and assigned to W.

Subj.	X_1	X_2	$X_1 - X_2$	$\|\|X_1 - X_2\|\|$	Rank	Signed Rank
1	14	14	0	0	—	—
2	17	18	-1	1	1	-1
3	22	20	$+2$	2	2.5	2.5
4	14	12	$+2$	2	2.5	2.5
5	28	32	-4	4	4	-4
6	31	25	$+6$	6	5	5
mean difference=			$+5$			$W = 5$

Table C.1: **First steps of the Wilcoxon Signed Rank Test. The observations, their difference and absolute difference are entered in the table. The subjects are assigned a rank and a signed rank, see the rightmost two columns.**

As the maximum possible value of W is given by

$$W_{max} = \sum_{i=1}^{N} i = \frac{N(N+1)}{2} \tag{C.1}$$

and the corresponding negative value is given by $W_{min} = -W_{max}$ (In this specific case, $N = 5$ and thus $W_{max} = 30$), if W was close to W_{max} one would say that the treatment had a positive change. Similar, if $W = 0$ one would be able to say that the treatment had no effect.

The aim of the test is to test whether or not the treatment has had any "significant" change to the condition of the subjects. So the NULL-hypothesis is H_0 formulated as:

$$H_0 : X_1 = X_2 \tag{C.2}$$

and the alternative hypothesis is either (for non-directional tests)"

$$H_a : X_1 \neq X_2 \tag{C.3}$$

or (for directional tests):

$$H_a : X_1 \geq X_2 \tag{C.4}$$

If W is positive, it indicates improvement from X_1 to X_2; if it is negative, a deterioration. If there is no tendency in positive or negative direction, the value of W is expected to approximate zero.

Stating H_0 as $X_1 = X_2$ is the same as expecting W to approximate 0, which in turn is the same as saying that W belongs to a sampling distribution with zero mean, $\mu_W = 0$.

The variation of this sampling distribution is based on the number of samples N in the test. If $N \geq 10$ it is a reasonably close approximation of the normal distribution, denoted "large sample case". However, if $N < 10$ a different approach is used - this is the "small sample case":

Small Sample Case

In small-sample cases, the value of W must be compared to the values presented in table C.2 for different confidence intervals α. The *non-directional test* is used if the aim is to decide whether X_2 is different from X_1 (equation C.3); *directional test* is used if the aim is to decide if X_2 is larger *or* smaller than X_1 (equation C.4).

	Desired level of significance for α			
	Directional Test			
	0.050	0.025	0.010	0.005
	Non-Directional Test			
	–	0.050	0.020	0.010
$N = 5$	15	–	–	–
$N = 6$	17	21	–	–
$N = 7$	22	24	28	–
$N = 8$	26	30	34	36
$N = 9$	29	35	39	43

Table C.2: **Table showing critical $\pm W$ values for small sample cases $(5 \leq N' \leq 9)$. Compare W to the values in this tables to decide whether or not to reject H_0.**

Returning to the medical example from table C.1, there were five samples, $N = 5$, and the sum of the signed ranks was also five, $W = 5$. As can be seen in the table, to reject H_0 at a significance level of $\alpha = 0.05$ would require $W > 15$, which clearly is not the case. Thus H_0 is accepted at $\alpha = 0.05$, which is to say that the treatment in this example did not yield a statistical difference in the subjects condition.

Large Sample Case

In large sample cases, with $N \geq 10$, the z is calculated:

$$z = \frac{(W - \mu_W) \pm 0.5}{\sigma_W} \tag{C.5}$$

where

$$\sigma_W = \sqrt{\frac{N(N+1)(2N+1)}{6}} \tag{C.6}$$

The ± 5 is correction for continuity, and chosen as -5 if W is less than μ_W and as $+5$ if W is greater. Because $\mu_W = 0$ a simpler computational formula is derived:

$$z = \frac{W - 0.5}{\sigma_W} \tag{C.7}$$

z must then be compared to the values in table C.3 on the next page in order to determine if H_0 should be rejected.

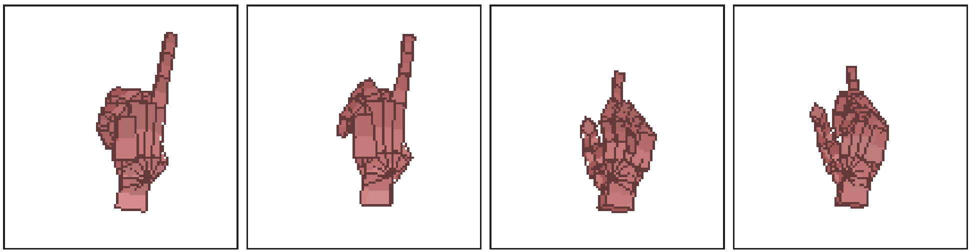
Desired level of significance for α				
Directional Test				
0.050	0.025	0.010	0.005	0.0005
Non-Directional Test				
–	0.050	0.020	0.010	0.0010
$z_{critical}$				
1.645	1.960	2.326	2.576	3.291

Table C.3: **Table showing the critical z values for large sample cases ($N \geq 10$). Compare z to the values in this tables to decide whether or not to reject H_0.**

Appendix D

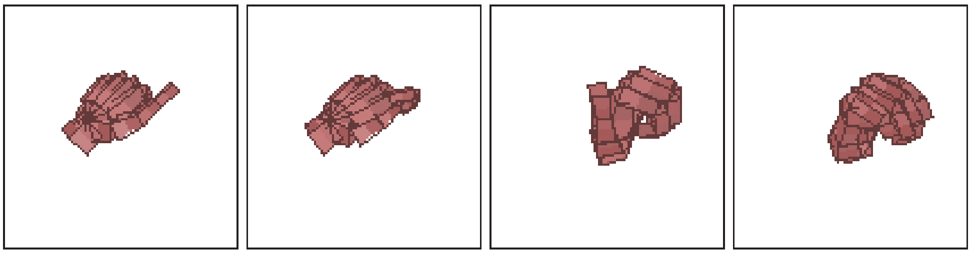
Error Examples

This appendix shows a series of error examples, hopefully giving an impression of the magnitude of pixel- and pose error in a few situations.



(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure D.1: **Pixel error: 0.0236, Pose error: 21 cm.**



(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure D.2: **Pixel error: 0.0374, Pose error: 51 cm.**

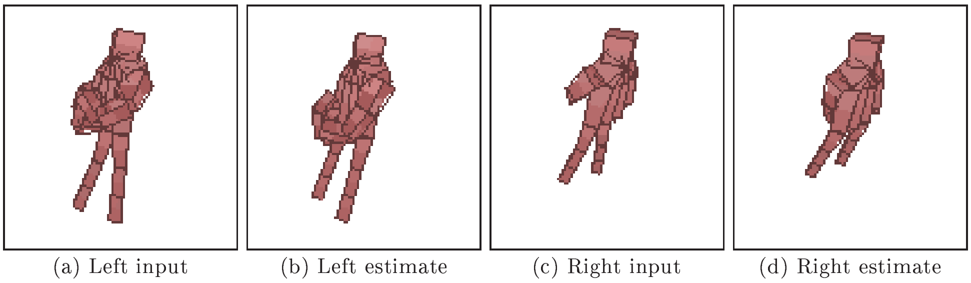

(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure D.3: **Pixel error: 0.0811, Pose error: 87 cm.**

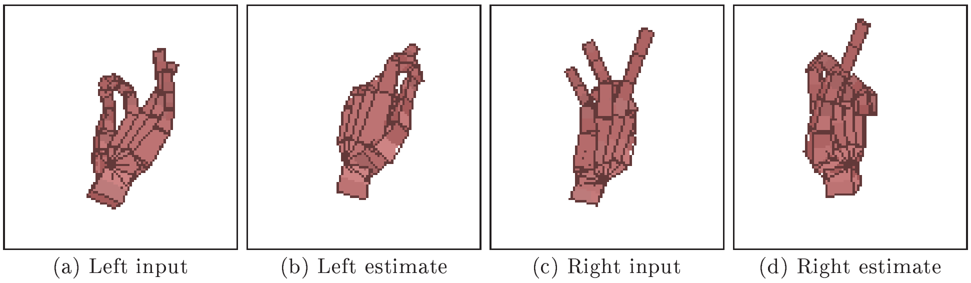

(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure D.4: **Pixel error: 0.0976, Pose error: 146 cm.**

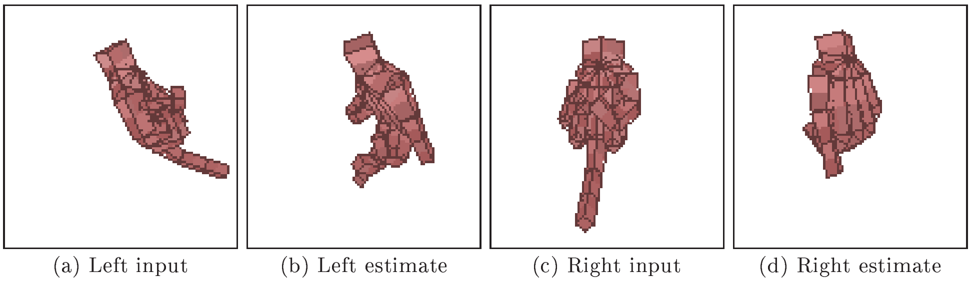
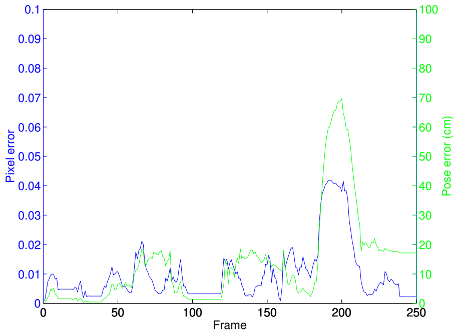

(a) Left input (b) Left estimate (c) Right input (d) Right estimate

Figure D.5: **Pixel error: 0.1258, Pose error: 160 cm.**

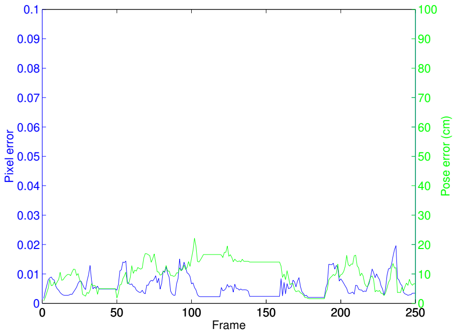
Appendix E

Test Results

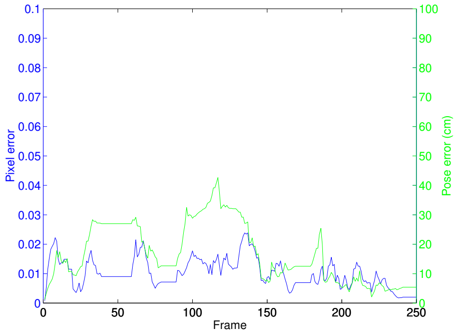
Spider Test - Initial Test



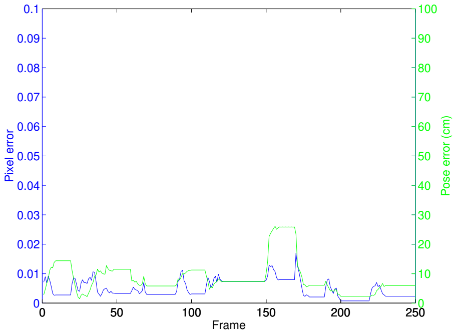
Sequence 1



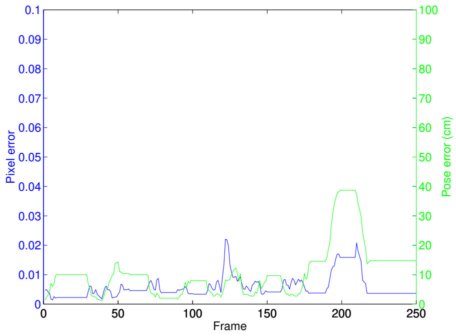
Sequence 2



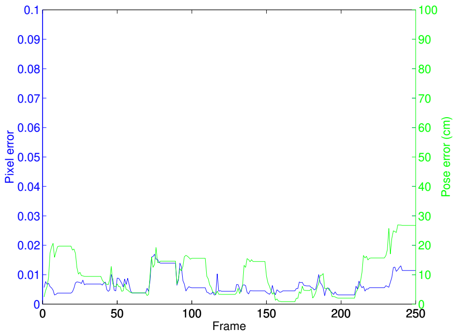
Sequence 3



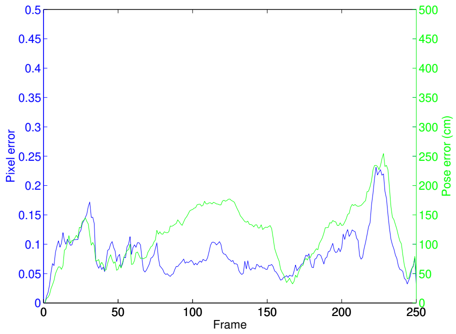
Sequence 4



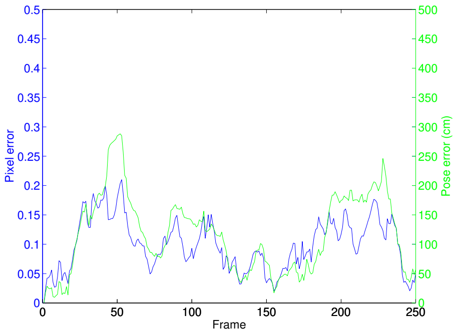
Sequence 5



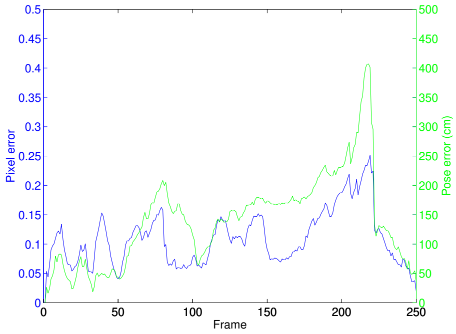
Sequence 6



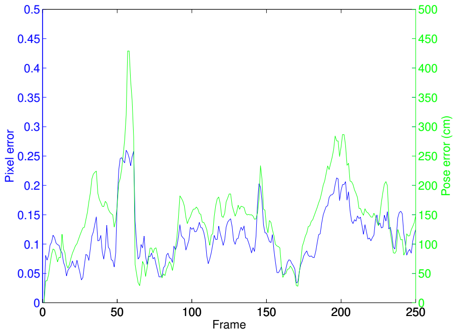
Sequence 7



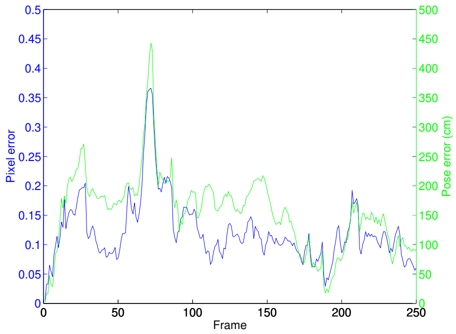
Sequence 8



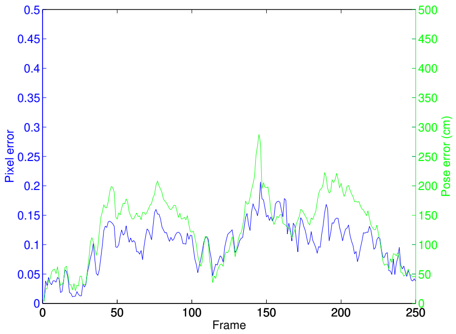
Sequence 9



Sequence 10

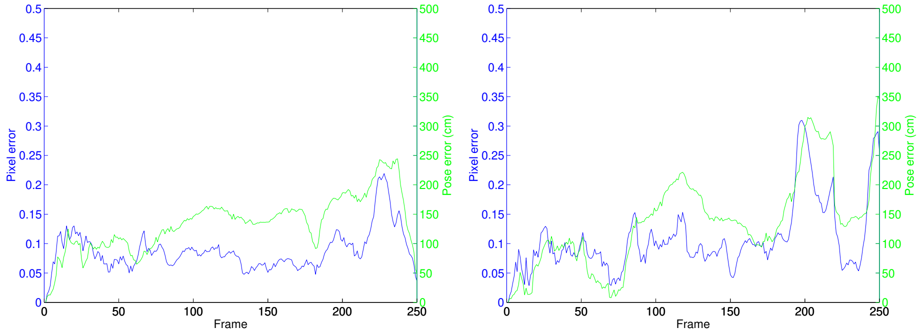


Sequence 11

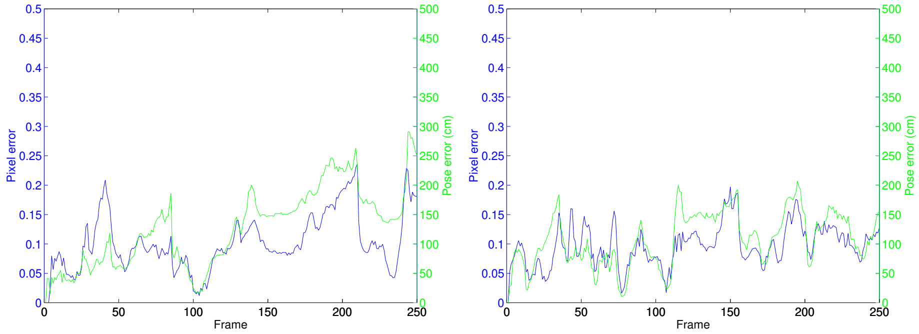


Sequence 12

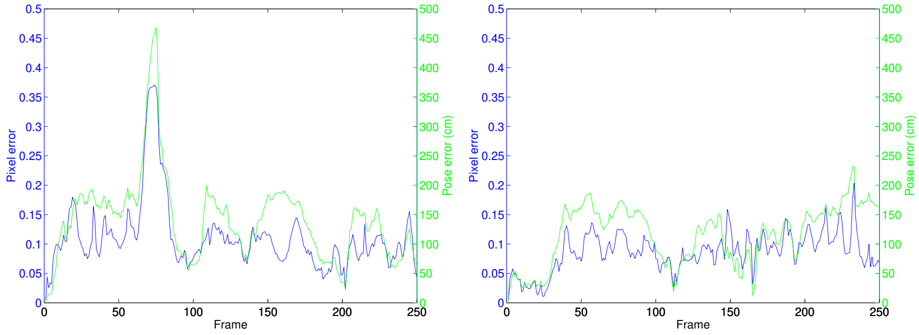
Spider Test - Restructured Spider Tree



Sequence 7 Sequence 8



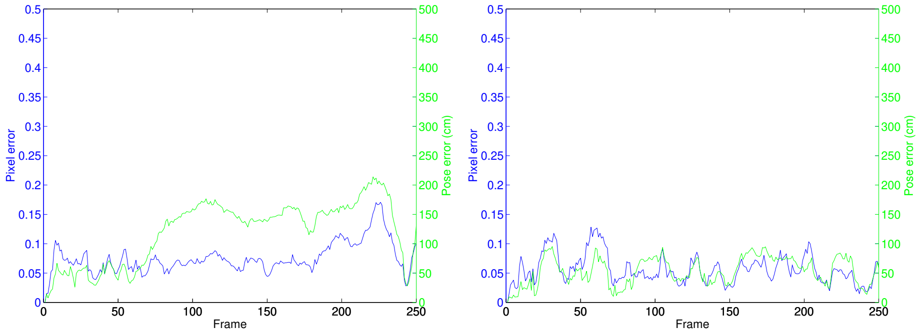
Sequence 9 Sequence 10



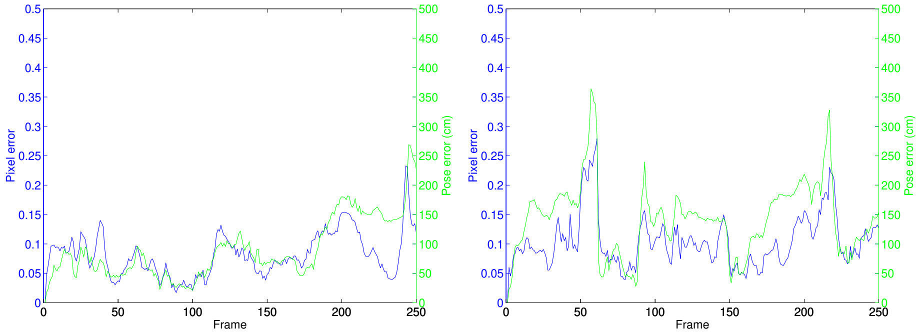
Sequence 11 Sequence 12

Figure E.1: **Test results for the Restructured Spider Tree. All sequences are with moving wrist.**

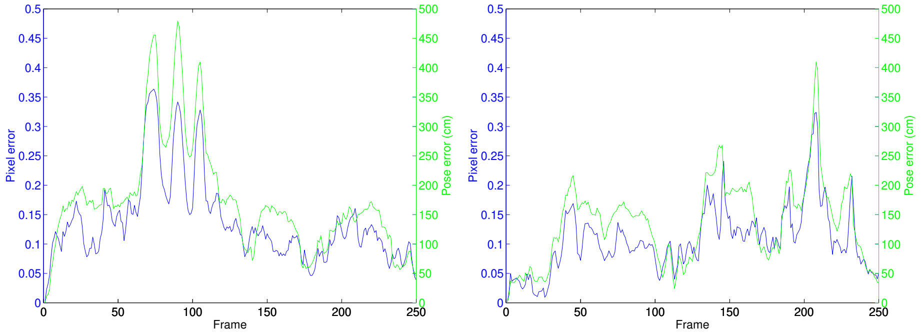
Spider Test - Three Dimensional Nodes



Sequence 7 Sequence 8



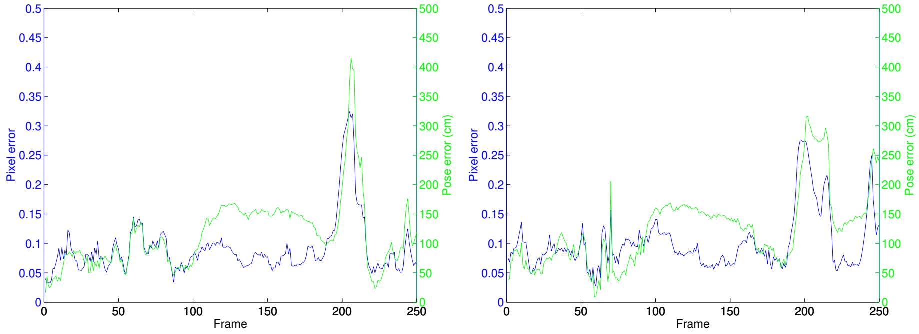
Sequence 9 Sequence 10



Sequence 11 Sequence 12

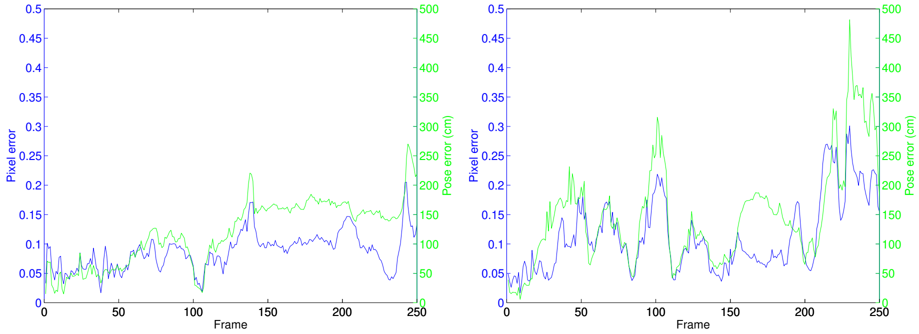
Figure E.2: **Test results for using Three Dimensional Nodes in the tree structure. All sequences are with moving wrist.**

Spider Test - Two Part Spider



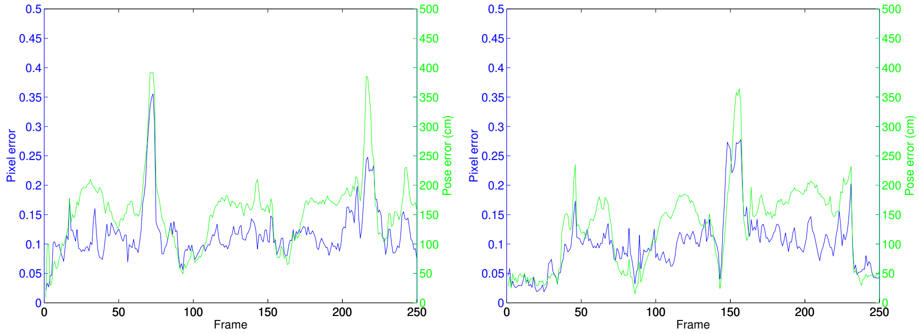
Sequence 7

Sequence 8



Sequence 9

Sequence 10

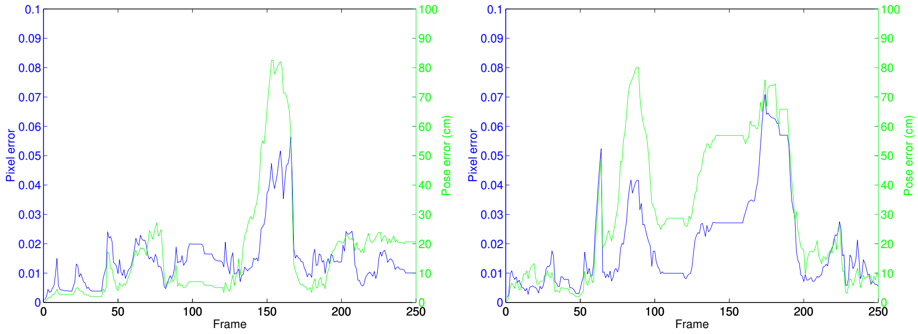


Sequence 11

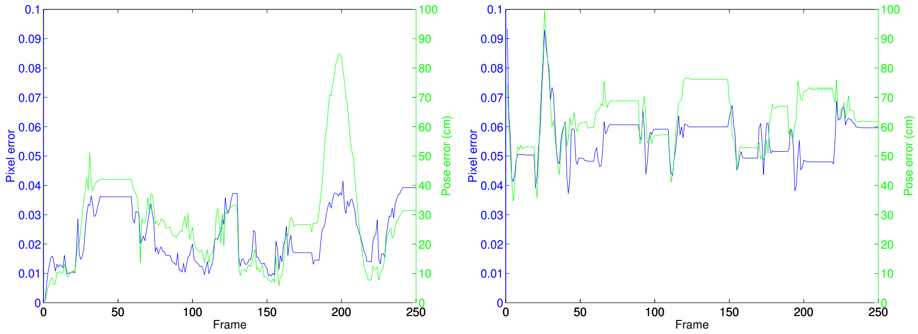
Sequence 12

Figure E.3: **Test results for the Two Part Spider. All sequences are with moving wrist.**

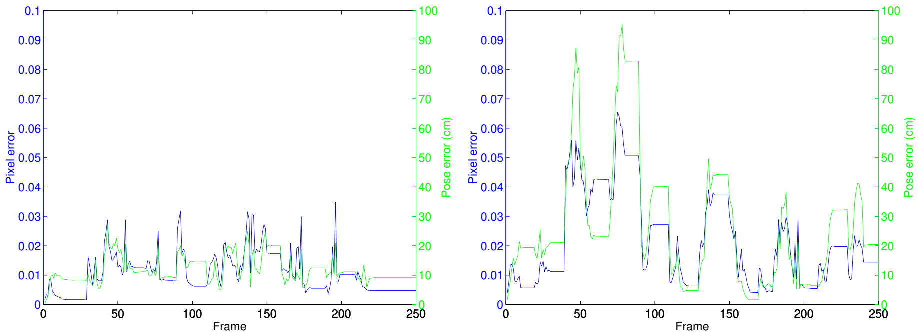
Downhill Simplex - Initial Test



Sequence 1 Sequence 2



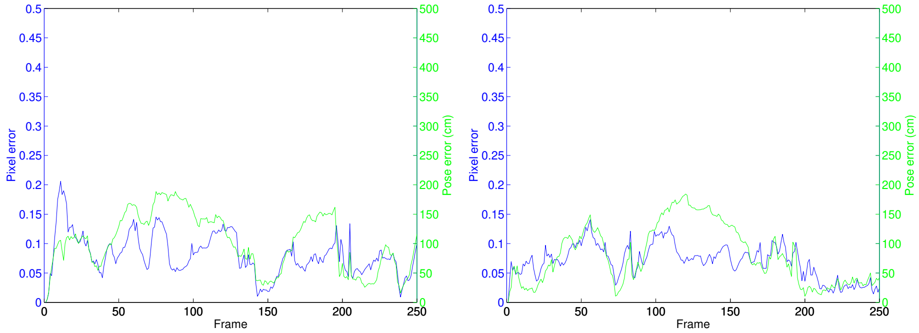
Sequence 3 Sequence 4



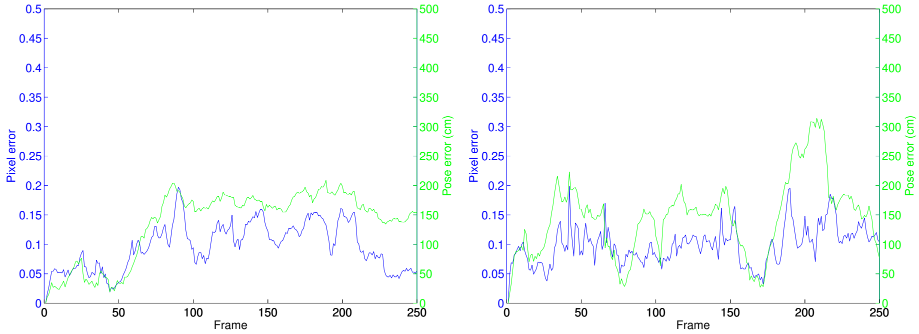
Sequence 5 Sequence 6

Figure E.4: **Test results for the Downhill Simplex initial tests. All sequences are with the wrist fixed.**

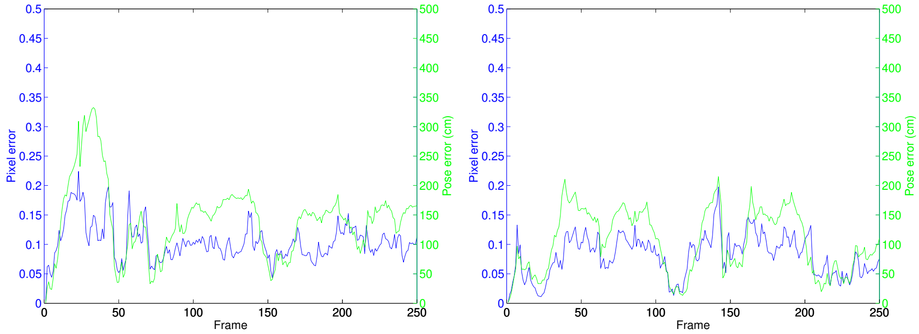
Downhill Simplex - Initial Test



Sequence 7 Sequence 8



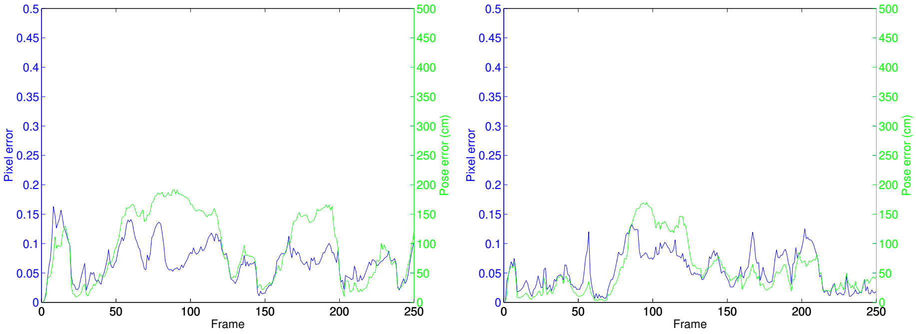
Sequence 9 Sequence 10



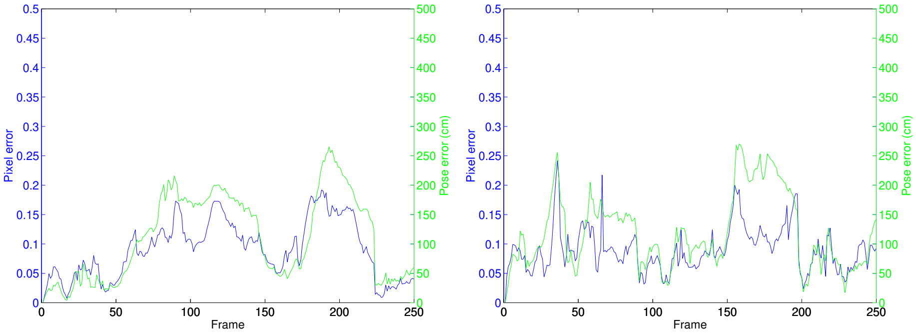
Sequence 11 Sequence 12

Figure E.5: **Test results for the Downhill Simplex Initial Test. All sequences are with the wrist moving**

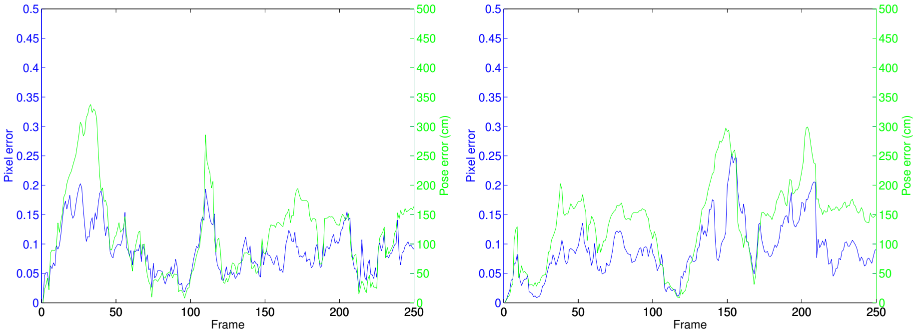
Downhill Simplex - One Restart



Sequence 7 Sequence 8



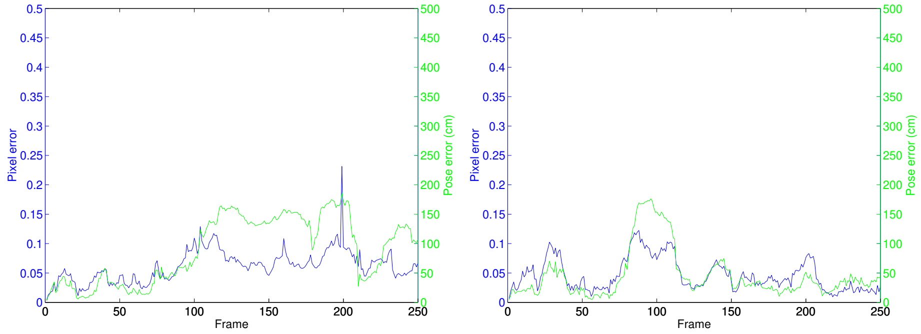
Sequence 9 Sequence 10



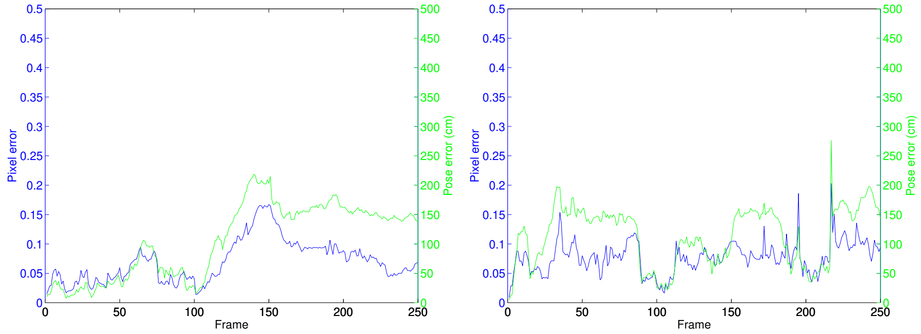
Sequence 11 Sequence 12

Figure E.6: **Test Results for the Downhill Simplex algorithm using one restart. All sequences are with moving wrist.**

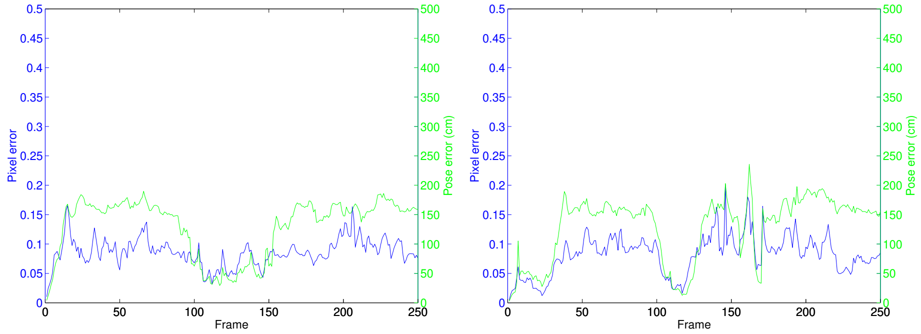
Simulated Annealing - Initial Test



Sequence 7 Sequence 8



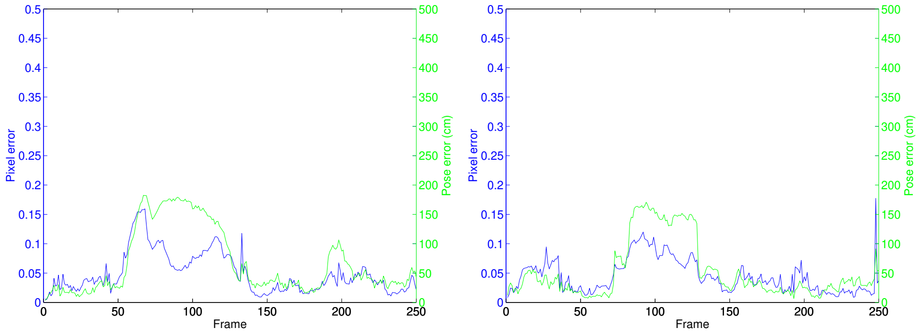
Sequence 9 Sequence 10



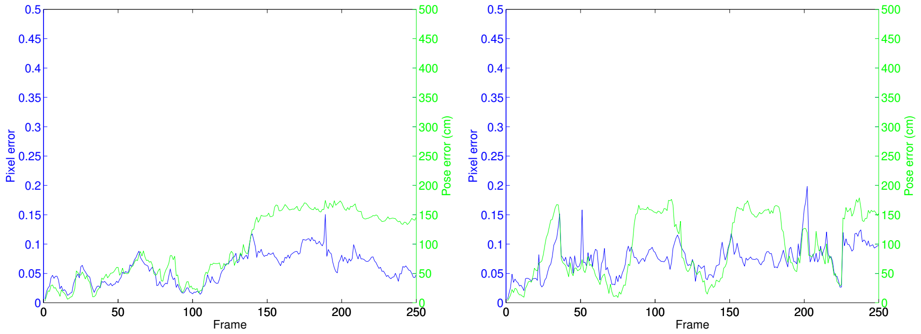
Sequence 11 Sequence 12

Figure E.7: **Test Results for the Simulated Annealing initial tests. All sequences are with moving wrist.**

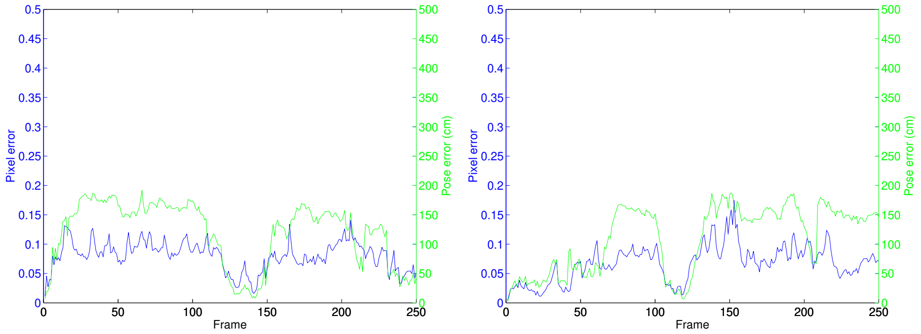
Simulated Annealing - Optimised Test



Sequence 7 Sequence 8



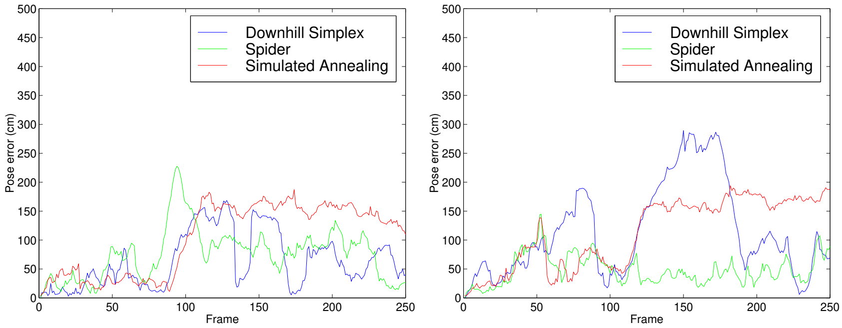
Sequence 9 Sequence 10



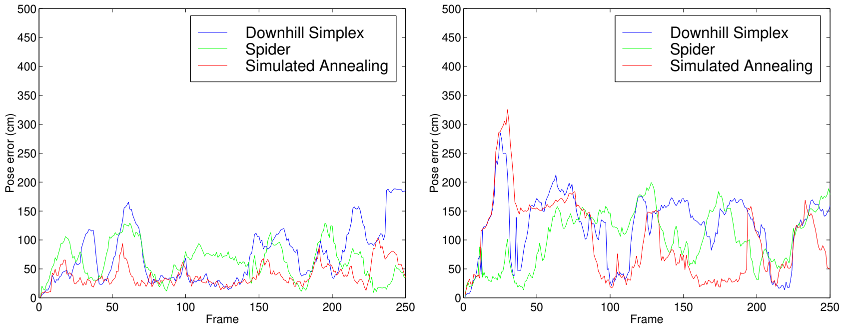
Sequence 11 Sequence 12

Figure E.8: **Test Results for the Simulated Annealing optimised tests. All sequences are with moving wrist.**

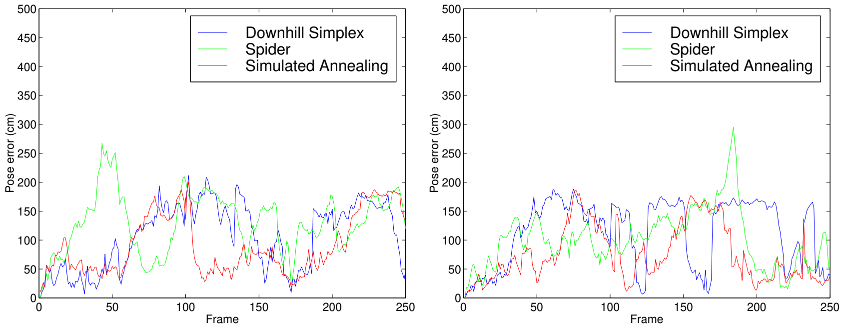
Test Results - Comparative Test



Sequence 1 Sequence 2



Sequence 3 Sequence 4



Sequence 5 Sequence 6

Figure E.9: **Pose error results from the comparative tests. These tests are done with moving wrist.**