Automatic definition of the objective function for model-based hand tracking

Konstantinos Paliouras

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes, Heraklion, GR-70013, Greece

Thesis Advisors: Prof. *Antonis Argyros*

University of Crete
Computer Science Department

Automatic definition of the objective function for model-based hand tracking

Thesis submitted by
Konstantinos Paliouras
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Konstantinos Paliouras

Committee approvals: _____
Antonis Argyros
Professor, Thesis Supervisor

_____
Panos Trahanias
Professor, Committee Member

_____
Manolis Lourakis
Principal Researcher, Committee Member

Departmental approval: _____
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, November 2014

Abstract

Estimating the configuration and pose of articulated objects such as the human body, has high theoretical interest and practical usage. In this work, we are particularly interested in methods for robust and efficient hand tracking and hand pose estimation. Quite recently, model-based approaches have produced very promising results with respect to these problems. The current state of the art method recovers the 3D position, orientation and 20 DOF articulation of a human hand from markerless visual observations obtained by an RGB-D sensor. According to this method, which is used as a baseline in this work, hand pose estimation is formulated as an optimization problem, seeking for the hand model parameters that minimize the discrepancy between the appearance of hypothesized hand configurations and the actual hand observation. The discrepancy between observations and hypotheses is quantified by an objective function that combines several features. The design of this function is of utmost importance to the optimization process as the quality of the final result depends critically on it. At the same time, the design of this function is a complicated process that requires a lot of prior experience with the problem as well as the integration of empirical evidence that is acquired in a time consuming iterative process.

The goal of this work is to define tools and processes that automate the definition of the objective function in such optimization problems. More specifically, we employed regression analysis techniques to define an objective function automatically, i.e., without requiring considerable prior experience to the problem at hand. First, a set of relevant, candidate image features is computed. Then, given data sets with ground truth information, regression analysis is used to combine features from the original set in an objective function that seeks to maximize optimization performance. Regression analysis was performed on datasets of quasi-random hand pose configurations as well as on hand poses obtained through hand tracking.

Extensive experiments study the performance of the proposed approach based on different regression methods, dataset generation strategies and feature selection techniques. A variety of tests has shown that the optimization results obtained by the derived objective functions are comparable to those obtained by using the objective function defined by problem experts. Thus, the process of objective function definition can be automated to a certain extent. Finally, we present topics of future work that could lead, eventually, to a fully automated methodology for determining the objective function in such problems.

Περίληψη

Η εκτίμηση της στάσης ενός αρθρωτού αντικειμένου, όπως το ανθρώπινο σώμα, έχει έντονο θεωρητικό ενδιαφέρον και πρακτική χρήση. Σε αυτή την εργασία, μας ενδιαφέρουν κυρίως οι μέθοδοι για εύρωστη και αποδοτική παρακολούθηση του χεριού και η εκτίμηση της στάσης του. Πρόσφατες προσεγγίσεις που βασίζονται σε 3Δ μοντέλα και μεθόδους βελτιστοποίησης έχουν δώσει πολύ υποσχόμενα αποτελέσματα. Η καλύτερη αυτή τη στιγμή σχετική μέθοδος ανακτά την τρισδιάστατη θέση, τον προσανατολισμό και τους 20 βαθμούς ελευθερίας της αρθρωτής κίνησης ενός ανθρώπινου χεριού από μη-επισημειωμένες (markerless) οπτικές παρατηρήσεις χρησιμοποιώντας έναν αισθητήρα RGB-D (χρώμα, βάθος). Σύμφωνα με αυτή την μέθοδο, η οποία αποτελεί μέθοδο αναφοράς για αυτήν την εργασία, η εκτίμηση της στάσης του χεριού διατυπώνεται ως ένα πρόβλημα βελτιστοποίησης όπου αναζητούνται οι παράμετροι ενός 3Δ μοντέλου του χεριού που ελαχιστοποιούν την ασυμφωνία μεταξύ της απεικόνισης υποθετικών στάσεων του χεριού και των πραγματικών παρατηρήσεών του. Η ασυμφωνία μεταξύ παρατηρήσεων και υποθέσεων ποσοτικοποιείται από μια αντικειμενική συνάρτηση που συνδυάζει πολλά χαρακτηριστικά που υπολογίζονται από τις εικόνες. Ο σχεδιασμός αυτής της συνάρτησης έχει ύψιστη σημασία για την διαδικασία βελτιστοποίησης, καθώς η ποιότητα των τελικών αποτελεσμάτων εξαρτάται άμεσα από αυτή. Ταυτόχρονα, ο σχεδιασμός της συνάρτησης είναι μια πολύπλοκη διαδικασία που απαιτεί προηγούμενη εμπειρία με το πρόβλημα και την συνεκτίμηση εμπειρικών δεδομένων που λαμβάνονται μέσω μιας χρονοβόρας επαναληπτικής διαδικασίας.

Ο σκοπός αυτής της εργασίας είναι να ορίσει εργαλεία και διαδικασίες που να αυτοματοποιούν την διατύπωση της αντικειμενικής συνάρτησης σε τέτοια προβλήματα βελτιστοποίησης. Συγκεκριμένα, χρησιμοποιούμε τεχνικές ανάλυσης παλινδρόμησης (regression analysis) για να ορίσουμε την αντικειμενική συνάρτηση αυτόματα, δηλαδή χωρίς να χρειάζεται μεγάλη προηγούμενη εμπειρία στο πρόβλημα. Στην αρχή υπολογίζεται ένα σύνολο από σχετικά και υποψήφια χαρακτηριστικά εικόνας. Στην συνέχεια εφαρμόζουμε ανάλυση παλινδρόμησης πάνω σε δεδομένα με παραδείγματα ώστε να συνδυαστούν τα χαρακτηριστικά σε μία αντικειμενική συνάρτηση η οποία προσπαθεί να μεγιστοποιήσει την απόδοση της βελτιστοποίησης. Τα δεδομένα προέρχονται από ψευδό-τυχαίες στάσεις χεριών όπως και από στάσεις που ανακτήθηκαν από ιστορικό προηγούμενων παρακολουθήσεων.

Εκτενή πειράματα μελετούν την απόδοση την προτεινόμενης μεθόδου βάση διαφορετικών μεθόδων ανάλυσης παλινδρόμησης, στρατηγικών δημιουργίας δεδομένων και τεχνικών επιλογής χαρακτηριστικών. Μια σειρά από διάφορα τεστ έδειξαν ότι η ακρίβεια της βελτιστοποίησης που επιτυγχάνεται με βάση τις παραγόμενες αντικειμενικές συναρτήσεις είναι συγκρίσιμη με αυτή της αντικειμενικής συνάρτησης που όρισαν οι ειδικοί του προβλήματος. Έτσι, η διαδικασία του ορισμού της αντικειμενικής συνάρτησης μπορεί να αυτοματοποιηθεί ως ένα βαθμό. Τέλος, παρουσιάζουμε θέματα ενδεχόμενης μελλοντικής δουλειάς που μπορεί να οδηγήσει τελικά σε μία πλήρως αυτοματοποιημένη μεθοδολογία για τον ορισμό της αντικειμενικής συνάρτησης σε τέτοια προβλήματα.

Acknowledgements

I would like to express my gratitude to my supervisor Mr. Argyros for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore I would like to thank Nikos, Iasonas, Kostas and Pasxalis for introducing me to the topic as well for the invaluable support on the way. Also, I like to thank my friends Kostas and Tzina for helping to fulfil the goals of this work with their endless comments and support.

Last but not least, I would like to thank my loved ones, Marilena, Brujah, Tiramola and my family who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love.

Contents

1 Introduction **3**
- 1.1 Challenges . . . 4
- 1.2 Existing approaches . . . 5
- 1.3 Optimization problems and objective functions . . . 5
- 1.4 The baseline hand pose estimation method . . . 6
- 1.5 Thesis structure . . . 7

2 Literature review **9**

3 Algorithmic Tools **11**
- 3.1 Optimization . . . 11
  - 3.1.1 Objective Function . . . 12
  - 3.1.2 Particle Swarm Optimization . . . 12
- 3.2 Machine learning . . . 14
  - 3.2.1 Regression . . . 16
  - 3.2.2 Decision Trees . . . 19
  - 3.2.3 Random Forests . . . 20
- 3.3 Baseline method . . . 22
  - 3.3.1 Observation . . . 22
  - 3.3.2 Hand Model . . . 23
  - 3.3.3 Distance between hand poses . . . 25
  - 3.3.4 Hypothesis evaluation (Objective Function) . . . 26
  - 3.3.5 Optimization . . . 27

4 Methodology **29**
- 4.1 Features . . . 29
  - 4.1.1 Sum of depth distances $D_1(.)$ . . . 30
  - 4.1.2 Deviation of depth distances $D_2(.)$ . . . 30
  - 4.1.3 Occupied area $D_3(.)$ . . . 31
  - 4.1.4 Accuracy of the skin map $D_4(.)$ . . . 31
  - 4.1.5 Depth map edges $D_5(.)$ . . . 31
  - 4.1.6 Hand contour $D_6(.)$ . . . 32
- 4.2 Dataset . . . 32

4.2.1 Sampling with low-discrepancy sequence 34
4.2.2 Sampling biased to optimization 35
4.3 Regression Model 36

5 Experimental Evaluation **37**
5.1 Features 37
5.2 Evaluation on synthetic data 41
5.2.1 Regression models performance 42
5.2.2 Tracking Performance 43

6 Discussion **47**
6.1 Future Work 49

List of Figures

3.1 Objective functions for evaluation optimization algorithms 13
3.2 A polynomial function fitted on a dataset 16
3.3 Performance of different functions on a dataset 18
3.4 A decision tree for modeling the task of *playing tennis*. 19
3.5 The bagging method on a dataset of ozone and temperature 21
3.6 Overview of baseline method pipeline. 23
3.7 Anatomy and kinematic modeling of the human hand 24
3.8 27-parameter representation of a hand pose. 24
3.9 The hand geometry model used by the baseline method. 25

4.1 Proposed changes on the *baseline method* pipeline 30
4.2 Steps to compare the depth map edges 33
4.3 The distribution of samples in datasets 34

5.1 The response histogram of D_i functions 38
5.2 Profile of feature functions $D_i(.)$ and baseline discrepancy function $D(.)$ on the Z axis of hand global position. The observation model that was used has exactly the same configuration as the tested hypothesis. 39
5.3 Profile of feature functions $D_i(.)$ and baseline discrepancy function $D(.)$ on the Z axis of hand global position. The observation model that was used has a different configuration than the tested hypothesis. 40
5.4 Model performance per feature 42
5.5 Tracking performance tested on the ground truth "cooljason" dataset. Models were trained on dataset generated by 4096 quasi-random poses. 44
5.6 Tracking performance tested on the ground truth "cooljason" dataset. Models were trained on dataset generated on tracking logs using the objective function. 45

Chapter 1

Introduction

Automatic capture and analysis of human motion has numerous potential applications and high scientific interest. Long standing unresolved *human-computer interaction* problems could be solved by directly using the human body for interacting with computers, especially for the case of estimating the configuration and pose of a human hand. Bypassing intermediate input devices like keyboard or mouse, can result in a more natural, efficient and high degree of freedom (DOF) control device. Some of the potential applications are: improved *HCI* experience, sign language decoder, tele-surgery, robot learning movements, controlled human body training etc. As an example, a designer could sculpt a 3D model using *Computer Aided Design* software by moving his hand, virtually touching the surface of his creation. In the field of medicine, a doctor could instruct a robot arm to replicate the movement of his hand, thus performing a remote surgery. Or a patient recovering from a stroke could be observed in his physiotherapy process in order to get accurate feedback for improvement. Another example could be a device that translates sign language to speech. Finally, even everyday user experience with computers could be greatly enriched. Imagine a person entering his living room and controling a music play-list, sound volume or any other desired action, by waving his hands.

In the past, the only technology that satisfied the advanced requirements of hand-based input for *HCI* is glove-based sensing. It is only in recent years that *computer vision* has been able to provide hand-tracking methods that could satisfy these requirements. Still though, these methods usually demand many computational resources and specialized hardware, preventing their usage in areas other than laboratories. Computer vision (CV) has the potential to provide a natural, non-contact solution, that could be easy to use, without requiring long calibration or setup procedures and at the same time being performed by modest hardware, like a laptop with a camera or even a smartphone.

The problem of vision-based hand tracking has two major categories, that of *hand pose tracking* and *hand pose estimation*. In the former case, the problem is to find a sequence of continuous hand poses. The latter requires estimation of a hand pose from a single frame, which is a more challenging problem than *hand*

pose tracking because it cannot exploit temporal continuity. A significant amount of literature has been devoted to the problem of vision-based hand tracking [1, 2]. Different setups for visual input have been tested: single camera, stereo cameras, multi cameras, structured lighting. One of the most effective methods to track articulated objects like human hands, is through placing visual markers on specific points of the observed skeleton [3]. Although this method produces results with high accuracy, it demands interference with the subject, which adds an undesired overhead of placing the markers and in some cases it is not even possible. Latest works [4, 5] have showed that the marker-less approach can be competitive in terms of accuracy and performance, and eventually eliminate the need for markers.

1.1 Challenges

The problem of hand-tracking introduces a lot of challenges that must be faced. Erol et al. [2], in their survey, have outlined the major difficulties as:

- *Dimensionality*: The human hand is an articulated object with more than 20 DOF. Studies have shown that even with dimensionality reduction there cannot be less than 6 DOF [2]. With the addition of global position and orientation this results in a considerable amount of parameters that must be estimated for every observed hand pose.

- *Self-occlusions*: Since the hand is an articulated object with many DOF, its projected shape can vary a lot depending on the pose of the hand. This can introduce a lot of self-occlusions from parts of the hand that hide portions of itself in a non predictable way.

- *Computational budget*: The problem of hand-tracking is computationally expensive mainly because of the high dimensionality of the problem. Even for a single image sequence, a real-time CV system needs to process a huge amount of data. With the current hardware technology, some existing algorithms require expensive, dedicated hardware, and possibly parallel processing capabilities to operate in real-time.

- *Uncontrolled environment*: Even the problem of locating a rigid object on arbitrary backgrounds is almost always a challenging problem. Tracking a hand is even worst as the skin color is a not a solid and the self-occlusions produce more variance on the color of the hand surface. For widespread usage, a hand tracking system is expected to work under non-restricted environments.

- *Rapid hand motion*: The hand has very fast motion capabilities with a speed reaching up to $5m/s$ for translation and $300°/s$ for wrist rotation. For the problem of hand-tracking, high speed motion and low sampling rates introduce extra difficulties, as the consecutive frames become more and more uncorrelated. Increasing the sampling rate can mitigate this problem, but this

is restricted by the current hardware technology and algorithmic complexity which currently cannot achieve more than $30Hz$ of tracking speed.

1.2 Existing approaches

Many of solutions have been proposed for the problem of the hand-tracking [2, 1]. These solutions can be divided into two main categories, the *appearance-based* and the *model-based* approach.

The *appearance-based* approach tries to solve the problem using a map between the feature space and the solution space. This map is usually constructed with offline training of a prediction model, which can be either a regression or classifier model. The regression models are used to predict the exact configuration of the hand in solution space, while the classifier models usually try to predict the gesture of the observed hand pose.

On the other hand *model-based* approaches seek the solution directly on the solution space. Usually this involves making multiple hypotheses in solution space that are evaluated in feature space by comparing the appearance of the observed hand and the estimated appearance of the hypotheses. To estimate the appearance of the hypotheses, a shape and kinematic model of the articulated object is needed by the system and this is the reason they are called *model-based*.

Both of the approaches have cons and pros and none of them provides a universal solution to the problem. Typically the *appearance-based* approaches are very fast on prediction but demand offline training on datasets that cover all cases extensively (different hand poses, illumination variances, self-occlusions, etc). The construction of this dataset is very important and the accuracy of these methods depends on the diversity of cases in this dataset. On the other hand, *model-based* approaches do not depend on prior training, thus making them unbiased to any dataset, and the accuracy is theoretically constrained only by the computational resources. However for the same level of accuracy, *model-based* methods require a larger computational budget than the *appearance-based* approach, because they seek the solution directly in the solution space, which is multi-dimensional and represents each DOF of the articulated object (in our case the human hand).

1.3 Optimization problems and objective functions

Model-based approaches typically formulate the problem as an optimization problem, where an objective function is used to evaluate hypotheses directly on the solution space. The goal of any optimization problem is to find a candidate as close as possible to the solution with the minimum number of steps (Section 3.1). For this reason, the objective function is of the utmost importance as the accuracy of the prediction and steps needed to approach the solution depends heavily on its response.

The construction of the objective function demands prior-experience in the problem and good knowledge of optimization strategies. Usually a lot of research is done to carefully craft a function that reflects the problem, and a lot of time is spent to improve this by following a trial and error methodology. An improved objective function that has a better correlation with the problem can reduce the optimization steps which with *model-based* hand-tracking, are very expensive to compute. This can result in reduced computational resources and/or improved accuracy in the same consumed time.

In this thesis we will see how to construct an objective function following a robust methodology that uses machine learning tools for this purpose (Chapter 4). This methodology does not demand deep prior-knowledge of the problem or heavy mathematical skills. The key concept is to use regression analysis which provides established and deterministic algorithms to construct a function with the desired response behaviour. The function is estimated by modelling the response of a pre-defined ideal objective function that is sampled on a synthetic dataset. This dataset is built by sampling randomly the multi-dimensional solution space and for each pair of object poses, the visual discrepancies and the response of the ideal objective function are measured and recorded. This approach has the advantage that the characteristics of the function can be defined before constructing it, tools to approximate this function are readily available and so are evaluation tools needed to estimate the accuracy of this process.

1.4 The baseline hand pose estimation method

For the purposes of studying and evaluating an alternative method for constructing the objective function of model-based hand-trackers, we used the work of Oikonomidis et al. ("Efficient Model-based 3D Tracking of Hand Articulations using Kinect" 2011) [4]. In their work these authors present a model-based approach for recovering and tracking the full configuration of a human hand with 27 DOF. The problem is formulated as an optimization problem seeking the best solution in the configuration space of 27 dimensions. The methodology can be divided into 3 main components.

- *Observation*: This component is responsible for acquiring input from the sensor and pre-processing data. At the pre-process stage, it detects and isolates all the areas with skin color. [6]

- *Hypothesis evaluation*: For each hypothesis made in configuration space the distance against the observed hand pose must be evaluated. This component is responsible for quantifying this distance by considering the visual discrepancies in feature space. To achieve this, each hypothesis is projected in feature space using rendering techniques to estimate its appearance. For this, a hand shape and kinematic model is supplied to the rendering pipeline.

- *Optimization*: Finally, a component performs optimization on solution space in order to find that hypothesis with the minimum distance to the observed hand. The PSO [7] optimization algorithm was chosen for this task and the hypothesis evaluation component is used to score all candidates during the optimization process.

1.5 Thesis structure

This thesis is organized as follows. In Chapter 2 the related works on hand pose estimation are presented. Chapter 3 provides the required theoretical background on *Optimization Theory*, *Regression Analysis* and a detailed presentation of the employed baseline method [4]. The proposed methodology is presented in detail in Chapter 4. Experimental results of this methodology are presented and discussed in Chapter 5. Finally, Chapter 6 concludes this thesis with a general discussion regarding the gained experience, conclusions, the limitations of the method and presents issues for future investigation.

Chapter 2

Literature review

A lot of published work exists for recovering the full 3D configuration of articulated objects, especially the human body or parts of it like hands, head etc. Moeslund et al. [1] have made an extensive survey on vision-based human body capture and analysis. Hand tracking and body tracking problems share many similarities, like hierarchical tree structure, problem dimensionality and complexity, occlusions and anatomic constraints. Erol et al. [2] present a variety of methods for hand pose estimation or tracking. Depending on the output of these methods they are divided in partial and full pose estimation methods. Further categorization is between appearance-based and model-based methods. Typically appearance-based methods [5, 8, 9] solve the problem by modelling the transition from feature space to configuration space. This is achieved either by using analytical expressions or by constructing a training dataset that is later considered on prediction to either find the closest example or perform regression. Appearance-based methods perform fast on prediction and are suitable for gesture recognition. A common problem though, for these methods, is that they usually demand large training dataset that must be specialized to the application environment. To compensate for potential bias of the training dataset, Shotton et al. [5], in their method, generated large a realistic synthetic dataset, sampling all the possible feature space, permitting good generalization.

On the other hand, model-based methods [10, 11, 12, 13] search directly for the solution in the configuration space and each hypothesis is evaluated in feature space, usually using rendering techniques to estimate the appearance of the hypothesis. For each hypothesis in configuration space, the appearance must be simulated. An error function is considered to evaluate the visual discrepancies, which usually demands high computational resources. Although, in these methods, prediction is a computationally complex problem, model-based methods do not need offline training, making them easier to employ, unbiased from any specific application problem and easily extensible.

Summing up, appearance-based methods employ statistical analysis or machine learning to learn the correlation from feature space to configuration space, while

model-based methods usually try to optimize a hypothesis in configuration space that has the least visual discrepancies in feature space.

In all optimization problems, there are two major components, the error/objective function and the optimization algorithm. Different options exist for both components but the performance is dependent on the correct combination of the two. As an example, Martin de La Gorce et el. [12] used a quasi-newton optimization algorithm and a hand made error function that considers textures and shading of the model which proved to perform well on the problem. Oikonomidis et al. [10] used the *Particle Swarm Optimization* algorithm and they also crafted a special error function for evaluating the visual discrepancies, taking into account the skin color and depth information provided by a RGB-D sensor or from a multi-camera setup [4]. The optimization strategy is usually difficult to find and consumes a lot of researching time, usually by trial and error. For this reason, work is done to provide other ways for selecting the strategy. As a result the typical classification of appearance-based and model-based pose estimation or tracking is not always clear.

Recent works have tried to combine the advantages of both methods basically using machine learning. Xuehan Xiong et al. [14] mentioned the effectiveness of 2^{nd} order descendent methods, as well as the difficulty in using them in computer vision, mainly because it is hard to analytically differentiate the objective function. They proposed instead, a *supervised descent method* that models the behaviour of an objective function in offline training. At prediction time, this method consults the learned model to estimate the descent direction, allowing the usage of any non linear objective function. Chen Cao et al. [15] have presented an appearance-based method for tracking facial expression. In their work, they have used regression analysis for modelling the correlation between the feature space and the 3D positions of a face shape model. The correlation model is built at the initialization stage, where the user is instructed to do a small set of predefined facial expressions. At this point a model-based procedure registers a generic parametric face shape model and its 3D configuration. The output of this procedure is then used to train the regression model that will be used later for tracking. In this way, the tracker is computationally efficient without demanding off-line training or a generation of complex datasets, thanks to this hybrid methodology.

Chapter 3

Algorithmic Tools

The proposed method of this thesis depends heavily on machine learning and solving problems using mathematical optimization methods. This chapter will provide an introduction to these methods. First we will see how optimization methods work and how they can be used to solve complex problems (Section 3.1). This will lead us to a description of objective functions and their key role in the performance of optimization methods. Then in Section 3.2 there will be an introduction to machine learning and how it can be used to transform a function. Finally the model-based hand-pose estimation method presented by Oikonomidis et al. [10] will be described (Section 3.3). This is the baseline method that is used to implement and evaluate the proposed methodology of this thesis.

3.1 Optimization

Mathematical optimization (alternatively, optimization or mathematical programming) is the selection of the best element from some set of available alternatives. In the simplest case, an optimization problem consists of an algorithm that will systematically try to estimate the input values of a real function that will give the maximum or minimum output. [16]. The real function that the optimization algorithm is using is called the *objective function*. [17]

There is no unique optimization method that can solve all kinds of problems. Some methods demand a derivative of the objective function and are usually referred to as gradient-based algorithms [18]. While other methods do not need a derivative and are referred to as derivative-free algorithms. Some use a direct solution approach while others use iterative and evolutionary approaches to improve the candidate solutions. Depending on the nature of the objective function, numerous optimization methods have been developed [19, 16, 20].

In computer vision, there are many examples [21, 10] that formulate a vision problem as an optimization problem. Using optimization methods, especially heuristic algorithms, it becomes possible to overcome the computational complexity needed by a direct solution. Apart from heuristic algorithms there are also

approximation algorithms, which consume a significantly smaller portion of computational resources. While the solution is not guaranteed to be the global best, usually it is a sufficient approximation to the solution.

3.1.1 Objective Function

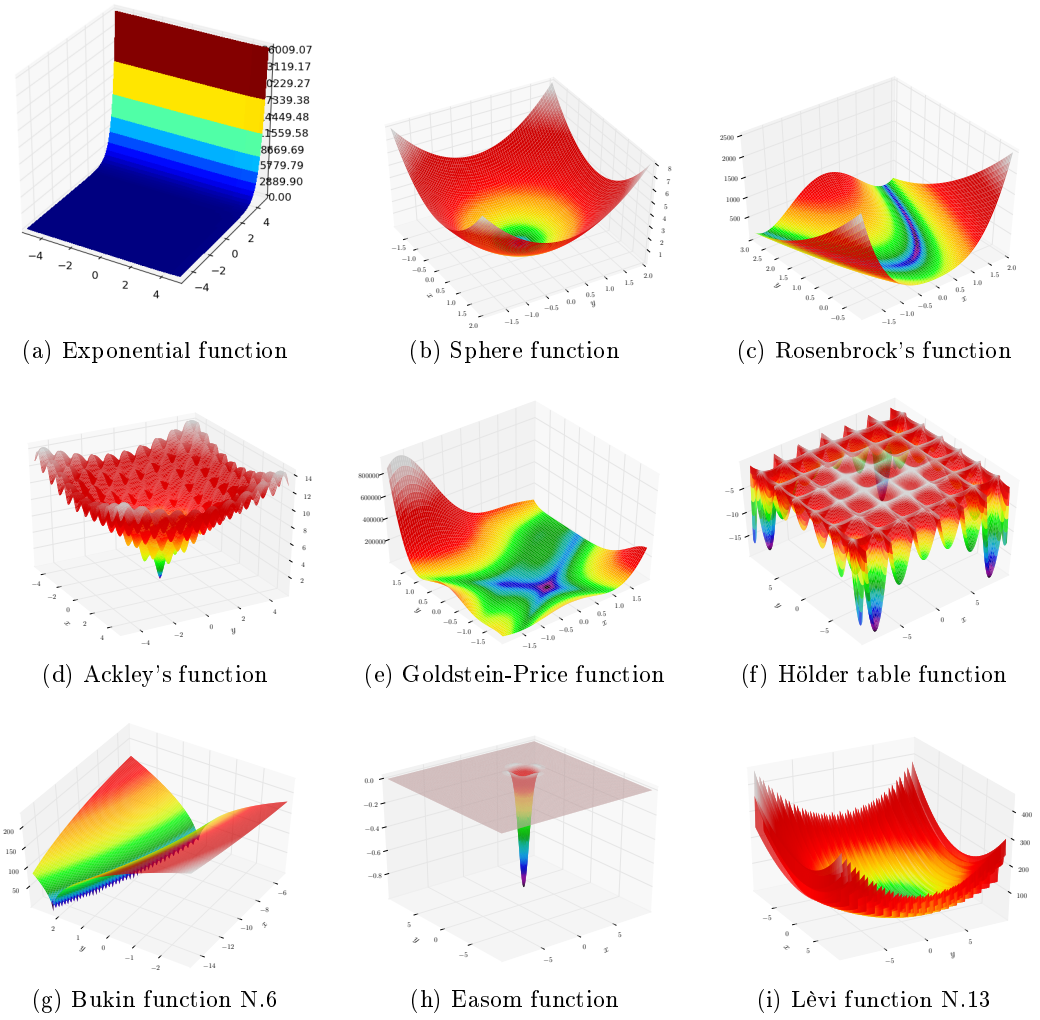
In mathematical optimization, an error (loss) function or cost function is a function that maps an event to a real number, intuitively representing some "cost" associated with the event. An objective function is either an error function or its negative, in which case it is to be maximized [17]. Usually the objective function is a loss function, meaning that an optimization problem seeks to minimize it.

An objective function defines the space where the optimization algorithm will search for the global minimum. In the extreme case where the optimization algorithm makes no assumption about this space, it has to check every point in this space to assure which one is the global minimum. In the majority of real problems this is not feasible as the size of the space does not permit exhaustive searching. The morphology of this space and the characteristics of the objective function can be exploited by optimization algorithms to move faster toward the solution. For example, if an objective function provides a derivative, it can be used to evaluate the direction of the next local minimum [18].

In Figure 3.1, examples of objective functions that can jeopardize the assumptions made by optimization algorithms are shown. Functions that do not have smooth surfaces (i,d) or have multiple local minima (d,f) or have big flat surfaces (e) are not easy to optimize. Thus when formulating an optimization problem it is essential to provide an objective function that has exploitable characteristics. This will permit better performance of the optimization solution in terms of speed and accuracy.

3.1.2 Particle Swarm Optimization

Some linear or non-linear problems can be solved by evaluating the first derivative of the objective function, however this is not always possible. In 1995 Kennedy and Eberhart presented a new stochastic, evolutionary algorithm that is related to genetic algorithms and can optimize derivative-free, non-linear objective functions [7, 24] using a few computational resources. This algorithm was named Particle Swarm Optimization because it was inspired by the social behavior of moving organisms, like bird flocks or fish schools, and tries to simulate their social interaction. In a nutshell, Particle Swarm Optimization or PSO optimizes the problem by having a population of solution candidates (called particles) and for each iteration (called generation) moving these particles around in the search space according to a simple mathematical formula. This formula defines that each particle is attracted by its local best known particle but also by the global best known position in the search space.[25] This trivial algorithm performs well in non-smooth surfaces and has good scalability with respect to the problem's dimensions.



(a) Exponential function (b) Sphere function (c) Rosenbrock's function

(d) Ackley's function (e) Goldstein-Price function (f) Hölder table function

(g) Bukin function N.6 (h) Easom function (i) Lèvi function N.13

Figure 3.1: Objective functions that have been used for evaluating optimization algorithms. The idiosyncrasy of these functions, shows the complexity of optimization problem depending on the nature of the objective function. [22, 23]

In the past years, researches worked on evolving and adapting the idea of particle swarm optimization in different problems. The result of their work lead to multiple PSO variants [26] that have better performance on specific problems. Though the general idea of the algorithm remains the same, the policy of interaction and the way the particles behave depends on the variant. Some variants define dynamic neighborhood topologies [27], others work on discrete parameter space [28] and some can solve dynamic optimization problems [29]. In this thesis, the simplest form of PSO is used, which is called the *canonical PSO*.

In the canonical PSO, every particle stores its current position in a vector x_k and its current velocity in a vector v_k. A vector P_k stores the position at which the particle achieved, up to the current generation k, the best score of the objective function. The swarm as a whole, stores in a vector G_k the best position encountered across all particles of the swarm (global optimum). Every particle is aware of the global optimum and taking into consideration its local optimum, it updates the velocity and position vector in every generation k as follows:

$$v_{k+1} = w(v_k + c_1 r_1(P_k - x_k) + c_2 r_2(G_k - x_k)) \tag{3.1}$$

and

$$x_{k+1} = x_k + v_{k+1} \tag{3.2}$$

where w is a constant constriction factor [30], c_1 is called the *cognitive component*, c_2 is termed the social component and r_1 , r_2 are random samples of a uniform distribution in the range [0...1]. Finally, $c_1 + c_2 > 4$ must hold [30].

For each dimension in search space, a boundary constraint is defined. In the first iteration particles are spread all over the search space according to a randomization function and their velocities are set to zero. If during an update of a particle, a velocity component forces the particle to move outside the boundaries, a handling policy is required. A variety of alternative policies have been proposed [31]. In the thesis the *nearest point* method is used, which defines that if velocity forces a particle to move in position P_o outside boundaries, the particle will be moved to a new position P_b in such way that $|P_o - P_b|$ is minimum.

3.2 Machine learning

Machine learning is a branch of artificial intelligence that consists of constructing and studying systems that can learn from data. For example, a machine learning system could be trained on email messages to learn to distinguish between spam and non-spam messages. After learning, it can then be used to classify new email messages into spam and non-spam folders.[32]. Tom M. Mitchell provided a widely quoted and more formal definition of machine learning which is [33]:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its per-

formance at tasks in T, as measured by P, improves with experience E"

In general any machine learning system can be seen as a problem of how to represent the knowledge that can be extracted from input training data and how to generalize from this information that is accumulated. Generalization is the ability to perform accurately on new, unseen examples/tasks after having experienced a learning data set.

Multiple algorithms have been developed to address different problems. A common categorization is based on the desired outcome of the algorithm or the type of input available during training the machine. The three major types of machine learning are:

- *Supervised Learning.* Algorithms are trained on labeled examples, where each example is a pair consisting of an input object (typically a vector) and a desired output value.
- *Unsupervised Learning.* Algorithms are trained on unlabeled examples. The goal here is to find hidden structure in unlabeled data.
- *Reinforcement Learning* is concerned with how intelligent agents ought to act in an environment to maximize some notion of reward. The agent executes actions which cause the observable state of the environment to change. Through a sequence of actions, the agent attempts to gather knowledge about how the environment responds to its actions, and attempts to synthesize a sequence of actions that maximizes a cumulative reward.

Another common categorization is based on the nature of the problem that machine learning is required to model. The three most common categories are presented below, though many more exist:

- *Classification* where the problem is to identify to which of a set of categories a new observation belongs.
- *Regression* solves the problem of estimating a relationship among some variables.
- *Clustering* is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups.

In the proposed methodology of this thesis, we will use supervised machine learning algorithms that solve the problem of *regression*. In Section 3.2.1 we will see how regression analysis works and how it can be used to predict the value of a variable. In Section 3.2.2 there will be an introduction to *Decision Tree* algorithms and their usage in regression problems. Finally we will see an ensemble technique called *Random Forests* (Section 3.2.3) that improves the performance of *Decision Trees*.

3.2.1 Regression

Regression analysis is a statistical process for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a response (dependent) variable and one or more explanatory (independent) variables. The goal of this process, is a function, called the *regression function*. This function will describe the relationship between the response variable Y and the explanatory variable(s) X, i.e. given X as arguments of the regression function it will estimate the value of Y [34]. Historically, the earliest form of regression was the method of least squares, which was published by Legendre in 1805 [35] and Gauss in 1809 [36].

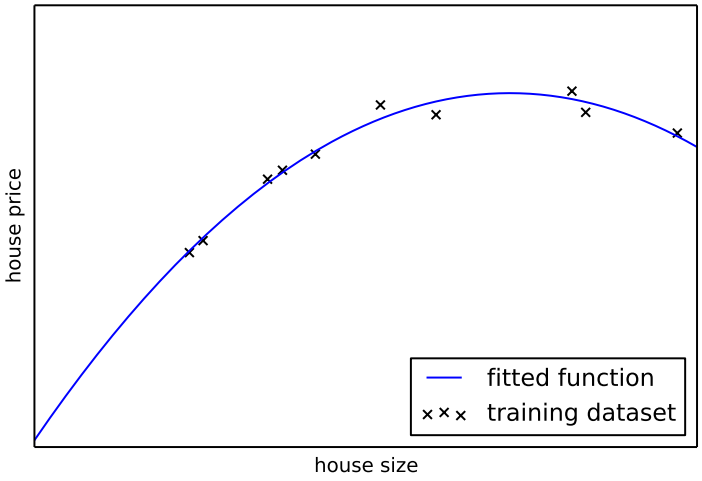


Figure 3.2: A dataset with examples of house sizes and their market price. On this dataset a 2^{nd} degree polynomial function was fitted.

For example, in the problem of predicting the price of a house given its size, we need to collect a training dataset of examples between house sizes and prices and then construct a regression function that can model this correlation with the highest accuracy. This function can be used to estimate the price of a house based on its size, even for values that were not included in the training dataset. In Figure 3.2 we can see a dataset and a 2^{nd} degree polynomial function that was fitted on these data. This function models the correlation of house size with house price and can estimate the price even for houses for which we do not have examples.

The simplest form of regression is *simple linear regression* which is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of N points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible [37, 38]. A simple linear function with only one explanatory variable

can be formulated as:

$$y = w^T \phi \tag{3.3}$$

where $w = (\alpha, \beta)^T$ and $\phi = (1, x)^T$. The parameter vector of the model is called w and is the vector of constant values that is calculated on training over a given training dataset, while x is the explanatory variable. It should be noted, that instead of using the linear formula $y = \alpha x + \beta$, it is preferred to increase the dimension of the X vector by one with the constant value 1. This mathematical trick, permits the calculation of all model parameters using matrix operations.

For a given w parameter vector we can calculate the training error using the least squared error:

$$E(w) = \sum_{\mu=1}^{N} (y^\mu - w^T \phi^\mu)^2 \tag{3.4}$$

where N, y, ϕ are the number of samples, the response variable and the explanatory variables respectively in the training dataset. With the cost function we can formulate the problem of finding the best parameter vector of linear model as minimization problem over the $E(w)$

$$w_{best} = arg \min_{w} E(w) \tag{3.5}$$

Using the w_{best} and the equation of the linear model (Eq. 3.3) we can then estimate the value of the response variable y for any given explanatory variable x.

The idea behind simple linear regression can be generalized for different cost functions. Thus we can say that the problem of fitting any linear model on a given training dataset, is the procedure of finding the w parameter vector that has the minimum training error based on a cost function $E(x)$. Regression methods are not used only for linear models but continue to be an area of active research. In recent decades, new methods have been developed for robust regression, regression involving correlated responses such as time series and growth curves, regression in which the predictor or response variables are curves, images, graphs, or other complex data objects [34].

Model performance, Over-fitting, and Under-fitting

A model is trained by minimizing the training error (Eq. 3.5) with the dataset, which will increase the accuracy of the model on the training data. But the training accuracy does not reflect the prediction accuracy on unseen data. So usually another dataset is used to calculate the performance of a model. There are multiple ways of improving the prediction accuracy of models.

- Increase the number of training points N. This might provide a training set with more coverage, and lead to better performance.
- Make model more flexible by changing its parameter, or choose a more flexible model. This might allow us to more closely fit the training data, and lead to

a better result.

- Add more features. If we were to, for example, perform a linear regression using x, x^2, $\sqrt{x}$, x^{-1}, or other functions, we might find a functional form which can better be mapped to the value of y.

The best course of action varies from situation to situation and from problem to problem and is a matter of experience and methodology.

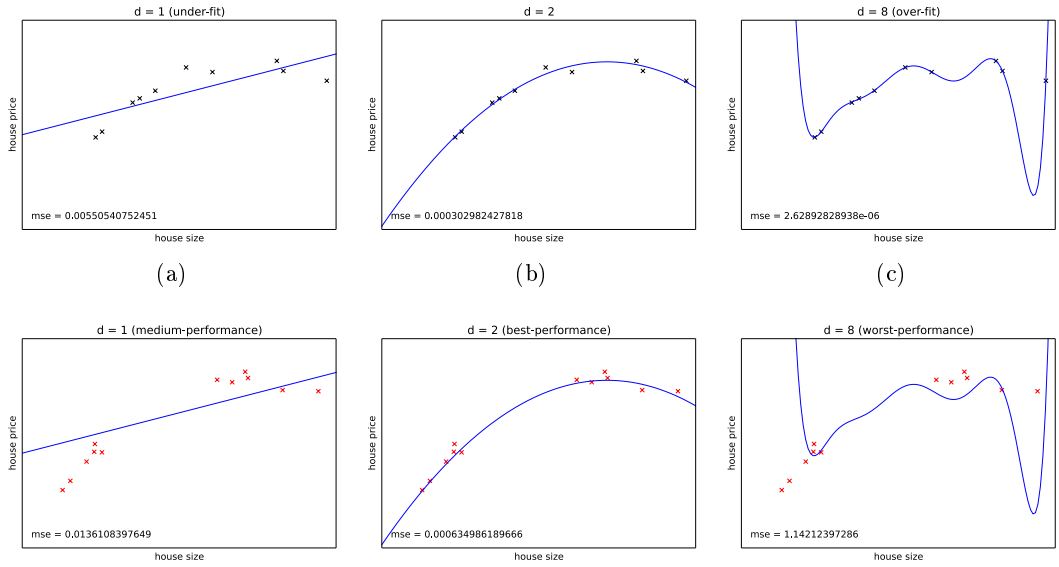


Figure 3.3: Polynomial functions of different degree are fitted on a dataset of house size versus price. In the first row the functions are fitted on the training dataset while in the second row the prediction accuracy is demonstrated over an evaluation dataset. The *mean squared error* (mse) was used to measure the divergence of functions from the dataset.

Improving the accuracy of a model over a given evaluation dataset is not enough to ensure that the model has a good performance. For example, in Figure 3.3 a linear model (1^{st} degree) was used to model the correlation between price and size of houses. Regardless of the choice of error function for fitting the linear model, the analysist has made an assumption that these two variables are linearly correlated, which does not hold. Thus, the model fits the training data poorly, which results in great loss of accuracy; this is a case of *under-fitting*. On the other hand, an 8^{th} degree polynomial was used, which permits to build a model with very low training error, but this is not the case for the evaluation dataset where the model has poor performance. This model has the problem of *over-fitting* on the training set and cannot be generalized well. In other words it is biased over the training set. In the same example, a 2^{nd} degree polynomial was used and although it didn't have the

best accuracy on the training dataset, it did have very good performance on the evaluation dataset. This model does not suffer from under-fitting or over-fitting. A lot of research is done for the estimation of model performance and various methods have been developed for this purpose, such as cross-validation [39] and learning curves.

3.2.2 Decision Trees

Decision tree learning is a method for approximating (preferably discrete-valued) target functions, in which the learned function is represented by a decision tree. Learned trees can also be re-represented as sets of if-then rules to improve human readability. These learning methods are among the most popular of inductive inference algorithms and have been successfully applied to a broad range of tasks [33].

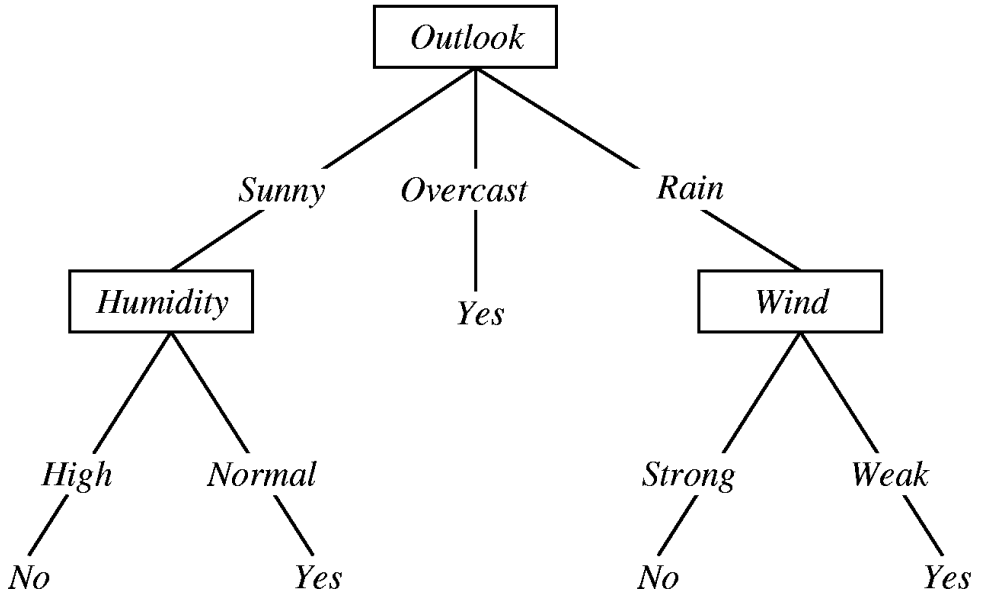


Figure 3.4: A decision tree for modeling the task of *playing tennis*. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf (in this case *Yes* or *No*) [33].

For example, if we wanted to estimate the possibility of playing tennis based on some forecasting variables, we could use the tree shown in Figure 3.4. In this model, the explanatory variables are *outlook*, *humidity*, *wind* and the response discrete-value (*Yes* or *No*) variable is *play*. In order to use this model for prediction, we walk through the tree, starting from the root node and at each node we choose the path based on the embedded rule and the values of the explanatory variables. At some point we will reach a leaf, which holds the value of the response variable for this example.

Prediction using decision trees is trivial, and very fast. However training is not an easy task, as decision trees are prone to over-fitting. A decision tree learning algorithm has to walk through the training set and create a tree, that can represent this data and generalize well on unseen data.

The most notable algorithms are ID3, C4.5 developed by Ross Quinlan [40, 41] and CART (Classification And Regression Tree) by Breiman et al. [42]. Although these algorithms were developed independently they share the same principals. The basic principal is to split the tree, based on the feature that has the bigger information entropy and continue this process until no further gain can be made, or some pre-set stopping conditions are met.

As mentioned before, the process of creating a decision tree is prone to over-fit on the training set. To overcome this problem, learning algorithms try to keep the trees small in order for them to be generalized well. A common approach to this problem is to create the tree and later apply a pruning algorithm that will detect and remove sections of the tree that provide little influence on prediction accuracy. Another approach is to put some criterion in the construction process of the tree, like the maximum height.

3.2.3 Random Forests

In Section 3.2.2 we saw that decision trees tend to over-fit on the training dataset and usually a pruning technique is applied to reduce model bias and improve prediction accuracy. However, these techniques sacrifice the accuracy on the training dataset for the sake of generalization. Tin Kam Ho (1995) showed a method of how to improve prediction accuracy of decision trees without sacrificing the accuracy on the training dataset [43]. The basic idea was that someone can use multiple trees that have been trained to random feature subspaces. These trees, called *Random Decision Trees*, were used to form an ensemble method for obtaining a better predictivity performance than could be obtained from any of the individual trees. Later, in 2001 Breiman combined the idea of bagging and the random selection of features in order to construct a collection of decision trees with controlled variation, i.e. *Random Forests* [44].

Bagging is a meta-algorithm designed to improve the stability and accuracy of machine learning algorithms. Although it is usually applied to decision tree methods, it can be used with any type of method. The methodology consists of splitting the training dataset D into m new training sets D_i, each of size n', by sampling from D uniformly and with replacement. Then m models are fitted at each D_i dataset and combined by averaging the output (for regression) or voting (for classification). For example, in Figure 3.5 a dataset is shown in which the bagging method was used. In this example the requirement is to fit a smoothed model over the given dataset. Instead of building one model over the complete dataset, 100 samples of the data were drawn. For each sample a smoothed model was built. In Figure 3.5 the 10 best fitted models are marked with grey lines. The final smoothed model, which is shown with the red line, was the average model over

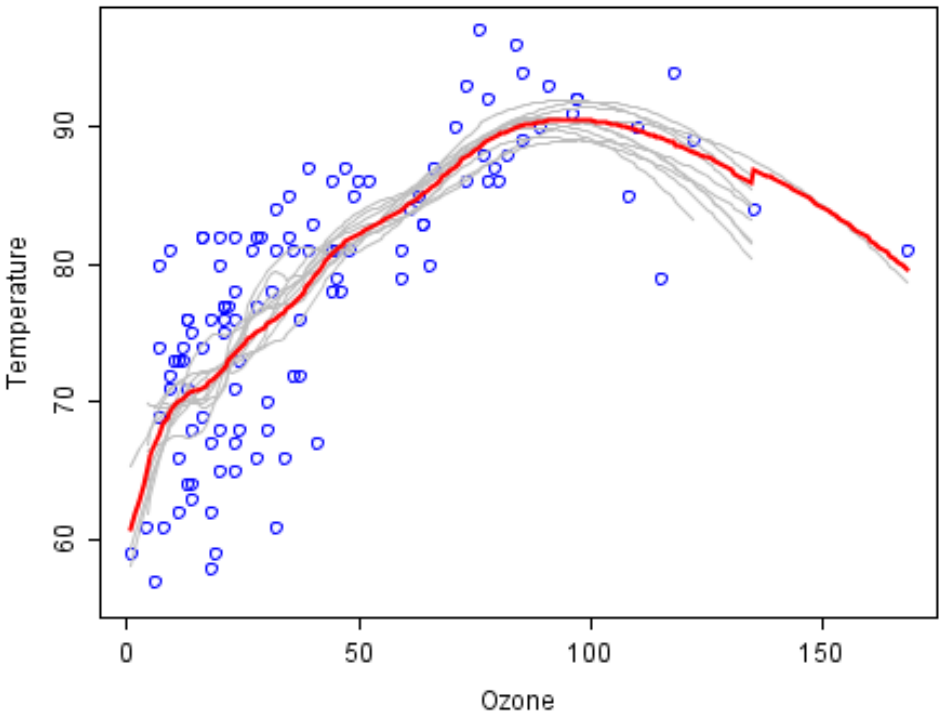


Figure 3.5: A dataset of ozone and temperature is shown in the graph where the bagging method was used to fit a smoothed predictor. The grey lines show 10 smoothed fits on samples of the dataset while the red line is the bagged predictor (average of all) [45].

the 100 built models. This averaged model is expected to have a higher predictivity performance than any of the individual models or by using only one model over the whole dataset.

Random forests use the *bagging* meta-algorithm to create a set of decision trees over the given dataset. These trees are combined to make a prediction over unseen data. As a result, *random forests* can fit to complex spaces with improved performance when compared to using a single decision tree.

3.3 Baseline method

The proposed methodology of this thesis will be evaluated over an existing working method. This method, will be the *baseline method* and any proposed change will be compared against its initial performance. The work of Oikonomidis et al. (*"Efficient Model-based 3D Tracking of Hand Articulations using Kinect"* 2011) [10] was chosen for this task. In their work they presented a model-based approach to the problem of recovering and tracking the 3D position, orientation and full articulation of a human hand from markeless visual observations obtained by a Kinect sensor. They treat this as an optimization problem, seeking for the set of hand parameters in 3D space that have the minimum discrepancy with those of the observed hand. In Figure 3.6 the multiple components of the methodology are shown and how they communicate to solve the problem. This method uses a variant of the Particle Swarm Optimization algorithm, that was described in Section 3.1.2, to optimize the 27 hand model parameters around the solution of the previous frame. A hand model (Section 3.3.2) and a renderer are supplied and are used to render arbitrary hypothesized poses based on a kinematic model. Each hypothesis is evaluated against the observed hand pose by a human-invented error function (Section 3.3.4) that the optimizer (PSO) will minimize. The hypothesis that resulted in the minimum error is expected to be most correlated with the pose that is observed, thus being the solution to the problem. In the proposed methodology we will show an automated method to construct an error function for this problem that can perform well without human intuition.

3.3.1 Observation

The input from the RGB-D sensor consists of an RGB image I and a corresponding depth map D. The dimensions of both arrays are 640×480. The depth map consists of depth values in mm for each corresponding pixel of I. Skin color is detected as in [6] and the resulting largest skin colored blob is kept for further consideration. A conservative estimation of the hands spatial extent is computed by dilating this blob with a circular mask of radius $r = 5$. Given the estimation of the 3D position of the tracked hand for the previous frame, skin colored 3D points that are within a preset depth range ($25cm$) from that estimation are kept, whereas the remaining depth map is set to zero. The observation model $O = (o_s, o_d)$ that feeds the

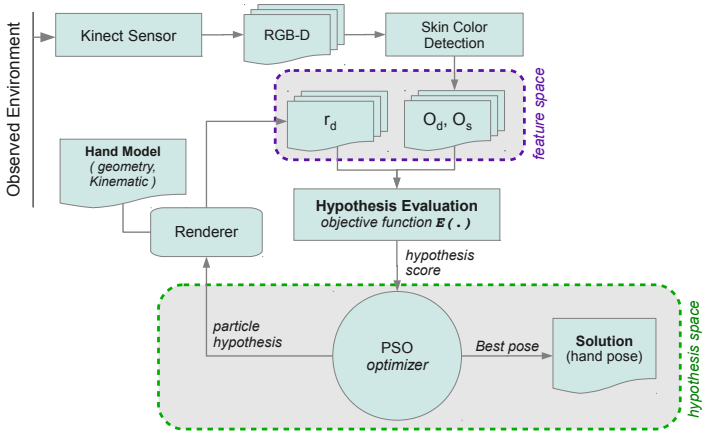


Figure 3.6: Overview of baseline method pipeline. The basic components are the observation (*Kinect Sensor*, *Skin Color Detection*), the optimizer (*PSO Optimization*) that searches for the best hypothesis and the *objective function* that is used to evaluate hypotheses.

rest of the process consists of the 2D map o_s of the segmented skin color and the corresponding depth map o_d.

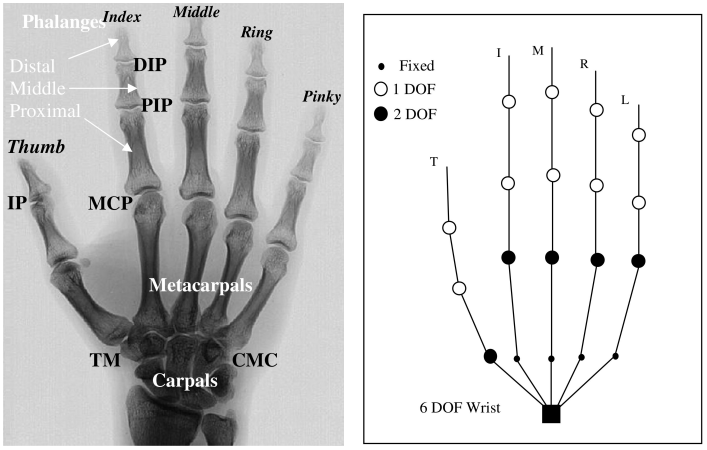
3.3.2 Hand Model

In order to generate the visualization of each hypothesis, a hand geometry model is supplied (Section 3.3.2) along with a kinematic model (Section 3.3.2) to control the elements of the model in a way similar to the human hand.

Kinematic Model

The kinematic model of the hand is based on anatomical studies [46]. The human hand is formed of 27 bones, 19 of which belong to the palm and fingers. The bones of the wrist are called carpals, those of the palm are called metacarpals and those of the fingers are called phalanges. Phalanges, depending on their proximity to the palm, are subdivided to proximal, middle and distal ones. If we add the hand's joints, a complex articulation model emerges. However, the skeleton of the hand can be abstracted as a stick figure with each finger as a kinematic chain having the base frame at the palm and each fingertip as the end-effector.

As shown in Figure 3.7 the kinematics of each finger, including the thumb, are modeled using four parameters that encode angles. More specifically, two are used for the base of the finger and two for the remaining joints. Bounds on the values of each parameter are set based on anatomical studies [46]. The global position of the hand is represented using a fixed point on the palm. The global orientation is parameterized using the redundant representation of quaternions. The final



(a) (b)

Figure 3.7: (a) The anatomy of a real human hand. (b) The kinematic model of the baseline method. Images taken from [2]

representation of a hand pose, encoding position, orientation and the articulation of the 5 fingers, is a point h that lies in a 27 dimensional space called *kinematics space*. An explanation of each dimension is shown in Figure 3.8. This space is also called the *hypothesis space* because it is where the optimization task of the baseline method takes place (Section 3.3.5).

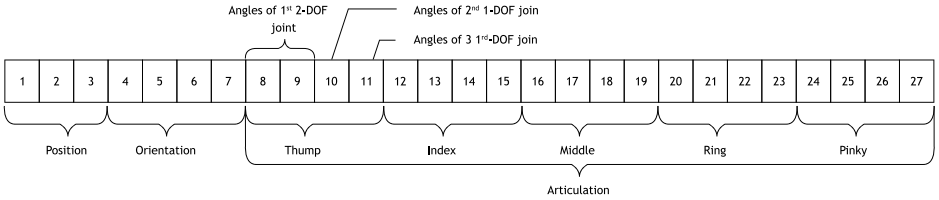
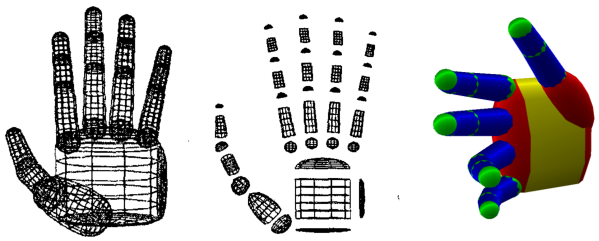


Figure 3.8: 27-parameter representation of a hand pose. The first 7 parameters encode position and orientation. The last 20 parameters, 4 for each finger, are used to encode articulation. The taxonomy illustrated for the thumb is the same for the remaining fingers. [47]

Geometry Model

In previous work done by Stenger et al. [48], they proposed the usage of quadrics to model the human hand. Their work resulted in a low triangle model that had enough detail for the problem of hand tracking. In the work of Oikonomidis et al. [10] the authors followed a similar approach but instead of using quadrics for describing the elements of the hand, they used basic geometric primitives, a

sphere and a truncated cylinder. These geometric primitives, subject to appropriate homogeneous transformations, yield a model of 37 geometry elements, similar to that of Stenger et al. [48] while being able to take advantage of GPU hardware geometry instancing for accelerated rendering.



(a) (b) (c)

Figure 3.9: The geometry model of the hand proposed by Stenger et al. [48] (a)(b) and the usage of basic geometry primitive proposed by the base line method (c) [10] (yellow for elliptic cylinders, red for ellipsoids, green for spheres and blue for cones)

Each element transformation performs two different tasks. First, it appropriately transforms primitives to more general quadrics and, second, it applies the required kinematics. In particular, a shape transformation matrix T_s can transform spheres to ellipsoids and cylinders to elliptic cylinders or cones. A rigid transformation matrix T_k is computed for a hand pose h using forward kinematics over the parameters of h (Section. 3.3.2). The final homogeneous transformation T for each of the 37 primitives (sphere or cylinder) is

$$T = T_k * T_s \tag{3.6}$$

In the implementation, the re-use of the same 2 basic primitives is performed using geometric instancing as described in [11].

3.3.3 Distance between hand poses

Given two points h_α and h_β in hypothesis space, we could assume that the Euclidean distance of these two points is a meaningful metric to express the discrepancy between the hand poses. However not all dimensions in the hypothesis space have the same effect for the shape of the hand. For example, an offset in the angle of a MCP joint has a bigger impact on the final pose of that finger than an IP joint would, because it is placed in an earlier position in the finger kinematic chain.

3D space is more suitable for creating a good metric for the distance between hand poses. Still, using the distance of only one point, like the global position of the hand, is not adequate for expressing the diversity of two hand poses. The average distance of some distributed points over the hand model can better reflect

the discrepancy of these poses. The center of the basic primitives in hand geometry model was chosen for sampling the distance. So, given two hand poses h_α and h_β their distance is expressed as:

$$\Delta(h_\alpha, h_\beta) = \frac{1}{37} \sum_{i=0}^{37} ||p_i(h_\alpha) - p_i(h_\beta)|| \tag{3.7}$$

which is the averaged Euclidean distance between the 37 p_i primitive elements (Section 3.3.2) of the hand geometry model of h_α and h_β hand poses. In this work, this distance is assumed to be the *true distance* and it will be used for training of machine learning models and evaluating the accuracy of tracking in experimental results.

3.3.4 Hypothesis evaluation (Objective Function)

The goal of the optimization procedure is to find a hypothesis h that is the most compatible with the visual observations. To do so, an error function was created which receives the configuration of the hypothesized hand h and the visual observation model of the tracked hand O (Section 3.3.1) and computes an error to quantify the discrepancy between the two. Given that one pose is in *hypothesis space* and the other in *feature space*, it is not feasible to make a direct comparison. To perform the comparison, a mapping of the hypothesis h to feature space is applied by means of rendering.

In order to minimize distortions in feature space due to the different projection methods, the process of rendering is done using a camera calibration information C, similar to the sensor that observes the real world. Given calibration information C and hypothesis h, a depth map $r_d(h, C)$ is rendered. By comparing this map with the respective observation o_d, a "matched depths" binary map $r_m(h, C)$ is produced. More specifically, a pixel of r_m is set to 1 if the respective depths in o_d and r_d differ less than a predetermined value d_m or if the observation is missing (signified by 0 in o_d), and 0 otherwise. This map is compared to the observation o_s, so that skin colored pixels that have incompatible depth observations do not positively contribute to the total score. So, the following term is formed:

$$\frac{2\sum(o_s \wedge r_m)}{\sum(o_s \wedge r_m) + \sum(o_s \vee r_m)} \tag{3.8}$$

We also want to measure the difference between depth maps in observed and hypothesized hand poses:

$$\frac{\sum min(|o_d - r_d|, d_M)}{\sum(o_s \vee r_m) + \varepsilon} \tag{3.9}$$

Equation (3.9) models the absolute value of the clamped depth differences between the observation O and the hypothesis h. Unless clamping to a maximum depth d_M is performed, a few large depth discrepancies considerably penalize an otherwise reasonable fit. A small value ε is added to the denominator of this term to avoid

division by zero.

Combining equations (3.9) and (3.8) yields the function $D(O, h, C)$ that calculates a score based on the visualization difference.

$$D(O,h,C) = \frac{\sum min(|o_d - r_d|, d_M)}{\sum(o_s \vee r_m) + \varepsilon} + \lambda \left(1 - \frac{2\sum(o_s \wedge r_m)}{\sum(o_s \wedge r_m) + \sum(o_s \vee r_m)}\right) \quad (3.10)$$

The final error function over a hypothesis h and an observation model O is formulated as:

$$E(h,O) = D(O,h,C) + \lambda_k \cdot kc(h) \quad (3.11)$$

In equation (3.11) $\lambda_k \cdot kc(h)$ is a penalty for kinematic constraints. The kc adds a penalty to kinematically implausible hand configurations. To accomplish this, it takes into account only adjacent finger interpenetration and returns a penalty score that is weighted by λ_k.

In this work, the error function $E(h, O)$ (Eq. 3.11) is also the *objective function* of the problem which the optimization module seeks to minimize.

3.3.5 Optimization

The problem of hand tracking is formulated as an optimization problem over the error function $E(h, O)$ (Eq. 3.11) using the PSO algorithm. The general mathematical form of the problem is:

$$h_{best} = arg\min_{h} E(h,O) \quad (3.12)$$

As described in Section 3.1.2, PSO is an evolutionary optimization algorithm that receives an objective function $F(.)$ and a search space S and outputs an estimation of the optimum of $F(.)$ in S, while treating it as a black box. In the baseline method, PSO operated in the 27-dimensional hypothesis space. By exploiting temporal continuity, the solution over frame F_t is used to generate the initial population for the optimization problem for frame F_{t+1}. More specifically, the first member of the population h_{ref} for frame F_{t+1} is the solution for frame F_t; The rest of the population consists of perturbations of h_{ref} . The variance of these perturbations is experimentally determined as it depends on the anticipated jerkiness of the observed motion and the image acquisition frame rate. The optimization for frame F_{t+1} is executed for a fixed amount of generations. After all generations have evolved, the best hypothesis h_{best} is dubbed as the solution for time step t + 1.

Chapter 4

Methodology

This thesis proposes an automated way to build an objective function that does not depend on human intuition. For this purpose, the process of feature extraction and the process of estimating the error are separated, from the otherwise combined processes inside the *baseline objective function* (Section 3.3.4). Given the observation model and a hypothesis in feature space, various trivial algorithms (Section 4.1) are applied to create a vector of scalar feature discrepancies. Then, by using *regression methods*, shown in Section 3.2.1, the correlation between the vector of feature discrepancies and the *true distance* (Section 3.3.3) is modeled and the machine-learned function $E_{ml}(.)$ is created, which will replace the *baseline objective function* $E(h, O)$ (Eq. 3.11). The new objective function $E_{ml}(.)$ is expected to have a smooth surface, a unique global minimum and a constant monotony which will eventually increase the performance of the optimization algorithm. Figure 4.1 shows how the new $E_{ml}(.)$ function is integrated in the *baseline method*.

The methodology can be considered a three-step procedure where the steps are:

- Find or create a set of algorithms to calculate per feature discrepancy, quantified in a scalar variable. Section 4.1
- Construct a dataset that will be used to train and evaluate the performance of various objective functions. Section 4.2
- Train a machine-learned function using the dataset from the previous step. This function, will form the new objective function E_{ml}. Section 4.3

4.1 Features

In the baseline method the input of the objective function consists of the observation model $O(o_d, o_s)$ and the rendered depth map r_d of the tested hypothesis. Given these two feature maps, we will apply algorithms $D_i(O, h)$ that will quantify, the discrepancies on these maps, i.e. for every hypothesis evaluation, a vector F will be generated from $D_i(O, h)$ functions. For the creation of $D_i(O, h)$ functions,

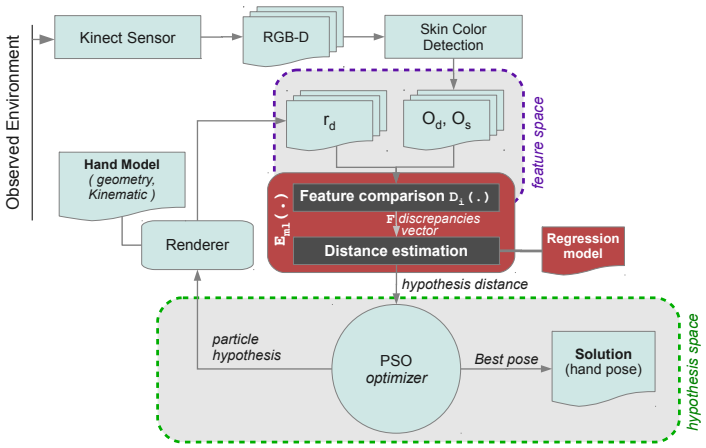


Figure 4.1: The *baseline method* pipeline after the proposed changes, marked wtih red. The $E(.)$ function has been replaced by the two-step $E_{ml}(.)$ function. An offline trained regression model is considered to map the correlation from feature discrepancies to the *true distance* of hand poses.

combinations of known algorithms were applied. Figure 5.1 shows the response histogram of D_i functions over the *true distance* on a dataset of 5 million examples. Although not much effort was put into creating these functions, there are easily identifiable correlations that are expected to be modeled by the new objective function E_{ml}

As described in Section 3.3.1 o_d is the depth map of the observation, o_s is the skin map segmented using skin color detection, and r_d is the rendered depth map of hypothesis h. In all cases N is the total number of pixels of the feature maps.

4.1.1 Sum of depth distances $D_1(.)$

Depth discrepancy is very informative of the correlation between two poses. In the same way as with the *baseline method*, we also introduce a metric of depth distances but without any data pre-processing before considering them in the final score. The simplest form of depth distances is used:

$$D_1(O,h) = \sum |o_d - r_d| \tag{4.1}$$

4.1.2 Deviation of depth distances $D_2(.)$

Not only the sum of difference of depth distances between two hand poses is a good metric, but also the deviation of these distances. Consider a hand that is moved away from the camera and a hand that is rotated around its axis that is vertical to the camera's view. Both movements could create exactly the same distance in depths, though the deviation of distances will be different. In the first case all

pixels will have a constant offset and almost zero deviation. While in the second case, the difference will be low near the axis and increase as you move away from it. The standard deviation was used for this purpose as:

$$D_2(O,h) = \sqrt{\frac{\sum (o_d - r_d)^2}{N}} \tag{4.2}$$

4.1.3 Occupied area $D_3(.)$

Another interesting metric is the area of the projection on the sensor. A hand that is very close will occupy a larger area on the projection than a hand that is further away. The area is also influenced by finger configuration and global orientation. A hand that is facing with the palm towards the camera will cover a larger area than if it were viewed edge-on. The same happens if the fingers of the hand are extended or flexed. The area is calculated based on depth maps as the number of non zero-valued pixels:

$$D_3(O,h) = \left| \sum_{pixel(o_d)\neq 0} 1 - \sum_{pixel(r_d)\neq 0} 1 \right| \tag{4.3}$$

4.1.4 Accuracy of the skin map $D_4(.)$

If the skin occupancy maps of the observed hand and the hypothesis h are given, the problem can be considered as a case of binary classification. The pixels of the hypothesis map vote for occupied or not-occupied pixels while the skin of the observed hand holds the truth. The performance of this classifier can be estimated if these two maps are similar. So every occupied pixel in the hypothesis h that corresponds to an occupied pixel in the observation O is a true positive, while if it is unoccupied in the observation it is a false positive. The F_1-score was selected to measure the classification performance, which is formed as:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{4.4}$$

where Precision $= \frac{tp}{tp+fp}$ and Recall $= \frac{tp}{tp+fn}$.

Given the $F_1(.)$ function (Eq. 4.4) the feature discrepancy function is formed as:

$$D_4(O,h) = 2\frac{TP}{(2TP + FP + FN)} \tag{4.5}$$

where $TP = \sum o_d \wedge r_d$, $TN = \sum \neg o_d \wedge \neg r_d$ and $FN = \sum o_d \wedge \neg r_d$.

4.1.5 Depth map edges $D_5(.)$

The edges on the color map are a common image feature. Edge comparison between two images can give an estimation of the similarity to the observed object.

But for the *baseline method*, the rendering of the hypothesis h does not follow a photorealistic approach and there is no usable color map of the hypothesis h. However, a photorealistic rendering is not suggested, as it is a very expensive process and it will greatly reduce the performance of the tracker. On the other hand, a descriptive depth map already exists for both observation and hypothesis, and the discontinuities of the depth distances can be detected by an edge detection algorithm. The comparison of edges on depth maps can give another hint of the distance between hand poses.

For this purpose, the canny edge detector algorithm[49] was used to create the edge maps o_e and r_e for observation and hypothesis respectively. Then the Euclidean distance map (ℓ_2 distance transformation) o_{ed} is generated over the edge map o_e, using the algorithm described in [50]. A map of all distances per edge pixel is created by masking the distance map o_{ed} using the edge map r_e as shown in Figure 4.2. A scalar value is extracted by reducing the resulting map using the sum operation:

$$D_5(O, h) = \sum o_{ed} \wedge r_e \tag{4.6}$$

4.1.6 Hand contour $D_6(.)$

The method used in feature discrepancy $D_5(.)$ can also be applied on skin contours. Specifically, the o_s map and the binary mask r_m that corresponds to the occupied pixels of r_d are used to generate the edge maps of the observation and the hypothesis. The comparison between edge maps is performed as explained in Section 4.1.5.

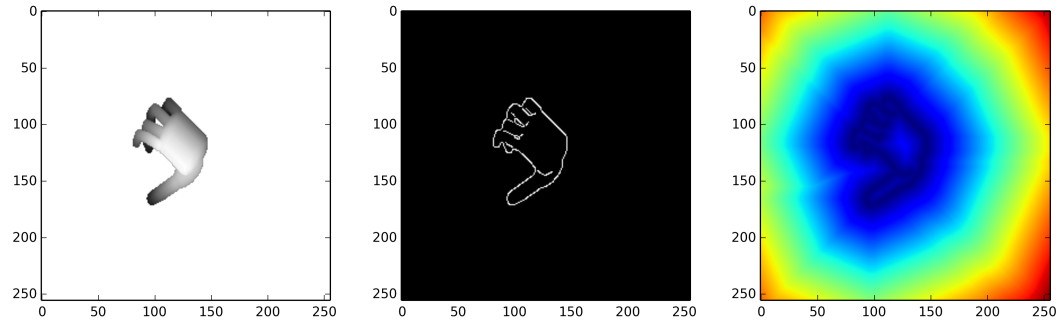
$$D_6(O, h) = \sum o_{sd} \wedge r_d \tag{4.7}$$

where o_{sd} is the distance transform of o_s.

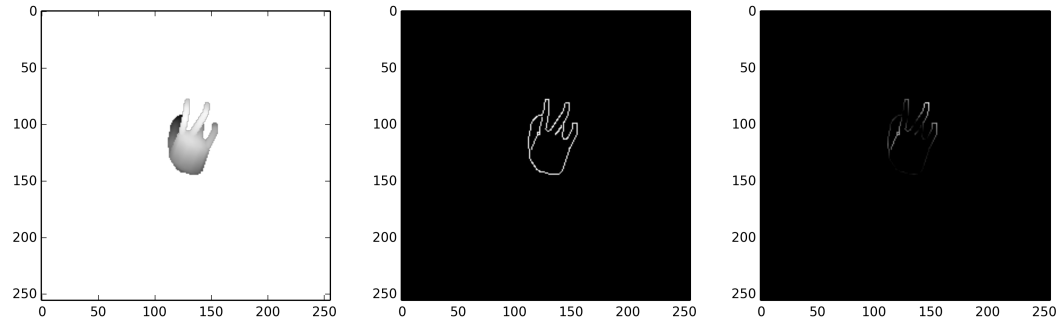
4.2 Dataset

A dataset with examples of compared hand poses is needed for modeling the correlation between feature discrepancies and *true distance*. Every example in this dataset consists of the feature discrepancy vector F_i and the *true distance* Δ_i between an observed hand model and a hypothesis. To create a dataset, the usage space of the learned model, which is the same as the search space of the optimization procedure, must be sampled.

For every example in the dataset an observation model O and a hypothesis h are selected, i.e. 54 total parameters (27 for the observed hand plus 27 for the hypothesis). The selection of these parameters are defined by the sampling strategy. To construct the example, the feature discrepancies F and the true distance between a random *observation model* and a random *hypothesis* are calculated. As



(a) Observed depth map o_d (b) Detected edges on o_d (c) ℓ_2 distance transform of (b)

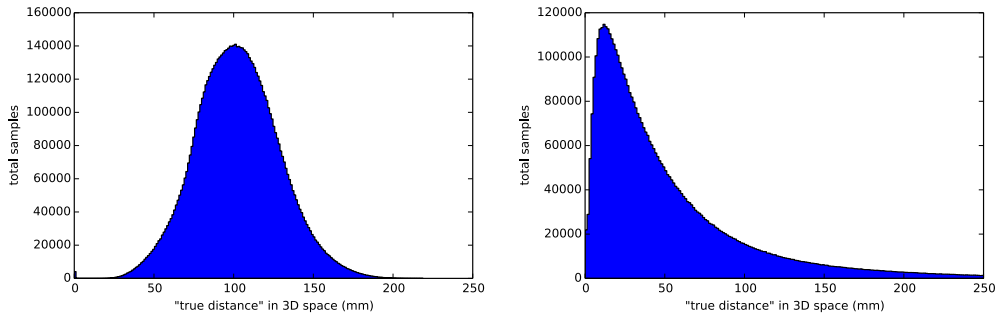


(d) Hypothesized depth map r_d (e) Detected edges on r_d (f) Per-pixel distances

Figure 4.2: The intermediate steps to compare edges in the depth map are shown. An edge map (b)(e) is generated from the depth maps of observation (a) and hypothesis (d). Using the distance transform (c) on the edge map (e) a map of edge pixel distances is created(f).

described in Section 3.3.1, the *observation models* are made by post-processing of the output of the kinect sensor. By capturing a real hand, it is impossible to select a specific configuration of the observation model. So, a simulation technique was used to create an observation model O from any arbitrary hand pose, based on the same method described in "hypothesis evaluation" (Section 3.3.4).

The size of this space is enormously big to create a dense sample. So two different sampling strategies are introduced. The first one uses a low-discrepancy sequence to quasi-random select hand poses. The second one uses introspection of the optimization procedure in order to create a dense distribution around the area that is mostly used. The histogram of samples *true distance* is illustrated in Figure 4.3.



(a) Quasi-random hand poses (b) PSO particles search space

Figure 4.3: The distribution of samples in datasets. Using the quasi-random method (a) over the effective frustum of the kinect sensor and (b) based on a trace of PSO particles search space.

4.2.1 Sampling with low-discrepancy sequence

In mathematics, a low-discrepancy sequence is a sequence with the property that for all values of N, its subsequence $x_1, ..., x_N$ has a low discrepancy. These sequences are commonly called quasi-random because they are used to replace numbers drawn from a uniform distribution. The advantage of using low-discrepancy sequences is that they offer better area coverage even for a few samples. A number of algorithms that create these kind of sequences have been developed. In this thesis, the Sobol sequence[51] is used.

This sampling strategy has no external bias over preferred areas. The generation procedure is divided in the following steps:

- Create a set P of n quasi-random hand pose configurations.
- Generate the feature maps p_d and p_s (depth, skin map) for every P_i

- For all possible combinations of P set, generate an example in the dataset that consists of the the true distance Δ_k and the F_k feature discrepancies vector calculated by the $D_i(.)$ functions.

The size of the generated dataset depends on the n generated hand poses and can be calculated as:

$$samples = \frac{n!}{2(n-2)!} \tag{4.8}$$

The boundaries of dimensions are selected based on the usable frustum of the kinect sensor and the possible movements of figures based on anatomical studies [46]. In particular for the 27-DOF parameters of the hand pose (see Figure 3.8), the boundaries of the global position are selected so that the hand is always inside the perspective frustum. The boundaries of each finger are the same as the boundaries of the PSO optimization module in *baseline method*. In the special case of global orientation, another quasi-random algorithm is used to create random quaternions over the hypersphere.

4.2.2 Sampling biased to optimization

The previous method provides a good strategy for uniformly sampling the search space. However, in practice the optimization module of the *baseline method* does not use the space evenly, it prefers cases where the true distance is small. This happens because the particles of PSO are distributed around the previous solution for the purpose of tracking; so that, the observation model O and the hypothesis h are close to each other (Section 3.3.5). In Figure 4.3 the different distributions of distances in a quasi-random dataset and those of hypotheses made by PSO optimization are illustrated. A model learned using a dataset generated with the former strategy is generalized well, however, it is expected to have poor performance for short distances because few examples exist in the dataset.

So another sampling strategy is proposed that will be biased in the area of search space where the *baseline method* usually searches. Although we could simulate the optimization procedure only to generate samples, reusing the logs of previous hand tracked poses was found to be easier and more accurate. In these logs the trajectories of each PSO particle is traced and all hypotheses that were evaluated as the best solution per frame were stored. A trivial algorithm could use the observation model $O(o_d, o_s)$ and the N hypothesis h_i per frame to create $N \times F$ examples in the dataset between O and h_i. But this is not possible because the $\Delta(.)$ function cannot be evaluated for O where hand configuration is unknown. Instead, for each frame we used the h_{best} solution as an approximation to the O model. The total number of samples in the dataset are proportional to the number of generation g, particles p and frames f of the tracking log:

$$samples = ((g \times p) - 1) \times f \tag{4.9}$$

4.3 Regression Model

The new $E_{ml}(.)$ will be constructed by modelling the *training dataset* that was created using one of the methods described in Section 4.2. The purpose of this step is to define the actual structure of function $E_{ml}(.)$ (Eq. 4.10), which given the outcomes of $D_i(.)$ functions (Section 4.1) will compute a value as close as possible to the $\Delta(.)$ function. To achieve this we use known regression analysis algorithms (Section 3.2.1) on the *training dataset* in order to model this correlation and construct a usable function.

$$E_{ml} = f(D_1(.), D_2(.), ..., D_n(.)) \tag{4.10}$$

Different regression analysis algorithms have been developed to find the correlation between parameters on a dataset, depending on the nature of their relation. The most common are closed form models that are trained by finding the best values for their parameters. These models have minimal memory footprint, are computationally efficient on training and prediction but their accuracy depends on the form of the model. Their performance is dependent on the skills of the analysist, so one chooses the best closed form mathematical expression that can actually model the correlation. Common models are *linear*, *polynomial* and *exponential* functions. Another category is that of models constructed at the time of training, like *Decision Trees* (Section 3.2.2). These models have no parameters, enabling usage without any deep prior knowledge of the problem but have limitations on which type of correlations they can reproduce.

On the problem of hand tracking we have proposed different models in order to evaluate the performance of our methodology. The models that have been found to perform sufficiently and deserve further evaluation are:

- Linear model using *mean squared error*
- Polynomial model of 2^{nd} degree
- Polynomial model of 3^{rd} degree
- Random Forests with 6 sub-trees (as the number of features)

Chapter 5

Experimental Evaluation

In this chapter we will evaluate the behaviour of feature discrepancy functions $D_i(.)$, dataset generation, model and tracking performance. The proposed feature discrepancy functions $D_i(.)$ are profiled to identify their behaviour in the 27 hypothesis space. Then an evaluation method based on the ground truth dataset, generated through synthetic samples, is presented and used to make further experiments. Finally, regression models are generated as proposed by the method (Section 4.3) and are extensively evaluated. This is done in two steps, first we evaluate the performance of the models in predicting the response of $\Delta(.)$ function (Eq. 3.7). Then we evaluate their influence on the hand tracking problem by replacing the baseline objective function and measuring the hand tracking accuracy.

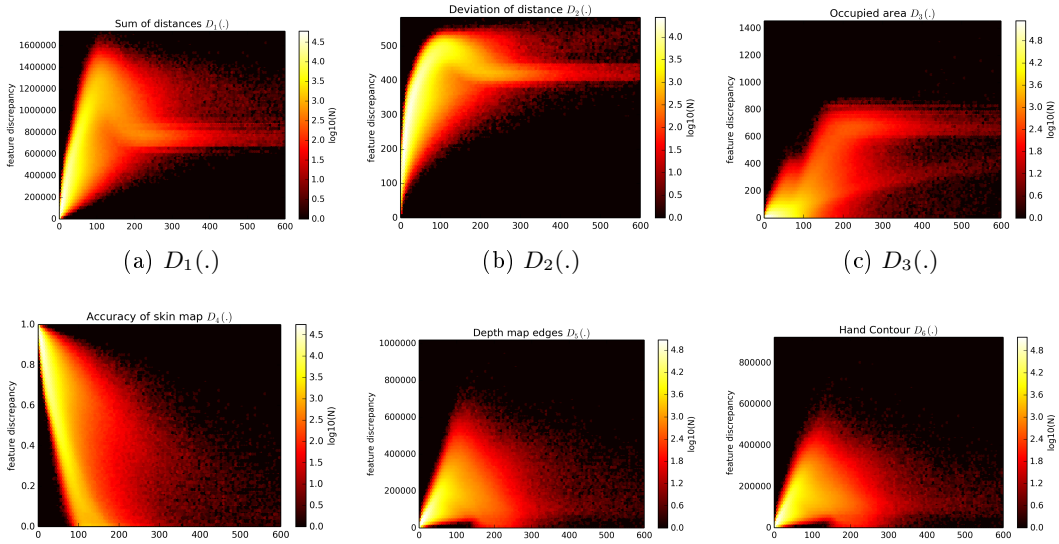
All experiments ran on a computer equipped with a quad-core intel i7 930 CPU, 16 GBs RAM and the Nvidia GTX 580 GPU with 1581 GFlops processing power and 1.5 GBs memory.

5.1 Features

In Section 4.1 six $D_i(.)$ functions were proposed to evaluate the visual discrepancies between an observation model O and a hypothesis h. In this section we analyse the behaviour of these functions and their potential to correlate with the $\Delta(.)$ function (Eq. 3.7). It is essential that some kind of correlation exists on which regression models can be based/trained and make efficient predictions.

In Figure 5.1 the response histograms of D_i functions is shown. This figure can give an estimation if there are any human recognizable patterns between $D_i(.)$ and *true distance* $\Delta(.)$. It is clear that for the majority of features there is a strong and almost linear pattern for short distances less than 100 mm. However, for distances larger than 100 mm, the response histograms show that output of $D_i(.)$ is unpredictable and in some cases, like $D_3(.)$, $D_5(.)$ and $D_6(.)$, is almost random.

To better understand the internals of $D_i(.)$ functions we profiled their response by changing the number of dimensions. So given two identical hypotheses h_α and h_β, iteratively we changed the value of the global position in the z axis for h_β in



(a) $D_1(.)$ (b) $D_2(.)$ (c) $D_3(.)$

(d) $D_4(.)$ Accuracy of skin map (e) $D_5(.)$ Depth map edges (f) $D_6(.)$ Hand contour

Figure 5.1: The response histogram of D_i functions over the *true distance* on a dataset of 5 million examples. Every example k, in this dataset, consists of the observation model O_k, the hypothesis h_k, the *true distance* Δ_k and the features comparison vector F_k.

the range between -700 mm and -1500 mm while keeping the same dimension for h_α constant at -875 mm. In Figure 5.2 we can see the results of this procedure and how the response of the $D_i(.)$ functions to changes in the z axis. In all cases we can identify a strong global minimum exactly at the point where the hypothesis h_α is. The functions $D_1(.)$, $D_2(.)$ and $D_4(.)$ have a smooth surface and unique global minimum. Functions $D_3(.)$, $D_5(.)$ have a bit of noise but still preserve a unique global minimum, unlike $D_6(.)$. Finally the baseline discrepancy function $D()$ (Eq. 3.10) is also tested in the same experiment, and also presents a prominent unique global minimum at the correct position.

The hypothesis configuration space is a high dimensionality space of 27 degrees of freedom. In order to expose its complexity we replicated the same experiment but this time h_α and h_β were not identical hypotheses, however all their parameters were kept constant throughout the test. In Figure 5.3 we see that the results of this experiment, are significantly different. The majority of functions have poor performance on identifying whether h_α and h_β have the same Z value. For example the functions $D_1(.)$ and $D_4(.)$ do not have any global minimum or maximum lying close to the solution. Functions $D_5(.)$ and $D_6(.)$ have increased noise compared to the previous test and the solution is close to their global maximum, instead of the global minimum as in the previous test. The solution for $D_2(.)$ is also close to its global maximum. Only $D_3(.)$ preserves a robust behaviour within

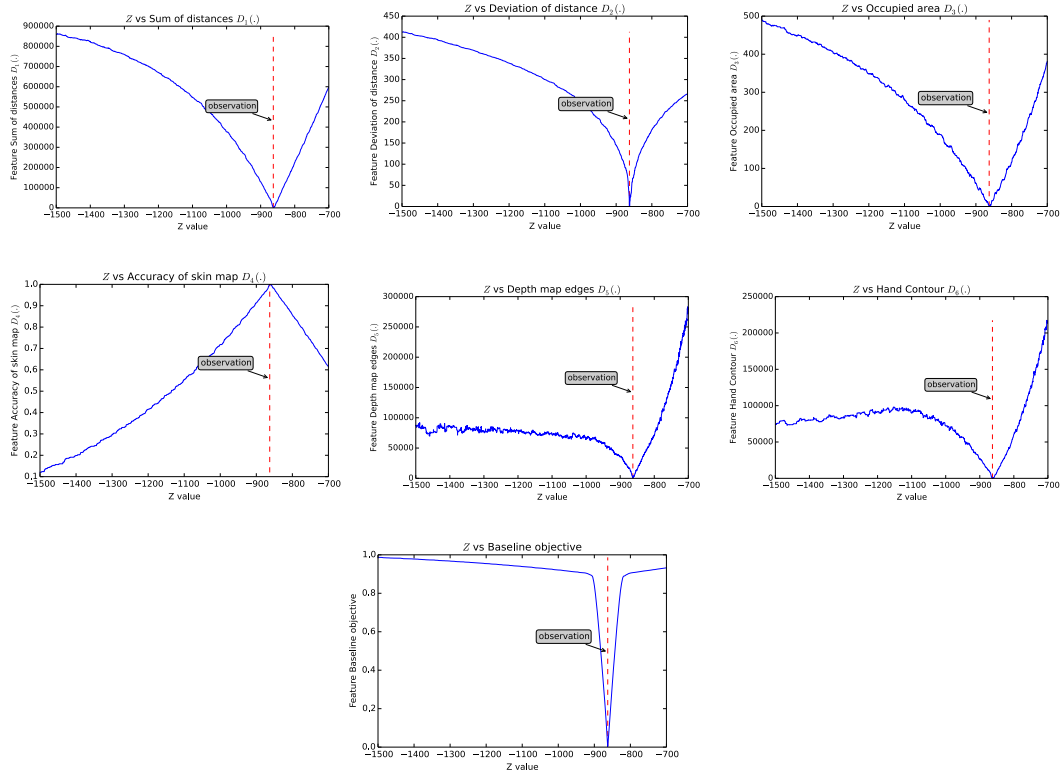


Figure 5.2: Profile of feature functions $D_i(.)$ and baseline discrepancy function $D(.)$ on the Z axis of hand global position. The observation model that was used has exactly the same configuration as the tested hypothesis.

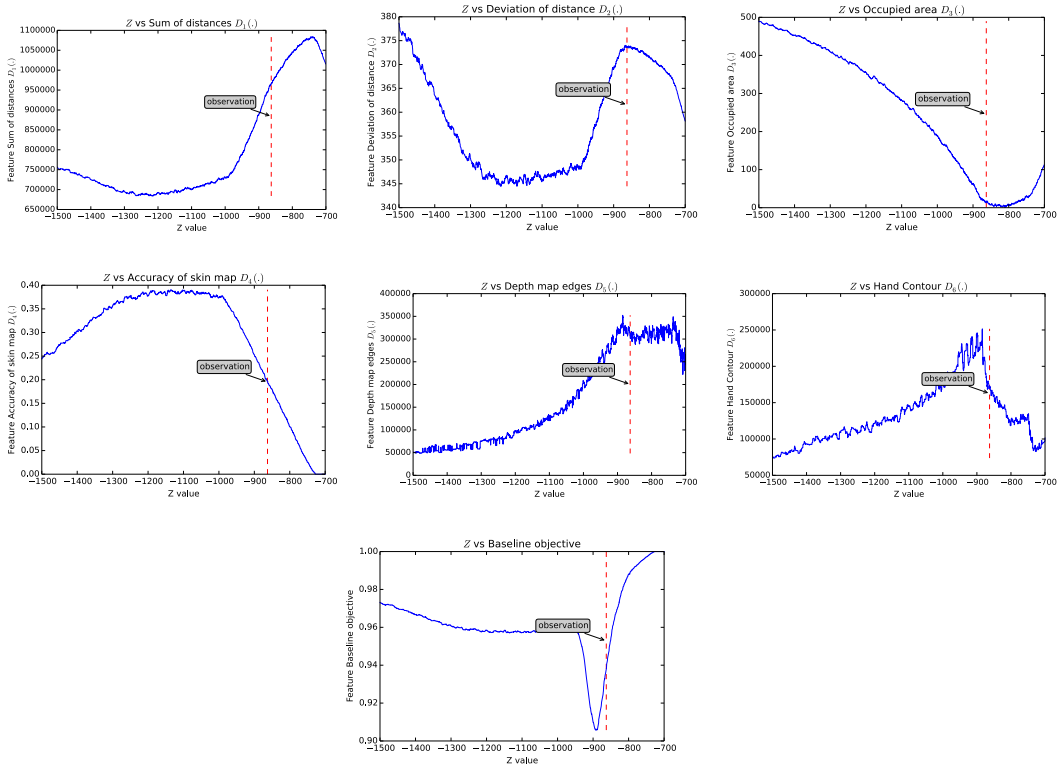


Figure 5.3: Profile of feature functions $D_i(.)$ and baseline discrepancy function $D(.)$ on the Z axis of hand global position. The observation model that was used has a different configuration than the tested hypothesis.

both tests, with a slight error of 50 mm. It is interesting that even the baseline discrepancy function $D(.)$ has an error of less than 50 mm although it was specially crafted for this problem. This experiment shows that due to the high complexity of the hypothesis space, changes in one dimension can affect behaviour in other dimensions. Regression models are challenged to learn this complexity and map the correlation between $D_i(.)$ functions and the $\Delta(.)$ function.

5.2 Evaluation on synthetic data

In order to evaluate the performance of tracking, a ground truth dataset is needed, that includes samples of captured observation model O and the actual configuration of the observed hand h^{true}. Unfortunately it is impossible to generate a dataset using a real observed environment. This is because there is no method that can detect the position of the hand with high accuracy without influencing the observation of the sensor. For example if we used markers on the hand, then the actual image of the observed hand will be altered, and skin color detection algorithm will be influenced, making the results of this experiment unreliable. This also holds for the case of a glove or any other method that demands fusion with the hand.

For this reason, we followed the synthetic approach to create a ground truth dataset. Using the software of the baseline method, it is possible to instruct the renderer to simulate the output of the sensor and create a synthetic observation model O for any hand configuration h. The problem in this case, is the selection of hand poses. Selecting poses that are implausible in a real hand because of articulation or motion constraints will produce a dataset of non-realistic scenarios, for example a hand that moves too fast, or too slow or a joint that moves unnaturally.

It is feasible to produce hand trajectories through the usage of a human kinematics simulator, however the quality of the dataset depends on the quality of the simulator. For this reason we preferred to capture real hand poses and use them to generate synthetic data. This could be done with any kind of hand tracker which includes also the *baseline method*. This led us to use the baseline method to observe a real hand trajectory in order to extract a hand configuration h_i^{true} per frame i. Although these hand poses had a small difference from the real hand pose, this error was small enough to still replicate a normal hand movement.

The final procedure of creating the ground truth dataset has the following steps:

- Use the base line method to observe a real hand trajectory, and extract the h_i^{true} hand configuration per frame i.
- Reprogram the base line method to only simulate the O_i observation model for each hand pose configuration h_i^{true}
- Create a Gtd dataset with samples h_i^{true} and O_i for each frame i

Having a ground truth dataset Gtd, one can evaluate the performance of a tracker $Tracker(.)$ by measuring the error between $Tracker(Gtd[O]_i)$ and $Gtd[h^{true}]_i$ using the $\Delta(.)$ function Eq. 3.7.

5.2.1 Regression models performance

Before evaluating the tracking performance, it is interesting to evaluate the influence of each feature $D_i(.)$ compared with different regression models. For this test we trained the four proposed models (Section 4.3) using only one feature discrepancy function $D_i(.)$ and using all six $D_i(.)$ functions. In all cases the models were trained on the same training dataset and were evaluated on the same ground truth dataset. The PSO was configured to 64 particles and 25 generations.

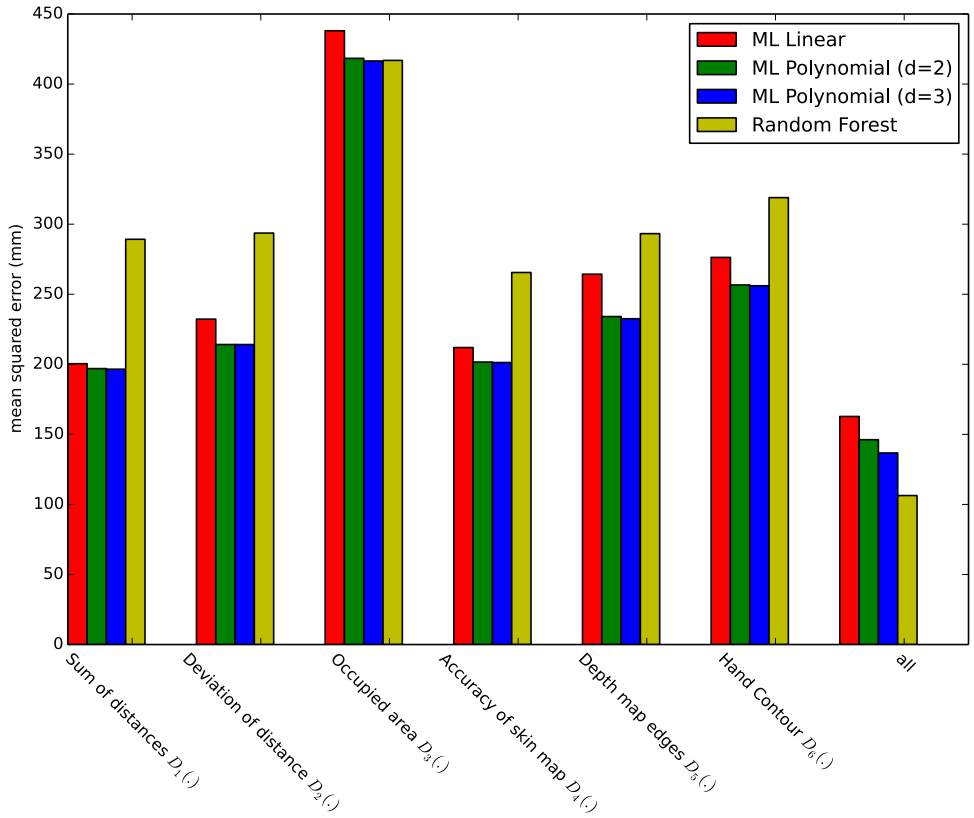


Figure 5.4: Performance of models when trained on dataset that contained only one feature, compared to performance when trained on all features simultaneously.

In Figure 5.4 we see the results of this test. Note that the combination of all features produced better results than any single feature. Using only one feature resulted in almost the same accuracy for all $D_i(.)$ functions except for $D_3(.)$ which showed reduced performance. It is interesting that the *random forest* algorithm had different performance compared to other models when using one $D_i(.)$ function or all six combined. Specifically it was significantly outperformed by all other closed form algorithms when using only one feature. But it had better performance than all other algorithms when using all $D_i(.)$ functions together. This was expected because the random forest algorithm is more suitable for modelling multi-dimensional

and non-linear problems, such as the hand-tracking problem.

5.2.2 Tracking Performance

Finally we need to evaluate the performance of the hand tracker employed by the new objective function. To evaluate the performance we use the ground truth dataset that was created using synthetic data. The test was done by combining different configurations for the regression model, the training dataset and the PSO algorithm. The error was measured in mm using the $\Delta(.)$ function Eq. 3.7. For each configuration profile, the test was run 20 times and the mean error was considered. This was done because PSO is a non-deterministic algorithm and the execution of it depends on the random seed.

More specifically the following configuration options were used:

- *Regression Model*: linear model, polynomial 2^{nd} degree, and random forests.
- *Training Dataset*: 1 using quasi-random with 4096 poses, 1 generated on previous tracking logs
- *PSO configuration*: 5, 10, 15, 20, 25 generations and 8, 32, 64 particles

Figure 5.5 shows the results of the tracking performance for all configurations using a training dataset generated by a *quasi-random* method. At first glance, the *baseline method* has $10.7mm$ minimum error which is the best for all cases. However, a simple linear method with no prior knowledge of the problem complexity is only $8mm$ worse than the *baseline method*. The polynomial method while being a superset of the linear method and therefore was expected to be at least identical to it, has an error of $28.0mm$. The best explanation for this is that the polynomial function was over-fitted on the dataset which had more samples at long distances as explained in Section 4.2.1. This led to poor generalization at small distances which are extensively searched by the PSO. Finally the *Random Forests* algorithm, had the worst performance with $100.4mm$ minimum error and $1500mm$ maximum error. *Random Forests* make no assumption of the modelled space which makes them a bad predictor for areas where no training samples exist.

Figure 5.6 shows the same test but using a training dataset generated from a previous tracking log. The major difference with Figure 5.5 is that all regression models have improved performance on low PSO budget. In the previous test, the worst error was $1500mm$, while now it is $407.4mm$. On PSO high budget configurations, there is a different behaviour according to which model is used. More specifically, the linear method has a drop in performance from $18.5mm$ error to $36.8mm$. This is explained because the linear method is less flexible and did not manage to fit on the training dataset, leading to unpredictable behaviour. The same is confirmed by the polynomial method, which had an increased performance in this test. Polynomials are more flexible than linear functions and are a better fit to the samples around small distances. In this version of the test the polynomial is better than the linear function, as expected. It is notable that *Random Forests*

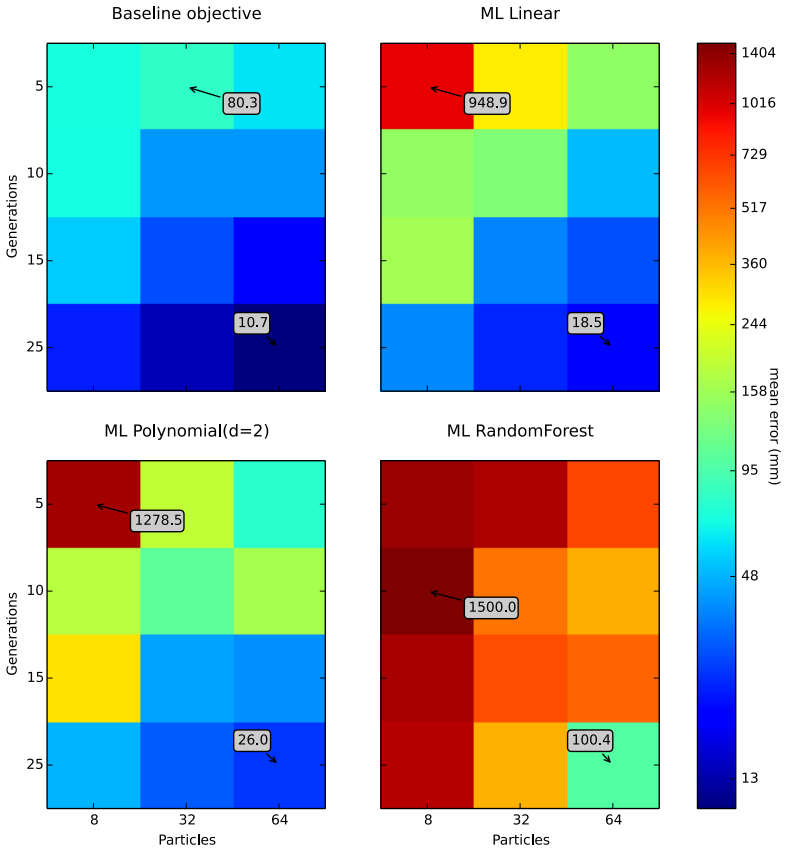


Figure 5.5: Tracking performance tested on the ground truth "cooljason" dataset. Models were trained on dataset generated by 4096 quasi-random poses.

have the most improvement using this dataset. This was also expected as this dataset had examples of compared poses with small distances and the *Random Forests* managed to learn the behaviour in this area.

The results of this test show that our method is sustainable and can, to some degree, replace the manual procedure of designing an objective function. It has been shown though that simple things such as the generation of a dataset have a key role on performance and should not be overlooked. It must be noted that even the procedure of dataset generation and model selection can be approached methodically, without any deep prior knowledge in the field of machine learning.

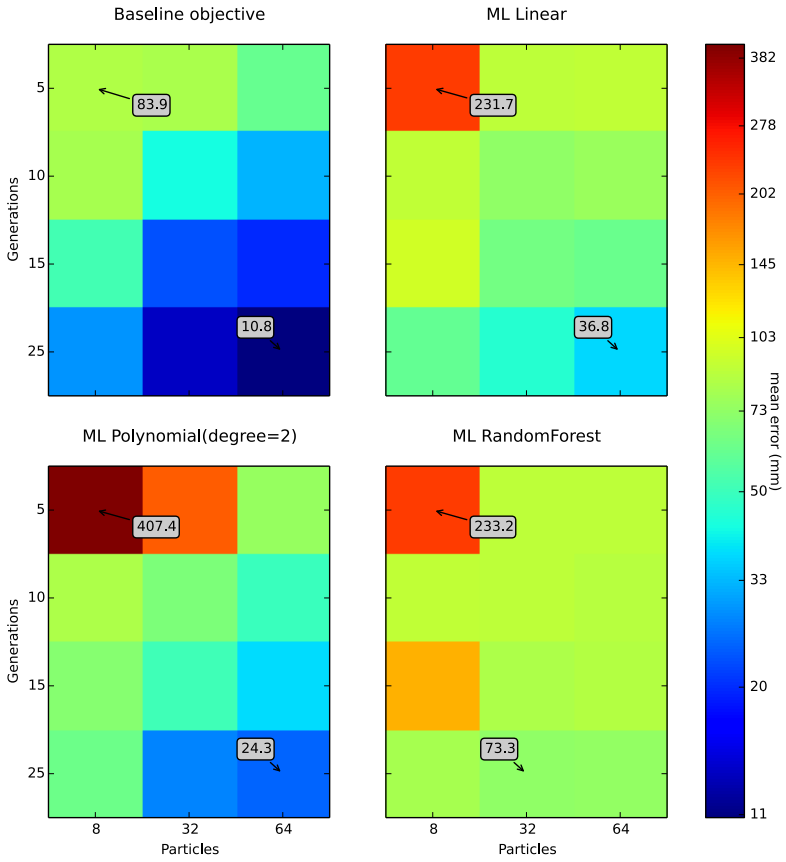


Figure 5.6: Tracking performance tested on the ground truth "cooljason" dataset. Models were trained on dataset generated on tracking logs using the objective function.

Chapter 6

Discussion

In the introduction we described an existing model-based hand tracking method [10] that can efficiently track a human hand from an RGBD sensor in real time using modest computational resources. This method approaches the problem of hand tracking as an optimization problem, where for each frame the optimizer seeks for the hand configuration considering the visual discrepancies between the hypothesis and the observation. The testing is performed using rendering techniques to simulate the output of the RGBD sensor as if it was observing the hypothesized hand configuration. This simulated observation is compared against the received observation from the sensor to estimate the distance between the observed and the hypothesized hand pose. For this comparison a well crafted objective function was built by the authors of the method that behaves well around the solution of the problem and can be effectively used by an optimizer and especially by the *Particle Swarm Optimization* algorithm.

In this work, we acknowledge the importance of the structure of the objective function on the optimization problem and the difficulties one has to face in order to construct a function that performs well within the problem. The goal of this work was to find a method that could be used to craft an objective function for this problem without deep knowledge of and experience on mathematical techniques or computer vision theory. For this reason we formulated the problem of building an objective function as a machine learning problem of estimating a function with the desired characteristics through the process of regression analysis on a training dataset. We started by defining the characteristics that would help the function perform well. The main conclusion of this exploration was that the ideal function would be the *true distance* $\Delta(.)$ Eq. 3.7. We have used machine learning to estimate a $\Delta(.)$ function since a direct calculation requires the knowledge of both hand configurations, and this is not available for the observed hand.

For the reasons mentioned previously, we have proposed a detailed approach to the problem that could effectively replace the current objective function. In the *baseline method* the input of the objective function Eq. 3.11 is the observation model O (the output of the RGBD sensor) and hypothesized hand configuration

h. The hand configuration h is converted to a simulated observation model O_s and the comparison is done between these two observation models inside the *baseline objective function*. In our work we split the procedures of evaluating the visual discrepancies and estimating a score. We have defined $D_i(.)$ functions that use trivial and well established algorithms for computing visual discrepancies, which are presented in Section 4.1. The output of these $D_i(.)$ functions is considered by a trained regression model in order to estimate the *true distance* $\Delta(.)$ between observed hand and hypothesis h. The problem of crafting an objective function was shifted from a *trial and error* to a *regression analysis* procedure in order to create a new objective function $E_{ml}(.)$ that accepts the output of the $D_i(.)$ functions and responds with an estimation of the $\Delta(.)$ function output. This was achieved using training datasets that were generated by two different approaches, see Section 4.2. One approach was to generate quasi-random hand poses using a low discrepancy sequence and generate the training dataset by comparing all these poses. The other approach was to observe the execution of the hand tracking procedure using the baseline method and create a dataset from all the hypotheses investigated by the PSO. The former was easier to build and was unbiased from any external parameter, while the latter resulted in a more detailed dataset for the specific problem, however it was biased to the observed environment at the time of execution.

To apply and evaluate our method, different regression algorithms were tested on both training dataset generation techniques. In Chapter 5, first we profiled and extensively evaluated the response of the $D_i(.)$ functions. It was shown that trivial functions can give hints about the correlation of two hand poses based on their observation but the high dimensionality and complexity of the problem made it difficult to model this correlation. This was also the challenge of the regression analysis procedure. Later, we presented a way to make an evaluation of the tracking performance using a synthetic dataset (Section 5.2). This procedure was greatly inspired from the evaluation procedure that was used in the *baseline method* [10]. Using this procedure we evaluated the influence of each visual discrepancy function $D_i(.)$ isolated against each tested regression model. The experiments showed that all $D_i(.)$ functions could approach the solution but the performance of their combination was by far the best. Another interesting result is that the *Random Forests* algorithm used for regression analysis was more effective in multi-dimensional space than any of the other methods. In the last step of evaluation, we measured the performance of hand-tracking by replacing the *baseline objective function* with the new objective function $E_ml(.)$. For this experiment we tested multiple configurations of the PSO algorithm, regression models and training dataset generation approaches. All the combinations of these configuration options were tested and presented in Figures 5.5 and 5.6. These two figures show that none of the configuration profiles managed to outperform the *baseline objective function* but many profiles approached the same performance.

In conclusion, we showed a methodical procedure to create an objective function for the model-based hand tracking problem. The procedure of crafting an objective function is turned from an iterative procedure to a regression analysis problem on a

training dataset. Although at the evaluation stage this method did not manage to perform better than the baseline method, it showed competitive results, especially considering the reduced demands of expertise on the problem of hand tracking or mathematical optimization. Finally, we should mention that this method is not constrained to the problem of hand-tracking but can be used in any optimization problem that depends on complex objective functions.

6.1 Future Work

In Section 5.2.2 we showed that the training dataset has a key role on the performance of hand-tracking. In Section 4.2 we proposed two different approaches on generating datasets. However none of them proposes a standard procedure that produces good quality samples and unbiased from any external parameter. Future work could be done to improve the quasi-random approach so that the generated samples have similar distributions to that used by the PSO algorithm (Figure 4.3). A possible solution is to first sample the hypothesis space using a quasi-random approach and then, for each pose re-sample the area around this pose, in order to generate examples of hand pose pairs having small distances.

Another issue, that was not considered in this work, is that of implausible hand poses. In the *baseline method* the objective function $D(O, h, C)$ apart from returning a score based on the visual discrepancies, also adjusts the score depending on whether the hand configuration is feasible or not. In our proposed method, we have not taken into consideration any implausible hand configurations. In principal a good hand-tracker would find only hand configurations close to the observed one, reducing the possibility of finding unnatural hand poses. However, including this knowledge in the hand tracker could make the optimization procedure faster and more accurate by excluding all implausible configurations. This could be solved by adding a pre-processing layer in the objective function that will reject completely any unfeasible configurations and continue to the next hypothesis.

In the baseline method, hand-tracking demands that in the first frame the observed hand starts from a predefined position. This is required because there is no method to make hand pose estimation without prior knowledge. In our work we have not relaxed this constraint and we have focused only on the problem of hand tracking that exploits time continuity. However, in contrast to the original objective function $D(O, h, C)$ Eq. 3.11, the new $E_{ml}(.)$ tries to approximate the $\Delta(.)$ function, which has constant quality on all distances. This makes it a more suitable objective function not only for hand-tracking but also for hand pose estimation. In future, it would be interesting to use the same methodology for solving the problem of hand pose estimation and incorporating this in the base line method, eliminating completely the constraint on the initial hand pose.

Finally, this work is based on the hand-tracking problem, but should not be limited to that. It can be generalized to any optimization problem with a complex objective function. It would be very interesting to find other known optimization

problems to adapt and evaluate this methodology. This could greatly help to improve this proposal and automate all the included steps.

Bibliography

[1] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis," *Computer vision and image understanding*, vol. 104, no. 2, pp. 90–126, 2006.

[2] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly, "Vision-based hand pose estimation: A review," *Computer Vision and Image Understanding*, vol. 108, no. 1–2, pp. 52 – 73, 2007, special Issue on Vision for Human-Computer Interaction. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314206002281

[3] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3. ACM, 2009, p. 63.

[4] I. Oikonomidis, N. Kyriazis, and A. A. Argyros, "Markerless and efficient 26-dof hand pose recovery," in *Computer Vision–ACCV 2010*. Springer, 2011, pp. 744–757.

[5] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

[6] A. A. Argyros and M. I. Lourakis, "Real-time tracking of multiple skin-colored objects with a possibly moving camera," in *Computer Vision-ECCV 2004*. Springer, 2004, pp. 368–379.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4. IEEE, Nov. 1995, pp. 1942–1948 vol.4. [Online]. Available: http://dx.doi.org/10.1109/icnn.1995.488968

[8] J. Romero, H. Kjellstrom, and D. Kragic, "Monocular real-time 3d articulated hand pose estimation," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. IEEE, 2009, pp. 87–92.

[9] Y. Wu and T. S. Huang, "View-independent recognition of hand postures," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2. IEEE, 2000, pp. 88–94.

[10] I. Oikonomidis, N. Kyriazis, and A. Argyros, "Efficient model-based 3d tracking of hand articulations using kinect," in *BMVC 2011*. BMVA, 2011.

[11] N. Kyriazis, I. Oikonomidis, and A. Argyros, "A gpu-powered computational framework for efficient 3d model-based vision," ICS-FORTH, Tech. Rep. TR420, July 2011.

[12] M. de La Gorce, N. Paragios, and D. J. Fleet, "Model-based hand tracking with texture, shading and self-occlusions," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference On*. IEEE, 2008, pp. 1–8.

[13] H. Hamer, K. Schindler, E. Koller-Meier, and L. Van Gool, "Tracking a hand manipulating an object," in *Computer Vision, 2009 IEEE 12th International Conference On*. IEEE, 2009, pp. 1475–1482.

[14] X. Xiong and F. De la Torre, "Supervised descent method and its applications to face alignment," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2013, pp. 532–539.

[15] C. Cao, Y. Weng, S. Lin, and K. Zhou, "3d shape regression for real-time facial animation." *ACM Trans. Graph.*, vol. 32, no. 4, p. 41, 2013.

[16] Wikipedia, "Mathematical optimization — wikipedia, the free encyclopedia," 2013, [Online; accessed 13-February-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Mathematical_optimization&oldid=587193239

[17] ——, "Loss function — wikipedia, the free encyclopedia," 2013. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Loss_function&oldid=582461084

[18] J. Snyman, *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Springer, 2005, vol. 97.

[19] C. T. Kelley, *Iterative methods for optimization*. Siam, 1999, vol. 18.

[20] Wikipedia, "Global optimization — wikipedia, the free encyclopedia," 2013, [Online; accessed 13-February-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Global_optimization&oldid=588447632

[21] P. Padeleris, X. Zabulis, and A. A. Argyros, "Head pose estimation on depth data based on particle swarm optimization," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. IEEE, 2012, pp. 42–49.

[22] Wikipedia, "Test functions for optimization — wikipedia, the free encyclopedia," 2014, [Online; accessed 15-February-2014]. [Online].

Available: http://en.wikipedia.org/w/index.php?title=Test_functions_for_optimization&oldid=593377680

[23] J. M. Dieterich and B. Hartke, "Empirical review of standard benchmark functions using evolutionary global optimization," *CoRR*, vol. abs/1207.4318, 2012.

[24] J. F. Kennedy, J. Kennedy, and R. C. Eberhart, *Swarm intelligence*. Morgan Kaufmann, 2001.

[25] Wikipedia, "Particle swarm optimization — wikipedia, the free encyclopedia," 2014, [Online; accessed 17-February-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Particle_swarm_optimization&oldid=593577321

[26] A. Banks, J. Vincent, and C. Anyakoha, "A review of particle swarm optimization. part i: background and development," *Natural Computing*, vol. 6, no. 4, pp. 467–484, 2007.

[27] P. N. Suganthan, "Particle swarm optimiser with neighbourhood operator," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3. IEEE, 1999.

[28] B. Al-kazemi and C. K. Mohan, "Multi-phase generalization of the particle swarm optimization algorithm," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 2, 2002, pp. 1057–1062.

[29] T. M. Blackwell and P. Bentley, "Don't push me! collision-avoiding swarms," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 2. IEEE, 2002, pp. 1691–1696.

[30] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 1, pp. 58–73, 2002.

[31] S. Helwig and R. Wanka, "Particle swarm optimization in high-dimensional bounded search spaces," in *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*. IEEE, 2007, pp. 198–205.

[32] Wikipedia, "Machine learning — wikipedia, the free encyclopedia," 2014, [Online; accessed 18-February-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=595728221

[33] T. M. Mitchell, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.

[34] Wikipedia, "Regression analysis — wikipedia, the free encyclopedia," 2013, [Online; accessed 19-February-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Regression_analysis&oldid=587277291

[35] A. M. Legendre, *Nouvelles méthodes pour la détermination des orbites des cometes.* F. Didot, 1805.

[36] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*, 1809.

[37] Wikipedia, "Simple linear regression — wikipedia, the free encyclopedia," 2014, [Online; accessed 19-February-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Simple_linear_regression&oldid=590845753

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[39] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection." Morgan Kaufmann, 1995, pp. 1137–1143.

[40] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[41] ——, *C4. 5: programs for machine learning.* Morgan kaufmann, 1993, vol. 1.

[42] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees.* CRC press, 1984.

[43] T. K. Ho, "Random decision forests," in *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1. IEEE, 1995, pp. 278–282.

[44] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[45] Wikipedia, "Bootstrap aggregating — wikipedia, the free encyclopedia," 2014, [Online; accessed 1-March-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Bootstrap_aggregating&oldid=591557530

[46] I. Albrecht, J. Haber, and H.-P. Seidel, "Construction and animation of anatomically based human hand models," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation.* Eurographics Association, 2003, pp. 98–109.

[47] P. Douvantzis, "Dimensionality reduction for efficient hand tracking, pose estimation and classification," Master's thesis, University of Crete, School of Sciences and Engineering, Computer Science Department, 2013.

[48] B. Stenger, P. R. Mendonça, and R. Cipolla, "Model-based 3d tracking of an articulated hand," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 2. IEEE, 2001, pp. II–310.

[49] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[50] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International journal of computer vision*, vol. 70, no. 1, pp. 41–54, 2006.

[51] I. M. Sobol, "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967.