COPPE
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia
UFRJ

ESTIMATING HAND POSES FROM RGB-D DATA

Pedro de Souza Asad

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Ricardo Guerra Marroquim

Rio de Janeiro
Maio de 2016

ESTIMATING HAND POSES FROM RGB-D DATA

Pedro de Souza Asad

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Ricardo Guerra Marroquim, D.Sc.

Prof. Marcelo Bernardes Vieira, Ph.D.

Prof. Claudio Esperança, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2016

Asad, Pedro de Souza

Estimating hand poses from RGB-D data/Pedro de Souza Asad. – Rio de Janeiro: UFRJ/COPPE, 2016.

XII, 66 p. $29,7$cm.

Orientador: Ricardo Guerra Marroquim

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2016.

Referências Bibliográficas: p. 55 – 59.

1. Estimação de postura. 2. Otimização por nuvem de partículas (PSO). 3. Câmeras RGB-D. I. Marroquim, Ricardo Guerra. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos que vieram antes.

Agradecimentos

Um trabalho como este é o produto de uma complexa rede de interações. Interações humanas, interações com o ambiente, interações bioquímicas, etc. Sua aceitação culmina na titulação de um indivíduo como *Mestre em Ciências*, mas o mérito por sua realização se estende, em maior ou menor grau de evidência, a toda uma comunidade e um planeta, historicamente vivos, em que o mesmo se insere. Por isso, meu primeiro agradecimento é para todos aqueles que não foram evidenciados no restante desta seção, que trata das bandas intermediárias, invisíveis, ou semi-transparentes, do espectro de participações neste trabalho, e faz isso em uma sequência sem nenhum compromisso com ordens de magnitude, mas tão somente ao gosto do autor.

A primeira menção nomeada é a meu orientador, Ricardo, agente tão fundamental quanto evidente, que demonstrou paciência perseverante diante da minha indisciplina com prazos, mas nunca deixou de "puxar minha orelha" com a regularidade adequada. Vale também dizer que não são muitos os orientadores que estão dispostos, ou disponíveis, a sentar ao seu lado para ajudar a depurar seu código ou a oferecer conselhos sobre suas incertezas profissionais e que tive a rara sorte de Ricardo ser um deles. Ele reflete bem um espaço querido, o LCG: laboratório repleto de pessoas, alunos e professores, entusiásticos e prestativos, prontos a te auxiliar com empecilhos técnicos fulminantes ou a te acompanhar para um lanche da tarde repleto de conversas deliciosas, que vão da política à crônica-ficção da vida cotidiana.

Este laboratório representa para mim, hoje, o patamar mais íntimo de uma comunidade acadêmica ampla que é a Universidade Federal do Rio de Janeiro. Desta, obtive também o meu grau de bacharel e tenho me beneficiado, há oito anos, de seu suporte constante para acessar conhecimento, firmar o caráter e fazer mais perguntas do que sou capaz de responder. São muitas as pessoas necessárias ao funcionamento desta máquina formidável: professores, alunos, secretários acadêmicos, bibliotecários, faxineiros, seguranças, administradores, motoristas e outros. A todos, agradeço pelas contribuições infinitesimais e anônimas que viabilizam a vida acadêmica nos campi.

É impossível andar sem que uma das pernas esteja firme no chão. Por isso, agradeço à minha família, porque ajudou-me a firmar a primeira perna e a saber,

mais ou menos, onde pisar com a segunda. E porque sei, apenas sei, que estão aí pro que der e vier. Nos anos mais recentes, para a sorte de todos, temos sido enriquecidos pela presença alegre e irreverente de minha companheira. Obrigado, por ser o contraponto ideal. Agradeço à presença de alguns amigos no dia da minha defesa e à torcida de tantos outros que tinham compromissos inadiáveis, ou que simplesmente não iam entender coisa alguma. Com muitos deles, compartilho a relação de amor e ódio com os computadores, mas com todos, compartilho a alegria de viver com pessoas diferentes que se respeitam, auxiliam e enriquecem.

Outra menção importante segue para a comunidade open source. Formada por um grande número de entusiastas ao redor do globo que trabalha, na maior parte das vezes, em seu tempo livre, este grupo formidável de desenvolvedores inventa (e às vezes, reinventa) ferramentas sensacionais, sem as quais tudo seria mais difícil e (muito) caro.

Meu último agradecimento é para você, leitor, por se importar em ler estas considerações e pelo interesse neste trabalho. Através de você, desejo sorte a todos os que virão e escreverão nas páginas em branco.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ESTIMANDO POSTURAS MANUAIS A PARTIR DE DADOS RGB-D

Pedro de Souza Asad

Maio/2016

Orientador: Ricardo Guerra Marroquim

Programa: Engenharia de Sistemas e Computação

A análise de movimentos humanos baseada em visão computacional é uma área de pesquisa ativa, devido às suas numerosas aplicações e problemas desafiadores que, em geral, não possuem soluções genéricas. Dentro deste amplo domínio, a análise de movimento das mãos se destaca como uma área de mérito próprio, devido à notável importância que elas exercem em múltiplas atividades humanas, como na operação de interfaces humano-máquina, comunicação gestual e linguagem de sinais, para mencionar algumas. Este trabalho se desdobra sobre avanços recentes em visão computacional para a detecção de posturas manuais, no caso particular de câmeras RGB-D. Buscamos reproduzir e aprimorar uma técnica existente que aplica otimização por nuvem de partículas para encaixar um modelo articulado com mais de 20 graus de liberdade a uma mão em movimento observada a partir de um sensor Kinect. Como resultado, construímos um sistema de rastreamento de mãos minuciosamente documentado que difere do trabalho original em alguns aspectos, apesar de manter a mesma estrutura geral. Nosso sistema produz resultados satisfatoriamente precisos, embora seu desempenho tenha sido consideravelmente inferior, devido a restrições de tempo que impediram a otimização de etapas de transferência e conversão de informações.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ESTIMATING HAND POSES FROM RGB-D DATA

Pedro de Souza Asad

May/2016

Advisor: Ricardo Guerra Marroquim

Department: Systems Engineering and Computer Science

Computer vision-based human motion analysis is an active research area, due to its many applications and challenging problems and due to the lack of generic solutions. Inside this broad domain, hand motion analysis arises as a specialized field on its own merit, notably because hands play an important role in many human activities, such as in operation of human-computer interfaces, gestural communication and sign language, to mention a few. This work develops on recent advances of computer vision techniques for hand pose estimation, in the particular case of RGB-D sensors. We sought to reproduce and improve an existing technique that applies particle swarm optimization to fit an articulated hand model with more than 20 degrees of freedom onto a performing hand observed from a Kinect sensor. As a result, we built a thoroughly documented hand tracking software that differs from the original work in some aspects, although keeping the same general layout. It produces satisfactorily accurate results, although its performance was considerably lower, due to time constraints that prevented an optimization of information transfer and manipulation phases.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Humans move. In fact, most living things move in some way. From the small cellular structures that propel many types of bacteria through liquid means, passing by the various tropic movements that plants exhibit, to the complex musculoskeletal systems possessed by vertebrates, the ability to move across the ambient is an evolutionary strategy that has long enabled organisms to interact with one another and with their habitats in intricate and diverse ways. During the course of human history, however, movement has not only been regarded as a vehicle for survival-related activities, but also as an ideal of transcendence and social accomplishment, influencing and being influenced by the development of culture. Nonetheless, in music, dancing, sports and martial arts, for instance, movements must be performed in specific ways in order to bring a sense of adequation, joy or fulfillment.

Not surprisingly, the automatic understanding of human motion by computers has been an active research area for decades. Some notable applications include, but are not limited to: medical diagnosis support, biomechanics research for the purposes of therapy and sports engineering, human-computer interaction, automated surveillance, robot vision and learning, and digital entertainment. The plethora of imaginable applications in this area is only matched, in numbers, by its challenges. Most existing solutions are tailored to particular conditions of applicability, environment, precision needs, motion restrictions and so on. For instance, motion analysis using special clothes with attached magnetic sensors or colored markers that are detected by cameras is very popular in film making due to its precision, but requires complex equipment setups that are unsuitable for the purposes of automated surveillance or casual entertainment.

The present work approaches human hand motion capture as a computer vision problem, in which an end consumer camera is used to observe a performing hand and an articulated hand model is fitted to the observation. Said device is a Kinect for Xbox 360 sensor, a low cost RGB-D camera created for the purpose of interactive digital entertainment that has received a lot of attention from the research community since its launch in 2010, because it provides not only the RGB color components for every pixel, but also a *depth map* (which stands for the D in RGB-D) that encodes orthogonal distance to the image plane, a very valuable information for computer vision algorithms. Our work sought, in the beginning, to reproduce and improve an existing method in the literature that applied the stochastic optimization algorithm known as *particle swarm optimization* (PSO) to obtain the hand model parameters in this scenario. A number of simplifications and adaptations were adopted to reduce implementation effort and provide a solution to details the reference papers were vague about. As a result, we developed a functional hand tracking system, that stems from the original work in a number of ways, reaching reasonable accuracy and robustness. Both an extensive quantitative evaluation of our method and a comparison to the original were not possible due to time constraints, rendering any claims about actual improvements pointless. However, we believe this study to be a relevant contribution to vision-based hand tracking literature, by confirming the viability of the previously existing framework, even under small variations, making some bottlenecks and limitations evident and providing an open source code [1] accompanied by the thorough and self-contained documentation that we believe this text to be.

This document is structured as follows: In the rest of this chapter, we discuss the motivation, scope, methodology and contributions of this work; Chapter 2 presents an overview of the related research literature, with a special emphasis on the method that served as our main reference; Chapter 3 details the implementation of our system; Chapter 4 presents a qualitative discussion of our system's accuracy and a quantitative measure of its performance; Chapter 5 discusses possible solutions to our main issues and suggests possible research directions in which this work might be further extended.

1.1 Objectives

We are motivated by the ideal of producing a reliable, precise and efficient hand motion tracking system that would be able to rival or even surpass a person's ability to deduce hand poses from visual observations. Such a system would, as mentioned before, allow many types of rich interactions and cooperation between human beings and computers. However, as the current state of the art in hand pose estimation suggests, such a goal is far from accomplishment. That being said, with no particular applications in mind, we decided to investigate an existing approach with the intent of reproducing it and, if possible, contributing to its improvement.

The chosen method, as mentioned before, was published by Oikonomidis et al. [1],

[1]Details on Section 1.3.1.

and consists in tracking a single hand by fitting the parameters of an articulated hand model with more than 20 degrees of freedom to the observed color and depth input data provided by a still Kinect sensor. We chose their work because it is reasonably accurate, as reported in their paper and suggested by the demonstration videos in its supplementary material [2] and because its best performance is sub-real time (about 15 Hz, compared to the Kinect's frame rate of 30 Hz). The last point reveals their method is computationally expensive, but also implies that subtle contributions might be able to lift its performance to the real time level. Another potential improvement regards hand pose stability, since the tracked poses tend to present jerky oscillations, even when the performing hand is still.

1.2 Methodology

We took a mostly experimental approach to the development and testing of our hand tracking system. The work we attempted to reproduce was divided in a number of steps, detailed in Section 2.3, and we implemented each one of them, from image acquisition to pose optimization. Every newly developed functionality was tested for correctness against third party datasets or video sequences recorded in our laboratory, as described in Section 4.1. When fixing errors or adjusting parameters, we aimed at approximating the observed hand poses in our videos, assessing the result's quality by visual inspection.

As mentioned before, we initially planned to reproduce an existing system before investigating possible improvements. However, faithful reproductions in computer vision are notoriously difficult, since apparently irrelevant environmental conditions and seemingly secondary parameters may considerably impact the quality and even the viability of the final solution, a fact that is frequently overlooked in published papers. This led us to: (a) adopt different parameters than the reference work for various parts in order to obtain consistent results; (b) simplify the algorithmic treatment where our scenario constraints allowed to and (c) develop custom solutions to details that were not clearly explained in the references. The three most distinctive aspects of our system are: (a) skin segmentation, which is approached by Section 3.2.3; (b) the adopted articulated hand model, that includes less DOFs than the original work and (c) a slightly different formulation of the objective function used for evaluating pose hypotheses, which is described in Section 3.5.2. All configurable system parameters are described in Appendix B.

A fundamental discipline of software development is planing the *system architecture*. A carefully planned architecture allows easy inclusion of additional functionality, replacement or removal of old modules and behavior multiplexing by combining exchangeable components. The detailing of an architecture usually leads to adopting multiple *design patterns*, that is, general solutions for common situations and problems in programming. Most literature on software architecture and design patterns is focused on enterprise and market solutions and not on academic necessities, where we perceive the adoption of such disciplines to be very little disseminated, in the present time. For the most part, we followed and *ad-hoc* approach to the architecture of our system, but applied some of the classical design patterns in the book [3] to improve the system's structure and the code's readability.

Our system was implemented mainly in the version 11 of the C++ language [4], a language that is commonplace in systems programming due to its performance features and extensive library support. The depth maps used to evaluate pose hypotheses during the optimization phase, as explained in Section 3.4, were generated by rendering a hand mesh model with custom vertex and fragment shaders in OpenGL Shading Language (GLSL) 4.0 [5], an open and extremely portable standard for 3D graphics programming. Finally, the GPU programs used for skin segmentation and hand detection, explained in Section 3.2.3, and for hypothesis evaluation, explained in Section 3.5.2, were implemented with the NVIDIA CUDA toolkit 7.5 [6], which permits better usage of the NVIDIA GPU our test machine was equipped with. The references provided in this paragraph are for extensive programming guides of these languages that we used frequently during this study. The reader seeking shorter, comprehensive introductions to these technologies may find many such materials in the web.

1.3 Results and contributions

The result of our work is a functional hand tracking application that is able to fit a 22 DOF articulated hand model to a single performing hand observed with a Kinect sensor. Our application allows to visualize the input video and depth frames overlaid with the rendered hand model in the tracked pose. It was based on an existing optimization-oriented approach by Oikonomidis et al. [1] and it differs from their method regarding hand detection and isolation, hand model constraints and a few more algorithmic details in later phases.

From an empirical perspective, visual inspection indicates that our implementation gives reasonably accurate results. Furthermore, our method starts tracking from an open palm pose in a fixed position in space that, in principle, may be misaligned with the performing hand, but nonetheless was able to translate the model to the proper location and alignment in all tested video sequences. It can be noted from our video sequences that rapid motion and hand poses in which the palm plane is nearly perpendicular to the camera's image plane are unfavorable to our method, although we believe this restriction to be inherent to the formulation of the optimization phase, as with the original work.

Quantitative evaluation in this area is usually performed by evaluating the target method against a synthetic dataset, due to the unavailability of annotated datasets of real hand motion. Such approach was followed by the reference work, but it does not detail how to synthesize motion sequences that reflect plausible motion. For that reason and due to time constraints, we provide no quantitative evaluation of our method, nor a comparison against the other one. In Chapter 4, we detail the experiments we performed and provide a quantitative measure of our system's performance. Being very close in structure to the reference work, we believe this performance evaluation to be a valuable information for similar implementations. Our schedule did not allow to focus extensively in optimizations and hence, our system did not reach desired performance, falling considerably behind the reference work in the final evaluation. However, one of our late analysis, as explained in Chapter 4, showed our major bottleneck was an inefficient conversion of rendered depth values during particle evaluation phase that could be accelerated with trickier usage of the graphics programming APIs involved, meaning our implementation is indeed close to obtaining comparable performance.

Finally, we believe this detailed document to be a valuable resource for implementations of similar systems. Being, in many senses, a variation of the reference work, it can guide the implementation of other derivative works. Our source code was also published in open source terms, meaning it may contribute to future research by providing a starting point or a reference for comparison. We furthermore intend to keep developing this technique and publish additional improvements to the code under the same terms.

1.3.1 Source code

The source code's development history was recorded with the `git` revision control system and its latest version can be found at

`https://gitlab.com/psa-exe/hand-tracking`

It is released under the MIT license (see [7] for more details), which is included with the code. The latest version of the code up to the conclusion of this text has version number 0.1 and is marked with the corresponding `v0.1` tag in the repository.

Chapter 2

Related work

Marker-based human pose estimation using body markers and colored gloves, as usually applied to motion capture, provides good overall accuracy. However, these devices are usually expensive and cumbersome, requiring users to break the interaction flow in order to wear them and limiting their application in some scenarios, such as automated surveillance and robot vision. Thus, obtaining reliable computer vision-based solutions for this problem is considered important. In this chapter, we provide an overview of computer vision-based human hand pose estimation approaches, with emphasis on methods that provide continuous tracking of a fully articulated hand model with the aid of a Kinect depth sensor. We also describe some studies in the area of skin detection that are relevant to our work. The last section is dedicated to describing our main references in more detail.

2.1 Vision-based hand pose estimation

Hands constitute a special case of the broader field of human motion analysis that presents its own challenges, such as highly dynamic movement, many degrees of freedom and frequent self occlusions, compared to other body parts. A comprehensive and singularly important survey on vision-based hand pose estimation research published until 2007 is given by [8]. Since human skeleton information is frequently used as input for gesture recognition systems, there are more surveys that cover the higher level applications of hand gesture recognition, such as [9]. Since we focus on pose estimation, regardless of the application, we also point at [10], which reviews general computer vision enhanced by the Kinect sensor, but dedicates a whole section to the latest advancements in hand pose recognition using this device. Throughout this chapter we also review other relevant and recent studies in this area that are not covered in any of these surveys.

The typical taxonomy of human pose estimation literature[8, 9] divides techniques in three main categories: (a) appearance-based (or discriminative), (b) model-based (or generative) and (c) hybrid. Appearance-based methods rely on machine learning frameworks, such as random forests and neural networks to map a set of observed features to a discrete pose space. They usually require an extensive training phase where numerous pairs of ground truth pose labels and corresponding observed features must be supplied, but once trained these methods are able to operate with real time performance. By performing frame-wise recognition, thy become resilient to estimation errors, since an incorrect pose is not propagated through the tracking sequence. On the other hand, they are unlikely to extrapolate their training sets and predict completely unknown poses. Moreover, pose transitions are not smooth.

Model-based methods are optimization-driven, as they iteratively improve a one or multiple hypotheses by reducing their estimated distance to the observed features. Successfully applied optimization algorithms include Gauss-Newton, bounded linear programming, spring-force systems, simulated annealing, genetic algorithms and particle swarm optimization. The elevated dimensionality of highly articulated hand models (usually 20 to 30 degrees of freedom per hand) raises the computation costs of such algorithms and requires careful design and incorporation of domain information, such as joint rotation constraints and the temporal coherence assumption, to reach acceptable performance and avoid local minima. On the other side, infinite combinations of pose parameters are possible. Temporal coherence is an important factor for performance improvement and smooth transitions that can be achieved by initializing the set of hypotheses from the previously deduced pose. On the other hand, it also renders them fragile to accumulation of tracking errors. Due to the size of the search space, initialization is non-trivial, but can be achieved by performing a predefined pose or with the aid of appearance-based recognition, for instance. In the next section, we detail model-based estimation.

The third category, that of hybrid methods, encompasses approaches that attempt to bring together the strongest features of the generative and discriminative worlds. The discriminative portion may be responsible for initialization, recovery, pruning of the search space or feature extraction while the optimization part is usually responsible for refining the coarse estimates.

2.2 Model-based hand pose estimation

In the past, one-shot hand pose recognition has been achieved with the help of a glove with colored markers and by solving an inverse kinematics problem, where joint motion is highly constrained[11]. Model fitting was driven by a spring-like force model in which some characteristic model points (consisting mainly of fingertips) were attracted to their likely spatial locations, estimated from a pair of calibrated cameras through feature detection.

Other works apply stochastic optimization to find the pose that better fits the observations. Particle Swarm Optimization (PSO), for instance, has been successfully applied for finding the 3D configuration of one [1, 12] and even two [13] interacting hand models. An objective function is applied to measure the discrepancy between each particle (model pose hypothesis) and the observed depth map. Then, every particle is updated according to its momentum, plus a random and a drifting component towards the current best particle at every iteration. This type of PSO-based technique may be even tuned to track a hand interacting with a simple rigid object [14]. PSO has also been successfully applied to other pose estimation problems such as head pose estimation[15].

Another course of action in estimating the hand pose is combining optical flow with salient points in order to obtain good descriptors for the fingers, since they usually present the problem of great self-similarity [16].

2.2.1 Hand modeling

One of the biggest issues with generative estimation of hand postures based on global search is that the high dimensionality of the search space (usually 20 to 30 degrees of freedom for a single hand) considerably raises the computational costs of finding a good solution. Besides, one incorrectly estimated parameter may cause an overall anatomically implausible configuration, since the articulated parts are hierarchically connected. To address both issues, some methods[11, 17, 18] apply constraints on joint movements to prohibit anatomically implausible motion, enforce naturalness of motion or both.

Applying finger motion constraints may drastically reduce the search space, but sometimes the computational requirements become a serious drawback[17]. Moreover, whereas the simplest constraints may be represented as equations and many as inequalities, many others simply cannot, specially the ones that enforce *natural motion* (produced solely by the hand's own motor structures under healthy conditions, and not involving external agents such as objects or other hands, for instance). In this case, dimensionality reduction may be employed to learn the subspace of natural motion from a set of training poses.

Some works go even further in the recreation of a realistic hand by modeling bones, muscles, skin and anatomic motion [19, 20]. Such highly realistic models are very suitable for augmented reality and animation systems, but present an exaggerated level of detail for applications where only articulated tracking is of importance.

Some works prune the search space by roughly identifying the position and/or orientation of the palm before applying a tight estimation of the palm and finger configurations[11, 21]. In [21], a small accelerometer is attached to the person's hand to obtain its orientation.

2.2.2 Kinect-based pose estimation

The arrival of the Kinect sensor[22] in 2010 made incorporating depth information into computer vision pipelines easier and cheaper than traditional stereo or Time of Flight cameras [10]. Many computer vision domains have since benefited greatly from this additional spatial dimension, including indoor scanning applications, person detection, gesture recognition, activity recognition and object tracking, for instance. An excellent survey on computer vision methods enhanced with Kinect, including pose estimation systems, is given in [10]. For a survey on depth-enhanced human motion analysis, which includes other types of sensors, the reader is referred to [23]. A brief description of the sensor's architecture is given in Appendix A.

One of the seminal papers in Kinect-based human pose tracking, which reveals the method behind the device's *Software Development Kit* (SDK), is based on a discriminative classification of body parts at pixel-level [24]. It describes a random forest classifier that maps a set of local depth-derived features into body part labels. After per pixel labeling is performed, mean shift mode finding is applied to estimate the joint positions, which constitute the output of the method. The classifier is trained with a huge exemplar set of actors of different body types performing varied actions, rendering the system robust against body shape variations and making it suitable for many entertainment scenarios. The resulting algorithm is fast (able to cope with the Kinect's framerate) and resilient, since it operates on individual frames separately. However, this method outputs a coarse skeleton which is not suitable for hand pose tracking, since the greatest level of detail for the hands contains only the palm center (end point) and wrist (joint). Another disadvantage is that it took an extensive training set to train the body part classifier.

Two systems modify the previous approach to the context of hand pose recognition, by training a random decision forest with synthesized depth images of an articulated hand model, which classifies depth pixels as belonging to one of 21 manually chosen hand parts[25, 26]. The latter study improves on the former by achieving realtime performance and better accuracy. Although a quantitative measure of the system's accuracy is not given, when combined with an SVM (support vector machine) module the resulting system achieves the ten digits of American Sign Language with correct gesture detection rate of 99.9%.

Another part-based hand pose recognition method uses distance transform to locate the palm center and fingertips and determine if they are extended or bent [27]. However, this simplified finger pose model does not translate into rotation parameters for an articulated model, rendering this method more useful for simple gesture recognition and mostly limiting its application to frontal hand poses.

2.2.3 Skin detection and hand segmentation

A fundamental step in hand pose tracking is segmenting the hands from the rest of the scene. Various simplifying conditions are usually assumed, such as the hands being the closest body part to the camera or the background being static. When depth information is available, it may be used alone or in conjunction with RGB data to segment the hands [28–30]. Nevertheless, it is also possible to use only color information. Video-based hand segmentation usually resorts to color-based skin segmentation and given the simplifying conditions, it achieves satisfactory results.

The skin segmentation problem is usually formulated as a two-class separation of image pixels, where the possible classes are *skin* and *non skin*. Various classification strategies are possible, including, but not limited to: simple range checking (a pixel is considered to be skin if, and only if its color components are within predefined ranges); Bayesian filtering based on color histograms; simple and multimodal Gaussian classification and several neural network approaches. Consult [31] for a comprehensive survey on color-based skin classification methods.

Except for non-parametric approaches, such as histogram-based ones, classification methods, such as Gaussian and range checking, require specific color representations to work. Surprisingly, the skin colors of different ethnic groups have been found to differ more in luminance than in chrominance [32], which led many researchers to adopt color spaces in which the luminance component is separated (HSV, YUV and Lab, for instance) and can be discarded. Although discarding the illumination information improves robustness against lighting variations, it has been found that it degrades the accuracy of predictions [31]. Nonetheless, color spaces like RGB and CIE-XYZ may be effectively employed in skin detection. A common practice when using the RGB color space is to normalize the (r, g, b) color components, such that $r + g + b = 1$, which reduces illumination dependency and allows for dimensionality reduction, since one of the components becomes redundant. Comparative studies of color spaces used in skin classification can be found in [33–35].

2.3 Overview of main references

In 2011, [1] presented a markerless vision-based method for retrieving the full 3D articulated pose of a moving human hand from a RGB-D camera by means of Particle Swarm Optimization. This method segments the observed hand and compares its skin and depth information against synthesized images of an articulated hand model in several hypothesized configurations, or *particles*. These candidate solutions are improved over a fixed number of iterations and the best ever matching pose for the present frame is chosen. Temporal coherence is achieved by using each frame's solution to spawn the particle swarm for the next one.

We give a brief overview of the system as follows: (a) hands are detected by means of color-based skin segmentation, yielding a binary mask of observed skin pixels S_O; (b) based on the estimated hand position for the previous frame, the input depth is filtered to remove irrelevant pixel depths, and an observed depth map D_O is obtained; (c) the particle swarm is initialized with pose hypotheses derived from the solution of the previous frame; (d) each hypothesis $\mathbf{h}$ applied to the hand model and rendered with the estimated camera parameters for the Kinect depth camera, producing a rendered depth map $D_R(\mathbf{h})$; (e) each hypothesis is compared against the observation $O = (S_O, D_O)$ by means of an objective function $F(\mathbf{h}, O)$; (f) particle positions and velocities are updated according to PSO algorithm and steps (d)-(f) are repeated for a fixed number of iterations. At the end of the last iteration, the best hypothesis produced is taken as the final solution for that frame. This high level view of the system is demonstrated by Figure 2.1.

This algorithm has been extended to work with multiple hands [13] and with hands interacting with objects[14]. As with most model-based approaches, this method still requires careful initialization and is prone to error accumulation if a bad pose is chosen at some time, specially under fast motion. Nevertheless, it presents a valuable indication that accurate, smooth and efficient model-based hand pose tracking through particle swarm optimization is, to some degree, feasible, with the currently available hardware platforms. It is the main reference for our work, hence it will be further described in the current section, together with closely related studies. We start by describing the modeling of hand kinematics, proceed to shape representation, then hand detection and finally, to pose estimation.

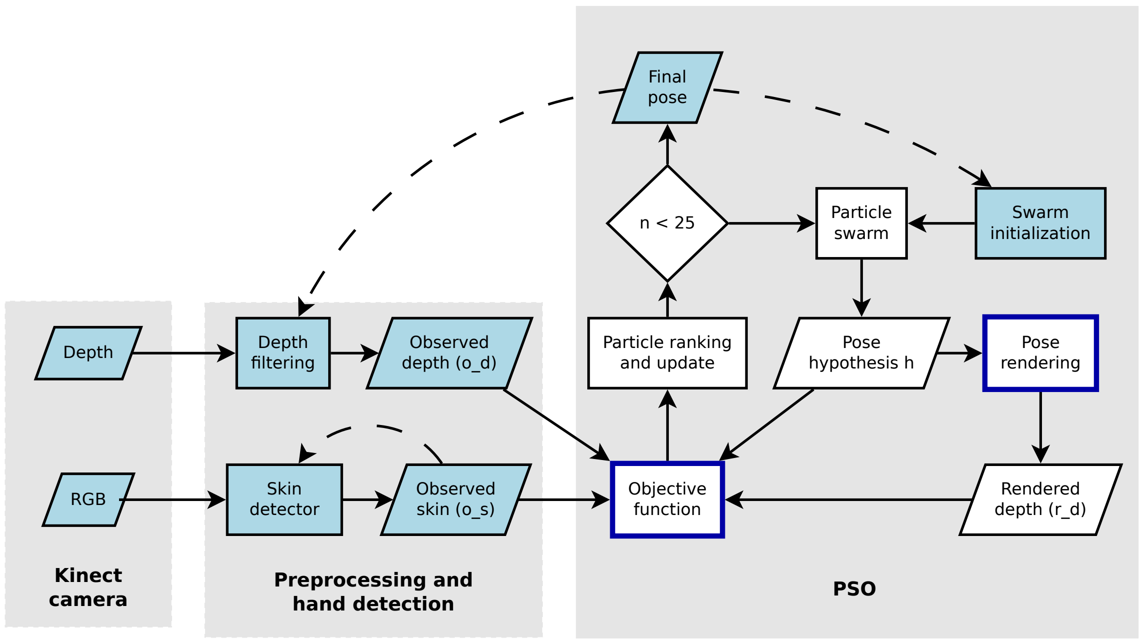


Figure 2.1: Overview of the pose estimation system developed in the main reference. Parallelograms indicate input, output and intermediate data, rectangles indicate modules/steps performed by the algorithm and the diamond indicates iteration control. Light blue boxes indicate data or steps that occur only once per frame. Dashed arrows indicate information that is used as feedback in the next frame. Rectangles with a blue outline indicate steps that are computed on the GPU.

2.3.1 Hand kinematics modeling

The human hand is a sophisticated biomechanical structure comprised of 27 bones: 14 phalanxes (the finger bones), 5 metacarpals (the bones that form the palm center and connect to the phalanxes) and 8 carpals (the ones that connect the metacarpals to the wrist). Figure 2.2 shows a picture of a right human hand skeleton with the names of the main bones [36].

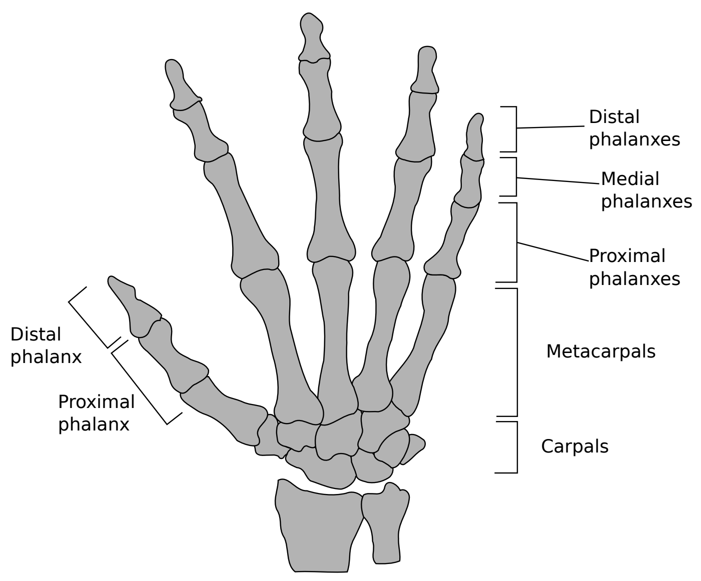


Figure 2.2: Right hand skeleton and main bone names. This is a free-hand reproduction of the skeleton picture in [11]. asçdfhaçs qwerqkçlwje q qwlekrjqwlke qwlekrjhqwlekr qwlekrjhqwelrk qwelkrjhqwer qlkwejrhqlwekr qwe rlkqjwehr qwer lkqjwehr

From an anatomic perspective, joints are contact surfaces between adjacent bones that allow them to rotate around one another [37]. From a kinematic perspective that is usually adopted [8], the hand may be modeled as a tree-like structure of undeformable parts connected at articulation points that allow for rotating around up to three orthogonal axes. From this last point of view, joints correspond to rotation centers and it is as such that we employ the term *joint* for the rest of this text. Figure 2.3 shows the joint names for the human hand and displays how many degrees of freedom (DOFs) are adopted for each in the main reference. While most of the carpals and metacarpals present little to no perceptible reciprocal motion at all (with the notable exception of the thumb metacarpal) and are left as part of a rigid palm part, the phalanxes are capable of very intricate movements. The most noticeable movements are those of flexion/extension (FE) and adduction/abduction (AA): the first revolves around axes that are parallel to the image plane and brings the phalanxes closer to or further from the palm; the second is only possessed by the index through little metacarpophalangeal and thumb carpometacarpal joints and rotates around axes perpendicular to the image plane, bringing adjacent fingers closer to or further from each other. Rotation around a third orthogonal axis, usually referred to as torsion, is too small and is usually ignored. In the case of the thumb, the FE axis is not parallel to the image plane and the AA axis is not perpendicular to it. Rather, the thumb joints' coordinate systems are slightly rotated around their torsion axes by an equal amount for all of them. The FE axes of one finger are always aligned, making each finger a planar manipulator.

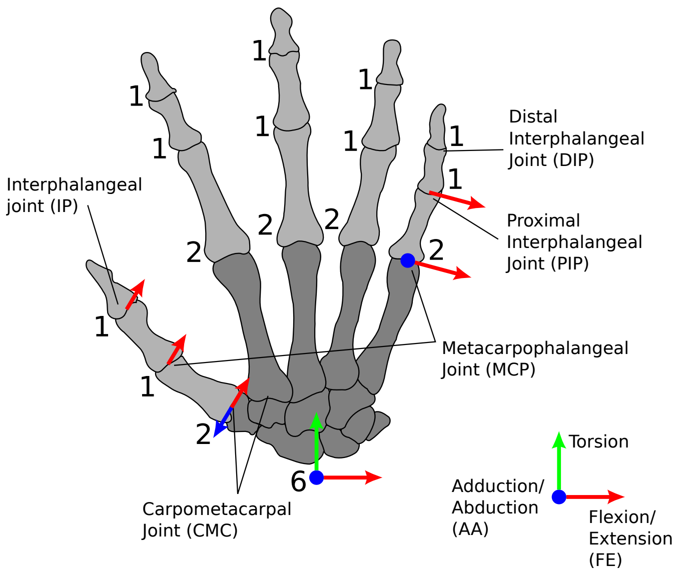

1
1
1
1
1
1
1
1
Distal Interphalangeal Joint (DIP)
Interphalangeal joint (IP)
Proximal Interphalangeal Joint (PIP)
2
2
2
2
1
1
Metacarpophalangeal Joint (MCP)
2
6
Torsion
Adduction/ Abduction (AA)
Flexion/ Extension (FE)
Carpometacarpal Joint (CMC)

Figure 2.3: Degrees of freedom most commonly presented in kinematic hand models. The number of DOFs is shown close to each joint and non-moving bones are painted in a darker shade of gray than the moving ones. AA and FE axes are depicted in blue and red, respectively, for the little finger's MCP and PIP joints and are omitted for the remaining joints, because joints with the same number of DOFs have similar axes. The axes for the thumb joints appear shorter, to suggest that the AA axes of the thumb joints' are not parallel to the other fingers'. A set of three axes is shown next to the wrist, which, including 3 translation coordinates, contains 6 DOFs. The key on the bottom right shows the names of the joint movements next to the respective color-coded rotation axes.

The most difficult finger motion to be modeled is that of the thumb carpometacarpal joint, because, from an anatomic perspective, it actually rotates around two non-orthogonal and non-intersecting axes [11], unlike the presented AA-FE orthogonal system, allowing the thumb to face the palm when gripping objects. Some works apply include the torsion movement to solve this issue, while others (such as the main reference) assume a more restrictive and less realistic model with indeed two orthogonal axes (like the other fingers' metacarpophalangeal joints). Finally, it is also possible to assume three orthogonal axes by modeling the torsion angle as a linear combination of the AA and FE angles. The resulting typical hand model has 26 or 27 DOFs, depending on the choice for the thumb [36].

The highest node in the movement hierarchy corresponds to the *wrist*, that is, the joint between the hand and the forearm, which takes 3 translation DOFs to be placed in space and another 3 DOFs to describe the hand orientation. Although the hand is not actually capable of unrestricted 3-axial rotation relative to the forearm, when the arm kinematics are not considered separately, the resulting wrist model has these total 6 DOFs.

As Figure 2.3 indicates, the most commonly used kinematic models allow for up to 20 or 21 degrees of finger motion (although more are possible), depending on the joint type used for the thumb CMC joint. But the complex structure of the human hand, albeit highly articulated, also causes its movement to be very constrained. Hence, under the assumption of realistic motion, it is reasonable to restrict the joint movements as a way to avoid implausible poses and reduce the size of the search space. The set of all possible joint configurations, considering the adopted constraints, is called *pose space*.

According to [11], hand part motion may be either *active* or *passive*. Active motion of any part may result only from the sole action of muscles and tendons connected to that part or from the chained active motion of its ancestor parts. Passive motion, on the contrary, includes motion performed under the influence of external forces, such as twisting fingers or bending them backwards with the help of another hand or by pressing against objects, for instance. Active motion constitutes a subset of passive motion, as the latter allows for a much greater motion amplitude.

Joint constraints for regulating both types of movements were classified as either *static* or *dynamic* by the same authors and a third type of constraint that we will call *natural* was described by [17]. Static constraints correspond to permitted value ranges for joint angles, and may be represented by inequalities of the form $\theta_{min} \leq \theta \leq \theta_{max}$. Dynamic constraints model the movement dependency between moving finger parts or adjacent fingers and reflect the overlapping action of muscles and tendons and their limited flexibility. The facts that most people are unable bend the little finger without also bending the ring one or that it is usually hard to bend or extend any distal phalanx without doing the same with the corresponding medial phalanx are examples of dynamic constraints. Dynamic constraints also regulate how one finger may limit the static amplitude of adjacent fingers even further when it moves. Natural constraints have not received much attention in the literature, as they correspond to subtle relations among moving fingers that cannot be easily described in closed equation/inequation form. A possible way to approach them is applying machine learning techniques to learn the subspace of natural motion from glove-captured motion data [17]. Finally, we note that static constraints considerably reduce the parameter ranges, while some dynamic constraints may be used to reduce these ranges further and others to remove redundant dimensions. Since only static constraints are used by the reference method, we defer a deeper discussion of dynamic constraints until Chapter 3.

Another important aspect of hand kinematics in hand modeling are the causes of finger motion. One possibility is to model the mechanics of musculotendon interaction with bones and move fingers as a result of applying forces to certain muscles, which has been demonstrated in the context of realistic hand animation [19]. Another approach consists in applying spring-like forces to the model fingertips in order to attract them to their estimated spatial positions and estimate the joint

MCP		PIP	DIP
θ_{AA}	θ_{FE}	θ_{FE}	θ_{FE}
$[-15°, 15°]$	$[0°, 90°]$	$[0°, 110°]$	$[0°, 90°]$

Table 2.1: Static constraints on finger DOFs The same constraints as in [17] are presented for the index, middle, ring and little fingers. The thumb CMC joint has the same constraints as the other finger's MCP joints and its MCP joint has no adduction/abduction movement and the flexion amplitude is the same as the other finger's PIPs.

parameters through inverse kinematics of the finger chains [11], which assumes that the palm location and orientation and the fingertip positions may be estimated directly from the image. A third possibility is to ignore the causes of motion and adopt specific strategies for exploring the pose space, which is the usual choice for optimization-based pose estimation algorithms, such as the main reference.

In [1, 13, 14], no twisting of fingers is considered, the metacarpals are fixed and the thumb CMC joint is modeled as a saddle-like joint (2 DOFs). Besides, only static constraints similar to the ones presented in [17] and summarized in Table 2.1 are considered. Interpenetration of adjacent fingers is not dynamically constrained, but rather penalized by the objective function formulation. Their resulting hand model has 20 finger DOFs that, combined with the 3D position and orientation of the wrist, the latter represented as a quaternion, constitutes a 27 dimensional pose space.

2.3.2 Hand shape modeling

Hand shape models include triangle meshes, quadrics, B-spline surfaces and even more complicated mesh models that allow for realistic skin deformation [19, 20]. However, the shape model employed by the main reference is geometrically simple: it is assembled from 37 triangle meshes, 15 being triangulations of cylinders and 22 of ellipsoids. In the present work, we implemented an analogous model using the same types of meshes for each respective hand part. Our model is further described from a kinematic point of view in Section 3.3.2, but we present it here in Figure 2.4 as part of the current discussion.

2.3.3 Hand detection

In [1], a single hand is tracked, and it is assumed to be the closest skin-colored object moving in the camera's field of view. As such, hand detection amounts to skin segmentation and connected component labeling. The skin segmentation technique employed is the same of [38], where color histograms in the YUV space (without the Y component) are used to classify pixels with color $\mathbf{c} = [u, v]^\intercal$ as belonging to

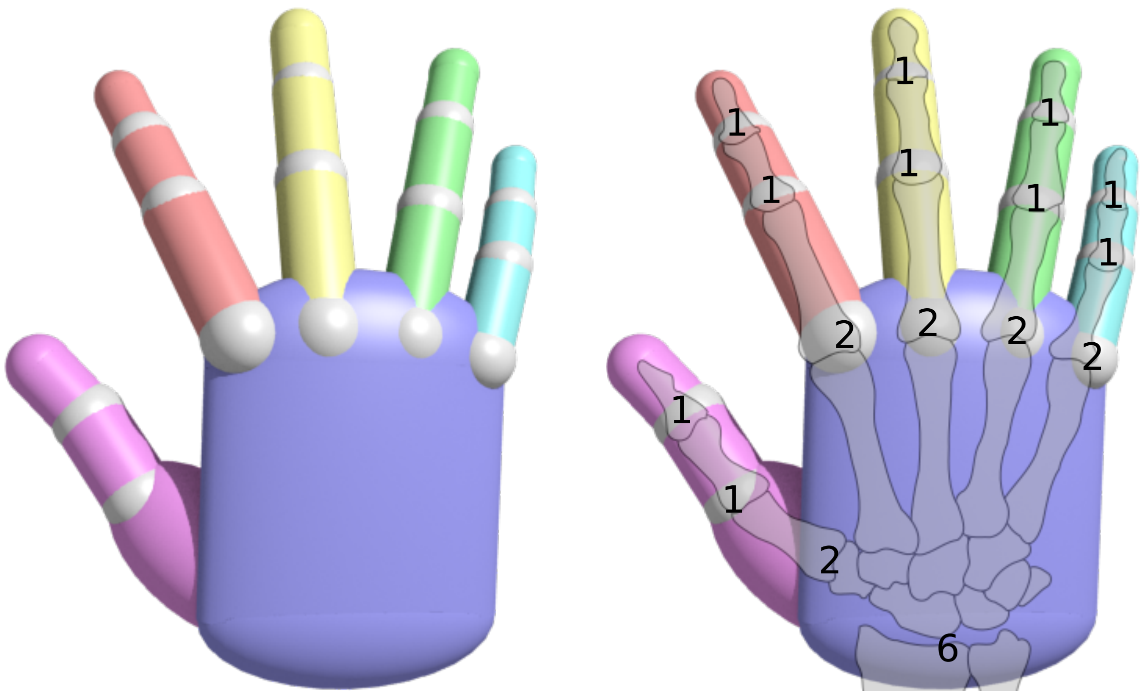


Figure 2.4: Our right hand shape model On the left, we show a top view of our shape model, inspired by [1]. On the right, we show the same picture, overlaid with the same hand skeleton of Figure 2.2 and with the matching DOF numbers. This shape mode is composed of mesh triangulations of 15 cylinders and 22 ellipsoids.

either the skin or non-skin classes, denoted by $\mathfrak{s}$ and $\bar{\mathfrak{s}}$, respectively, according to Bayes rule:

$$P(\mathfrak{s}|\mathbf{c}) = \frac{P(\mathbf{c}|\mathfrak{s})P(\mathfrak{s})}{P(\mathbf{c})} \tag{2.1}$$

where $P(\mathfrak{s}|\mathbf{c})$ is the *posterior probability* of $\mathbf{c}$ belonging to $\mathfrak{s}$, $P(\mathbf{c}|\mathfrak{s})$ is the *likelihood* of $\mathbf{c}$ among skin colors, $P(\mathfrak{s})$ is the *prior probability* of finding skin pixels (the expected ratio of $\mathfrak{s}$ in the observations) and $P(\mathbf{c})$ is the *evidence probability*, that is, the probability of observing color $\mathbf{c}$ among all others.

The posterior probability in the previous equation is calculated for each pixel and compared against two hysteresis thresholds $\tau_{\mathfrak{s}}$ and $\tau_{\bar{\mathfrak{s}}}$, with $\tau_{\mathfrak{s}} > \tau_{\bar{\mathfrak{s}}}$. A pixel $\mathbf{c}$ is considered to belong to $\mathfrak{s}$ if $P(\mathfrak{s}|\mathbf{c}) > \tau_{\mathfrak{s}}$ or if $P(\mathfrak{s}|\mathbf{c}) > \tau_{\bar{\mathfrak{s}}}$ and it is neighbor to some skin pixel, where the definition applies recursively. The resulting skin pixels are grouped in connected components and the components with less than a certain number τ_B of pixels are filtered out. Figure 2.5 illustrates this process.

Given the simplifying assumptions in [1], all blobs but the largest are discarded. But when dealing with more complicated scenarios where multiple hands [13], hands interacting with objects [14] or hands subject to partial occlusion are tracked, the full technique of [38] is more suitable, as it manages object hypotheses (creates, updates, evaluates against connected components and removes when needed) to track observed blobs while dealing with occlusion.

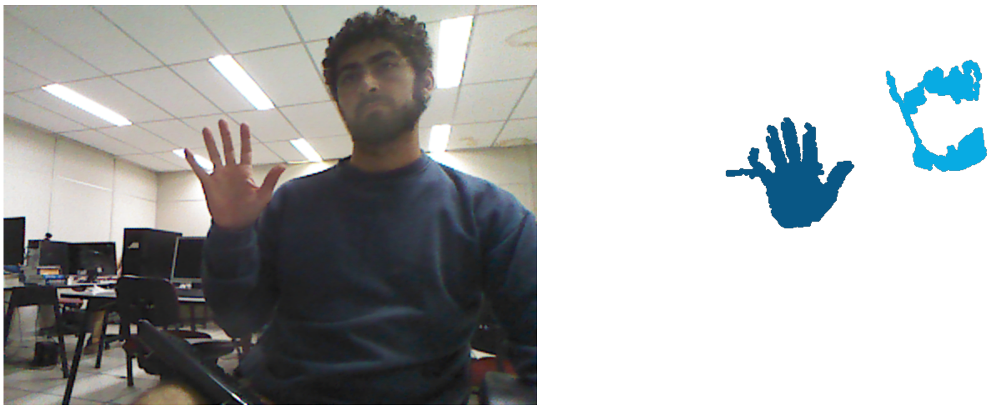

(a) Source RGB frame. (b) Resulting skin blobs.

Figure 2.5: Skin blobs produced during hand detection.
Only the two larger blobs remain, the largest of which corresponds to the hand. This example is repeated in Section 3.2.3.

Two final steps are required to obtain the binary mask of observed skin pixels S_O and isolate the relevant observed pixel depths D_O, in [1]: (a) the selected blob is dilated with a circular mask of radius 5 and (b) given the estimated 3D position of the hand in the previous frame, all depth information further than 25 cm of the hand model's center is discarded. The authors are not precise as to how pixel depths are discarded during initialization phase (the first frame) or what model point is considered to be the hand's center. Together, the pair of images $O = (S_O, D_O)$ constitutes the output of this phase and is fed to the pose estimation phase in order to calculate the objective function.

Calculation of skin histograms

One interesting contribution of [38] is the development of a semi-automatic skin classifier training procedure. Since manual annotation of skin pixels in data sets is a laborious task, color histograms are updated iteratively as follows: (a) an arbitrarily small non-empty set of ground truth images is provided by a human operator; (b) the system computes the histograms based on the input; (c) the system automatically segments further images provided; (d) the operator may manually fix incorrectly classified pixels; (e) color histograms are updated in order to segment the next image (if any). This procedure may be carried on until the operator is satisfied with the classifier results. The authors report an initial training set of 20 images and 80 additional images used in their experiments, with no quantitative results.

In order to make the system more robust against illumination changes, a common challenge for skin detectors, two classifiers are actually employed: the offline classifier trained with the previously described procedure and an online classifier that is updated assuming detected skin is correctly classified. When comparing the posterior probability of a pixel color being a skin color with the hysteresis thresholds, a linear combination of both classifiers is actually used:

$$P(\mathfrak{s}|\mathbf{c}) = \gamma P(\mathfrak{s}|\mathbf{c}) + (1-\gamma)P_w(\mathfrak{s}|\mathbf{c}) \tag{2.2}$$

where P_w comes from the online classifier, considering only the w most recent frames and the $\gamma \in [0,1]$ parameter controls the influence of the adaptive part. The reported values for the authors' experiments are $w = 5$ and $\gamma = 0.8$.

2.3.4 Pose estimation

Hypothesis evaluation

Assuming the Kinect's depth camera calibration matrix is known, a depth map $D_R(\mathbf{h})$ of the same size as the input frames is generated for each hand pose hypothesis $\mathbf{h}$ by rendering the articulated hand model parameterized by $\mathbf{h}$ with a virtual camera setup that mimics the known camera matrix. The rendered depth map is then compared to the observed data $O = (S_O, D_O)$ through the GPU-accelerated calculation of the energy function below

$$F_{12} = \lambda_A \frac{\sum \min\{|D_O - D_R|, \tau_{M_2}\}}{\sum (S_O \vee D_M) + \epsilon_1} + \left(1 - \lambda_B \frac{2\sum(S_O \wedge D_M)}{\sum(S_O \wedge D_M + S_O \vee D_M)}\right) \tag{2.3}$$

where ϵ is described as a small constant to avoid division by zero, λ is a scaling factor and $D_M(h, C)$ is a binary map of matched depths between D_R and D_O, defined as

$$D_M(O) = \begin{cases} 1, \text{ if } |D_R - D_O| < \tau_{M_1} \text{ or } D_O = 0 \\ 0, \text{ otherwise} \end{cases} \tag{2.4}$$

that is, each pair of pixels in the depth maps is considered to be a match (value 1) if the depth difference between rendered (D_R) and observed (D_O) is smaller than the threshold τ_{M_1} or if the observed data is reported as unknown by the depth camera ($D_O = 0$).

The complete evaluation of hypothesis $\mathbf{h}$ contains another term $F_2(\mathbf{h})$, which is responsible for penalizing anatomically unfeasible poses and is defined as the clamped sum of adduction-to-abduction angle differences between adjacent fingers, that is

$$F_K = -\sum_{p \in Q} \min\{\Theta(p, \mathbf{h}), 0\} \tag{2.5}$$

where Q corresponds to the three pairs of adjacent fingers excluding the thumb and $\Theta(p, \mathbf{h})$ is the signed angle difference between the angles of finger pair p in $\mathbf{h}$. The complete formulation of the objective function is then given by

$$F(\mathbf{h}, O) = F_{12}(\mathbf{h}, O) + \lambda_K F_K(\mathbf{h}, O) \tag{2.6}$$

where λ_3 is yet another scaling factor.

Particle update

Each particle in the swarm corresponds to a point $\mathbf{h}$ in the pose space $\mathcal{P}$. As several iterations, or *generations*, are computed, they move across the pose space influenced by: (a) inertia, (b) individual memory of best solution found so far, (c) their perception of the fittest particle and (d) subtle random perturbations that occur at regular intervals. Individual particle memory at k-th generation is represented as a vector ${}^{(k)}\mathbf{b}_i$ that corresponds to the i-th particle's best scoring position so far and the fittest particle, indicated by ${}^{(k)}\mathbf{b}$, is shared across the whole swarm. At every generation, particle velocities are steered towards ${}^{(k)}\mathbf{b}_i$ and ${}^{(k)}\mathbf{b}$ by random proportions, contributing to exploratory behavior, as outlined in the following equations

$$\begin{align} {}^{(k+1)}\mathbf{v}_i &= w\left[{}^{(k)}\mathbf{v}_i + c_1 r_1\left({}^{(k)}\mathbf{b}_i - {}^{(k)}\mathbf{h}_i\right) + c_2 r_2\left({}^{(k)}\mathbf{b} - {}^{(k)}\mathbf{h}_i\right)\right] \tag{2.7} \\ {}^{(k+1)}\mathbf{h}_i &= {}^{(k)}\mathbf{h}_i + {}^{(k+1)}\mathbf{v}_i \tag{2.8} \end{align}$$

where r_1, r_2 are two independent uniformly distributed variables in the range $[0, 1]$ and c_1, the *cognitive component*, and c_2, the *social component*, control the maximum influence of the per-particle memory and the global best solution, respectively, on each particle's velocity. Because of these random components, the resulting optimization scheme may be characterized as a stochastic algorithm, since separate executions may obtain different results. The adoption of the w constriction factor, which assumes values in the $(0, 1]$ interval (as can be deduced from its definition in Equation 2.9) limits velocity magnitude, so that the chances of particles leaping over good solutions are reduced. The authors of the reference work mention that determining these parameters may be achieved experimentally, but do not give further details. Consequently, we adopted the same values, described in Appendix B.

$$w = \frac{2}{|2 - \psi - \sqrt{\psi^2 - 4\psi}|}, \text{ with } \psi = c_1 + c_2 \tag{2.9}$$

In order to respect static joint constraints, updated particles must be corrected if they violate the permitted ranges after applying Equation 2.8. The adopted correction method was projecting to the nearest pose-space surface point.

An additional measure applied to prevent bad finger estimation as a result of local minima is to perturb a randomly chosen finger joint parameter for half of the particles at every i_r iterations. The chosen joint parameters are replaced by samples of a uniform distribution in the corresponding permitted value ranges.

Temporal coherence and initialization

Temporal coherence is achieved by initializing the swarm members for frame $k+1$ with randomly disturbed copies of the k-th frame's solution. The authors are not precise as to how the variance of these disturbances is determined, although they mention it should be inferred from the jerkiness of observed motion through time.

Initialization on the first frame is also not explained in great detail. The authors report choosing $[x, y, z]^\intercal$ palm coordinates in the camera space that match the estimated position of the hand blob, so that, when rendered, the model and the blob appear as close as possible to each other. Calculation of the the hand model orientation that collaborates for this approximate placing is also not described.

Chapter 3

Method description

The hand pose tracking method we developed is divided in a number of steps, implemented by dedicated modules. It stems from the main reference work [1, 13, 14] in a number ways, including algorithmic details that were not thoroughly documented by them, parts where an initial simplified solution proved to be a working replacement, or when environmental conditions, such as lighting and background, where very particular to the performed tests. An overview of its structure is presented in Figure 3.1. In terms of structure, by comparing Figure 2.1 with this one, three facts become apparent: (a) we perform no depth filtering based on the last estimated pose; (b) our skin detector is composed only by a static part and is not subject to feedback training; (c) we perform skin detection and blob isolation on the GPU. In the next sections, we describe each of the system's major aspects, in order: preprocessing of input data, hand detection and isolation, modeling of hand kinematics and shape, hypothesis rendering and, finally, pose estimation.

3.1 Acquisition and preprocessing

The Kinect camera is briefly describe in Appendix A. The RGB frames produced by it may present some common image artifacts in low cost CCD cameras, such as *chromatic aberration* and *purple fringing* [39]. Such phenomena are not fully understood and constitute a challenge for digital camera manufacturers, but they are known to appear in some conditions, such as when a dark object is in front of a strong natural or halogen lamp light source. To reduce the negative influence of these artifacts during our color-based skin detection, we applied a median filter of diameter 5 to all video frames. As Figure 3.2 exemplifies, these phenomena usually manifest as an aura of sparse oddly colored pixels in skin borders. Since they have unlikely colors, replacement by the median value in each 5×5 window removes a considerable part of them.

The Kinect cameras are calibrated during manufacturing and the corresponding

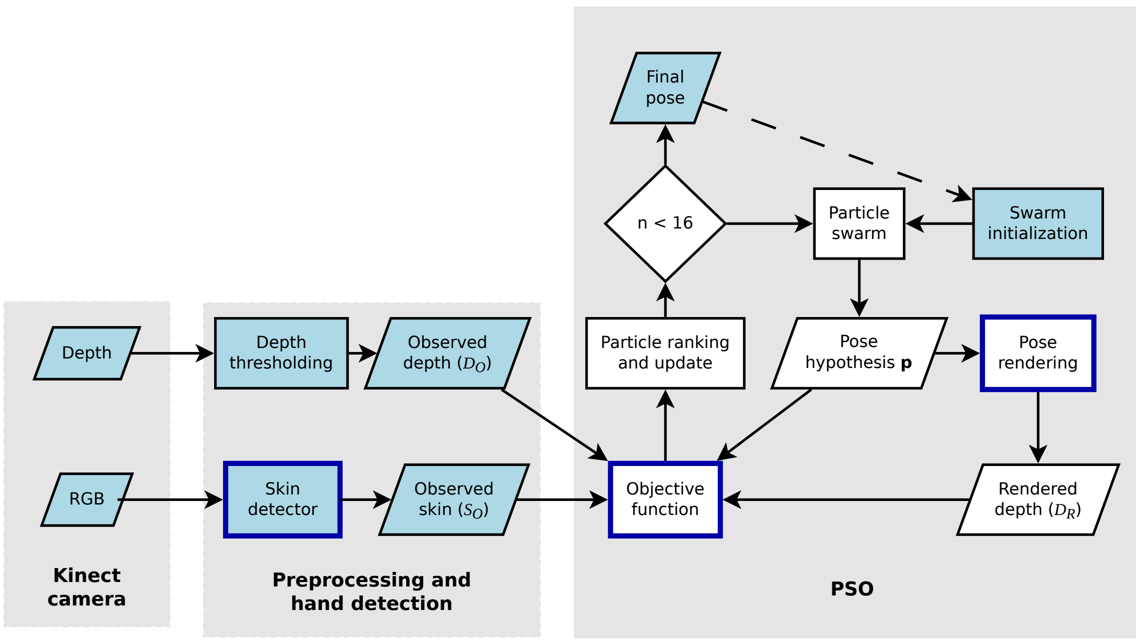


Figure 3.1: Overview of our system modules
Parallelograms indicate input, output and intermediate data, rectangles indicate modules/steps performed by the algorithm and the diamond indicates iteration control. Light blue boxes indicate data or steps that occur only once per frame. Dashed arrows indicate information that is used as feedback in the next frame. Rectangles with a blue outline indicate steps that are computed on the GPU.

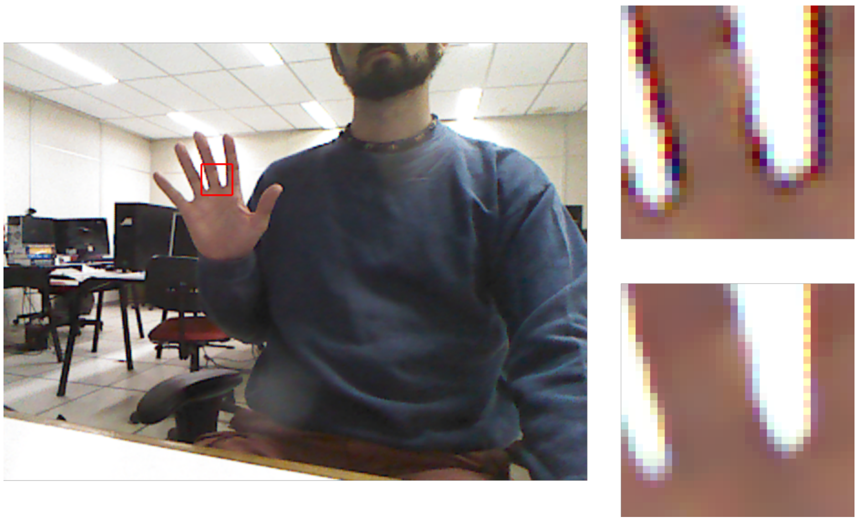


Figure 3.2: Kinect video frame before and after median filtering The frame on the left shows a RGB image just as it was captured from the camera. Two enlarged versions of the rectangular area surrounded by a red frame are shown on the right, both scaled up using nearest neighbor interpolation to improve clarity: the top one contains the same pixels as the left frame and the bottom one comes from the image transformed by a median filter.

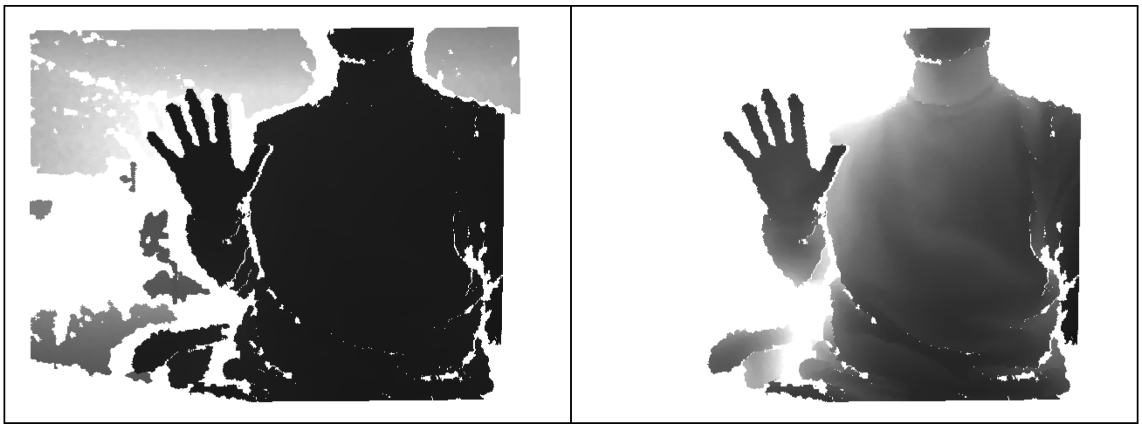


Figure 3.3: Improved view of a Kinect depth frame

On the left, a depth frame captured in one of our experiments using the own camera's registering and millimeter conversion facility. The farthest pixel is about 6 meters from the camera plane, so the 16-bit pixel range $[0, 6000]$ mm was linearly scaled down into an 8-bit grayscale. The image on the right represents the same frame, but depth values greater than 0.8 meters were discarded and the depth range of $[0, 800]$ mm was exponentially scaled into $[0, 255]$ to increase contrast and give an idea of the Kinect's precision at close range.

parameters are stored in the device's memory [10], so Kinect device drivers usually provide a depth camera mode in which the image is automatically re-projected in alignment with the RGB camera. The re-projected depth frame covers a smaller area than the video frame, as can be noted from the frame of missing depth values around the left image in Figure 3.3, which is the corresponding depth frame to the RGB frame in the left of Figure 3.2. Thus, all experiments were performed near the center of the depth camera's field of view. The registered image depth camera mode also provides automatic depth to real distance conversion, so that the values obtained represent the distance, in millimeters, from the camera plane.

All of the previously described steps are performed in CPU. The resulting video and depth images are transferred to GPU memory and then passed to the next module in the system.

3.2 Hand detection

The approach we adopted to detect and isolate the performing hand was color-based skin classification, followed by connected component labeling and a simple component filtering strategy. In the following subsections, we provide more details on the implementation of our detector, on its training phase, including used datasets, and on the final isolation of the hand.

3.2.1 Skin detector

As explained in Chapter 2, it is possible to build robust skin detectors based only on per-pixel color information. The reference hand pose estimation methods implement the skin detection algorithm of [38], which classifies pixels as either skin or non-skin based on a combination of two chrominance histogram models, one trained offline using a semi-automatic training phase and an online model that adapts to the most recent frames. Gaussian-based classifiers, although usually recognized as less robust than histogram models, are also reported in the literature as having satisfactory results [31], which was confirmed by our experiments.

Just as the other method we described, our single 2D Gaussian model of skin color, based on the description provided in [31] distinguishes between skin and non-skin pixels by operating on the chrominance subset of the YUV color space, that is, we strip the Y channel and use only the UV components, which gives us the color space $\mathcal{C}_{UV} = \{[u, v]^\intercal \ : \ 0 \leq u, v \leq 255\}$. Instead of an assisted training phase, Gaussian classifiers are usually trained using a maximum likelihood estimation strategy over the available datasets. In our case, we applied two skin detection datasets consisting of pictures containing people and manually segmented binary skin masks. The computation of model parameters is described in the next paragraphs.

nD Gaussian model

In this section, we review some important models and results of probability and statistical theory on normally distributed variables. The reader unfamiliar with the basics of probability and statistics, or seeking an in-depth discussion of the subject, is referred to [40]. We start by recalling that a random variable X following a normal, or Gaussian, distribution has the following probability density function (pdf)

$$\phi_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{3.1}$$

where μ is the distribution's mean and σ is the standard deviation. A common way of describing the normally distributed variable X in terms of these parameters is writing, simply, $X \sim N(\mu; \sigma)$.

The standard normal distribution $N(0; 1)$ has null mean and unit standard deviation and any variable $X \sim N(\mu; \sigma)$ may be transformed into a variable Z that follows a standard normal distribution by carrying out the transformation

$$Z = \frac{X - \mu}{\sigma} \tag{3.2}$$

It may be verified, as shown in [41], that Z's pdf is, indeed,

$$\phi_{0,1}(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}z^2\right) \tag{3.3}$$

In the most general case of a normally-distributed, n-dimensional random variable $\mathbf{X} = [X_1, \ldots, X_n]^\intercal$ assuming values in $\mathbb{R}^n$, the distribution $\mathbf{X} \sim N(\mu; \Sigma)$ is characterized by its mean point $\mu = [\mu_1, \ldots, \mu_n]^\intercal$ and its covariance matrix Σ, which has the form

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \ldots & \sigma_{1n} \\ \sigma_{21} & \sigma_2^2 & \ldots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n2} & \sigma_{12} & \ldots & \sigma_n^2 \end{bmatrix} \tag{3.4}$$

where σ_{ij} indicates the covariance of the X_i and X_j components. The joint probability density function for $\mathbf{X}$ is described in terms of points $\mathbf{x} \in \mathbb{R}_n$ and the distribution parameters as

$$\phi_{\mu,\Sigma}(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^n |\Sigma|} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^\intercal \Sigma^{-1} (\mathbf{x} - \mu)\right) \tag{3.5}$$

Just as with the one-dimensional case, it is possible to map $\mathbf{X} \sim N(\mu; \Sigma)$ into a vector variable $\mathbf{Z}$ that follows a standard n-dimensional normal distribution $\mathbf{Z} \sim N(0; I)$. That is accomplished by the transformation

$$\mathbf{Z} = \left(\Lambda^{-\frac{1}{2}} V^\intercal\right) (\mathbf{X} - \mu) \tag{3.6}$$

where V is an orthonormal matrix (since Σ is symmetric) containing the eigenvectors of Σ and Λ is a diagonal matrix containing its eigenvectors. Furthermore, as Σ is symmetric and positive definite, Λ has only positive entries, so $\Lambda^{\frac{1}{2}}$ indicates the matrix where the entries are the square roots of the corresponding entries in Λ. This full eigen-decomposition Σ may be expressed by the equation

$$\Sigma = V(\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}})V^\intercal \tag{3.7}$$

It then becomes apparent that, by employing the variable substitution of Equation 3.6, the argument to the exponential function in Equation 3.5 becomes simply $-\frac{1}{2}\mathbf{z}^\intercal\mathbf{z}$. The complete demonstration that this transformation implies $\mathbf{Z} \sim N(0; I)$, with pdf

$$\phi_{\mathbf{0},I}(\mathbf{z}) = (2\pi)^{-\frac{n}{2}} \exp\left(-\frac{1}{2}\mathbf{z}^\intercal\mathbf{z}\right) \tag{3.8}$$

comes from the definition of a pdf and from the theorem of change of variable in integrals [42], but we do not reproduce the proof here. Rather, we limit ourselves to arguing intuitively that: (a) subtracting μ from the points $\mathbf{x}$ translates them to the origin of the coordinate system; (b) then, multiplying by $V^\intercal$ rotates those points into a basis where all components are independently distributed; (c) which means the only non-null entries in the diagonal matrix Λ should be the variance entries and (d) multiplying $V^\intercal(\mathbf{x} - \mu)$ by the inverse of $\Lambda^{\frac{1}{2}}$ on the right should do the equivalent of Equation 3.2, that is, dividing every component by its standard deviation and obtaining independent standard normally distributed components. The eigen-decomposition of Σ and the discussion on the nD Gaussian distribution, shown above, were presented in [41].

Classification

The histogram-based double threshold approach adopted by [38] considers pixels with prior skin probability greater than some level $\tau_{\mathfrak{s}}$ to be skin and saves pixels with skin probability between $\tau_{\bar{\mathfrak{s}}}$ and $\tau_{\mathfrak{s}}$ for later consideration by hysteresis. In the case of a Gaussian model, the description in terms of a probability density function is continuous, hence unsuitable for computing discrete probabilities at specific color points. Instead, it is possible to compute the cumulative probability of finding skin colors inside delimited regions of the color space and test pixels to see if they are inside those regions. In the following paragraphs, we discuss how classification is performed in a Gaussian framework and also propose: (a) a geometric interpretation that is not commonly discussed in the literature and (b) a new way of choosing the classification threshold.

As we can see from Equation 3.8, the n-dimensional Gaussian probability density is a monotonic function of the point's distance to the distribution mean and can be expressed in a form very similar to the one-dimensional pdf of Equation 3.3, as

$$\phi_{\mathbf{0},I}(\mathbf{z}) = \phi(r) = (2\pi)^{-\frac{n}{2}} \exp\left(-\frac{1}{2}r^2\right) \tag{3.9}$$

where $r = \sqrt{\mathbf{z}^\intercal \mathbf{z}}$. Since this function decreases as r increases, Gaussian detectors are parameterized by a positive value a and consider a pixel to be a skin pixel if, and only if, its color $\mathbf{c}$ satisfies

$$r \leq a \tag{3.10}$$

after normalization, since $r \leq a \implies \phi(r) \geq \phi(a)$. The parameter a works as a threshold for selecting pixels with a minimum probability density and is empirically chosen by observing the trade off between true and false detection rates that it controls. The geometric interpretation of Equation 3.10 is that color pixels are selected if, and only if, after transformation by Equation 3.6, they belong to a hypersphere of radius a. On the UV plane, instead of a circle (a 2-dimensional

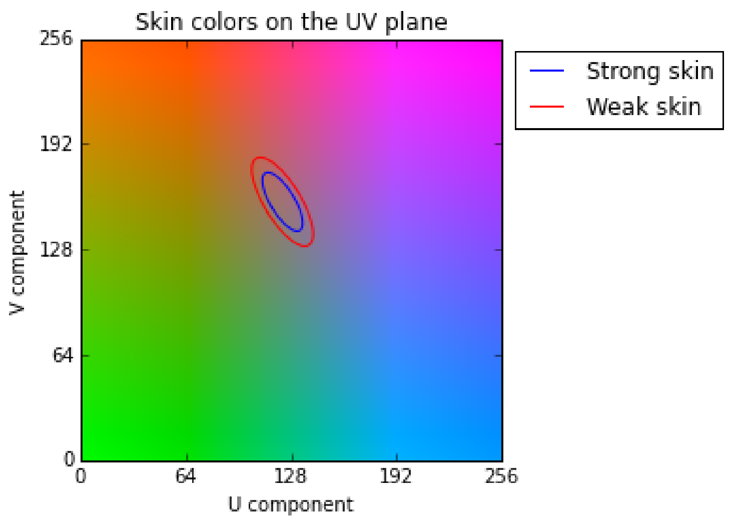


Figure 3.4: The UV chromaticity plane and the skin class boundaries. The chromaticity plane corresponds to the $Y = 255$ plane in the YUV color space. The two ellipses show the boundaries of the strong (blue, inner line) and weak (between blue and red) skin classes.

hypersphere), we have an ellipse that is centered in the color distribution mean and aligned with the the directions in which skin color concentration is higher, as show in Figure 3.4. The effect of the transformation in Equation 3.2 is, precisely, mapping that ellipse into a circle and turning the problem of determining if a $\mathbf{c}$ is a skin color into simply testing if $\mathbf{z}$ is no greater, in magnitude, than a.

Gaussian classification is usually performed as was just explained. We however, tested a mixed approach, inspired by the hysteresis thresholding of [38], with satisfactory results. We choose two different radii r_S and r_W and classify each pixel color $\mathbf{c}$ as:

- a strong skin pixel, if $\mathbf{z}^\intercal\mathbf{z} \leq r_S^2$
- a weak skin pixel, if $r_S^2 < \mathbf{z}^\intercal\mathbf{z} \leq r_W^2$
- a non-skin pixel, otherwise

Again, Figure 3.4 clarifies this description, by showing the UV plane with the corresponding strong and weak skin ellipses.

The actual implementation of the skin classifier corresponds to a single CUDA kernel that receives an array of 24-bit RGB pixel values and outputs a segment mask with 3 values that encode the possible skin classes.

Choosing the threshold radii

Choosing a threshold radius for a Gaussian classifier is usually described as an experimental process to balance between correct and incorrect detection rates. We

	Images	Total pixels	% of skin pixels
FSD	4000	1.1 billion	23%
IBTD	555	100 million	25%

Table 3.1: Summary of FSD and IBTD datasets.

make succinct contribution here, by choosing the values of the r_W and r_S radii with the aid of the cumulative distribution function. Computing the cumulative distribution function (cdf) of a standard 2D Gaussian distribution over an arbitrary region may be a complex task, but when this region is a circle of radius a centered on the origin, the cdf is, simply,

$$p_a = P(\mathbf{Z}^\intercal \mathbf{Z} < a) = 1 - e^{-\frac{a^2}{2}} \tag{3.11}$$

This of the probability p_a in terms of the radius a is easily invertible, giving

$$a = \sqrt{-2 \ln (1 - p_a)} \tag{3.12}$$

Considering the probabilities of 40% for r_S and 65% for r_W, as used in our classifier, we obtain radii 1.18 and 1.45, respectively. Although choosing the percentages is still an empirical process, we believe this procedure to be much clearer, since percentages relate to the population of skin colors, contrary to radii, and have a non linear mapping to the latter. Equation 3.11 is demonstrated in Appendix C.

3.2.2 Skin classifier training

Our Gaussian model parameters were estimated from two skin detection datasets containing pictures of people and corresponding binary skin masks created by hand, namely the FSD [34] and IBTD [43] datasets, summarized in Table 3.1.

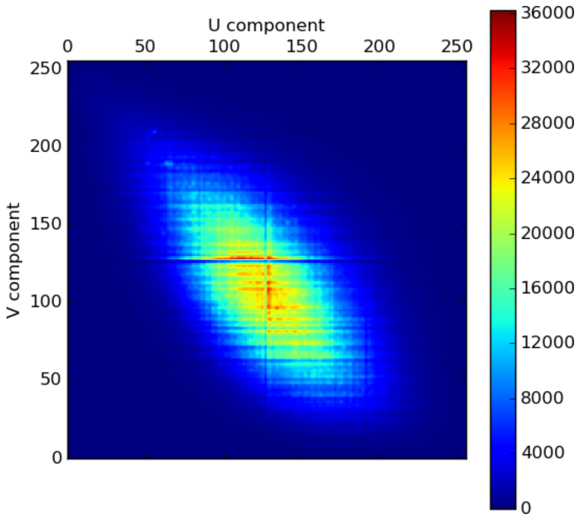
In order to learn the Gaussian model parameters from the datasets we calculated the maximum likelihood estimators for the mean and the covariance, which correspond to the sample mean and sample covariance

$$\hat{\mu} = \frac{1}{|D|} \sum_{\mathbf{x} \in D} \mathbf{x} \tag{3.13}$$

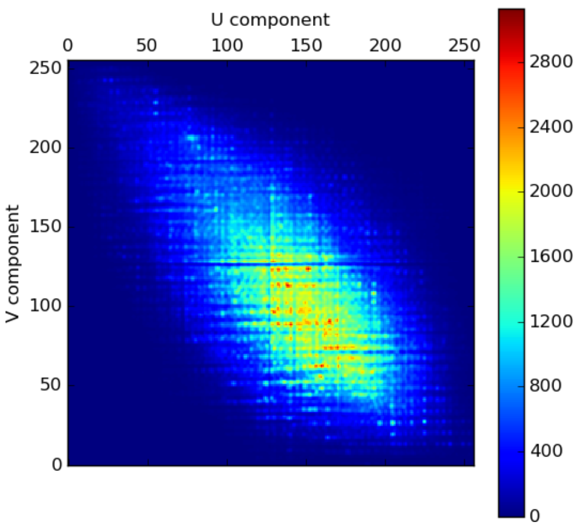
$$\hat{\Sigma} = \frac{1}{|D| - 1} \sum_{\mathbf{x} \in D} (\mathbf{x} - \hat{\mu})(\mathbf{x} - \hat{\mu})^\intercal \tag{3.14}$$

where D represents the set of skin color samples from all datasets. The estimators obtained for these datasets are given in Table B.1.

In previous experiments with a histogram-based classifier, we noticed that the JPG compression applied to the images in these datasets produced regularly spaced



(a) Skin colors in FSD



(b) Skin colors in IBTD

Figure 3.5: Skin colors histograms in the FSD and IBTD datasets. Color bars indicate incidence of UV colors, spread accross the UV plane. The images in the datasets are stored with considerable JPEG compression, causing pixel values to concentrate around sparse peaks in the histogram, with a lot of leaps caused by colors that do not occur in the images. Gaussian filtering (with a Gaussian kernel of radius 5) was applied on the color incidence matrices prior to plotting the histogram, in order to spread the pixel concentrations and give a better visual notion of skin color concentration.

holes in the gamut of the images. To counter this fact, we applied Gaussian filtering to the images before computing the histograms. However, when latter adopting a Gaussian approach, the distribution mean and covariance was not significantly affected by the missing colors. Figure 3.5 shows the color histograms of the referred datasets.

3.2.3 Hand isolation

Tracking a single posing hand in a controlled scenario where no interaction with other hands or objects takes place may be achieved with a simple tracking approach. Assuming that the performing hand is the only part of the tracked person or rather, the biggest visible part in front of the camera, we may rank the blobs by size and filter out the small ones. This requires calculation of connected components and theirs sizes, which we describe next.

Blob generation

Just as previously described, the classification step produces an output table were each pixel is assigned one of the *non-skin*, *strong skin* or *weak skin* classes. In order to obtain the skin-colored blobs, we first dilate the strong skin pixels with a

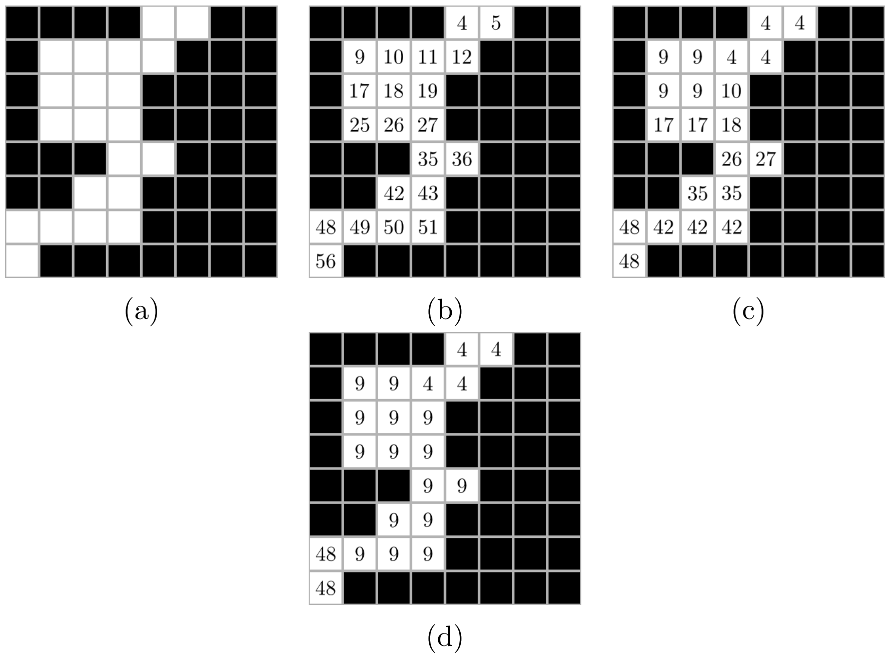


Figure 3.6: Connected component labeling on the GPU.
Snapshots of Kalentev's CCL algorithm on an example 8×8 binary image – white is foreground – at the end of each phase. The binary input (a) is first assigned provisional labels (b); then, each iteration performs a scan (c) and an analysis (d) steps to propagate minimal labels among neighboring pixels and solve equivalence chains.

circular mask of radius 5, then apply a variation of the GPU connected component labeling (CCL) method of [44]. This method is composed of one initialization step, in which each pixel is assigned a unique label corresponding to its offset in the image array, and two iterative steps that are executed until the label array converges: (a) a scan step, during which each pixel is checked for higher precedence neighboring labels and (b) an analysis step, that applies a parallel version of the union-find algorithm to solve label equivalence chains and ensure that all connected pixels will ultimately share a single label, called *representative label.* This process is illustrated by Figure 3.6.

In order to reproduce the hysteresis behavior of [38] we modify pixel labels by storing a segment part and an address part in each label. The segment part identifies which skin class the pixel belongs to and the address part corresponds to the pixel offset. During the scan phase, each foreground pixel $\mathbf{p}$ has its 8-connected neighborhood $N_8(\mathbf{p})$ checked for a label l' with higher precedence than the pixel's own label l, and if such exists, it is assigned to $\mathbf{p}$. Non skin labels are considered background and are ignored by the process. Otherwise, given foreground labels l and l', l' has higher precedence if: (a) l is a weak skin label and l' is a strong skin label or (b) they are of the same skin type and l' has smaller address. After the algorithm converges, an extra pass erases all pixels with weak skin labels, since

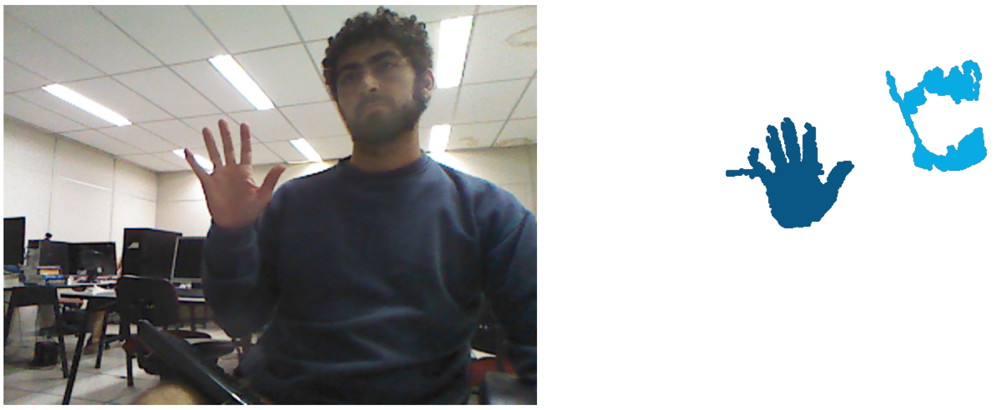

(a) Source RGB frame. (b) Resulting skin blobs.

Figure 3.7: Skin blobs produced during hand detection.
Skin classification in this case was applied with very tight model parameters, resulting in a strong skin region that contains the whole hand, but only scarcely detecting portions of the face.

they are certainly not adjacent to any strong skin region. Figure 3.7 displays some skin-colored blobs produced during the process.

Blob selection

In order to rank the blobs by size and choose the biggest one more efficiently, they need to be filtered by size, as a considerable number of small spurious blobs are produced, and relabeled sequentially. The latter may be accomplished, as noted by [45] by filling an array of the same size as the input image with a 1 for every representative pixel and a 0 for all others, then applying an *exclusive prefix sum* operation. The exclusive prefix sum of an array $A = \{a_1, a_2, \ldots, a_n\}$ is the array where each element is replaced by the sum of its predecessors, that is, $A' = \left\{0, a_1, \ldots, \sum_{i=1}^{n-1} a_i\right\}$. The final numbers in A' constitute a sequential relabeling to the representative pixels in the label array.

In order to calculate blob size or any other integral property of blobs, a properties array may be used in addition to the label array [44]. This array is initialized with per-pixel properties (such as unit area, for instance) during the initialization step and, during the analysis step, the partial properties are accumulated in the position of the representative pixel (the one corresponding to the highest precedence label). Finally, at the end of the process, the representative pixel of every region will contain the sum of the region's properties.

We finish the selection step by copying label properties and their original labels to CPU memory and selecting the blob with greatest area. Finally, a last GPU pass over the label array is performed to eliminate all pixels but the ones corresponding to the selected blob, so that we are left with a binary mask of skin pixels.

3.3 Hand modeling

We adopted the same hand kinematic and shape models as the ones presented in Section 2.3. Here, we detail our implementation of both models according to our interpretation of the reference works and related literature in the areas of hand kinematics modeling [11, 17, 19] and forward kinematics. For the latter, we do not provide an specific source, but rather describe our approach in detail.

3.3.1 Kinematics modeling

Pose space

In addition to the static joint constraints described in Section 2.3, we employ an intrafinger dynamic constraint that is modeled after the observation that few people can actively bend one finger's distal phalanx without bending the corresponding medial phalanx and vice versa, at least without forceful motion [17]. This restriction ties the values of each PIP-DIP pair, except for the thumb (since it has a single IP joint that moves independently from the MCP joint), as follows

$$\theta_{DIP,F} \leq \frac{2}{3}\theta_{PIP,F} \tag{3.15}$$

With that restriction, 4 degrees of freedom are spared and hand poses need only 23 numbers to be fully represented, which encode 22 actual DOFs because of the quaternion used for wrist orientation. Of those 23 components, 7 specify hand position and orientation, and 16 are Euler joint angles, 10 being adduction-abduction (AA) movement angles and 6 being flexion-extension movement (FE) angles. The complete description of this vector is given in Table 3.2 and we therefore indicate the set of all possible vectors, given the described static joint constraints, by $\mathcal{P}$ and name it the *pose space*. The elements $\mathbf{p} \in \mathcal{P}$ will be referred to as *pose vectors* or simply *poses*.

Forward kinematics

We treat hand posing as a forward kinematics (FK) problem, that is, hand parts are arranged in a tree-like hierarchy in which one moving part causes all its descendant parts to move by the same amount. The complete FK chain has 16 nodes, one for each articulated joint and it spans from the wrist, the root node, towards the fingertips. Each node has a corresponding transformation matrix that is derived from the product of local transformations with the accumulated product of its ancestor nodes' transformations.

The local coordinate frame L_i of an arbitrary joint j_i may contain a combina-

Palm		Fingers						
		T				I/M/R/L		
Orientation	Location	CMC		MCP	IP	MCP		PIP
$[x, y, z, w]$	$[x, y, z]$	θ_{AA}	θ_{FE}	θ_{AA}	θ_{FE}	θ_{AA}	θ_{FE}	θ_{FE}
(quaternion)	(meters)	(Euler angles in radians)						

Table 3.2: Components of the pose vector. For the palm, global orientation and positioning are described by a 4-component quaternion and a 3-dimensional vector. Most fingers have proximal and distal interphalangeal (PIP and DIP) joints with a single flexion/extension angle θ_{FE}, although only the former is a free parameter, and a metacarpophalangeal (MCP) joint with both flexion/extension and adduction/abduction (θ_{AA}) angles. The only exception is the thumb (T) , which has a carpometacarpal (CMC) joint with two degrees of freedom, in addition to a MCP and a single interphalangeal (IP) joint, each with a single θ_{FE} DOF. The index (I), middle (M), ring (R) and little (L) have identical parameters.

tion of axis rotations, scaling and translation relative to its parent's transformed coordinate system, all of which may be compactly represented as a 4×4 matrix in homogeneous coordinates. In a simplified linear exposition of FK chains where node j_i is the only child of node j_{i-1} and j_0 is the wrist joint, the transformation matrix of j_i in world coordinates, is given by the recursive equation

$$G_i = G_{i-1}(L_i P_i), \forall i \geq 1 \tag{3.16}$$

where G_i is j_i's transformation matrix in world coordinates, L_i is called *resting pose matrix* or *local matrix* and P_i is the *pose matrix*. The pose matrix is a rotation-only matrix parameterized by joint rotation angles and when it equals the identity, Equation 3.16 resorts to

$$G_i = G_{i-1} L_i, \forall i \geq 1 \tag{3.17}$$

which is why we name L_i as resting pose matrix. In all cases, we assume that $G_0 = P_0 L_0$ and that L_0 and P_0 satisfy two additional properties: (a) since j_0 has no parent node, L_0 contains no translation and (b) for the same reason, P_0 is the only pose matrix that may contain a translation, which corresponds to the wrist translation in global coordinates.

The pose matrices are calculated by the product of relevant joint Euler rotations angles

$$P_i = R_Z(\theta_{AAi}) R(\theta_{FEi}) \tag{3.18}$$

where θ_{AAi} and θ_{FEi} are the j_i's AA and FE movement angles, respectively. If j_i does not present AA movement, the last equation amounts to an X-axis rotation.

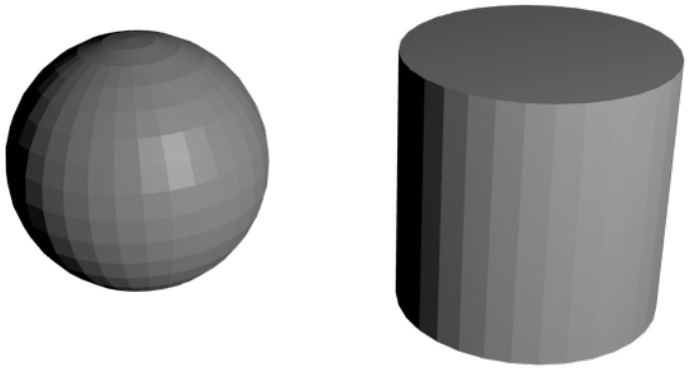


Figure 3.8: 3D primitives employed in the assembly the hand model. The shape primitives correspond to triangulations of a delimited cylinder with radius 1 and height 1, centered on the base disc and a unit sphere. The cylinder is triangulated with 32 longitudinal sections and the sphere with 32 longitudinal and 16 latitudinal sections. The hand model was created with the help of the Blender 3D modeling software and the correspondence we adopted between its metric unit and the International System was 1 bu. (blender unit) to 1 m.

Euler rotation matrices for the X, Y, Z axes are shown in Equation 3.19

$$\begin{array}{ccc} R_X(\theta) & R_Y(\theta) & R_Z(\theta) \\ \shortparallel & \shortparallel & \shortparallel \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 1 & 0 & 0 \\ \cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \tag{3.19}$$

3.3.2 Shape modeling

The adopted hand shape model is analogous to the mesh model used in the reference works and previously shown in Figure 2.4. It was assembled from 37 instances of the two basic primitives shown in Figure 3.8, by copying, scaling, rotating and translating the shapes based on a photograph of the author's right hand. Once the model was completed, the transformation matrices for each instance of the 37 primitives were imported into our system.

The FK chain modeling joint positions and orientations that was described in the previous subsection contains only 16 nodes. However, to apply a pose vector to the shape model, every primitive instance needs to be transformed and hence, needs a node of its own. The actual FK tree we employed contained 38 nodes, nearly one for every primitive, since the ellipsoids centered around joint positions were used to implicitly represent the joint nodes themselves. The only notable exception occurs with the thumb CMC joint and the ellipsoid that forms the thumb base, because this joint is located inside the palm whereas the rounded shape of the thumb base

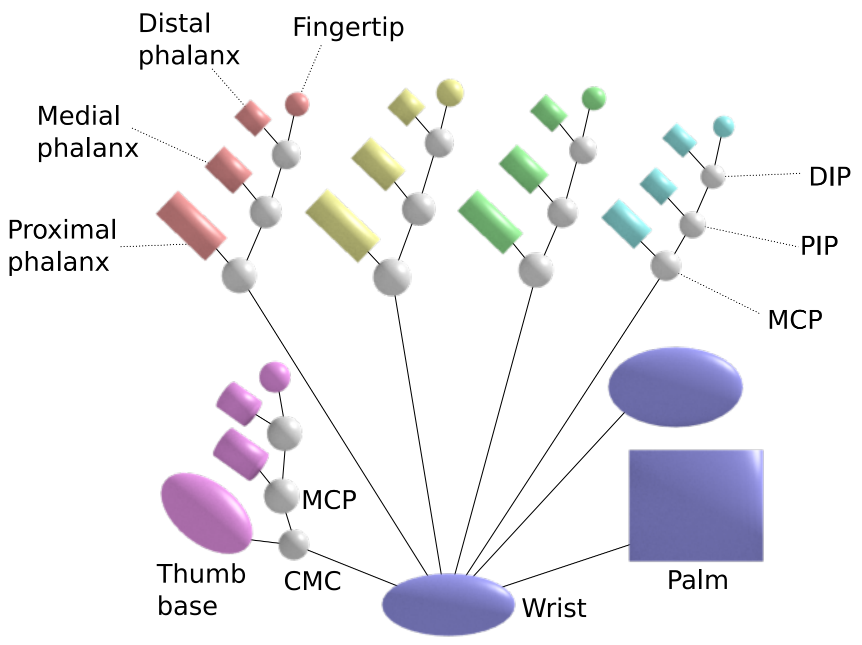


Figure 3.9: Full forward kinematics tree, with joints and shapes.

is prominent in relation to the palm surface, so they do not share a single node as it might seem appropriate at a first glance. Figure 3.9 gives a good understanding of the full tree-like structure of our model we just described.

3.4 Hand rendering

In order to compare each pose vector hypothesis **p** to the actual observations, **p** is applied to the hand model, which is then rendered with OpenGL to obtain the relevant depth map. Each depth map produced by OpenGL in this setting is a 640×480 image of 32-bit floating point depth values z_D in the range $[0, 1]$, which are later mapped into distance values in millimeters.

OpenGL camera model

OpenGL usually assumes two matrices in order to render primitives: a modelview matrix that scales, rotates and translates objects in space relative to the camera and a projection matrix that dictates the form of the viewing frustum and the type of projection (usually perspective or orthogonal) applied to the scene. The model view matrix may be seen as the product of two other matrices: a model matrix W that transforms the local vertex coordinates $\mathbf{v}$ into world coordinates $\mathbf{v}_W$ and a view matrix E that maps these into camera or eye-space coordinates $\mathbf{v}_E$. We simplify the discussion by considering only their product $G = EW$, since G may be obtained directly from the FK calculation step. The adopted projection matrix

$$P = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & -\frac{z_N+z_F}{z_F-z_N} & -2\frac{z_N z_F}{z_F-z_N} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.20}$$

aims to approximate the Kinect's depth camera (radial distortion not included) and it is constructed from the Kinect's relevant depth camera parameters listed in Table A.1. Other formats of projection matrix are possible, but are not relevant to our discussion.

Through the course of the rendering pipeline, after eye-space coordinates $\mathbf{v}_E$ are transformed by the projection matrix P into projection coordinates $\mathbf{v}_P$, perspective division is carried out as follows

$$\mathbf{v}_P = P\mathbf{v}_E = [x_P, y_P, z_P, w_P]^\intercal \tag{3.21}$$

$$\mathbf{v}_C = \frac{\mathbf{v}_P}{w_P} = \left[\frac{x_P}{w_P}, \frac{y_P}{w_P}, \frac{z_P}{w_P}, 1\right]^\intercal \tag{3.22}$$

In our case, the resulting point in clipping-space coordinates $\mathbf{v}_C$ are

$$\mathbf{v}_C = \left[-\frac{f_x x}{z}, -\frac{f_y y}{z}, \frac{z_N + z_F}{z_F - z_N} + \frac{2}{z}\frac{(z_N z_F)}{z_F - z_N}, 1\right]^\intercal \tag{3.23}$$

Depth value correction

The raw depth values produced by the Kinect camera have a non-linear relation to the actual distance from the camera center, so that closer objects are represented with better precision. The usual projection matrix employed in OpenGL rendering, shown in Equation 3.20, fulfills a similar purpose, since the calculated clipping depth $z_D \in [0, 1]$ is inversely proportional to its eye-space distance $z_E \in [z_N, z_F]$ after perspective division, as we can see from the clipping coordinates in Equation 3.23.

Since we are interested in the actual distance to the camera center, we use the special Kinect camera mode in which the depth image is automatically registered onto the RGB camera coordinate system and the raw depth values are converted into actual distance values in millimeters. In order to also obtain a linear mapping from eye coordinates to clipping coordinates during rendering, we modify the vertex shader by altering the value of the projected z_P coordinate

$$z_P \leftarrow -z_P \left(2\frac{-z_P - z_N}{z_F - z_N} - 1\right) \tag{3.24}$$

so that, after perspective division, we end up with a clipping-space z coordinate that linearly maps distances in the $[z_N, z_F]$ range into $[0, 1]$

$$z_C = \left(2\frac{-z - z_N}{z_F - z_N} - 1 \right) \tag{3.25}$$

After the single pass of the vertex and fragment shaders, the rendered z_D of each fragment is returned in the range $[0, 1]$, and we finally unproject it back to an eye-space value in millimeters to obtain compatible depth pixels by calculating

$$z_E = 1000 \left(\frac{z_F - z_N}{2}(z_D + 1) + z_n \right) \tag{3.26}$$

3.5 Pose estimation

Our optimization phase employs a variant of Particle Swarm Optimization (PSO) similar to that of the main reference works, that was previously explained in Section 2.3. In this section we just highlight the differences in our approach and some relevant implementation details.

3.5.1 Computation of generations

For a given frame in which the computed skin mask and observed depth map are given by $O = (S_O, D_O)$, the algorithm initializes the first swarm S_0 and performs a preliminary evaluation of its hypotheses ${}^{0}\mathbf{h}_i$, as follows:

1. The scores ${}^{(0)}\xi_i = F\left({}^{(0)}\mathbf{h}_i, O\right)$ are computed for every ${}^{(0)}\mathbf{h}_i$ by evaluating the objective function F
2. All ${}^{(0)}\mathbf{h}_i$ are ranked according to their scores ${}^{(0)}\xi_i$
3. Best local positions ${}^{(0)}\mathbf{b}_i$ are initialized and the global best ${}^{(0)}\mathbf{b}$ is annotated
4. The local best scores ${}^{(0)}\xi_i$ and the global best score ${}^{(0)}\xi$ are stored

From this point, each k-th generation, in a total of g generations, performs these steps

1. A new swarm S_k is generated from the previous by updating the particles in S_{k-1} according to Equation 2.8 (page 20)
2. The new particles are evaluated as in the preliminary step, but each ${}^{(k)}\mathbf{b}_i$, ${}^{(k)}\mathbf{b}$ and their respective scores are only kept if they represent an improvement of the previously stored result

At the end of g generations, the best particle across all generations is output as the estimated pose.

Modified function

We performed one modification in the objective function $F(\mathbf{h}, O)$, in the formulation of $D_M(\mathbf{h}, O)$, previously defined in Equation 2.4. We make further restrictions on the logical condition for assigning a positive depth match, as follows: (a) a depth match is confirmed if both the observed and rendered depth maps (D_O and D_R, respectively) are inside a tracking depth range $[z_{T_{min}}, z_{T_{max}}]$ and $|D_R - D_O| < \tau_{M_1}$ or (b) $D_R \in [z_{T_{min}}, z_{T_{max}}]$ and $D_O = 0$. The reason for this is that, instead of filtering depth values further than a certain amount from the hand blob, as in the reference method, we filter depth values outside of the fixed depth range $[z_{T_{min}}, z_{T_{max}}]$. Hence, if we did not modify the function, it could produce positive matches in unclear situations, such as when both depth values are 0 or when there is a whole in the observed depth ($D_O = 0$) and the rendered depth is at infinity. In order to rephrase the definition of the energy function, we also define the three following *energy accumulators*

$$A_\Delta = \sum \min\{|D_O - D_R|, \tau_{M_2}\} \tag{3.27}$$

$$A_\vee = \sum (S_O \vee D_M) \tag{3.28}$$

$$A_\wedge = \sum (S_O \wedge D_M) \tag{3.29}$$

considering the definition of D_M we just presented. These functions are useful for describing the energy function in a more compact way and they will be useful in describing its implementation in the following paragraphs. We rephrase the energy terms as

$$F_1(\mathbf{h}, O) = \frac{A_\Delta}{A_\vee} \tag{3.30}$$

$$F_2(\mathbf{h}, O) = 1 - \frac{2A_\wedge}{A_\wedge + A_\vee} \tag{3.31}$$

$$F_3(\mathbf{h}) = -\sum_{p \in Q} \min\{\Theta(p, \mathbf{h}), 0\} \tag{3.32}$$

The energy function is finally described as

$$F(\mathbf{h}, O) = \lambda_1 F_1(\mathbf{h}, O) + \lambda_2 F_2(\mathbf{h}, O) + \lambda_3 F_3(\mathbf{h}) \tag{3.33}$$

Except for the modified D_M function, the energy function is equivalent to the one defined in the previous work. We just separate the energy terms further and rearrange the weight coefficients for each term. For instance, the original energy function could be described in terms of our energy terms as

$$F(\mathbf{h}, O) = \lambda_A F_1 + 1 - \lambda_B(1 + F_2) + \lambda_K F_K \tag{3.34}$$

One other relevant deviation is the adoption of weight parameters of the objective function with significantly different magnitudes. Because the energy term F_{12} defined in Equation 2.3 is significantly affected by its first sub-term, that corresponds to a sum of depth differences, the second sub-term may be severely diminished and contribute little to the overall energy even if a substantial number of depth pixels do not match.

Objective function implementation

In order to implement the objective function calculation efficiently, we adopt three strategies: (a) perform as few rendering passes as possible; (b) calculate the energy accumulators for all particles at once; (c) sum the accumulators for each hypothesis in a single operation. The first is accomplished by vertically stacking as many 640×480 framebuffers as possible. Since the current limits on OpenGL framebuffers is 16 K by 16 K pixels, we may render up to 34 particles in a single render pass. The second is done after all rendering passes are completed with a single CUDA kernel launch, since the current CUDA architecture supports computation on a large number of elements in a single call, making better usage of memory bandwidth. The third is carried out by computing a prefix sum over each array of energy accumulators with a custom associative operation. We mark the last pixel in each accumulators array (that is, every $(640 \cdot 480)$-th pixel) with a special value α and use, as associative operation

$$f(a, b) = \begin{cases} a + b, \text{ if a,b} \neq \alpha \\ 0, \text{otherwise} \end{cases} \tag{3.35}$$

That way, energy accumulators are not summed beyond each rendered image's limit. Finally, in order to obtain the accumulated energy components, we just download every $(640 \cdot 480)$-th value in those arrays and multiply them on the CPU side to compute the energy values for all hypotheses.

3.5.2 Tracking initialization

The first swarm member of the first generation of the first frame is initialized from a predefined pose, where the hand fingers are spread open and the palm is facing the camera plane, as shown in Figure 3.10. Random variations of these pose provide the remaining $p - 1$ members of S_0. When tracking over frames, the estimated solution

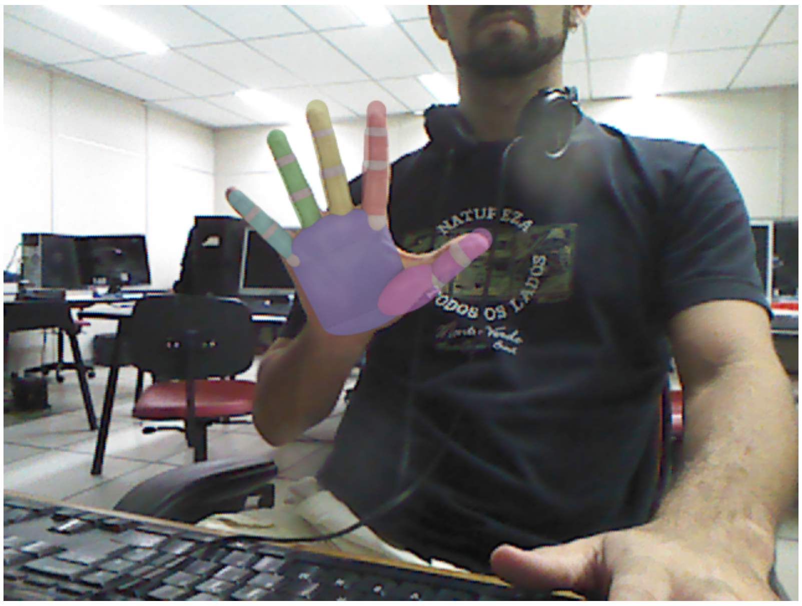


Figure 3.10: Initial hand pose overlays the image that was used for configuring it.

for the previous frame is, as in the reference method, picked as the first swarm member in S_0 for the newest frame and the remaining members are generated as in initialization. Recovering from aborted tracking works similarly, by choosing the best evaluated frozen swarm member for the current frame to be the first member in the reinitialized swarm.

Another modification we propose is defining a maximum disturbance amplitude for each type of DOF. Because the many different DOFs such as FE angles, AA angles, quaternion components and hand coordinates influence hand appearance in drastically different ways, we argue that the $p-1$ members generated from the first swarm member should not be subject to the same ratio of disturbance. For instance, a 25% variation in the angle of the thumb IP joint produces a subtle pose variation, whereas a 25% change in the hand x coordinate may drive the model away from the approximate hand location.Therefore, we introduce 5 parameters: 3 for maximum relative variation of FE angles, AA angles and quaternions components; and 2 parameters for maximum offset of xy-direction movement and z-direction movement during pose variant generation. The default values are described in Appendix B.

Chapter 4

Discussion

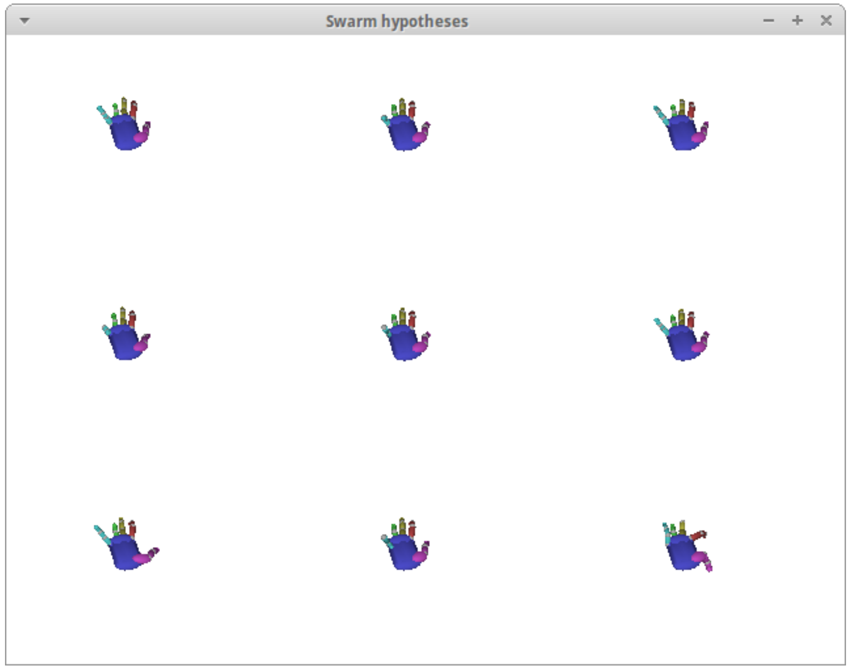
This chapter presents an analysis of our method's results. The first section describes our testing environment in terms of ambiance, hardware and software and the video sequences we recorded in order to evaluate our work. The second one discusses the effectiveness (that is, accuracy, flexibility and resilience) and performance of our method. Since an open dataset for assessment of RGBD-based articulated hand pose estimation with ground truth is not available and the preparation of such is a time-consuming task, we resorted to a qualitative assessment of effectiveness, considering our time constraints. We, however, present the average time per frame for various operations and modules, in order to quantify performance and make the bottlenecks evident.

4.1 Experiments

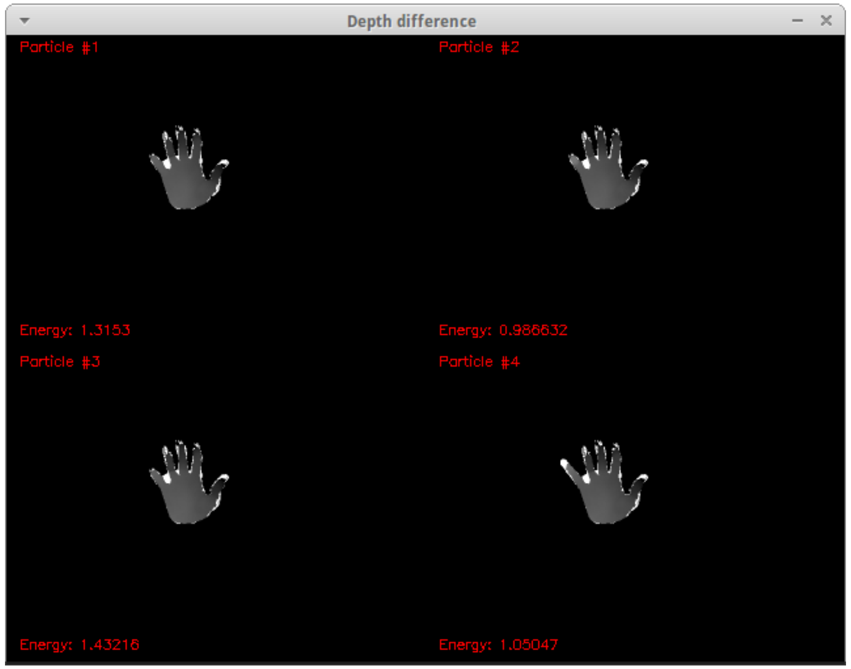
In order to make a qualitative analysis of our method and aid in the development process, we created a visualization module for our system and modified the acquisition module to allow both processing of live video streams and of pre-recorded video sequences. For each displayed pair of video and depth frames, we paint the skin mask over the depth frame and overlay the video with a transparent colored rendering of the shape model in the estimated pose for that frame. The corresponding objective function score is also displayed in the bottom left corner of the screen. Two optional windows may also display, as depicted in Figure 4.1: (a) miniatures of the last generation swarm members and (b) energy accumulators for every hypothesis.

Test videos

We recorded two input video sequences with our Kinect device in order to evaluate our system at multiple development stages and test it with different parameters. In all sequences, the author sat in front of the Kinect camera less than one meter away,



(a) Debug window for visualizing rendered hypotheses.



(b) Debug window for visualizing accumulation of depth differences

Figure 4.1: Some auxiliary debug windows

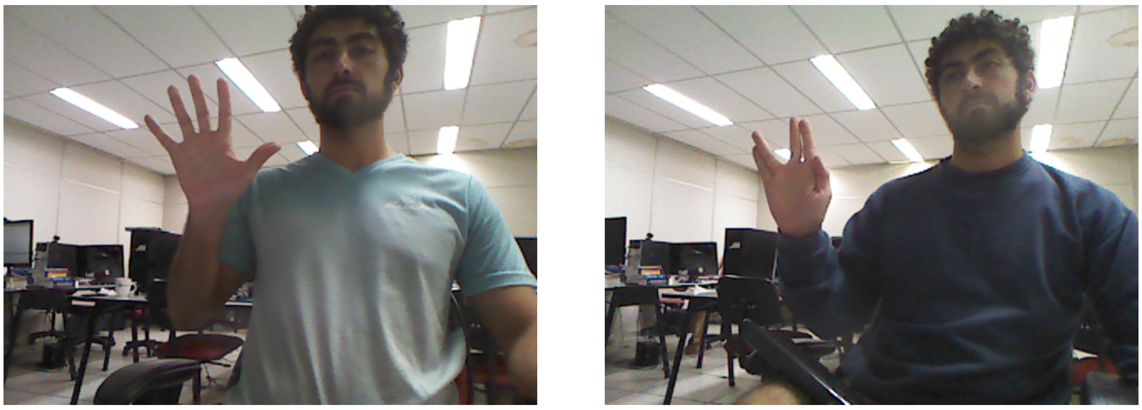


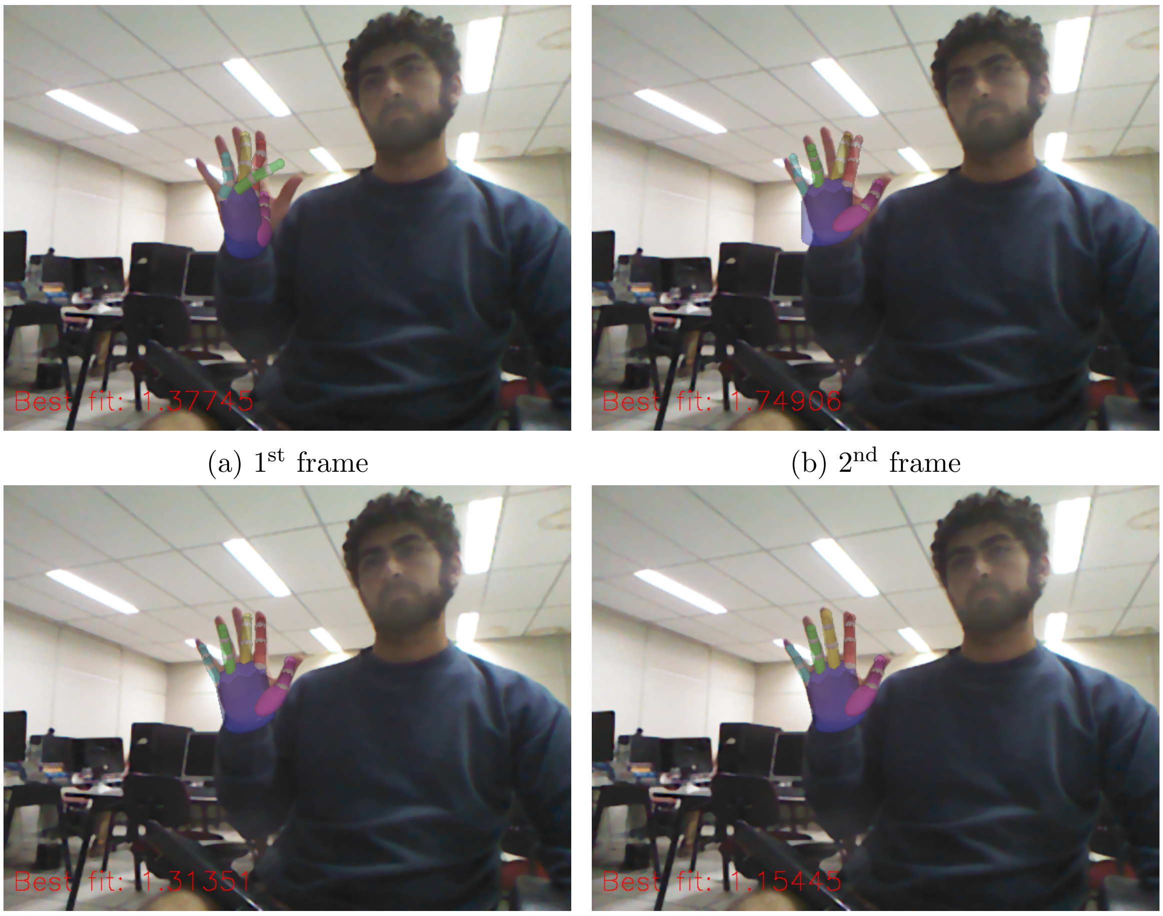
Figure 4.2: Example frames from our test video sequences. The first sequence (represented by the left frame) contains 1038 RGBD frames and the second (represented by the right frame) contains 841 RGBD frames.

so when his right hand was raised ahead of the body it stayed about half a meter from the device. Then, the author performed a sequence of hand movements, such as waving, moving one and multiple fingers at a time, rotating the hand and clenching the fist, for instance. Figure 4.2 shows example frames from the two sequences. Recording of the frames was performed in PNG format just as they were obtained from the camera, without performing any compression or image processing.

Environment and equipment

Our experiments were performed on a desktop computer with an 8-core i7 Intel processor, 16 GB of memory and a NVIDIA GTX 770 video card, featuring 1536 CUDA cores, 2 GB of dedicated memory and OpenGL 4.3 support. We developed, compiled and tested all our software with the NVIDIA CUDA Toolkit version 7.5 [6] (the latest version to date), in a 64-bit Ubuntu 15.04 operating system environment. The toolkit includes a CUDA/C++ compiler with support for the version 11 of the C++ language [4], which we used to compile and link code. The Kinect device used was a Kinect for Xbox 360, model 1473. Fragment and vertex shaders used for rendering of hand pose hypotheses were implemented in version 4 of OpenGL shading language [5]. The references presented in this paragraph are for extensive reference manuals of these languages, which we used frequently.

We believe that a description of the environment and light conditions of the room where the videos were recorded is very important for anyone trying to produce a similar scenario for experimentation. Since a very precise description would require advanced lighting measurements that are out of scope, we give an overview on those conditions, as follows: the room where the videos were recorded is approximately 65m^2 in size, with light beige plates on the floor, walls and a white ceiling; it is illuminated by 20 32W Philips fluorescent lamps with color temperature of 4100K



(a) 1st frame (b) 2nd frame (c) 3rd frame (d) 4th frame

Figure 4.3: Rapid convergence at initialization The first four frames of the sequence show the favorability of the position in which the palm is facing the camera: although the first prediction is considerably misaligned with the performing hand, it is corrected in just 3 frames (a tenth of a second).

that are laid out as almost uniformly spaced pairs.

4.2 Analysis

4.2.1 On the effectiveness

During the longer video, it was observed that the algorithm was able to correctly predict the performed pose most of the time. It gave better results when the palm plane was nearly parallel to the image plane, which may be explained by the larger observable hand surface. In such occasions, it was able to quickly (in less than 20 frames) improve bad predictions and converge to a good estimation. Figure 4.3 shows the first four frames of this video.

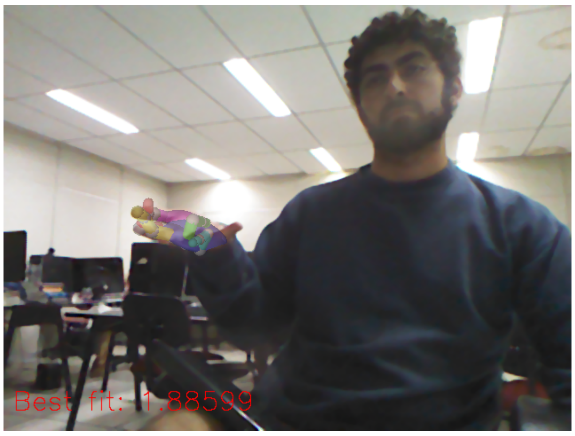
Palm alignment was lost three times throughout the first sequence. In all times, the hand was rotating in such a way that the palm plane became perpendicular to the camera plane, reducing the total area of the hand observation. Between the first occurrence and the recovery of the second one, illustrated in Figure 4.4, 85 frames elapsed. In the third occurrence, the apparent recovery leads to a hand rotated around the arm longitudinal axis by 180 degrees, inverting the positions of the thumb and little finger. This inverted position persisted until the end of the sequence, as shown in Figure 4.5d.

In all described experiments, we used a relatively low number of particles (only 16) compared to the reference works (that employed 64), and also a smaller number of generations (16 against 25) while still attaining average-quality results. We experimented with these resulted numbers because our code was only recently finished and since there was no time to optimize it, using the same parameters caused an even slower processing rate. The experimental evidence suggests that with this reduced number of particles and iterations our method is better suited for straight-up hand poses with a low degree of self-occlusion, although finger bending is still correctly detected in many situations. Testing the same video 3 times gave similar results for the presented frames. Figure 4.5 finally shows some varied poses tracked by our method with varied degrees of quality, including rotated hands and a clenched fist. It is possible to see, from those videos, that the tracking algorithm mostly favors overlaying fingers over letting them occupy empty positions in space, that is, positions that do not contain depth or skin information. So, when a wrong pose is estimated, it usually involves a mistaken association of fingers, and not finger floating loose in space.

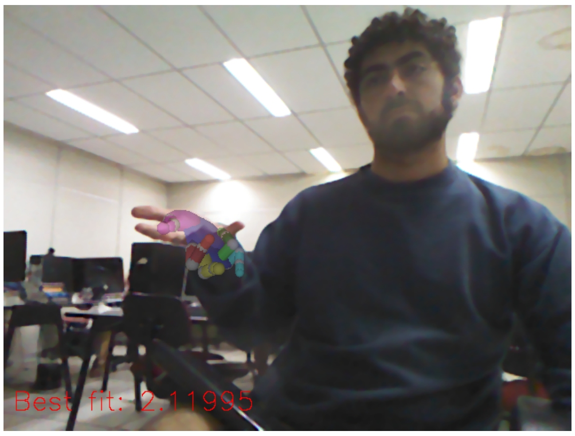
4.2.2 On the efficiency

The average framerate is 0.34 Hz, as the system takes 2.94s to process a single RGBD frame, in average. Most of this time is spent in the rendering process, which includes FK calculation, data uploading to GPU and actual rendering. The graph of Figure 4.7 shows the time spent by our method on each phase of the PSO module. Figure 4.6 also shows the time spent on other modules of our system. Although PSO time dominates most of the processing, it would still be relevant to reduce the processing times for these modules if one were to optimize the system for realtime operation.

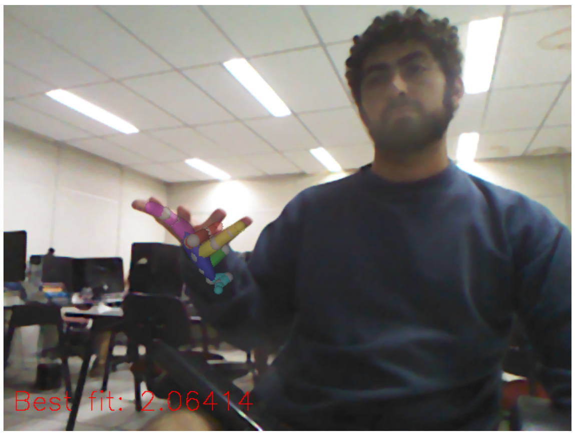
We verified, on a late analysis, that most time spent on particle rendering was due to an inefficient conversion of depth values. In order to compute the energy accumulators using a CUDA kernel, the rendered depth values for each particle must be readable by the kernel. Although mapping of OpenGL framebuffers to the CUDA global memory address space is possible, this topic poorly documented in the CUDA API, so there was no time to implement this mapping. Instead, we



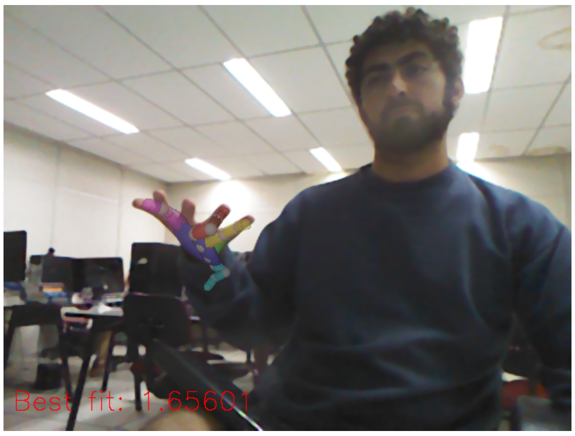
(a) 765th frame



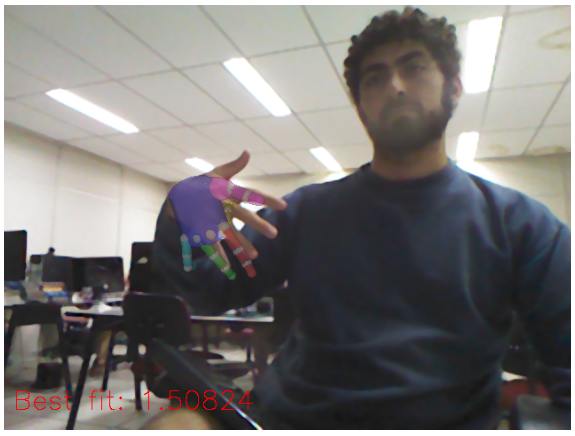
(b) 766th frame



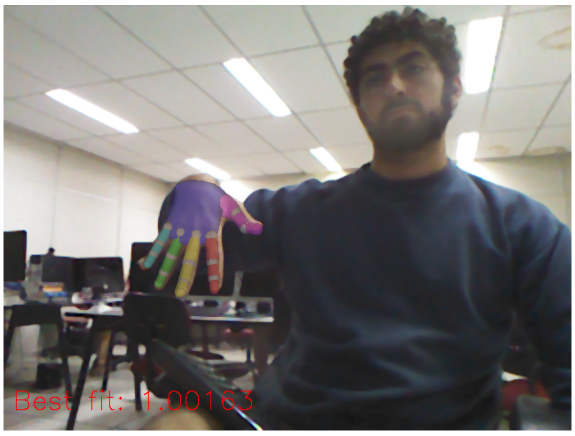
(c) 773th frame



(d) 774th frame

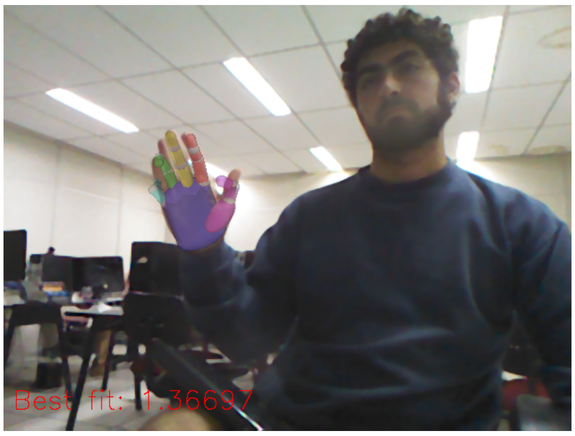


(e) 829th frame

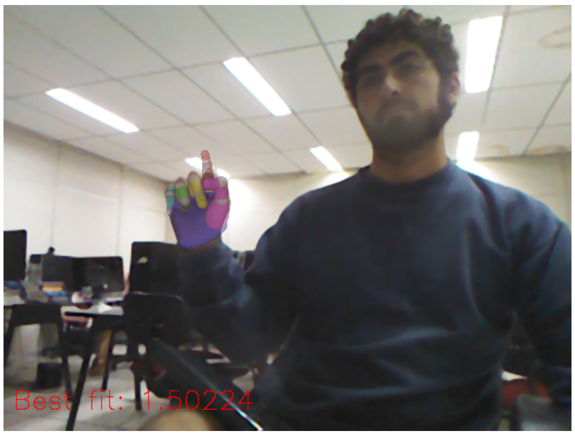


(f) 850th frame

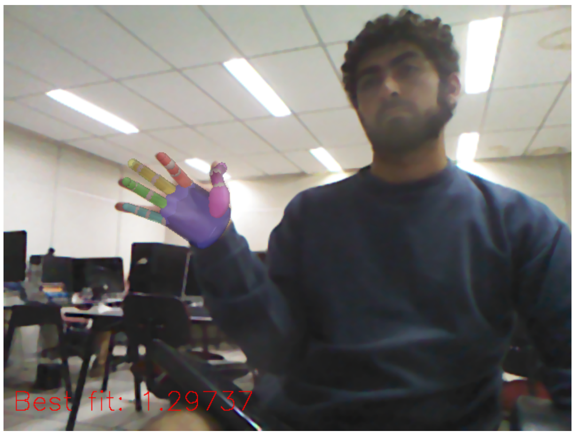
Figure 4.4: Tracking fail and recovery It is possible to observe how tracking accuracy is degraded by the unfavorable perspective that takes place when the palm plane is almost perpendicular to the image plane, a process that becomes more critical from (a) to (b), as confirmed by the increasing value of the objective function. Palm alignment starts to recover between (c) and (d), where there is a sudden drop of the objective function, although the index finger assumes the position of the middle, which assumes the position of the little, which is hanging. In frame (e), as the palm plane becomes parallel to the image plane again, palm alignment is considerably restored and in (f), all observed fingers are finally matched by their model counterparts.



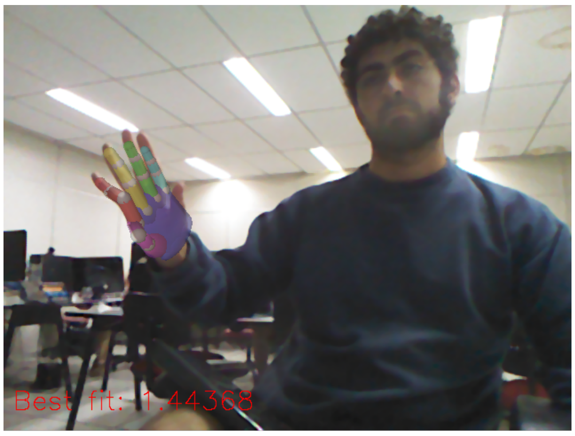
(a) 449th frame



(b) 559th frame



(c) 670th frame



(d) 919th frame

Figure 4.5: Various poses tracked by our method

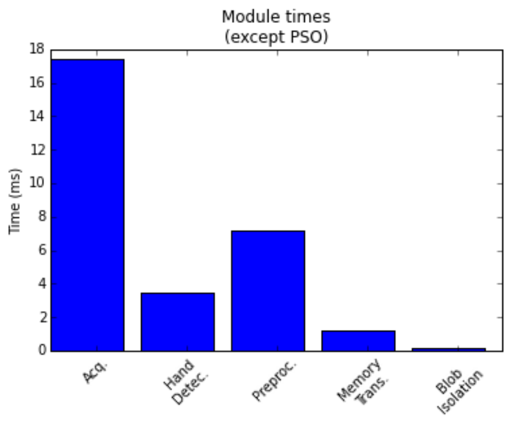


Figure 4.6: Average time per frame spent on each module, except optimization

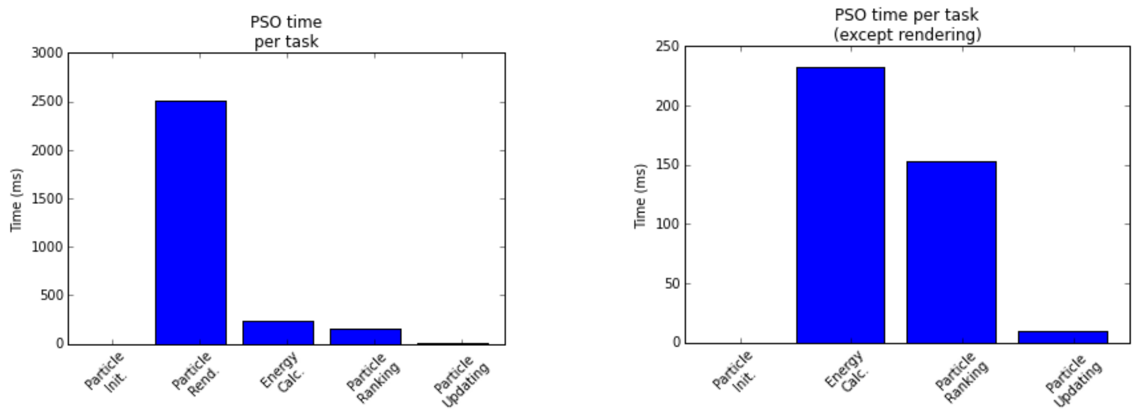


Figure 4.7: Average time per frame spent on each phase of optimization

sticked with the inefficient prototype implementation, which downloaded the depth buffer from the GPU, converted those values from floating point into 16-bit integers comparable to the Kinect's depth format and uploaded them back to the GPU.

Chapter 5

Conclusion

All we have to decide is what to do
with the time that is given to us.
– Gandalf, The Lord of the Rings:
The Fellowship of the Ring
J.R.R. Tolkien

Although glove-based motion capture systems offer excellent precision for human hand tracking, their application in human-computer interaction is limited by the cost and reduced naturalness of motion associated with these devices. On the other hand, computer vision-based solutions are promising alternatives in terms of cost and applicability, but must face many problems like noisy input, occlusion, varying illumination conditions and ambiguity, hence constitute and important research direction.

In this text we described the implementation of a computer vision system that uses RGBD data from the low-cost, end consumer Kinect sensor to track the articulated movement of a performing human hand. Our system, based on the work of Oikonomidis et al. [1], was developed with the intent of reproducing that work and identifying possible improvements. During development, we adopted a number of different strategies and modified problem formulations based on related literature and, as a result, we built a slightly different system that keeps the same overall structure. Our experiments provide additional evidence that Particle Swarm Optimization is, under certain conditions, a plausible technique for solving the problem of fully articulated hand tracking. Although our system's performance did not achieve the expected marks, we identify most bottlenecks and some improvable features and point at them in this text. Furthermore, we release our publish our system's code under open source terms [1], with the intent of helping future research in this area.

[1]Details on the source code repository are given in Section 1.3.1.

5.1 On the results

The image-based qualitative evaluation of our system provided in the last chapter indicates that it is able to track hand poses with acceptable accuracy and has some capacity to initialize tracking and recover from tracking errors. It deals most effectively with open hand poses where the palm is not perpendicular to the image plane, although it can track finger bending if they are not very cluttered. Although the experiments suggest that our accuracy was lower than the main reference's, it should be noted that we used a smaller swarm population (16 against 64) with less generations (16 against 25), which were chosen as such in order to reduce processing time.

The performance was highly affected by the conversion of rendered depth values from a floating point representation into a 16-bit format compatible with the Kinect depth frames, which took a considerable part of hypothesis rendering. The total average time per frame for hypothesis rendering was approximately 2.4 s, which corresponds to 9.7 milliseconds per particle per generation. Second to this, the most critical phases are the objective function calculation, followed by particle ranking, which take 243 ms and 152 ms on the average, respectively. If we were to improve performance to reach processing times similar to [1, 13, 14], the total time of these modules would need to be reduced by around 70%.

It is apparent, from the

5.1.1 Possible immediate improvements

The skin detection and energy accumulators calculation phases were implemented in the GPU, but virtually all image processing and optimization steps of our algorithm are parallelizable, including preprocessing and computation of swarm generations. If PSO related phases were all computed on the GPU, expensive synchronization steps and memory transfers would be saved. Computation of the FK chain on the GPU, in particular, would allow to output modelview matrices directly to OpenGL uniform buffers that are later used for rendering, and rendered images may be conversely mapped into CUDA buffers in order to calculate the energy accumulators. Efficiency of energy calculation could be further addressed by using a compact representation for the two of the three energy accumulators used by the objective function, since they are Boolean indicators with values of 0 or 1 and could be accumulated in a single 32-bit integer by separation of the lower order 16-bits from the higher order 16-bits.

A possible direction for accuracy enhancement is to improve objective function input filtering by applying one of many existing depth hole filling approaches or even alternative hand detection and segmentation approaches described in [10].

Better swarm initialization could also rely on information from blob tracking phase, so that the initial swarm member is placed on the estimated location of the moving hand blob and the palm is oriented to align with the principal axes of the blob. Parallel to this work, we developed a new method for computing connected component labeling on the GPU while calculating the first and second order moments of blobs and predicting their positions in consecutive frames that could be used for this purpose. However, we did not employ the method, which we call *Time Coherent Label Equivalence*, because it was not completely tested and verified at the time of writing.

5.1.2 Limitations

One limiting factor we can anticipate for our system, which is also a limitation of similar methods is rapid hand motion. The human hand is, after all, capable of 5 m/s translation and $400°/s$ rotation, whereas and most RGBD cameras have a 30 Hz [8] framerate. That said, a possible way to counter rapid motion is to track the hand on multiple levels, using object tracking techniques to make temporal predictions about hand positioning and orientation and use them to improve swarm initialization.

The performance of the skin classifier was considered satisfactory, although it is possible to make it more robust against illumination variations that might be present in other scenarios by refining our detector with information from other skin detection datasets, such as the ones listed in [46].

5.2 Contributions

In this work, we did not propose a completely new approach to the problem of fully articulated hand pose estimation from RGB-D data. Instead, we based ourselves on an existing approach and produced a complete system that performs hand tracking. We stemmed from the previously existing work in three broader aspects: hand detection method, hand model constraints and perturbation, and optimization details.

On the first matter, instead of a histogram-based per-pixel skin classifier with semi-assisted training, we used a Gaussian model with parameters estimated from two open datasets for skin detection. Our detector, however, performs hysteresis threshold in a similar way. In the description of our detector, we also introduce a new strategy for choosing the thresholds that is not common in the literature. Contrary to the other work's, our detector was implemented in the GPU.

On the second matter, we used a simplified hand model with one less degree of freedom per finger, according to another work on hand modeling that points out that each such finger DOF is highly correlated to a second one in the same finger. We also applied a different variation amplitude per model DOF, since some of them have greater impact on the overall pose than others, a consideration that was not present

On the third matter, our objective function was modified to deal with a simpler strategy of depth filtering and the weights used for each term of the objective function differ greatly from the original work. We experimented with many combinations of values and found out these provided a better balancing between the terms, preventing one from shadowing the others.

Although the depth of our analysis was restricted by time constraints, we built a system that achieves comparable functionality to the reference work and provide a valuable source of information for future research in the form of this detailed study. Also, we publish our complete system's code under open source terms [2] , allowing the research community to benefit from an existing implementation, which may be further improved or modified.

5.3 On future directions

As noted by [10], the Kinect's application is still narrow, due to many implicit facilitating environment restrictions that are present in the gaming scenario and experiments performed, such as the persons being far from the device (compensating for its narrow field of view) and mostly standing parallel to the image plane. Outdoor scenarios, partial occlusions, fast motion and unpredictable person/object poses and change of point of view may severely restrict the Kinect's application in robotic vision and automated surveillance, for instance. When it comes to hand tracking, the set of simplifying assumptions we assumed such as the hands being the closest skin-colored object to the camera, may render the method unusable in real world scenarios. To the purpose of hand tracking in unrestricted scenarios, there exists no studies as of now, to the best of our knowledge. An evaluation of this method with other types of indoor and outdoor depth sensors possessing different characteristics of precision, noise, field of view and costs could prove useful in such direction.

Combining appearance-based methods with model-based ones, such as this, is an interesting direction: on one hand, appearance-based approaches are fast and provide accurate classification of known poses if trained with large enough databases, allowing to obtain a rough estimate of the overall pose; on the other hand, model-based methods have elevated computational costs, but if correctly constrained, capture the details of each observed instance and provide smoother transitions between

[2]Details on the source code repository are given in Section 1.3.1.

consecutive similar instances. Hence, in a hypothetical hybrid system, discriminative recognition might be employed for initialization, error recovery and auxiliary pose space restriction, whereas a later generative phase might be used to refine the pose estimation, track over adjacent frames and even to obtain additional exemplars for feedback training of the discriminative recognizer, extending its knowledge to unknown poses.

It should be emphasized that careful incorporation of anatomic studies, as partly achieved in this work by means of finger joint constraints, is fundamental in pruning the search space and that investigating motion capture sequences as in [17] may help to obtain motion subspaces with reduced dimensionality that give good predictions of what can be instinctively understood as realistic human motion.

Bibliography

[1] OIKONOMIDIS, I., KYRIAZIS, N., ARGYROS, A. A. "Efficient model-based 3D tracking of hand articulations using Kinect". In: *BMVC*, v. 1, p. 3, 2011.

[2] "Efficient model-based 3D tracking of hand articulations using Kinect". `https://www.youtube.com/watch?v=Fxa43qcm1C4`. Accessed: 2016-05-30. Published: 2011-06-29.

[3] GAMMA, E., HELM, R., JOHNSON, R., et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.

[4] MEYERS, S. *Effective Modern C++*. O'Reilly Media, 2014. ISBN: 978-1-4919-0399-5.

[5] SELLERS, G., WRIGHT JR, R. S., HAEMEL, N. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley, 2015.

[6] NVIDIA, C. "C Programming Guide, version 7.5". 2015.

[7] "The MIT license". `https://opensource.org/licenses/MIT`. Accessed: 2016-06-02.

[8] EROL, A., BEBIS, G., NICOLESCU, M., et al. "Vision-based hand pose estimation: A review", *Computer Vision and Image Understanding*, v. 108, n. 1, pp. 52–73, 2007.

[9] RAUTARAY, S. S., AGRAWAL, A. "Vision based hand gesture recognition for human computer interaction: a survey", *Artificial Intelligence Review*, v. 43, n. 1, pp. 1–54, 2012.

[10] HAN, J., SHAO, L., XU, D., et al. "Enhanced computer vision with microsoft kinect sensor: A review", *Cybernetics, IEEE Transactions on*, v. 43, n. 5, pp. 1318–1334, 2013.

[11] LEE, J., KUNII, T. L. "Model-based analysis of hand posture", *Computer Graphics and Applications, IEEE*, v. 15, n. 5, pp. 77–86, 1995.

[12] OIKONOMIDIS, I., KYRIAZIS, N., ARGYROS, A. A. "Markerless and efficient 26-dof hand pose recovery". In: *Computer Vision–ACCV 2010*, Springer, pp. 744–757, 2010.

[13] OIKONOMIDIS, I., KYRIAZIS, N., ARGYROS, A. A. "Tracking the articulated motion of two strongly interacting hands". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1862–1869. IEEE, 2012.

[14] OIKONOMIDIS, I., KYRIAZIS, N., ARGYROS, A., et al. "Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2088–2095. IEEE, 2011.

[15] PADELERIS, P., ZABULIS, X., ARGYROS, A. A. "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*, pp. 42–49. IEEE, 2012.

[16] BALLAN, L., TANEJA, A., GALL, J., et al. "Motion capture of hands in action using discriminative salient points". In: *Computer Vision–ECCV 2012*, Springer, pp. 640–653, 2012.

[17] LIN, J., WU, Y., HUANG, T. S. "Modeling the constraints of human hand motion". In: *Human Motion, 2000. Proceedings. Workshop on*, pp. 121–126. IEEE, 2000.

[18] WU, Y., LIN, J. Y., HUANG, T. S. "Capturing natural hand articulation". In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, v. 2, pp. 426–432. IEEE, 2001.

[19] ALBRECHT, I., HABER, J., SEIDEL, H.-P. "Construction and animation of anatomically based human hand models". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 98–109. Eurographics Association, 2003.

[20] SUEDA, S., KAUFMAN, A., PAI, D. K. "Musculotendon simulation for hand animation". In: *ACM Transactions on Graphics (TOG)*, v. 27, p. 83. ACM, 2008.

[21] TRINDADE, P., LOBO, J., BARRETO, J. P. "Hand gesture recognition using color and depth images enhanced with hand angular pose data". In: *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, pp. 71–76. IEEE, 2012.

[22] ZHANG, Z. "Microsoft kinect sensor and its effect", *MultiMedia, IEEE*, v. 19, n. 2, pp. 4–10, 2012.

[23] YE, M., ZHANG, Q., WANG, L., et al. "A survey on human motion analysis from depth data". In: *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications*, Springer, pp. 149–187, 2013.

[24] SHOTTON, J., FITZGIBBON, A., COOK, M., et al. "Real-time human pose recognition in parts from single depth images". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1297–1304. IEEE, 2011.

[25] KESKIN, C., KIRAÇ, F., KARA, Y. E., et al. "Hand pose estimation and hand shape classification using multi-layered randomized decision forests". In: *Computer Vision–ECCV 2012*, Springer, pp. 852–863, 2012.

[26] KESKIN, C., KIRAÇ, F., KARA, Y. E., et al. "Real time hand pose estimation using depth sensors". In: *Consumer Depth Cameras for Computer Vision*, Springer, pp. 119–137, 2013.

[27] REN, Z., YUAN, J., MENG, J., et al. "Robust part-based hand gesture recognition using kinect sensor", *Multimedia, IEEE Transactions on*, v. 15, n. 5, pp. 1110–1120, 2013.

[28] TARA, R., SANTOSA, P., ADJI, T. "Hand segmentation from depth image using anthropometric approach in natural interface development", *Int. J. Sci. Eng. Res*, v. 3, n. 5, pp. 1–4, 2012.

[29] LIANG, H., YUAN, J., THALMANN, D. "3D fingertip and palm tracking in depth image sequences". In: *Proceedings of the 20th ACM international conference on Multimedia*, pp. 785–788. ACM, 2012.

[30] HACKENBERG, G., MCCALL, R., BROLL, W. "Lightweight palm and finger tracking for real-time 3D gesture control". In: *Virtual Reality Conference (VR), 2011 IEEE*, pp. 19–26. IEEE, 2011.

[31] KAKUMANU, P., MAKROGIANNIS, S., BOURBAKIS, N. "A survey of skin-color modeling and detection methods", *Pattern recognition*, v. 40, n. 3, pp. 1106–1122, 2007.

[32] IGARASHI, T., NISHINO, K., NAYAR, S. K. "The appearance of human skin: A survey", *Foundations and Trends® in Computer Graphics and Vision*, v. 3, n. 1, pp. 1–95, 2007.

[33] KHAN, R., HANBURY, A., STÖTTINGER, J., et al. "Color based skin classification", *Pattern Recognition Letters*, v. 33, n. 2, pp. 157–163, 2012.

[34] PHUNG, S. L., BOUZERDOUM, A., CHAI SR, D. "Skin segmentation using color pixel classification: analysis and comparison", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, v. 27, n. 1, pp. 148–154, 2005.

[35] VEZHNEVETS, V., SAZONOV, V., ANDREEVA, A. "A survey on pixel-based skin color detection techniques". In: *Proc. Graphicon*, v. 3, pp. 85–92. Moscow, Russia, 2003.

[36] TAYLOR, C. L., SCHWARZ, R. J. "The anatomy and mechanics of the human hand", *Artificial limbs*, v. 2, n. 2, pp. 22–35, 1955.

[37] "The free dictionary". `http://medical-dictionary.thefreedictionary.com`. Accessed: 2016-05-30.

[38] ARGYROS, A. A., LOURAKIS, M. I. "Real-time tracking of multiple skin-colored objects with a possibly moving camera". In: *Computer Vision-ECCV 2004*, Springer, pp. 368–379, 2004.

[39] KANG, S. B. "Automatic removal of chromatic aberration from a single image". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8. IEEE, 2007.

[40] EVANS, M. J., ROSENTHAL, J. S. *Probability and Statistics: the science of uncertainty.* W.H. Freeman, 2010. ISBN: 978-1429224628.

[41] DUO, C. B. "The Multivariate Gaussian Distribution". `http://cs229.stanford.edu/section/gaussians.pdf`, 2008. Lecture notes on Machine Learning, Stanford University. Acessed: 2016-05-30.

[42] "Integration by substitution – Application to statistics". `https://en.wikipedia.org/wiki/Integration_by_substitution#Application_in_probability`. Accessed: 2016-06-02.

[43] ZHU, Q., WU, C.-T., CHENG, K.-T., et al. "An adaptive skin model and its application to objectionable image filtering". In: *Proceedings of the 12th annual ACM international conference on Multimedia*, pp. 56–63. ACM, 2004.

[44] KALENTEV, O., RAI, A., KEMNITZ, S., et al. "Connected component labeling on a 2D grid using CUDA", *Journal of Parallel and Distributed Computing*, v. 71, n. 4, pp. 615–620, 2011.

[45] STAVA, O., BENES, B. "Connected component labeling in CUDA". In: Wen-Mei, W. H. (Ed.), *GPU computing gems emerald edition*, Elsevier, p. 569, 2010.

[46] MORRIS, T. "Skin Locus Based Skin Detection for Gesture Recognition", 2010.

[47] GENG, J. "Structured-light 3D surface imaging: a tutorial", *Advances in Optics and Photonics*, v. 3, n. 2, pp. 128–160, 2011.

[48] CRUZ, L., LUCIO, D., VELHO, L. "Kinect and rgbd images: Challenges and applications". In: *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on*, pp. 36–49. IEEE, 2012.

Appendix A

The Kinect device

The Kinect sensor's vision apparatus is comprised of three main components: a structured infrared (IR) light projector, an infrared camera and a visible light camera, both producing frames at the resolution of 640×480 pixels and frequency of 30 Hz. The structured IR pattern projected onto the scene is reflected and captured by the IR camera, which allows for deducing the depth at every pixel, a process more thoroughly described by [47]. A typical Kinect sensor is represented in Figure A.1. The Kinect sensor is also equipped with a microphone array and a tilt motor, which were not used in our work.

Video frames are 24-bit RGB-encoded 640×480 pixel images and depth frames are 16-bit images of the same size, where only the 11 lower bits are actually used, providing 2048 levels of sensitivity. Since distances are measured using infrared light, the Kinect does not operate correctly in an outdoor setting, where sunlight interferes with the structured light pattern [48]. Because of the 7.5 cm gap between the IR emitter and camera, a lot of blind spots are produced by occluding objects, resulting in black shadows in the depth image where the depth value is unknown. Video and depth cameras are also separated by a 2.5 cm gap. The depth frame stores depth values as integral values in millimeters and the operation range of the camera is from 0.4m to 3.5m. Table A.1 shows the intrinsic parameters of the depth camera.

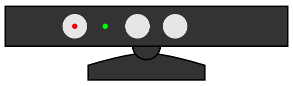


Figure A.1: Depiction of a Kinect Sensor.
From left to right, the gray circles portray: the infrared projector, the RGB camera and the infrared camera.

Field of view		**Clipping planes**	
f_x	f_y	z_N	z_F
58°	45°	0.4 m	8 m

Table A.1: Kinect depth camera intrinsic parameters

Appendix B

Parameters

In this appendix, we list the default parameters we applied during our experiments.

The hysteresis thresholds used for skin detection were $r_S = 0.4$ and $r_W = 0.65$. The actual Gaussian model parameters calculated for FSD and IBTD are listed in Table B.1. We used the parameters estimated from the combination of both datasets.

The number of generations used was $g = 16$ and the number of particles per generation was $p = 16$. Half of the particles joints are disturbed every $i_r = 3$ iterations. Pose variations generated at every first iteration from the first swarm member included different amplitudes for each type of DOF. The maximum amplitudes are: for θAA, 0.15 radians; for θ_{FE}, 0.15 radians; for quaternion components, 0.5; for translation in the xy plane (parallel to the image plane), 3 cm; for translation in the z direction (perpendicular to image plane), $1,5$ cm. All these amplitudes are multiplied by a common uniformly distributed scaling factor in the range $[0, 0.3]$. Figure 3.10 shows the initial pose used for the first swarm of the first frame.

The energy coefficients of F are: $\lambda_1 = 0.006$, $\lambda_2 = 1.0$ and $\lambda_3 = 0.5$. Division by zero in E_1 is avoided by setting $\epsilon_1 = 0.001$. The cognitive and social components have values $c_1 = 2.8$ and $c_2 = 1.3$, respectively. The absolute depth clipping range we applied in our modified function was $[z_{min}, z_{max}] = [0.4, 0.9]$ m. Other parameters

$$\mu \quad \begin{bmatrix} 113.7 \\ 156.0 \end{bmatrix} \qquad \Sigma \quad \begin{bmatrix} 69.6 & -73.4 \\ -73.4 & 153.3 \end{bmatrix}$$

(a) FSD only

$$\mu \quad \begin{bmatrix} 104.3 \\ 166.9 \end{bmatrix} \qquad \Sigma \quad \begin{bmatrix} 86.9 & -87.1 \\ -87.1 & 181.6 \end{bmatrix}$$

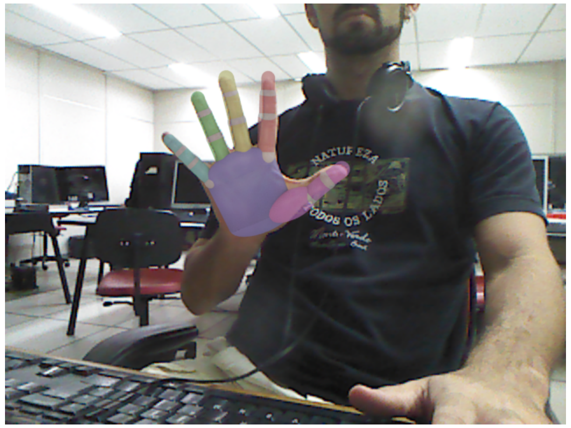
(b) IBTD only

$$\mu \quad \begin{bmatrix} 112.8 \\ 157.0 \end{bmatrix} \qquad \Sigma \quad \begin{bmatrix} 78.4 & -83.0 \\ -83.0 & 165.5 \end{bmatrix}$$

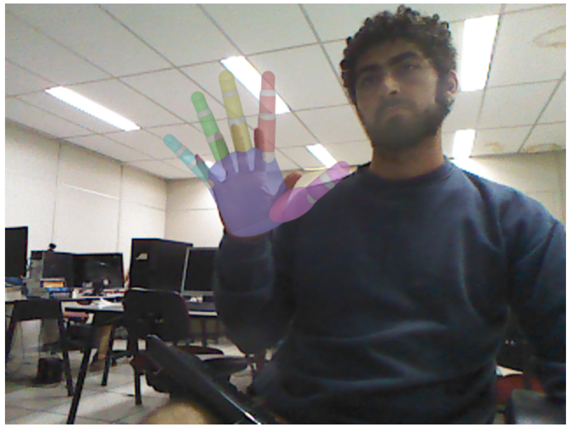
(c) FSD and IBTD combined

Table B.1: Gaussian skin model parameters

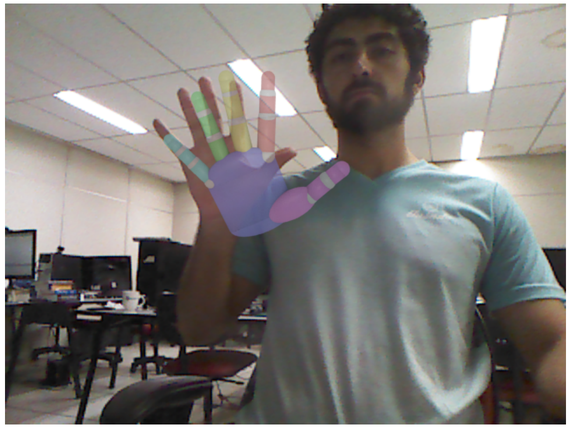
used for energy calculation are given as follows: $\tau_{M_1} = 4$ cm, $\tau_{M_2} = 4$ cm and $\tau_{M_3} = 5$ cm.



(a) Initial hand pose overlays the image that was used for configuring it.



(b) Initial hand pose overlays the first frame of the first video sequence.



(c) Initial hand pose overlays the first frame of the first second sequence.

Figure B.1: Initial hand pose for tracking Initial hand pose used as first swarm member in all tested video sequences. The other $p-1$ swarm members are generated as random variations of this pose.

Appendix C

Demonstrations

In this appendix, we demonstrate the auxiliary result on the computation of the cumulative distribution of the Gaussian distribution, used in Section 3.2.3.

Theorem 1. *Let* $\mathbf{Z}$ *be a random vector variable with a 2D standard Gaussian distribution. The cumulative probability* p_a *of drawing a point* $\mathbf{Z}$ *from this distribution, such that it is inside a circle of radius* a *with center* $[\begin{matrix} 0 & 0 \end{matrix}]^\intercal$*, is*

$$p_a = 1 - e^{-\frac{a^2}{2}}$$

Proof. The probability density function of $\mathbf{Z}$, previously presented in Equation 3.5 for the general nD case, is

$$\phi(\mathbf{z}) = \frac{1}{2\pi} \exp\left(-\frac{1}{2}\mathbf{z}^\intercal\mathbf{z}\right)$$

and the circle of radius a and center $[\begin{matrix} 0 & 0 \end{matrix}]^\intercal$ is the set of points $S_a = \{\mathbf{x} \in \mathbb{R}^2 \ : \ |\mathbf{x}| \leq a\}$. Hence, the cumulative distribution of drawing a random point $\mathbf{Z}$ inside S_a is given by the area integral

$$p_a = P(\mathbf{Z} \in S_a) = \int_{S_a} \phi(\mathbf{z}) \ dS_a =$$

which, expressed in polar coordinates, becomes

$$p_a = \int_a^0 \int_0^{2\pi} r\phi(\mathbf{z}) \ d\theta \ dr = \int_a^0 \int_0^{2\pi} r\frac{e^{-r^2/2}}{2\pi} \ d\theta \ dr$$

since $r^2 = \mathbf{z}^\intercal\mathbf{z}$. By solving the last integral to the end, we obtain

$$p_a = \int_a^0 re^{-r^2/2} \ dr = e^{-r^2/2}\Big|_0^a = 1 - e^{-a^2/2}$$

as stated. *Note: The integration limits for the radius r are chosen such that, if we compute the limit integral from ∞ to 0, that is, over the whole $\mathbb{R}^2$, we get the value of 1, as expected.* $\square$