UNIVERSITY OF CRETE

SCHOOL OF SCIENCES AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

PROBABILISTIC GESTURE RECOGNITION

MARKOS SIGALAS

M. Sc. THESIS

HERAKLION, DECEMBER 2008

Περίληψη

Η επικοινωνία με τη χρήση χειρονομιών αποτελεί μία τόσο κοινή όσο και ζωτική μορφή αλληλεπίδρασης μεταξύ των μελών της ανθρώπινης κοινωνίας. Εκτός από την αλληλεπίδραση με άλλα άτομα ή αντικείμενα, οι χειρονομίες πολλές φορές αντικαθιστούν κάθε άλλη μορφή επικοινωνίας, όπως στην περίπτωση των κωφών. Από την άλλη, η χρήση ηλεκτρονικών υπολογιστών, αποτελεί ένα αναπόσπαστο κομμάτι της κοινωνίας μας, επηρρεάζοντας ποικιλοτρόπως την καθημερινότητα των ανθρώπων. Η εξέλιξη στον τομέα της πληροφορικής, καθιστά πλέον δυνατή την ανάπτυξη και χρήση νεών μεθόδων αλληλεπίδρασης μεταξύ ανθρώπων και μηχανών οι οποίες εκμεταλλεύονται στο έπακρο τη δυναμική των χειρονομιών.

Η παρούσα εργασία παρουσιάζει μια πιθανοκρατική προσέγγιση στην αναγνώριση χειρονομιών. Βασιζόμενη στην υπόθεση πως η μοντελοποίηση διαφόρων κοινών χειρονομιών δεν απαιτεί πληροφορία υψηλού επιπέδου, η προτεινόμενη προσέγγιση επιτυγχάνει να μειώσει την πολυπλοκότητα του εξεταζόμενου προβλήματος μειώνοντας τις διαστάσεις του χώρου παραμέτρων οι οποίες περιγράφουν τη θέση του κάθε χεριού.

Η χρησιμοποιούμενη μεθοδολογία για την παρακολούθηση των παραμέτρων αυτών εξάγει μια εύρωστη αναπαράσταση της θέσης του χεριού, επιτυγχάνοντας έτσι την αποτελεσματική χωροχρονική μοντελοποίηση των χειρονομιών. Αρχικά, περιοχές με χρώμα παρόμοιο με αυτό του δέρματος ανιχνεύονται στις εικόνες. Δεδομένου πως, συνήθως, το κεφάλι είναι η υψηλότερη ανιχνεύσιμη περιοχή, το ύψος μπορεί εύκολα να υπολογιστεί και, μέσω αυτού, να εκτιμηθούν τα μήκη των άκρων, με τη χρήση απλών ανθρωπομετρικών αναλογιών. Κατόπιν, οι εξισώσεις της αντίστροφης κινηματικής εξάγουν μια αρχική εκτίμηση των παραμέτρων του βραχίωνα, οι οποίες και παρακολουθούνται στο χρόνο μέσω particle filters. Η χρησιμοποίηση των particle filters σημαίνει πως παρακολουθούνται ταυτόχρονα πολλαπλές υποθέσεις, καθιστόντας δυνατή την ανάκαμψη από περιπτώσεις λανθασμένων εκτιμήσεων. Για να διασφαλιστεί η χρονική σταθερότητα και να προληφθούν ασυνέχειες, οι εξαγόμενες παράμετροι φιλτράρονται ανάλογα με τη σχετικότητά τους με τις προηγούμενες εξόδους, με αποτέλεσμα την εξαγωγή

ομαλών ακολουθιών, οι οποίες και χρησιμοποιούνται για τη μοντελοποίηση της κάθε χειρονομίας.

Το τελικό στάδιο της αναγνώρισης χειρονομιών αποτελείται από ένα σύνολο νευρωνικών δικτύων, υπεύθυνα, το κάθε ένα, για την αναγνώριση μίας μόνο χειρονομίας. Η χρησιμοποίηση πολλαπλών νευρωνικών δικτύων –αντί ενός γενικού- αποκλείει πιθανές αμφιβολίες, οι οποίες εγείρονται εξαιτίας των επικαλυπτώμενων μονοπατιών των χειρονομιών. Δεδομένου πως δεν υπάρχει κάποια πρότερη γνώση σχετικά με την εκδηλούμενη χειρονομία, οι ακολουθίες παραμέτρων τροφοδοτούνται ταυτόχρονα σε όλα τα νευρωνικά δίκτυα. Η κατάλληλη εκπαίδευση των δικτύων, εγγυάται πως μόνο ένα δίκτυο θα έχει υψηλή έξοδο σε κάθε χρονική στιγμή, καταλήγοντας στην αποτελεσματική αναγνώριση της εκτελούμενης χειρονομίας.

Abstract

Communication with the use of gestures is a very crucial and common form of interaction in human societies. Gestures not only allow us to interact with other people and objects, but, in some cases, substitute every other form of communication –deaf people for example. On the other hand, computers have become an inseparable part of our society, influencing many aspects of our daily lives in terms of communication and interaction. Evolution in the field of informatics has seen tremendously high speeds, mostly in the last few decades, enabling new forms of /Human-Computer Interaction/ (HCI) which fully exploit the dynamics of hand gestures.

In the current thesis, a probabilistic approach towards Hand Gesture Recognition is proposed. Based on the assumption that various common gestures can be modeled without the need of high-level information, the proposed approach achieves to reduce the complexity of the problem by decreasing the space dimensionality of the parameters, which describe the configuration of the arm.

The methodology for tracking the mentioned parameters, manages to extract a robust representation of the arm's pose and to end up with an efficient spatio-temporal gesture model. Initially, skin-colored blobs are being detected on the images. Since, usually, the highest detected skin-colored blob is the head, the height of the actor is easily calculated, which leads to an estimation of the size of the limbs, with the aid of simple anthropometric proportions. Once this is done, inverse kinematics equations serve for the extraction of an initial estimation of the arm's parameters, which are then tracked over time with the use of particle filters. The usage of particle filters implies that multiple hypotheses are being tracked simultaneously, enabling the recovery from cases where erroneous estimations occur. In order to assure time invariance and to prevent discontinuities, the extracted parameters are being filtered according to their relevancy to previous outputs, resulting with smooth parameter sequences, which are, therefore, used in order to model each hand gesture.

The final, gesture recognition, step consists of a set of neural networks, each of them responsible for the recognition of a single gesture. The usage of multiple neural networks –instead of using a global one- ensures the elimination of possible ambiguities due to overlapping gesture paths. Since there is no prior knowledge regarding the possible gesture being performed, the parameter sequences are being fed to all neural networks simultaneously. Appropriate supervised training of the networks, ensures that only one network at each time will produce high output, resulting in the successful recognition of the performed gesture.

Table of Contents

Table of Figures

Chapter 1

Introduction

Communication with the use of gestures is a very crucial and common form of interaction in human societies. Gestures not only allow us to interact with other people and objects, but, in some cases, substitute every other form of communication –deaf people for example. On the other hand, computers have become an inseparable part of our society, influencing many aspects of our daily lives in the meaning of communication and interaction.

Evolution in the field of informatics has seen tremendously high speeds, mostly in the last few decades, enabling new forms of *Human-Computer Interaction* (HCI) and giving birth to new technologies such as *Virtual Environments (VEs)* and *Intelligent Machines*. Despite the great advance, current ways of interaction –keyboard, mouse- limit the potential, together with the effectiveness and naturalness of HCI. Recent studies have shown that it is very natural to point at an object with our index finger or manipulate objects with our hand [37]. Moreover it is easier to understand other people while seeing them manipulating objects [24, 48]. These make clear the fact that new technologies should be emerged in order to achieve naturalness in interaction with computers, which, consequently, will increase the effectiveness of such systems.

The first step has been made towards interaction using speech [48]. The idea was to make computers understand our language, so that they could complete tasks, help in education (*e-Learning*), interact or just communicate with humans, with the use of audio analysis and speech synthesis. Combined with *Human Face Analysis and Recognition*, a more sophisticated –or semantic- way of interaction has been achieved, by exploiting the variety of emotional states a human face can express, which also means a "better" communication with computers [48, 58, 71]. However, interaction using speech and facial expressions covers just a part of human-to-human interaction. In the last several years there has been an effort so that other

means of communication are introduced in HCI. “These new means include a class of devices based on the spatial movement of the human arm: *hand gestures”* [73].

Hand gesture recognition first implementations were launched in the late 70’s, when gloves, equipped with sensors and processors, were recognizing movements of the hand and its fingers. However, glove-based gesture recognition systems, although accurate, impose limitations because of the equipment needed, which leads to a reduction of naturalness, the basic requirement for HCI. Therefore, research turned to vision-based gesture recognition systems [51], which is also the subject of the current thesis.

1.1 Problem Statement

The recognition of hand gestures using visual input is not a trivial task. It involves several steps which need to be carefully designed. The first step is the *modeling* of gestures. Usually this means a mathematical representation of hands, pose and gesture trajectories which will be used in order to interpret a gesture. *Interpretation* of each gesture contains a large amount of ambiguity, as gestures are directly dependent on the context. Thus, it is clear that the success of a gesture recognition system relies on the gesture model to be chosen.

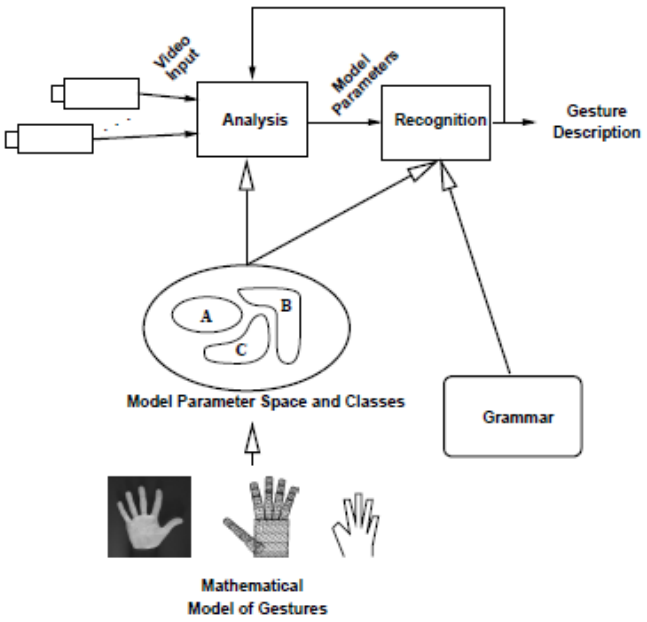


Figure 1: Block diagram of vision-based gesture interpretation system. [48]

Once the gesture model has been determined, the main task takes place. The input stream –video(s) or image(s) - is processed and some *features* are extracted.

The extracted set of features –which is consequently used in order to recognize a gesture- is another important factor in the gesture recognition process. They are context dependent and the effectiveness of the system is based on them.

These image features are being analyzed and in combination with a *Grammar,* they lead to a gesture description –interpretation. The *Grammar* contains the syntax of each gesture (for example a pointing gesture should consist of a specific sequence of movements) together with possible interaction with other communication modes like speech and facial expressions. In order to demonstrate the phases of gesture recognition process, Pavlovic in [48] presents the following global vision-based gesture interpretation system (Figure 1).

1.2 Approaches Towards Gesture Recognition

As mentioned earlier, gesture modeling and analysis is the most important part of Gesture Recognition. Huang and Pavlovic discriminate hand gesture recognition techniques according to the feature extraction method, and therefore to the gesture analysis approach, that each of them uses. Therefore we end up with two categories [37]:

- Glove-Based Techniques
- Vision-Based Techniques

Further –or different- classification is also possible. However, this categorization highlights the two basic approaches towards hand gesture recognition. Glove-based systems use a cloth-made glove equipped with sensors in order to capture hand and finger movements. As mentioned earlier, these techniques, although robust, constraint the user, as he has to be equipped, apart from the glove, with sensors and wires. On the other hand, vision-based techniques process video sequences of the user, and try to detect hand movements –and consequently hand gestures- using features detected on the images. This section presents major works based on these techniques.

1.2.1 Glove-based techniques

A Glove-Based Gesture Recognition system consists of cloth made glove, sensors, electronics for data processing and power supply. While worn by a user, the glove extracts features concerning the configuration of his/her hand together with its movement (trajectories) [81]. By analyzing and, thereafter, interpreting these features, one can extract information about the ongoing gesture.

The first glove-based system was introduced by Rich Sayre in 1977. By using flexible tubes and photocells, it could sense finger bending/movement. Gary Grimes, in 1983 designed the Digital Data Entry Glove, which could recognize up to 80 characters of the Single Hand Manual Alphabet for Deaf. These gloves were hard wired and served a very specific number of applications; moreover they were never commercialized [28, 81].

The VPL DataGlove, developed by Zimmerman [15], was, on the other hand, the most successful glove and the one that made glove-based systems popular. Similarly to Sayre's glove, the DataGlove could recognize finger bending and hand movement using sensors both for the fingers and the hand itself. After that, many more glove-based gesture recognition systems were implemented for both research and commercial purposes. Indicatively some of them are cited. The PowerGlove introduced by Mattel in 1989 was used for Nintendo game consoles [19]. CyberGlove (1992) and Humanglove (1997) are considered two of the most accurate gloves currently available whilst DigjiGlove and StrinGlove are the most recent ones [81]. All of the mentioned glove-based systems sense the bending of fingers and track hand movements.

1.2.2 Vision-based techniques

Vision-based techniques use visual input(s) in order to extract the features to be used in the gesture analysis phase. Based on the nature of these features, vision-based techniques can be further broken into three categories [73]:

- Model-Based Approaches
- Appearance-Based Approaches
- Low-Level Features Approaches

1.2.2.1 Model-based approaches

Features used in this first category are derived from kinematics models. The aim is to compute the pose of the arm and/or hand together with the joint angles. Kinematics parameters are being extracted from the 2D projections of a 3D hand model. Most models are based on the simplified skeletons of the human hand/arm [48] as shown in Figure 2, or more complicated models designed with the help of CAD systems (Figure 3).

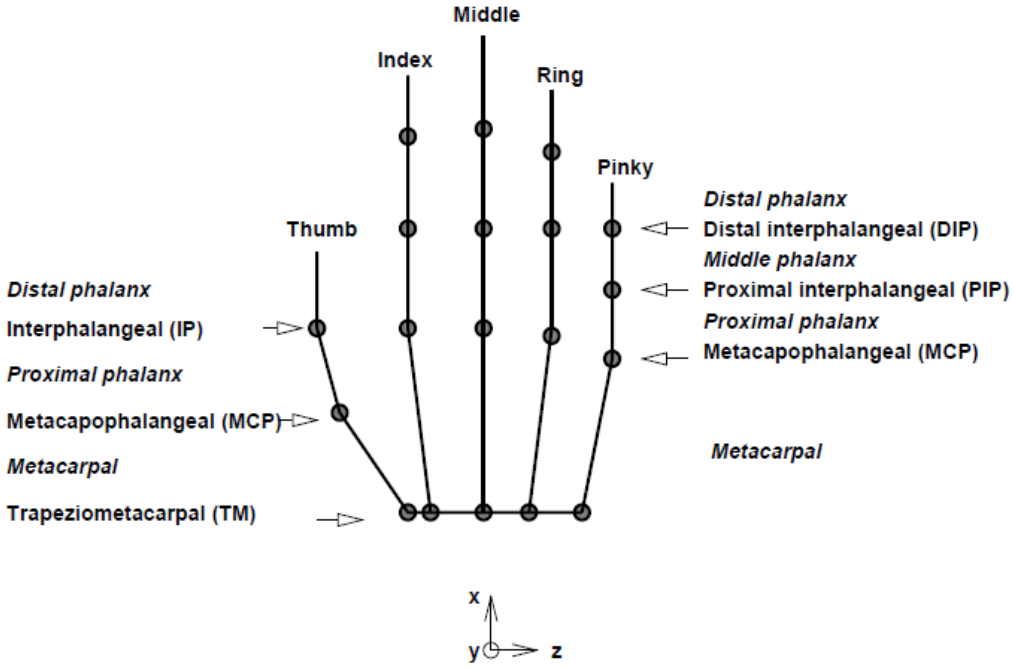


Figure 2: Skeleton-based model of the human hand.

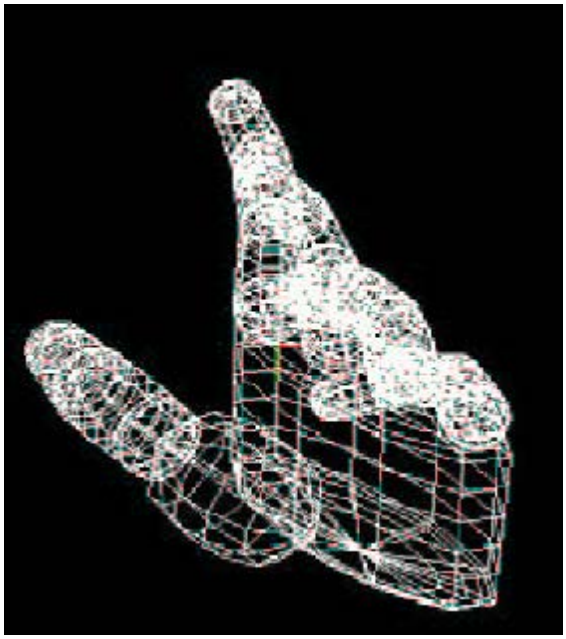


Figure 3: Hand 3D model.

Based on the model tracking work of Lowe [21], Rehg and Kanade, in 1994, [33] proposed one of the very first approaches towards hand tracking. By fitting a 3D hand model into the image, the goal is to extract a total of 27 parameters, 21 for joint angles and 6 for the hand pose. At each image, this method generates several

hypotheses for the parameters, selecting the one with the less *miscorrespondence.* The number and the range of parameters to be estimated, impose a great disadvantage to this method. Apart from the fact that the initial parameter estimation has to be close to the real values, this approach is vulnerable to image noise and hand self-occlusions. Similar approaches with equal or lesser DoFs have been used by other authors [1, 26, 39].

Recent works on model-based hand gesture recognition use probabilistic methods in order to estimate hand parameters. Wu et al. [66], use a Bayesian framework for the optimal estimation of the parameters. *Kalman filter* can also be used to "estimate, interpolate and predict the motion parameters" [64], although the assumptions of linear systems and Gaussian noise do not always stand. Due to high-dimensionality of the parameter space, probabilistic techniques usually turn to be very expensive (in computational terms), and, thus, far from real time implementations.

The *Condensation* algorithm and its extensions are also used in some other works, for hand tracking and model fitting. Black and Jepson in [53] and Rittscher and Blake in [61] estimate motion parameters and trajectories using extensions (and mixtures) of the *Condensation* algorithm.

1.2.2.2 Appearance-based approaches

Appearance or View Based Approaches model the hand as a collection of 2D intensity images [73] (can be thought of as a template) and relate the appearance of the hand (in the image) with it. Consequently each gesture is modeled as a sequence of views. In order to relate each pose to a template image, a *similarity* factor has to be used.

Towards this direction, eigenspace seems to be an efficient approach as it can represent a high-dimensional space with a small set of vectors. One of the first tasks that was addressed using eigenspace formulation was face recognition. Turk and Pentland in 1991 [23], instead of trying to estimate 3D geometry parameters of the face, attempt to describe it with a set of 2D characteristics and, thus, transform the face recognition problem to a 2D one. The features used for recognition are called "*eigenfaces",* as they are eigenvectors of the set of faces.

Face recognition using eigenspace methods has been quite successful and, therefore, the same framework has been applied for hand gesture recognition. Black and Jepson [52] proposed such an approach. Their work introduces major improvements to the original eigenspace approach [73] being able to cope with occlusions, background and transformation invariance. The method developed

works adequately for a small set of gestures –the authors track four gestures- while its efficiency is reduced for larger ones.

Contours is another approach used for hand gesture recognition. Usually, a contour is formed from the edge of the hand [27, 30] or from the polar coordinates [34] –termed as "*signature*". The idea in these approaches is to match the contour of the hand with the template model. Moreover contours can be used as the basis for further eigenspace analysis [35, 40, 48].

Finally, another technique is the use of fingertips positions as features for the gesture analysis, based on the assumption that the palm is rigid and that finger DoF number is limited [25, 39, 48]. Most of these approaches use the 2D locations of fingertips and palms in order to match the image with the template gesture. The works of Davis and Shah [31] and Quek et al. [42] are examples of fingertips approach.

1.2.2.3 Approaches based on low-level features

Because of the fact that previous approaches are vulnerable to image noise, many researchers turned to methods that use low-level image characteristics. This idea is based on the assumption that in hand gesture recognition application, "all that is required is a mapping between input video and gesture" [73], and therefore, the full reconstruction of the hand is not needed. The centroid of the palm , ellipsoid descriptors [56] and optical flow of the hand [54, 68] are examples of low-level image features used for hand gesture recognition. Although accurate and noise invariant, low-level features seem inefficient in arbitrary scenes.

1.3 Applications of Hand Gesture Recognition

As gesture recognition techniques turn to be more and more accurate, they offer efficient solution to various applications. Some of them are *Sign Language, Virtual Environments (VEs), 3D Modeling, Human-Robot Instruction, Multimodal interaction, Gesture-to-Speech, Presentations, Television Control* and other [60]. In this section we present several works done on some of these fields.

1.3.1 Sign language

An important area where gesture recognition techniques apply is that of Sign Language. Since sign languages consist of gestures, it came naturally for gesture recognition research to help towards this direction. Starner [50] and Kadous [38] managed to recognize forty words of the American Sign language and 95 words of

the Australian Sign Language, respectively. Murakami and Taguchi [22] recognized both finger and sign words of the Japanese Sign Language, while Imagawa et al. [55] implemented a bi-directional sign language translator.

1.3.2 Virtual environments

Virtual Environments is a relatively new field in HCI. Virtual conferencing or chatting (like the new *lively* from Google [82]) or virtual reality games (as *Second life [70]*) are some applications where users can interact with other people and objects, while navigating through the virtual world. The need for more *naturalness* in such applications gave birth to the idea of using Gesture Recognition together with the VEs. *Battle-View* of Pavlovic and Berry [51] is an example of a virtual battlefield, where hand gestures are used in order both to navigate the VE and manipulate objects in it. Other works as well [8, 31, 43, 44] use hand gestures in order to interact with and manipulate objects in VEs.

1.3.3 3D modeling

3D modeling applications can gain easiness by using hand gestures. Users can design, create and manipulate 3D objects faster and in a more natural way since they don't have to use a keyboard and/or mouse; devices which limit the potentials of a 3D designer. Zeleznik's SKETCH [75] or VLEGO of Kiyokawa et al. [45] are examples of applications where hand gestures help to the creation of 3D models.

1.3.4 Human-robot manipulation and instruction

Hand gestures can also facilitate the manipulation and "*teaching*" of robots and therefore boost the effectiveness of human-robot interaction. GripSee of Becker et al. [59] and Rogalla et al. [67] present platforms for manipulating and instructing robots, while Lee and Xu developed a system which allows robots to interactively *"learn"* new hand gestures.

1.3.5 Multimodal interaction

The field of Multimodal Interaction applications can also be thought of as an extension to the previous section or, moreover, as a subpart of Virtual Environment applications. The combination of speech and hand gestures offers more naturalness to the human-machine interaction together with the raise of its effectiveness, as speech recognition mistakes can be corrected –or minimized- by hand gesture recognition and vice-versa [46]. Brewster et al. [69] propose a method for multimodal interaction using both audio and gestural input.

1.3.6 Television control

Towards the idea of the "*Intelligent House*", where the user can automatically adjust the lightning, humidity, or answer phone calls, it came naturally for the hand gestures to play an important role. An aspect of *Intelligent House,* in which hand gesture recognition techniques apply, is the control of a television. Freeman's and Weissman's system [36] is a paradigm of such an application, where users can adjust the volume, change channels or turn on and off a television by the use of their hands.

1.4 Proposed Approach

In this work, a probabilistic approach towards hand gesture recognition is proposed. The modeling of the hand has been made using Kinematics equations for the hands [4, 77, 79], the features extracted from images are skin color and centroids of face and hands, and, finally, the tracking of the hands and the recognition of the pose is being done using *Particle Filtering* [74]. Neural networks are being used in order to recognize the ongoing gesture.

The proposed approach relies on the assumption that human perception of basic gestures is mainly based on the arm and not on the end effector (palm and fingers). One does not need to know the exact position and pose of fingers and the values of joint angles or the orientation of the palm in order to recognize a pointing or a "STOP" gesture. This allows us to reduce the number of DoFs for each hand to a minimum of four –three for the shoulder and one for the elbow [79]. For both hands and by taking into account the rotation of the body (around the y-axis) we end up with a total of nine DoFs.

First, face and hands are being determined by detecting skin color areas in the image [72]. The 2D projections of face and hands are being used in order to estimate their position in the world. Once the 3D coordinates have been calculated, the image is being represented by the kinematics model. As mentioned earlier, the points of interest are solely the face and the joints of shoulders and elbows, as shown in Figure 4. In order to have an initial estimation for the model parameters (joint angles) –which will serve as seed for hand tracking- as close as possible to the real values, we use the Inverse Kinematics method proposed in [77].

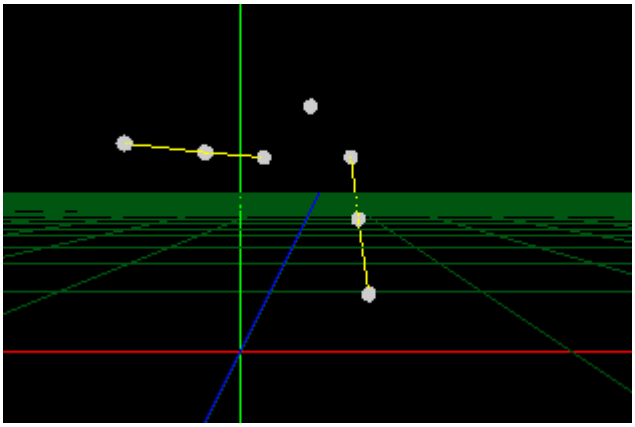


Figure 4: Pose 3D model.

This initial estimation is the seed for new hypotheses generation from the parameters tracking mechanism implemented with particle filters. Our weighting factor consists of simple Euclidean distance of the 2D projections of the estimated pose and the actual hand. Foreground-background subtraction is also used in order to ensure that joints that cannot be tracked (the elbow) will match the model.

The tracked parameters serve as input to the Neural Networks for the determination of the current gesture. In this phase, we face three problems: complexity, scalability and time/duration dependency of gestures. The first two problems have been approached with the use of a separate neural network for each gesture. The aim of this decision was to keep neural networks as simple as possible and to overcome any limitations on the number of gestures to be recognized in the future. The third problem concerns the training phase of the neural networks (will refer to it in Section 5). We want to ensure that a gesture will be recognized even if the ongoing gesture differs in duration from the one used in the training data. Therefore, input is given to neural networks whenever there is a *significant change* of the parameters values.

Chapter 2

Hand Gestures

Finding a suitable definition for hand gestures is not an easy task. There have been many psycholinguistic studies, trying to describe and analyze human hand gestures. Thieffry in [10] states that "*every gesture is the physical expression of mental concept*". Webster dictionary definition for gestures is: "… the use of motion of the limbs or body as a means of expression; a movement usually of the body or limbs that expresses or emphasizes an idea, sentiment, or attitude" [48]. In general, gestures can be conceived as a non-verbal form of communication and expressions of emotions and information.

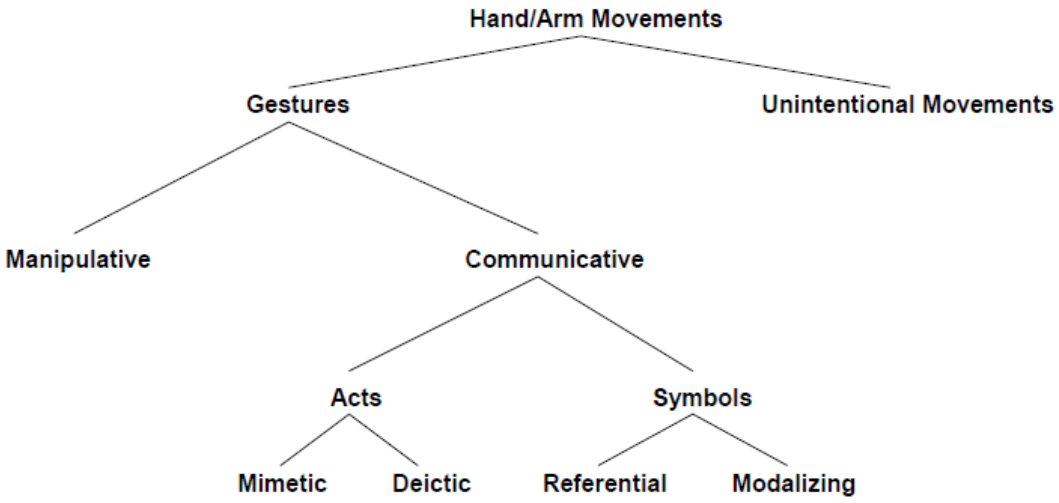


Figure 5: Gestural Taxonomy [48]

Hand gestures have a wide variety, depending on the context. Thus, several categorizations can arise such as *conversational, controlling, manipulating and communicative gestures [63].* One can assume that *conversational* and *controlling* gestures consist subsets of *communicative* gestures. Sign languages for deaf or a

navigation gesture –"go there"- are examples of communicative and controlling gestures, respectively. Communicative and controlling gestures are the classes of gestures that research is mainly turned to, as vision-based recognition systems can efficiently help. The next figure, borrowed from Pavlovic [48], shows the various aspects of hand gestures.

2.1 Hand Gestures in HCI

Hand gestures, in the context of HCI, have a somewhat different meaning, while equally difficult to define. Apart from the natural use of the hand as a manipulator, in HCI, one should emphasize on its use for interacting with a computer –the "practical gestures" [13]. This implies that gestures could –or even should- be represented differently in the scope of HCI than in real life, in order to exploit to the maximum current technologies. In many cases, simple models or representations of hand gestures turn to be very efficient.

A high-level classification of hand gestures is into static and dynamic [78]. Static gestures assume a certain pose of the body and hand, while dynamic ones present temporal and spatial variation. It is obvious that for static gestures, the points of interest are solely the pose and the position in space of the hands and/or arms, while, for the dynamic, the movement(s) of the hand/arm is also needed to describe the gesture. Although static gestures do not need any information on the trajectories of hands/arms, they can be thought of as being dynamic gestures without changes through time. By gathering these together, we end up with the following general definition [48]:

Definition 1: *Let* $\mathcal{H}(t) \in \mathcal{S}$ *be a vector that describes the pose of the hands and/or arms and their spatial position within an environmental at time* t *in the parameter space* $\mathcal{S}$*.* $\mathcal{S}$ *is application dependent and should be defined accordingly. A hand gesture is represented by a trajectory in the parameter space* $\mathcal{S}$ *over a suitably defined interval* $\mathcal{I}$*.*

Generally speaking, the gesture modeling phase *defines* the corresponding gesture. As obvious, from the above definition, a *spatiotemporal* model is suggested. In the next sections, we describe the *spatial* and *temporal* model of hand gestures used in the current work.

2.1.1 Spatial modeling of gestures

Determination of the parameter space of a hand gesture is strictly dependent on the context of each application. For some applications the parameter space consists only of the positions of the palm or fingertips while for other the values of all joints of the arm and fingers are needed to form the parameter space. For the purpose of this work (as stated in 1.4), the parameter space $\mathcal{S}$ consists of the angles of shoulder and elbow joints. Although 3D-space information is needed to describe position and movements of hand/arm, we prefer using the angles since, by using the *forward* and *inverse kinematics* equations (we will refer in a following section), we can easily map one parameter space to the other –angles and 3D parameter space.

Definition 2: *Parameter space* $\mathcal{S}$*:*

$\mathcal{S} = \{x : x =$ *angles of shoulder and elbow joints*$\}$

The above definition allows us to minimize the size of parameter space and thus the complexity of the model. An assumption made is that, since deformations of human skin do not provide any additional information [48], a human arm can be represented as an articulated object as shown in Figure 4.

2.1.2 Temporal modeling of gestures

What remains is the determination of a suitable time interval $\mathcal{I}$. Kendon [13] analyzes dynamic gestures into three phases: *preparation, stroke and retraction. Preparation* and *retraction* phases consist of movement from and towards resting position, before and after the gesture, respectively. As *stroke* contains most –if not all- of the information –*"definite for and enhanced dynamic qualities" [13]*- of the gesture, it can be clearly distinguished [47]. Moreover, gesture phases can also be distinguished by the speed of changes. *Preparation* and *retraction* show rapid position changes, while *stroke* in general presents slower hand motion [48].

In the current research, in order to define the gesture temporal model, we adopted a set of rules proposed by Quek [32, 41] and Pavlovic [48]:

Definition 3: Temporal segmentation of gestures:

1. *Gesture interval consists of three phases: preparation, stroke and retraction.*
2. *Hand configuration during the stroke follows a classifiable path in the parameter space.*
3. *Gestures are confined to a specified spatial volume (workspace).*

4. *Repetitive hand movements are gestures.*

While this set of rules is sufficient for most of gestures, it fails to describe gestures related to the speech (“beats”). Since, however, this is not the subject of this work, we won’t expand further.

Chapter 3

Background Tools and Mathematics

The method used in the proposed gesture recognition approach can be broken down into four phases: *preliminary*, *head and palms location, hand kinematics tracking* and *gesture recognition.*

- In the *preliminary phase*, cameras are calibrated in order to extract the intrinsic and extrinsic parameters of the stereo system. These parameters –although not needed to be very accurate- are vital for the system, since all following calculations are based on them.
- After having calibrated the cameras, the 3D position of the head and palms is extracted during the second phase. Firstly, a skin-color tracker is responsible for locating skin-color blobs on the images. In order to cope with depth ambiguities, particle filters are applied, resulting with an estimation about the location in space of the detected skin-colored blobs.
- Each arm is then represented by a set of four angles which can fully describe its pose. These parameters are called *kinematics* and are being extracted and tracked during the third phase. The calculated 3D position of skin-colored blobs is used for the initial estimation of kinematics, while particle filters are responsible for tracking these parameters over time.
- Neural networks, are finally responsible for processing the extracted kinematic parameters and recognize a possible ongoing gesture.

As it may be obvious from the above, this work covers several research fields. Image processing techniques are combined with probabilistic tracking methods, kinematics equations and neural networks so that an efficient gesture recognition system is composed as depicted thoroughly in Figure 6. This section is intended to describe the Mathematics borrowed from each research field and/or the tools used for their implementation.

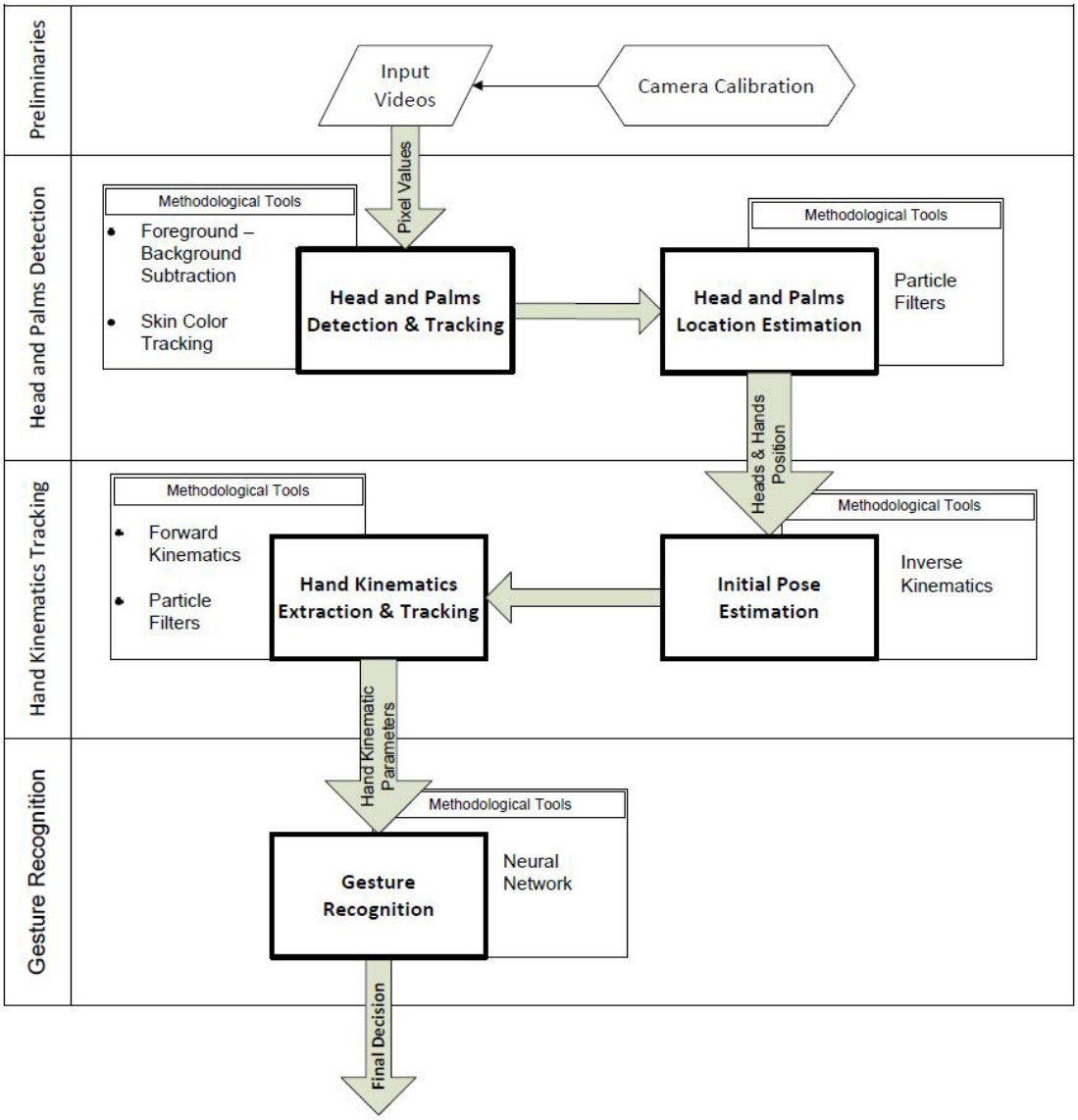


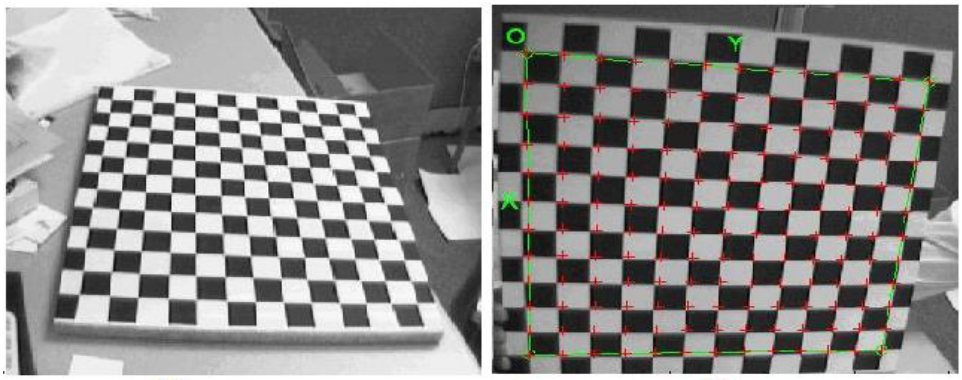
Figure 6: Gesture Recognition System Overview

3.1 Preliminary Phase -- Camera Calibration

During the *preliminary phase,* parameters which describe the camera model(s) are being extracted. This process is called *Camera Calibration,* and is essential for the system's operation, as the extracted parameters determine –up to a scale- the way that the input will be translated. The accuracy of the *Calibration parameters* affects the accuracy of all future calculations. Therefore, minimization of the estimation error is one of the main goals.

In order to preserve depth information, a stereo pair is used to capture the input video. This fact implies that cameras should be calibrated before processing the input. From the camera calibration process, *intrinsic and extrinsic* parameters (or

simply *intrinsics* and *extrinsics,* respectively) are derived for the stereo pair. *Intrinsics* concern the *internal* parameters of each camera, such as focal length(s), aspect ratio, principal points, and distortion coefficients, together with the corresponding uncertainty. *Extrinsics* refer to the relative position of the two cameras in space. Both *intrinsics* and *extrinsics* are necessary and a good calibration of cameras is crucial for the accurate operation of the system.



(a) (b)

```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.39535   657.76309 ] ± [ 0.34691   0.37111 ]
Principal point:       cc = [ 302.98368   242.61630 ] ± [ 0.70546   0.64553 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]
                          => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25584   0.12758   -0.00021   0.00003   0.00000 ]
                          ± [ 0.00271   0.01076   0.00015   0.00014   0.00000 ]

Pixel error:          err = [ 0.12668   0.12604 ]
```

(c)

Figure 7: (a) Chessboard pattern for camera calibration. (b) Extracted grid. (c) Camera intrinsics.

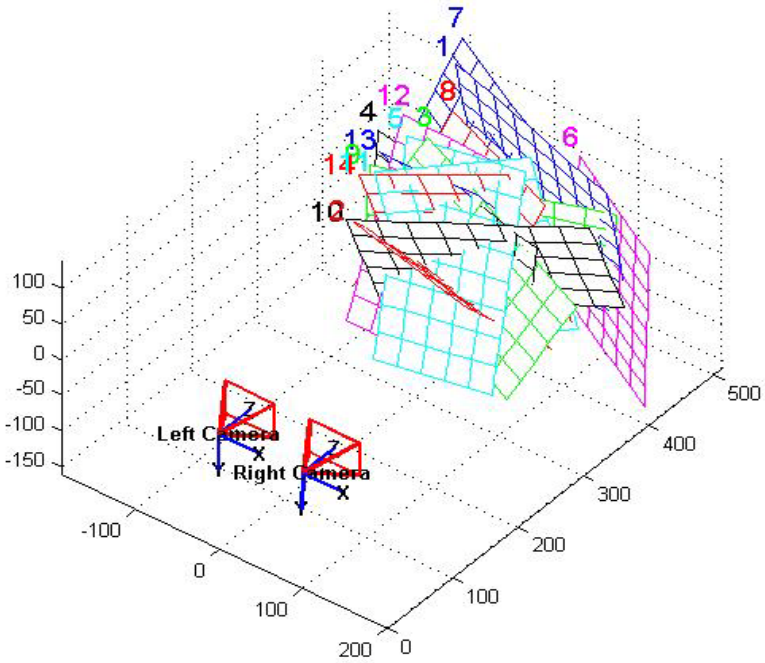


Figure 8: Extrinsics of the stereo pair.

For the purposes of camera calibration, the Matlab® toolbox of Jean-Yves Bouguet [3] has been used. The user marks the four external corners of a calibration pattern, as the one shown in Figure 7a, and defines the number of bounded squares and their dimensions. Once this is done, the corners on the grid are detected (Figure 7b). Since the distance between two corners is known and, thus, their relative (not absolute) position in space, the projection matrix can be calculated[1]. The intrinsic parameters can now be derived from the estimated projection matrix (Figure 7c).

For the estimation of the extrinsics, the process is more or less the same. Note that the images used for the calculation of the intrinsics of each camera should be snapshots of the same scene (from the corresponding point of view). This implies that the correspondence problem is solved, and therefore, the *Fundamental* as well as the *Essential Matrices* can be estimated, from which, the extrinsic parameters are extracted (Figure 8).

3.2 Skin-Color Detection and Tracking Tools

The Skin-*color Detection and Tracking Phase* provides the necessary information for the *kinematics tracking phase*. Skin-colored object, namely head and hand palms, are extracted and tracked over time. Argyros and Lourakis [72] proposed a method for detecting and tracking skin-colored objects over time. An improved version of the tracker [80], implements foreground-background subtraction prior to skin-color detection. By subtracting the foreground from the background (namely the dynamic from the static area of the image), the image area to be processed is minimized, while ambiguities due to color similarities of the background are practically eliminated.

3.2.1 Foreground-background subtraction

Stauffer and Grimson [62] proposed a recursive algorithm for the problem of foreground-background subtraction, by imposing a Gaussian Mixture Model (GMM) on each pixel of the image. The parameters of the model are updated for each input sample (image frame) and by simultaneously selecting the appropriate number of its (model's) components, foreground and background areas are determined.

In general, by denoting the value of a pixel at time t in RGB by $\vec{x}^{(t)}$, a pixel will probably belong to the background when

[1] Projection is discussed thoroughly in section 1.7.1.

$$\frac{p(BG|\vec{x}^{(t)})}{p(FG|\vec{x}^{(t)})} = \frac{p(\vec{x}^{(t)}|BG)p(BG)}{p(\vec{x}^{(t)}|FG)p(FG)}, \quad (1)$$

is larger than 1 and vice versa. In the general case, however, it is more likely that no information about the place, time and frequency of appearance of a foreground object is a priori known. Moreover, changes to the scene due to illumination changes and shadows, or even the addition (or equally the subtraction) of an object, should be adapted by the algorithm [76]. Therefore, a background model $p(\vec{x}^{(t)}|BG)$, which takes into account the *history* of each pixel and is updated through time, should be formed.

3.2.1.1 Background model

By assuming a uniform distribution for the appearance of the foreground objects $p(\vec{x}^{(t)}|FG)$, a pixel belongs to the background if

$$p(\vec{x}^{(t)}|BG) > c_{thr} \ (= p(\vec{x}^{(t)}|FG)p(FG)/p(BG)), \quad (2)$$

where c_{thr} is an appropriate threshold value. The background model is estimated from a training set $\mathcal{X}$ and is denoted by $\hat{p}(\vec{x}^{(t)}|\mathcal{X}, BG)$. In order to cope with the scene changes, the training set should be updated for each new sample so that old ones are discarded and the model's density is re-estimated [76].

At time *t,* the training set consists of $\mathcal{X}_T = \{x^{(t)}, \dots, x^{(t-T)}\}$ where T is a reasonable adaptation period and $x^{(t)}$ denotes the corresponding sample. Whenever a new sample arrives, the training set is updated and the density is re-estimated. As it is probable that foreground objects will be contained in the new sample, the model is now denoted as $\hat{p}(\vec{x}^{(t)}|\mathcal{X}_T, BG + FG)$. The model is described by a GMM with M components:

$$\hat{p}(\vec{x}|\mathcal{X}_T, BG + FG) = \sum_{m=1}^{M} \hat{\pi}_m \, \mathcal{N}(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m^2 I), \quad (3)$$

where $\hat{\vec{\mu}}_1, \dots, \hat{\vec{\mu}}_M$ are the estimates of the means, $\hat{\sigma}_1^2, \dots, \hat{\sigma}_M^2$ the estimates of the variances of the Gaussian components and $\hat{\pi}_m$ are the estimated mixing weights.

The number of the GMM's components basically denotes the *states* of the pixel and is self-determined. Assume for example that the scene consists of tree leafs which are moved by the wind. Therefore a certain pixel's value might change continuously from green (leaf) to blue (sky). In this case M=2 and whenever this pixel gets a value *close* to the means of either of two components, the pixel should be *labeled* as belonging to the background. However if the value of the pixel turns to white, a new component will be created. The update functions together with the

decision of whether the pixel belongs to the foreground are described in the next section.

3.2.1.2 Update equations

When a new data sample $\vec{x}^{(t)}$ arrives, the model parameters are recursively updated as follows [11]:

$$\hat{\pi}_m \leftarrow \hat{\pi}_m + \alpha(o_m^{(t)} - \hat{\pi}_m), \tag{4}$$

$$\hat{\vec{\mu}}_m \leftarrow \hat{\vec{\mu}}_m + o_m^{(t)}(\alpha / \hat{\pi}_m)\vec{\delta}_m, \tag{5}$$

$$\hat{\sigma}_m^2 \leftarrow \hat{\sigma}_m^2 + o_m^{(t)}(\alpha / \hat{\pi}_m)(\vec{\delta}_m^T \vec{\delta}_m - \hat{\sigma}_m^2), \tag{6}$$

where $\vec{\delta}_m = \vec{x}^{(t)} - \hat{\vec{\mu}}_m$. α defines an exponential decaying envelope so that old samples influence is decreased and is equal to 1/T so that the components of the GMM add up to 1. $o_m^{(t)}$ is the *ownership* factor and is equal to 1 for the "*close*" component with the largest weight and 0 for the rest. A component is "*close*" to the sample when the Mahalanobis[2] distance is less than a predefined value. If there is no "*close*" component, a new component is generated with $\hat{\pi}_{M+1} = \alpha$, $\hat{\vec{\mu}}_{\mathrm{M+1}} = \vec{x}^{(t)}$ and $\hat{\sigma}_{\mathrm{M+1}} = \sigma_0$ where σ_0 an appropriate initial variance.

A method for determining whether a component refers to a foreground object –if any- is finally needed. During the GMM update, the algorithm simultaneously clusters the components. Usually, foreground objects will be represented by clusters with relatively small weights. So, the background model can be approximated by the first B largest clusters [76]:

$$\hat{p}(\vec{x} \mid \mathrm{X}_T, BG) \sim \sum_{m=1}^{B} \hat{\pi}_m \mathrm{N}(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m^2 I). \tag{7}$$

By sorting the components according to their weight in descending order we can easily define B as:

$$B = \arg\min_b \left(\sum_{m=1}^{b} \hat{\pi}_m > (1 - c_f)\right), \tag{8}$$

[2] The Mahalanobis distance general form is $D_m(x) = \sqrt{(x-\mu)^{\mathrm{T}} \Sigma^{-1} (x-\mu)}$.

Here the squared distance from the m[th] component is calculated as $D_m^2 = \vec{\delta}_m^T \vec{\delta}_m / \hat{\sigma}_m^2$.

where c_f denotes the temporal portion of the data that can belong to foreground objects. For example, let c_f=0.3. This means that an *intruding* object is determined as foreground for time up to 0.3T. If that object remains for more than 0.3T, it will form a new cluster and, thus, will be considered as background.

3.2.1.3 Examples

The results of the foreground-background subtraction algorithm are demonstrated in the following figure. In Figure 9(a) the original input is shown. Figure 9(b) shows the result of the application of the algorithm on the original image, where the foreground object is indeed distinguished. Finally, in Figure 9(c) the foreground mask is demonstrated. Pixel values represent the probability of the corresponding pixel to belong to the foreground. This is the reason why in some parts of the mask image, pixels are not white (increased certainty) but grey. Shadow or illumination changes, impose changes to the background which, however, are not strong.

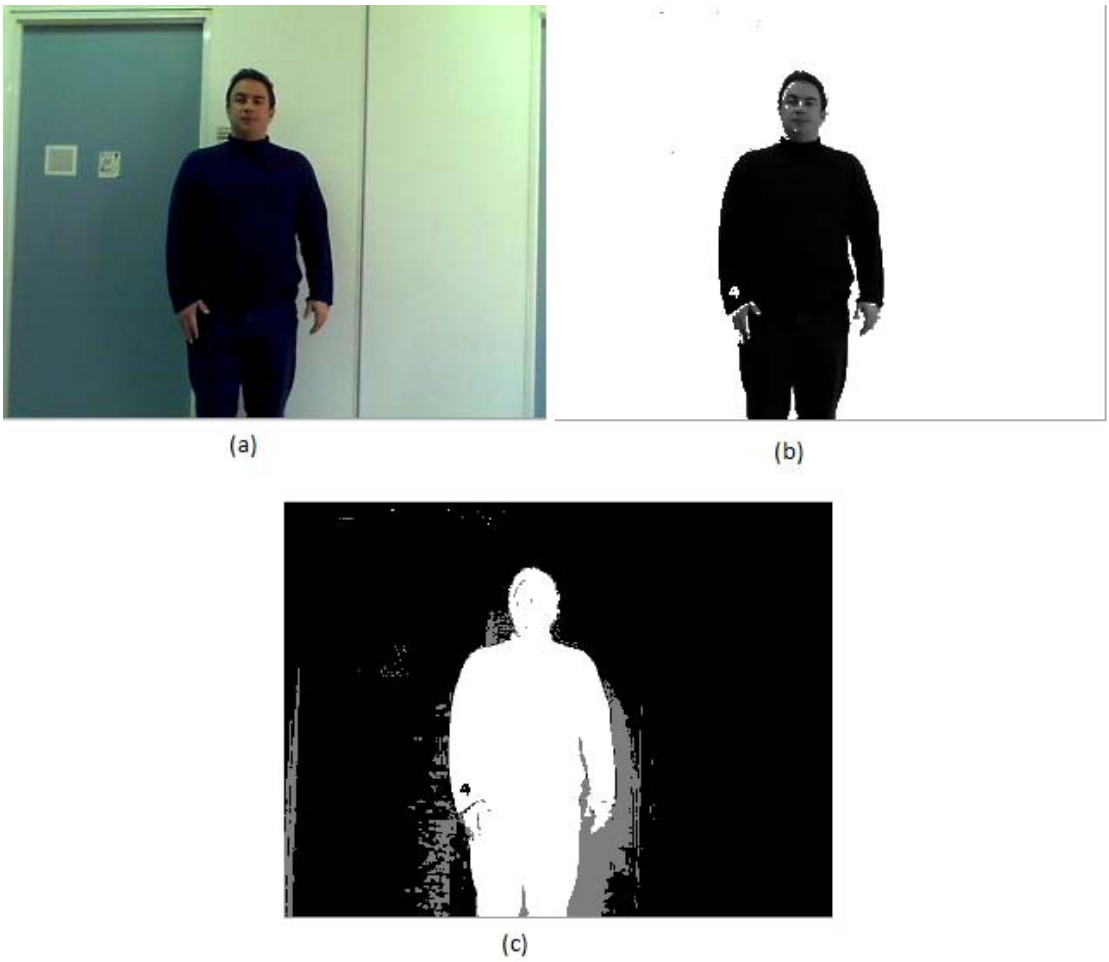


Figure 9: (a) Original Input, (b) Subtracted Foreground, (c) Foreground Mask

3.2.2 Skin-color detection and tracking

For the purposes of this thesis, a method that can detect and track over time multiple skin-colored objects, namely the features of interest, should be implemented. Argyros's and Lourakis's method [72] suitably satisfies our system's need, as it offers the possibility of real-time (low-cost) detection and tracking of multiple skin-color objects. Moreover, this method copes with the problem of illumination changes, as it self-adapts skin-color probabilities over time.

The skin-color tracker can be broken down into three phases: *off-line training, skin-color detection and hypothesis tracking*. During the training phase, the user manually marks skin-color areas so that prior probabilities are calculated. On detection phase these probabilities are being used to detect skin-color *blobs* and are updated simultaneously. The detected blobs give birth to object hypotheses, which are then tracked over time on the last phase.

3.2.2.1 Off-line training

For training, a small set of YUV 4:2:2 input images is used, on which, the user manually marks skin-colored regions (*ground-truth)*, as shown in Figure 10. As the Y-Component of this color representation corresponds to image illumination it can be easily omitted in order achieve two goals:

- Robustness to illumination changes and
- Increase of tracker's efficiency as problem's dimensionality is reduced.

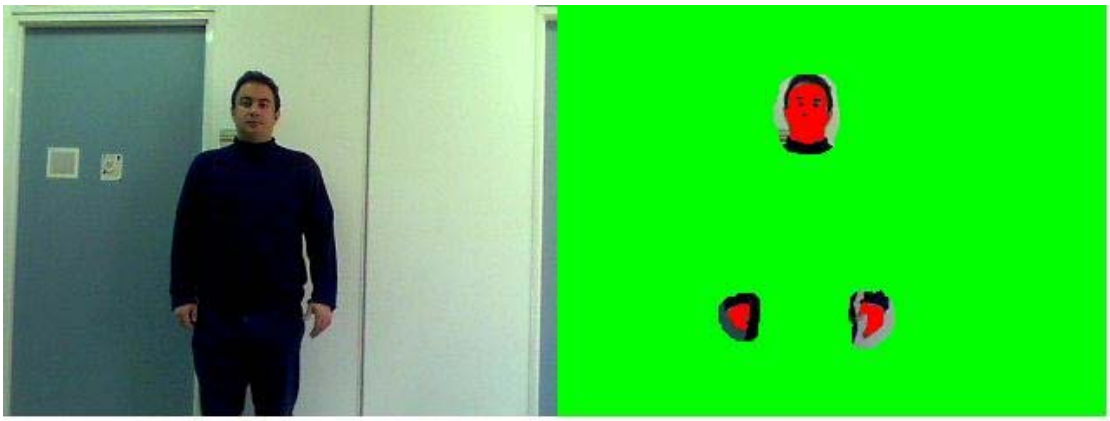


Figure 10: (a) The original image, (b) Marked image for tracker's training. Non skin-color regions are marked with green and skin-color ones with red. Some areas have not been marked in order to avoid ambiguities.

Let *c(x,y)* be the color of image point *I(x,y).* The marked input set is used to compute the prior probability *P(s)* of skin color in the image, *P(c)* of the occurrence of each color in the image and *P(c|s)* of a color being skin color.

3.2.2.2 Skin-color detection

After the calculation of prior probabilities, the wanted probability *P(s|c)* –namely the probability of each color being skin-color- can be easily derived from the Bayes rule as follows:

$$P(s \mid c) = \frac{P(c \mid s)P(s)}{P(c)}. \tag{9}$$

In order to decide if a color is skin-color, $P(s \mid c) > T_{\max}$ should stand, where $T_{\max}$ is a suitable selected threshold (based mostly on the scene). Points which satisfy the above condition are the seeds of potential blobs. Adjacent points with $P(s \mid c) > T_{\max}$ belong to the same blob. For the neighboring points of the seed ones, hysteresis thresholding is imposed to determine whether or not they should be treated as skin-color. Therefore, adjacent to skin-color point with $P(s \mid c) > T_{\min}$, where $T_{\min} < T_{\max}$, are recursively added to the corresponding blob. A connected components algorithm is then responsible to assign labels to image points of each blob. Finally, in order to eliminate small blobs, formed due to noise, size filtering is applied.

Because of the fact that the input set is relatively small, wrong results may occur – false positives or negatives. In this case, the user can manually correct this error by providing the ground-truth to the tracker. Moreover, the results of the tracker can be used as a self-adaptation method by continuously updating the prior probabilities.

Despite the fact that the chosen color representation provides illumination invariance up to a scale, poor results may occur due to illumination changes. In order to cope with this problem, the tracker maintains two sets of prior probabilities. Off-line training ones (*P(s), P(c) and P(c|s))* and $P_w(s)$, $P_w(c)$, $P_w(c|s)$ which correspond to the updated on-line probabilities for the detections in the *w* most recent frames. As illumination variations are context-dependent, the second set represents better the actual probabilities and can adapt to illumination conditions. Skin-color detection is then performed based on:

$$P(s \mid c) = \gamma P(s \mid c) + (1-\gamma) P_w(s \mid c), \tag{10}$$

where *P(s|c)* and $P_w(s|c)$ can be computed by Eq.9 using, however, the updated prior probabilities produced from both the training set and the detections during the

last *w* frames. *γ* is a sensitivity parameter which defines the influence of the training set in the detection process.

3.2.2.3 Skin-colored object tracking

The detected blobs are associated with object hypotheses, which are tracked over time. The relation here is not necessarily 1-1 which means that a blob can be supported by more than one hypothesis and vice versa. An example of such a case is two crossing hands which in reality are two different skin-colored objects but are detected as one blob.

As the features needed for this work are the head(s) and the hand palms, the assumption that the spatial distribution of skin-colored objects can be approximated by an ellipse seems both valid and suitable. Let N be the number of skin-colored objects in the scene at time *t* and $o_i, 1 \le i \le N$, the set of skin-colored pixels of the i^{th} object. The ellipse model is denoted as $h_i = h_i(c_{x_i}, c_{y_i}, \alpha_i, \beta_i, \theta_i)$, where (c_{x_i}, c_{y_i}) is the ellipse centroid, α_i and β_i the length of its major and minor axis, respectively and θ_i its orientation on the image plane. $B = \bigcup_{j=1}^{M} b_j$ denotes the union of skin-colored pixels (detected blobs), $O = \bigcup_{i=1}^{N} o_i$ the union of object pixels and $H = \bigcup_{i=1}^{N} h_i$ the union of the hypotheses[3].

During *tracking phase* hypotheses are generated, removed and tracked. The proposed data association algorithm is analyzed in the following sections. For exemplifying each task, we borrow the next Figure from Argyros's and Lourakis's work [72], with three skin-colored blobs (b_1, b_2 and b_3) and four object hypotheses (h_1, h_2, h_3 and h_4) produced from the previous frame.

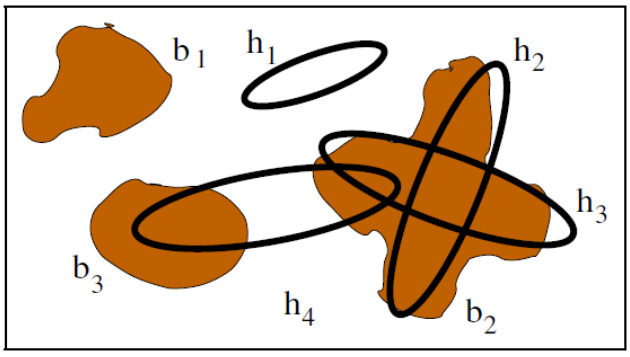


Figure 11: Cases of skin-colored blobs and object hypotheses.

[3] For compatibility purposes, we use the notations and examples used by the authors.

Object hypothesis generation

In order to generate a new object hypothesis, there should be at least one skin-colored object which is not *supported* by any of the existing hypotheses. This means that none of that blob's pixels lie into any of the ellipses of the object hypotheses, namely the intersection of the blob with all existing ellipses is empty. Such a case is blob b_1 in Figure 11.

A safe metric for deciding whether or not a blob lies into an ellipse is the distance of each blob's pixels from the ellipses. The distance $D(p,h)$ of a point $p = p(x,y)$ from an ellipse $h(c_x, c_y, \alpha, \beta, \theta)$ is defined as follows:

$$D(p,h) = \sqrt{\vec{u} \cdot \vec{u}}\,, \tag{11}$$

where

$$\vec{u} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \left(\frac{x - c_x}{\alpha}, \frac{y - c_y}{\beta} \right).$$

A value, of $D(p,h)$, equal or smaller than 1.0 means that a point lies on or inside the ellipse respectively, while $D(p,h) > 1.0$ means that the point is outside the ellipse. Therefore, generation of a new object hypothesis for a blob b is triggered whenever:

$$\forall p \in b, \min_{h \in H}\{D(p,h)\} > 1.0\,. \tag{12}$$

The parameters of the new object hypothesis's ellipse can be derived directly from the statistics of the distribution of blob's points. The centroid of the blob becomes the center of the ellipse while the rest of the parameters can be computed from the covariance matrix of the distribution of blob's points on the image plane. Let $\Sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix}$ be the covariance matrix of blob's points distribution. Then, ellipse's parameters are defined as:

$$\alpha = \sqrt{\lambda_1}\,,\ \beta = \sqrt{\lambda_2}\,,\ \theta = \tan^{-1}\left(\frac{-\sigma_{xy}}{\lambda_1 - \sigma_{yy}}\right), \tag{13}$$

where $\lambda_1 = \frac{\sigma_{xx} + \sigma_{yy} + \Lambda}{2}$, $\lambda_2 = \frac{\sigma_{xx} + \sigma_{yy} - \Lambda}{2}$ and $\Lambda = \sqrt{(\sigma_{xx} - \sigma_{yy})^2 - 4\sigma_{xy}{}^2}$.

False hypothesis removal

An object hypothesis has to be removed when it is not supported by any skin-colored blob. This can occur when the object moves out of the camera's view or when the object is occluded entirely by a non skin-colored object. Hypothesis h_1 in

Figure 11 demonstrates such a case. A hypothesis is allowed to *exist* for a certain amount of time before being removed, in order to cope with situations of poor skin-color detection or temporal occlusion of the skin-colored object due to movement.

Object hypothesis tracking

After finishing with hypotheses generation and removal, all of the remaining blobs should support the existence of past object hypotheses. This data association problem's solution is based on two rules:

- If a skin-colored pixel of a blob lies inside the ellipse of some object hypothesis, then this pixel is considered to belong to this hypothesis.
- If a skin-colored pixel of a blob lies outside all ellipses, then it is associated with its closest object hypothesis, using the distance metric of Eq.11.

These two rules manage to cope with two problems that can arise during the data association process. Apart from the simple case, where a skin-colored blob is easily associated with an ellipse, cases, where a) a blob is located within two or more hypotheses (blob b_2 in Figure 11) or b) a hypothesis covers two or more blobs (hypothesis h_4 in Figure 11), can occur.

In a situation where two or more object hypotheses are "competing" for a single skin-colored blob, the pixels of the blob which lie inside an ellipse are assigned to the corresponding object hypothesis. If there are pixels which lie inside more than one ellipse, then they will be assigned to all corresponding hypotheses. Finally, pixels outside all ellipses are associated with their closest hypothesis, by using the distance metric of Eq.11. It is not safe to make any assumption and remove any of the existing hypotheses because there's no knowledge of whether the blob is actually one skin-colored object or multiple occluded ones.

In the case where the ellipse of a hypothesis covers more than one blobs, similar strategy is followed. If the hypothesis was assigned to one of the blobs at a previous frame, then it is assigned to that blob. If none of the blobs is predicted by that hypothesis, then the hypothesis is assigned to the blob with which shares the largest number of skin-colored pixels.

Prediction

The process of hypothesis tracking involves prediction of the hypothesis next position. This is easily achieved with a linear way and by using the location of hypotheses in the previous frames. Therefore the estimated hypothesis model becomes $\hat{h}_i = h_i(\hat{c}_{x_i}, \hat{c}_{y_i}, \alpha_i, \beta_i, \theta_i)$ where $\hat{C}_i(t) = C_i(t-1) + \Delta C_i(t)$. $C_i(t)$ denotes $(c_{x_i}(t), c_{y_i}(t))$ and $\Delta C_i(t) = C_i(t-1) - C_i(t-2)$. Although the above equations assume

that the direction remains unchanged, experimental results have proved that this prediction mechanism performs efficiently.

3.3 Hand Kinematics Tracking

3D position of the detected skin-color objects can be easily estimated by *triangulation*[4]. Unfortunately, this method can lead to very poor results as it is sensitive to noise and strictly dependent to the calibration process –which by its own bears estimation errors. Therefore, instead of calculating 3D coordinates by *triangulation*, the "inverse" way is chosen. Particles are deployed in space and, by projecting them to the image planes of the cameras, head(s) and palms positions are estimated. Using the resulting 3D positions, an initial estimation for the kinematics parameters of the hands is derived by the use of *Inverse Kinematics Equations.* Finally the *kinematics parameters* are being tracked over time with the use of *Particle Filters* and *Forward Kinematics.*

3.3.1 Perspective projection

The intrinsic and extrinsic parameters derived from the camera calibration (3.1), are essential for the projection of a 3D point P=(X, Y, Z) onto the image plane of a camera (p_x, p_y). The interesting point here is that a point's location in space is expressed in the world frame, whereas its projection onto the plane is expressed according to the image reference frame (pixels). Therefore, a transformation between the world and the image reference frame is needed.

Let f_x and f_y[5] be the focal lengths of the camera and c_x and c_y its principal points (or image center); namely the *intrinsics* of the camera. The matrix which describes the *intrinsics* is therefore denoted as:

$$M_{\text{int}} = \begin{bmatrix} -fx & 0 & cx \\ 0 & -fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

and performs the transform between the camera frame and the image reference frame [57].

As stated in 3.1, the *extrinsics* of a camera represent its relative position according to the principal point of the world. A *rotation matrix R* and a *translation*

[4] The process for estimating the 3D position of a point using a stereo pair of cameras.

[5] $f_x=f/s_x$ and $f_y=f/s_y$, where s_x and s_y are the effective pixel sizes in the horizontal and vertical direction respectively.

vector T describe this transformation from O=(0, 0, 0) to the position of the camera. We define M_{ext}, the matrix which describe the *extrinsics*, as follows:

$$M_{ext} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & -R_1^T T \\ r_{21} & r_{22} & r_{23} & -R_2^T T \\ r_{31} & r_{32} & r_{33} & -R_3^T T \end{bmatrix}, \quad (15)$$

where $r_{11} \dots r_{33}$ are the elements of the rotation matrix $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$, T is the translation vector $T=[T_x, T_y, T_z]$ and R_i, i=1..3, denotes the i^{th} row of the rotation matrix. M_{ext} performs the transformation between the world and the camera reference frame [57]. The product of M_{int} and M_{ext} expresses the transformation between the world and the image plane reference frame.

By expressing the point in 3D space in homogeneous coordinates as $P_w=[X_w, Y_w, Z_w, 1]^T$ and by forming the product M_{int} M_{ext} P_w we end up with *"a linear matrix equation describing perspective projections" [57]*:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = M_{\text{int}} M_{ext} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \quad (16)$$

Finally, the pixel coordinates of the projection on the image plane are derived by normalizing the resulting vector by its 3rd element:

$$x_{im} = x_1 / x_3,$$

$$y_{im} = x_2 / x_3. \quad (17)$$

3.3.2 Particle filters

The *particle filter* is a nonparametric Bayes implementation which can model nonlinear transformations of random variables [74]. The key idea is to represent the *target distribution* by a set of random weighted samples drawn by this distribution and to compute estimates based on these samples and on these weights. The next figure (borrowed from Miodrag Bolic [2]) illustrates the general idea of Particle Filters on a nonlinear distribution. Random samples (measures), which approximate the distribution of the unknowns, are recursively generated. The posterior to be approximated is denoted by blue and the samples with yellow. As new observations arrive, the samples and weights are propagated by exploiting Bayes theorem.

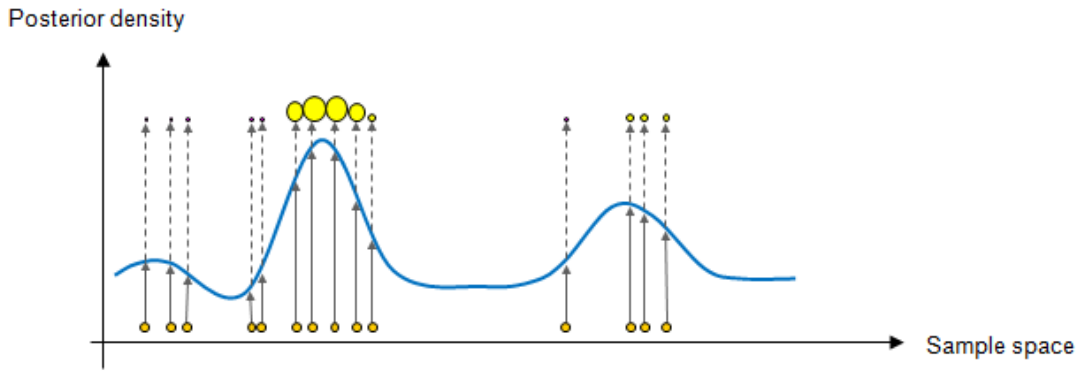


Figure 12: Particle Filter. Particles are drawn over the posterior distribution and propagated according to their weights.

The samples of the posterior distribution are called *particles* and are denoted as [74]:

$$\mathrm{X}_t := x_t^1, x_t^2, ..., x_t^M, \tag{18}$$

where M is the number of particles in the particle set X_t. Each particle represents a hypothesis on the current state at time *t* and its likelihood is (ideally) proportional to its Bayes posterior

$$x_t^m \sim p(x_t \mid y_0, y_1, ..., y_t) \tag{19}$$

where $1 \le m \le M$ and y_i with $0 \le i \le t$ the observations up to time *t*. The above figure also illustrates that the true state is more likely to be approximated by particles sampled from a dense area of the sample space.

The general idea of the particle filtering algorithm can be described as follows:

1. Initialization
   - $\bar{\mathrm{X}}_t = \mathrm{X}_t = \varnothing$
   - $t = 0$.
   - For $m = 1, \ldots, M$ sample $x_0^m \sim p(x_0)$. An initial estimation of the distribution.
   - Set $t = 1$.

2. Importance Sampling
   - For $m = 1, \ldots, M$:
     i. sample $\tilde{x}_t^m \sim p(x_t \mid x_{t-1}^m)$.
     ii. evaluate importance weights $w_t^m = p(y_t \mid \tilde{x}_t^m)$.
     iii. $\bar{\mathrm{X}}_t = \bar{\mathrm{X}}_t + \left\langle x_t^m, w_t^m \right\rangle$.
   - Normalize importance weights $\tilde{w}_t^m = w_t^m / \sum_{j=1}^{M} w_t^j$.

3. Resampling
   - For $m = 1, \ldots, M$:
     i. Draw i from $\bar{\mathrm{X}}_t$ according to $\tilde{w}_t^i$.
     ii. Add x_t^i to X_t.

4. Set $t \to t+1$ and repeat steps 2 and 3 till no other observations arrive.

Figure 13: The particle filter algorithm.

In order to simplify the particle filter algorithm, the next flow diagram [2] is cited:

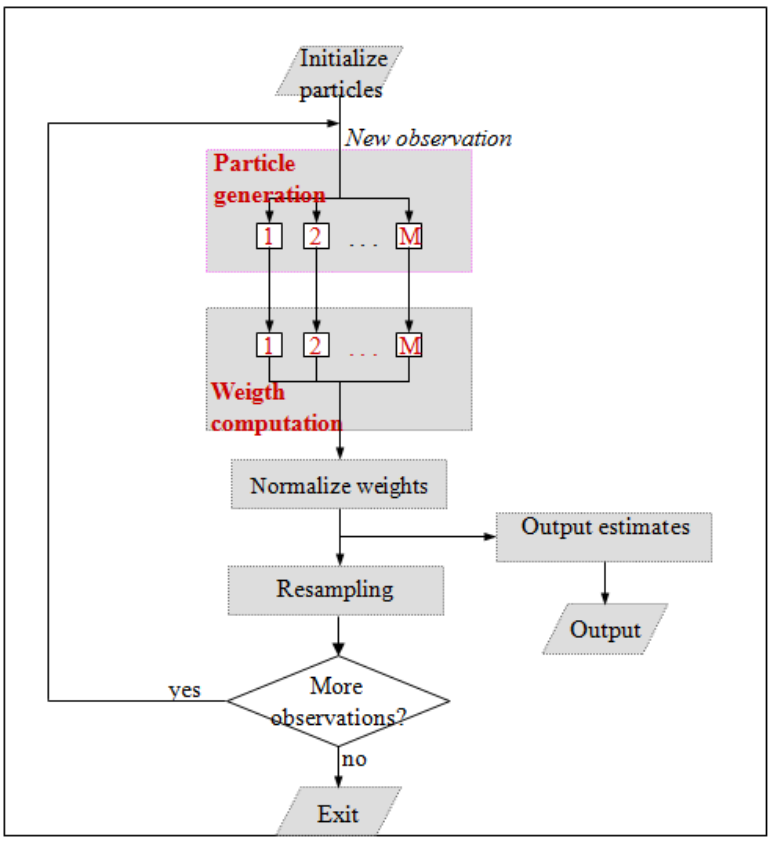


Figure 14: Flow Diagram of the Particle Filter Algorithm.

The efficiency of the particle filter lies basically on the *resampling* (or *importance sampling*) step which transforms the temporary set of M particles into another set of particles of equal size which will finally be propagated and will constitute the seed for the next resampling. M particles are being drawn with replacement from the temporary set $\bar{X}_t$. The importance weight of each particle defines its probability to be drawn. Therefore the metric of the importance factor should be carefully chosen according to the context of the implementation.

3.3.3 Human arm kinematics

"***Kinematics*** *is the science of motion that treats motion without regard to the forces which cause it" [17].*

The human arm consists of limbs (links) and joints which are described by their angles. Since the length of each limb is known and constant, if the angles of the joints and the location of the shoulder are also known, then the position of the elbow and the palm –namely the end effector- can be estimated. The process, during which the location of the end effector is calculated by using the joint angles, is called *forward kinematics.* The inverse process, that is estimating joint angles by knowing the location of the end effector, is called *inverse kinematics.*

3.3.3.1 Forward kinematics

In some simple cases, the calculation of the location of the end effector is straightforward. However, in most cases –such as that of the human hand- more effort is required. The next figure[17] illustrates the parameters which will be used for estimating the forward kinematics equations.

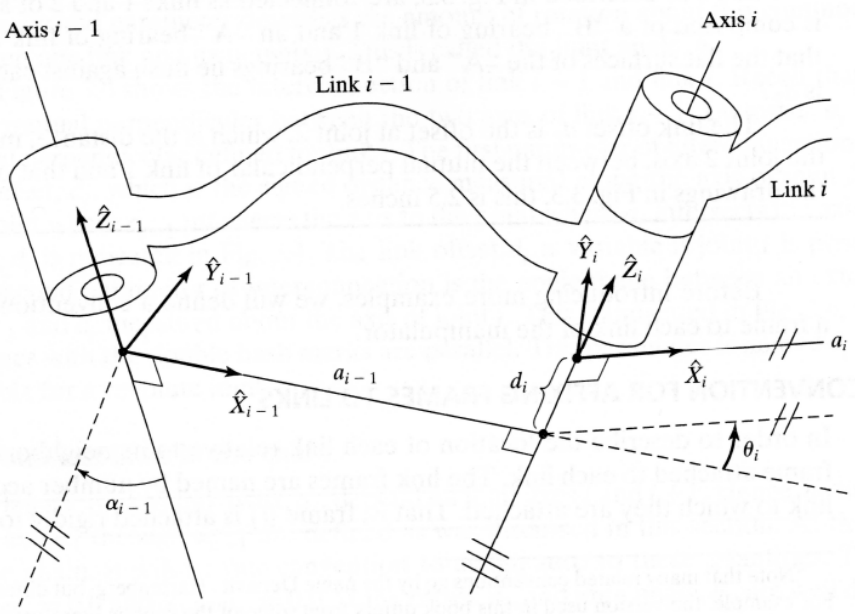


Figure 15: Frame {*i*} is attached rigidly to link *i*.

Figure 15 shows two effectors with different coordinate systems. Therefore, a transformation which will lead from one system to the other, and thus depicting the locations according to the reference system (either of two or even the base) is required. The important parameters derived from the manipulator illustrated above are [17]:

- a_i: the distance of the two *rotation* axes ($\hat{Z}_i$ and $\hat{Z}_{i+1}$) measured along $\hat{X}_i$.
- α_i: the angle between the two axes measured about $\hat{X}_i$.
- d_i: the distance between $\hat{X}_{i-1}$ and $\hat{X}_i$ measured along $\hat{Z}_i$.
- θ_i: the angle between $\hat{X}_{i-1}$ and $\hat{X}_i$ measured about $\hat{Z}_i$.

The parameters described above will be used for the derivation of the kinematics equations.

Consider now, a manipulator with N effectors, with any one of them having a separate frame. As stated before, the required transformation should transform the parameters from one's reference frame to the others, and, finally, to the reference frame of the end effector. A very efficient way of representing these transformations is the one proposed by Denavit and Hartenberg in [9]. The transformation which transforms the i-1th to the ith frame is denoted as ${}^{i-1}_{i}T$:

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -d_i \sin(\alpha_{i-1}) \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & d_i \cos(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (20) $$

Finally the product of all matrices leads to the parameters of the end effector:

$$ {}^{0}_{N}T = {}^{0}_{1}T\,{}^{1}_{2}T \ldots {}^{N-1}_{N}T\,. \quad (21) $$

In order to switch over two coordinate systems, a rotation matrix *R* and a translation vector *T* is needed. ${}^{i-1}_{i}T$ is nothing more than a representation of *R* and *T,* where *R* is the upper left 3x3 matrix and *T* is the leftmost 1x3 vector such as:

$$ {}^{i-1}_{i}T = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}. \quad (22) $$

3.3.3.2 Arm modeling

The model of the arm chosen for the purposes or this work is the one proposed by Tsetserukou et al. in [79]. Although the work refers to a robotic arm, it can efficiently represent a human one as it supports the same degrees of freedom. In Figure 16 the chosen model is illustrated, together with its kinematics parameters expressed to both the global reference frame and the reference frame of each joint.

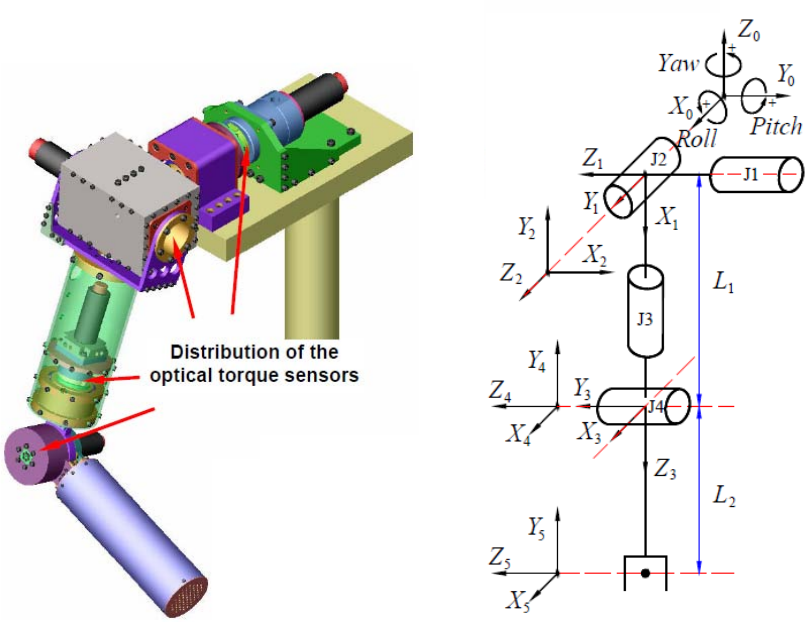


Figure 16: The model of the robotic arm (left) and its parameters (right).

J_1, J_2 and J_3 represent the three degrees of freedom of the human shoulder while J_4 represents the human elbow. L_1 is the distance between the shoulder and the elbow and L_2 is the forearm's length. Finally the angles of joints J_1, J_2, J_3 and J_4 are denoted as ϑ_1, ϑ_2, ϑ_3 and ϑ_4 respectively. The parameters which describe this arm model are given by the authors as follows:

i	α_{i-1}[deg]	a_{i-1}[m]	d_i[m]	ϑ_i[deg]
1	90	0	0	ϑ_1-90
2	-90	0	0	ϑ_2+90
3	90	0	L_1	ϑ_3+90
4	-90	0	0	ϑ_4-90
5	0	L_2	0	0

Figure 17: Arm Model Parameters.

By using the Denavit – Hartenberg representation, as described in Eq.20 and Eq.21, the location in space of both the elbow and the palm can be easily extracted from ${}^{0}_{3}T$ and ${}^{0}_{5}T$ respectively.

3.3.3.3 Inverse kinematics

Although the calculation of *forward kinematics* equations is more or less straightforward, calculation of *inverse kinematics equations* is a more complex process, strictly dependent to the context of the application. In many cases there is not a single solution for the parameters' values, whereas in some others, finding a solution is almost impossible (cases where ${}^{i-1}_{i}T$ cannot be inverted for example). Fortunately, this is not the case for the human arm model *inverse kinematics*, which can be solved analytically as described in [77].

Let

$$ {}^{0}_{5}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_{11} \\ r_{21} & r_{22} & r_{23} & t_{12} \\ r_{31} & r_{32} & r_{33} & t_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{23} $$

be the matrix which leads us from the shoulder to the palm (or wrist). Then, the vector $p = [t_{11}, t_{12}, t_{13}]^T$ depicts the location of the wrist, measured in the base reference frame (in our case the shoulder's coordinate system). Since the length of links is known, ϑ_4 can be calculated from the cosine rule as:

$$\theta_4 = \pi \pm \arccos\left(\frac{{L_1}^2 + {L_2}^2 - \|p\|^2}{2L_1L_2}\right). \quad (24)$$

Clearly, only one solution is physically acceptable due to limitations imposed by the elbow joint.

The interesting part though is the estimation of elbow's position. As Korein [12] notes, even if wrist's and shoulder's position is fixed, the elbow is still free to rotate about the shoulder-wrist axis. Figure 18 illustrates this case, where the elbow is free to move on a circle lying on a plane whose normal is parallel to the shoulder-wrist axis. Shoulder's, elbow's and wrist's positions are denoted by **s**, **w** and **e** respectively.

Elbow's position can therefore be expressed as a function of the circle's center and the rotation angle φ as follows:

$$\mathbf{e} = r[\cos(\varphi)\hat{\mathbf{u}} + \sin(\varphi)\hat{\mathbf{v}}] + \mathbf{c}, \quad (25)$$

where $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ are the unit vectors that form the elbow's local coordinate system, **c** is the center of the circle and φ is the rotation angle. $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ can be expressed as a function of the normal vector of the circle's plane $\hat{\mathbf{n}}$ and the **z** axis of the system:

$$\hat{\mathbf{u}} = \frac{-\mathbf{z} + (\mathbf{z}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}}}{\left\|-\mathbf{z} + (\mathbf{z}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}}\right\|}, \quad (26)$$

$$\hat{\mathbf{v}} = \hat{\mathbf{n}} \times \hat{\mathbf{u}}, \quad (27)$$

where

$$\hat{\mathbf{n}} = \frac{\mathbf{w} - \mathbf{s}}{\|\mathbf{w} - \mathbf{s}\|}. \quad (28)$$

Finally, the circle's center **c** and its radius **r** can be calculated by simple trigonometric functions:

$$\mathbf{c} = \mathbf{s} + \cos(\alpha)L_1\hat{\mathbf{n}},$$

$$\mathbf{r} = L_1\sin(\alpha) \quad (29)$$

with

$$\cos(\alpha) = \frac{{L_2}^2 - {L_1}^2 - \|\mathbf{w} - \mathbf{s}\|^2}{-2L_1\|\mathbf{w} - \mathbf{s}\|},\ \sin(\alpha) = \frac{L_2\sin(\psi)}{\|\mathbf{w} - \mathbf{s}\|},$$

$$\psi = \pi - \theta_4. \quad (30)$$

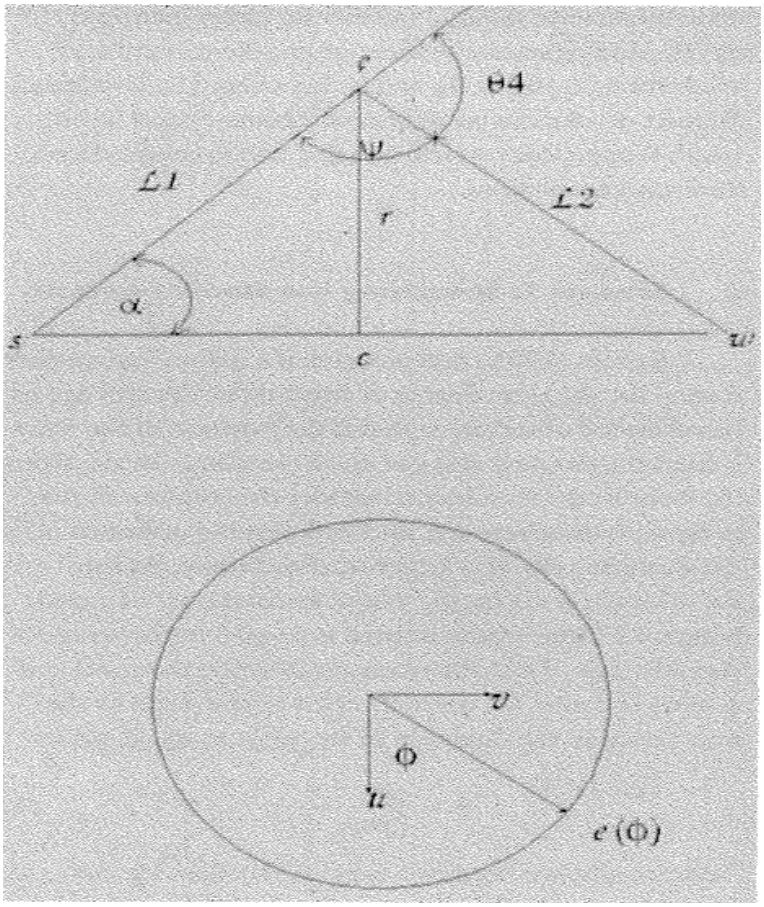


Figure 18: Elbow position based on swivel angle.

By assuming that the value of φ is known, and therefore the elbow's position, the inverse kinematics problem can be solved analytically for the rest of the joint angles. Note that the elbow's position is a function of the first three joints:

$$ {}^{0}_{3}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T = \mathbf{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \\ 1 \end{bmatrix}. \tag{31} $$

Therefore, by solving Eq.31 we get the values of the rest joint angles ϑ_1, ϑ_2 and ϑ_3.

3.4 Neural Network - Multi-Layer Perceptron

"A network of simple processing elements (neurons), which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters." [65]

Neural Networks are the result of an effort to describe how human mind works and, although the idea behind them was first started in late 1800s, they are currently used for performing several complex tasks as clustering, function estimation, robot navigation etc. The term "Neural Network" stands for two distinct cases: The Biological Neural Network which consists of real biological neurons in the human brain and the Artificial Neural Network (ANN), or simply Neural Network (NN), which is a mathematical or computational model based on biological neural networks [7, 65].

A neural network consists of sets of interconnected neurons. Each neuron can perform a simple 2-class classification, assuming that the classes are linearly separable. By adding neurons, additional classifications are feasible. The number of neurons to be added depends on the task to be performed, together with the input and output dataset provided to the network.

A typical neural network is consisted of three layers of neurons, as depicted in Figure 19; input, output and hidden layer. The neurons of each layer are connected to all neurons of the *next* layer. However, there is no interconnection between neurons of the same layer. The hidden layer is task specific and is constructed during the training phase, depending on the given input dataset and the corresponding output, and is responsible for the modeling of the function that translates the input to the output.

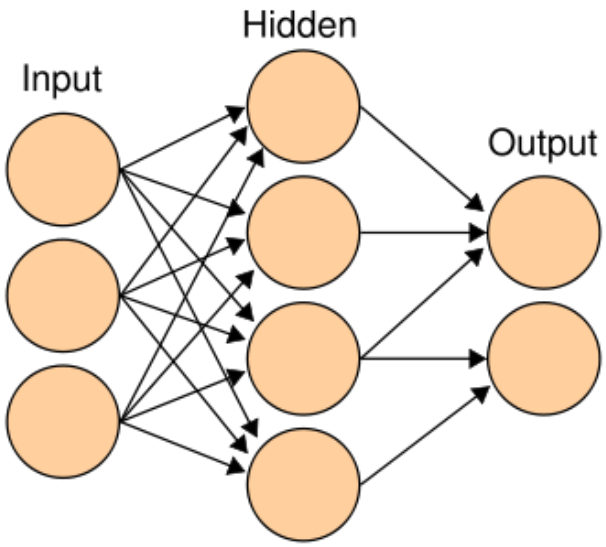


Figure 19: Typical Neural Network Layout

The number of neurons of each layer depends on the task. Therefore there are as many as input and output neurons as the input and output data respectively. The number of neurons in the hidden layer affects directly the effectiveness of the

network. Too few and the network will produce poor results (unable to learn the problem), too many and the network will be over-determined.

For each task to be performed, the neural network has to be trained first. Training phase is crucial for the efficiency of the network and, thus, needs to be designed carefully. During training, the structure of the network changes so that it adapts to the needs of the information provided and the desired output. In general, training can be thought of as the estimation of a function f that solves optimally a specific task by using a set of observations (inputs) together with the minimization of the error of that function.

Three are the dominant training techniques:

- **Supervised Training** where both input and the corresponding desired output are provided to the neural network and the network estimates the function $f : X \to Y$, where X and Y stand for the set of inputs and outputs, respectively.
- **Unsupervised Training** where there is no correspondence between the input and the output, and the network tries to relate them –and thus estimate the proportional function f - by detecting patterns in the data set.
- **Reinforcement Training** where there is neither given input nor output dataset, but are generated from the interaction between the network and the environment.

In this work, Multi-Layer Perceptron Neural Networks are being used for the task of gesture recognition, having as input the kinematic parameters which describe the arm. This section intends to present the Multi-Layer Perceptron Networks and describe the rules that govern its training phase.

3.4.1 Multi-layer perceptron

Multi-Layer Perceptron (MLP) is the most widely used class of neural network, mostly due to its success on many information processing tasks, such as pattern classification, function estimation and time series prediction. Moreover, MLPs have been used in many practical fields, such as speech recognition, medical diagnosis, autonomous vehicle control and financial prediction [49].

MLP's architecture follows the classical neural network layout, with input, output and hidden layers[6]. Nodes (neurons) of different layers are connected with real valued weights, whereas there is no connection between nodes of the same layer.

[6] Note that a hidden layer may consist of several layers.

Figure 20 illustrates a minimal 2-2-1 MLP –that is two nodes for the input and hidden layer and one output node.

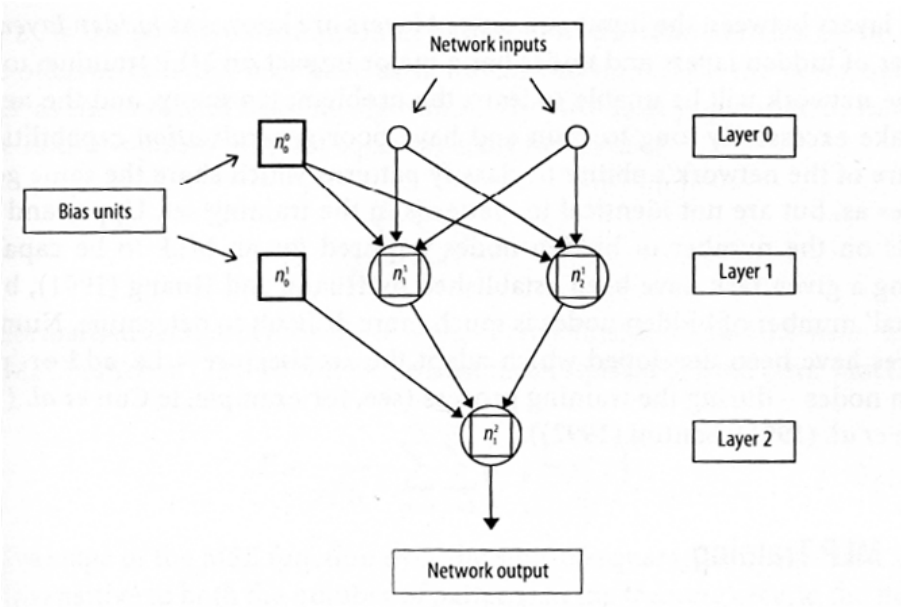


Figure 20: Minimal 2-2-1 MLP architecture.

The notation of Shepherd [49] is used here as well. Therefore:

- The network consists of L layers, with $l = 0$ denoting the input layer and $l = L$ denoting the output layer.
- Each node is denoted as n_i^l, with $(1 \le i \le N^l)$ where N^l is the number of nodes in layer l.
- The *strength* of a node's output –namely the activation- depends on the strength of the input to that node with respect to a threshold value. In order for the thresholds to be treated uniformly, an extra node with fixed output of 1.0 –called the *bias unit* and denoted as n_0^l for $l \neq L$ - is added to all but the output layer.
- Each node n_i^l has a set of *source nodes* S_i^l and a set of *target nodes* T_i^l. Let n_j^m and n_i^l two connected nodes. If $m < l$, node n_j^m is a source node of n_i^l (i.e. $n_j^m \in S_i^l$) and n_i^l is a target node of n_j^m (i.e. $n_i^l \in T_j^m$). It is obvious that all input nodes and bias units have no source nodes and all output nodes have no target nodes.
- Network weights are denoted with respect to the nodes they connect. Therefore w_{ij}^{lm} connects n_j^m and n_i^l with $m < l$.

Despite the fact that MLPs support networks of arbitrary connectivity, it is more convenient to consider architectures of restricted form, namely MLPs with fully connected adjacent layers but with no connections between nodes of non-adjacent

layers. This architecture is denoted as “standard” MLP architecture. The MLP of Figure 20 is an example of a “standard” architecture.

The number of input and output nodes depends on the pattern and target size, respectively. An MLP is supervised trained, using a fixed training set of P training pairs, with each training pair consisting of two real-valued vectors – a pattern p_q with $1 \le q \le P$ and the corresponding desired output t_q. Individual pattern and output elements are denoted as $p_{i,q} (1 \le i \le N^0)$ and $t_{j,q} (1 \le j \le N^L)$, respectively. The output $y_{i,q}^l$ of node n_i^l is a function of its activation $a_{i,q}^l$:

$$a_{i,q}^l = \sum_{n_j^m \in S_i^l} w_{ij}^{lm} y_{j,q}^m , \quad l > 0, \; m < l \tag{32}$$

$$y_{i,q}^l = f(a_{i,q}^l), \quad l > 0. \tag{33}$$

It is obvious that the output of an input node n_i^0 is $p_{i,q}$ for pattern q. The *squashing* or *activation function* $f(x)$ is both monotonic and continuously differentiable. The most frequently used function is the *sigmoid* function:

$$f(x) = \frac{1}{1+e^{-x}}. \tag{34}$$

3.4.2 MLP training

MLP training is a supervised iterative process. At each iteration (or *epoch*) the network output for each pattern in the training set is calculated and the weights are being progressively adjusted according to the difference between the actual and the desired output, so that the output of the network is *acceptably* close to the desired output. How much “acceptably close” is the actual to the desired output is measured by an *error* (or *energy*) function for each epoch. Therefore, the main target of the training phase of an MLP is to minimize this error.

Before training starts, the weights are initialized to small random values, in order to prevent *saturation* (where nodes are highly active or inactive for all patterns, and therefore insensitive to the training process) and *symmetry*. The choice of the initialization range of weights affects directly the performance of the MLP training. Weight initialization should ensure that the initial standard deviations of the network activations are in the same range for each node and lie within the normal operation region of the squashing function. A widely used weight initialization which satisfies all previously mentioned prerequisites is [18]:

$$r(w_{ij}^{lm}) = \frac{2.4}{N^m + 1}, \tag{35}$$

which sets each weight randomly with a distribution in the range of $[-r, +r]$.

The error function E for each epoch is the sum of all partial errors E_p produced from each pattern p :

$$E = \sum_{p=1}^{P} E_p \,. \tag{36}$$

Two are the most popular functions for the error of each pattern: the *sum-of-squares error function* (SSE)

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{i=1}^{N^L} (t_{i,p} - y_{i,p}^L)^2 \tag{37}$$

and *mean-squared-error* (MSE) which is a normalized version of *sum-of-squares*

$$E = \frac{1}{2PN^L} \sum_{p=1}^{P} \sum_{i=1}^{N^L} (t_{i,p} - y_{i,p}^L)^2 \,. \tag{38}$$

The advantage of the MSE over the SSE is that it is insensitive to both the number of patterns in the training set and the number of output nodes in the network.

However, both MSE and SSE proved to be sensitive in cases where the error function presents local-minima and multi-dimensional “plateaus”. In such cases, the network result is suboptimal, since the minimum number of misclassifications has been achieved. Therefore, other functions which improve network’s ability to escape from such regions are adopted. An example of such a function is the *cross-entropy error function* [16]:

$$E = -\sum_{p=1}^{P} \sum_{i=1}^{N^L} \ln[(y_{i,p}^L)^{t_{i,p}} (1 - y_{i,p}^L)^{1-t_{i,p}}] \,. \tag{39}$$

The above error function gives the ability to the network to progress in (and escape from) flat regions in weight space, since error gradients for poorly classified patterns are higher than MSE or SSE.

The success of MLP training depends on how “acceptably small” E becomes. Once more, “how small” is task specific and needs not to be as small as possible –i.e. finding the minimum achievable E -, since, in such a case, the network will become *over-trained*, and therefore, MLP ability to generalize will decrease. The characteristics of the surface of the error function, indicates the strategy which most

probably lead to a reliable training. Most MLP error surfaces share a number of broad characteristics, such as [49]:

- a high degree of smoothness,
- extensive *"plateaus"*,
- *"narrow valleys"* or *"ravines"*,
- many weak minima, some of which local ones,
- a degree of symmetry around the origin of the coordinate system used to plot the error surface.

An example of an error surface with some of these characteristics is illustrated in the next figure.

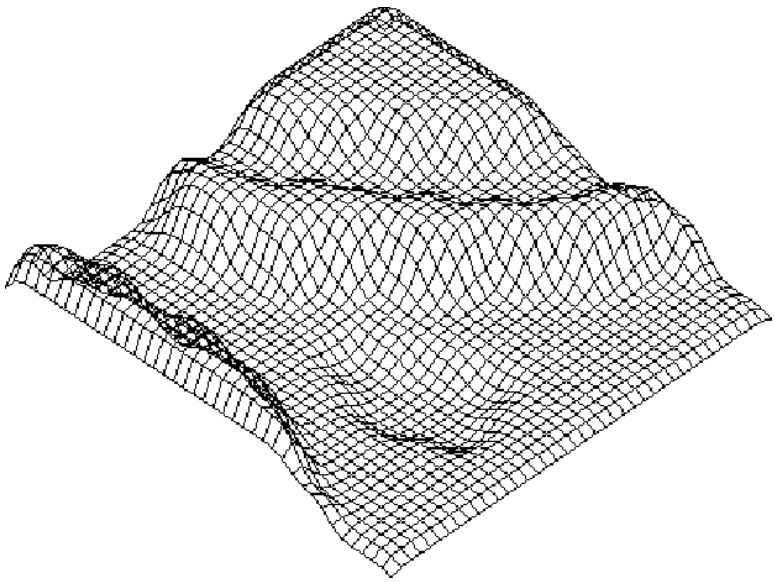


Figure 21: Example of MLP error surface.

The role of the hidden nodes is crucial for the MLP training dynamics. A hidden node which duplicates the function of another hidden node and, therefore, is not needed for the training phase, is called *redundant. Annema et al. [29]* in their analysis of the MLP training dynamics, depict the importance of redundancy for the training phase. Let w_i^l be the weight vector consisting of all weights w_{ij}^{lm}, connecting node n_i^l to its source nodes. The weight vector of a neuron corresponds to a hyperplane dividing the input space into two classes. During training, the weight vectors tend to converge towards specific attractor in the weight space [20].

3.4.3 Backpropagation

Error-backpropagation, or simply backpropagation, is a reliable method for updating the MLP weights during the training phase. The backpropagation algorithm

for an MLP, implements the *steepest* (or *gradient*) *descent* method. The traditional backpropagation algorithm is known as *batch* or *offline backpropagation* and works as follows [14]: at each epoch k, the gradient g_k is calculated and the weights of the network are updated according to

$$w_{k+1} = w_k + \Delta w_k$$
$$\Delta w_k = -\eta g_k, \qquad \eta > 0 \tag{40}$$

where η is a constant heuristically chosen scalar, usually in range (0,1). The above equation is known as the *generalized delta rule* and η as the *training rate.* It can be also thought of as the backpropagation algorithm sets the *search direction* to $-g_k$ and moves towards this direction by a *step length* of η for every iteration k. This rule ensures the reduction of total network error E as long as g_k is greater than zero, but, however, it does not guarantee that the network will escape from a local minimum.

The calculation of the gradient g_k is being done in two phases: a forward and a backward pass. The forward pass generates the output $y_{i,p}^l$ (see Eq.33) for the pattern layer $l = 1$ to $l = L$. The backward pass calculates the partial error E_p for pattern p and the corresponding partial gradient g_p with elements $\frac{\partial \mathrm{E}_p}{\partial w_{ij}^{lm}}$ such as:

$$\frac{\partial \mathrm{E}_p}{\partial w_{ij}^{lm}} = -\delta_{i,p}^l y_{j,p}^{l-1}, \tag{41}$$

where the error term δ is given by

$$\delta_{i,p}^L = (t_{i,p} - y_{i,p}^L) f'(a_{i,p}^L)$$
$$\delta_{j,p}^m = f'(a_{j,p}^m) \sum_{n_i^l \in T_j^m} \delta_{i,p}^l w_{ij}^{lm}, \quad m < L. \tag{42}$$

If the activation function is the sigmoid, f' is simply

$$f'(a) = y(1-y). \tag{43}$$

Although efficient for many training tasks, batch backpropagation has several important drawbacks:

- Whenever flat regions or narrow valleys appear, it often takes long to converge to a satisfactory error level.
- Training effectiveness depends directly on the training rate. If training rate is too small, the network error will not be reduced sufficiently, whereas if it is too large, network error may stuck to a high level.

- It tends to get trapped in local minima.

Due to these vulnerabilities, many variations of the original backpropagation algorithm have been proposed, in order to cope with problems such as local minima and flat regions. We present two of them, not necessarily the most efficient ones, but very helpful for this work.

3.4.3.1 Backpropagation with momentum

Rumelhart et al. [14] proposed a simple variation of the batch backpropagation algorithm, in order to speed up the training phase, by adding a *momentum term* in the generalized delta rule of Eq.40:

$$\Delta w_k = -\eta g_k + \alpha \Delta w_{k-1}. \tag{44}$$

The parameter α is user defined and is set in the range $0 \le \alpha \le 1$.Although α is task specific, it is typically set to 0.9. This approach actually made the algorithm to cope more efficiently with problematic regions –i.e. plateaus and narrow valleys- and therefore speed up the whole training process. However, there might have to be a reduction in η, in order to maintain network stability –i.e. constraint excessive weight changes.

3.4.3.2 On-line backpropagation

The most important variation of the original batch backpropagation algorithm is known as on-line backpropagation. The main difference between batch and on-line backpropagation, is that instead of updating the weights once per P backward passes (once per epoch), they are updated at each backward pass (i.e. P times per epoch). The weight update rule for the on-line backpropagation becomes as follows:

$$\begin{aligned} w_{k,p+1} &= w_{k,p} + \Delta w_{k,p} \\ \Delta w_{k,p} &= -\eta g_{k,p}, \eta > 0 \end{aligned}. \tag{45}$$

An important issue aroused here is the choice of a suitable η for each iteration k. Darken et al. [5] proposed the *search then converge* (STC) schedule, given by:

$$\eta_k = \eta_0 \frac{1 + (c/\eta_0)(k/\tau)}{1 + (c/\eta_0)(k/\tau) + \tau(k^2 + \tau^2)}, \tag{46}$$

where η_k is the training rate at iteration k, parameter c is set greater than a threshold $1/2\lambda_{\min}$, with $\lambda_{\min}$ being the smallest eigenvalue of the Hessian matrix of E. Parameter τ is related to the number of training epochs. The above guarantees an asymptotic rate of convergence for the on-line backpropagation. However,

convergence is highly dependent on parameters η_0 and τ which are estimated either heuristically or by prior knowledge about the task.

Theoretically, on-line backpropagation has certain disadvantages compared with batch backpropagation. Although requiring more computational effort per epoch, on-line backpropagation does not ensure the reduction of total network error and does not provide highly accurate solutions. On the other hand, on-line backpropagation characteristics make it preferable in many cases. Due to its stochastic nature, on-line backpropagation prevents MLP to get trapped in local minima. Additionally, in cases where training set contains redundant information –as it usually does- since weights are updated more often, on-line propagation converges faster. Finally, on-line training is essential in cases where not all training patterns are known at the start of the training [49].

Chapter 4

Hand Parameters Extraction and Tracking

The extraction and tracking over time of the kinematics parameters, which express the position and pose of the human arm, constitutes the core of this work. The tracked parameters, namely the angles of arm's joints, will be, eventually, the input of the neural network during the last phase of gesture recognition.

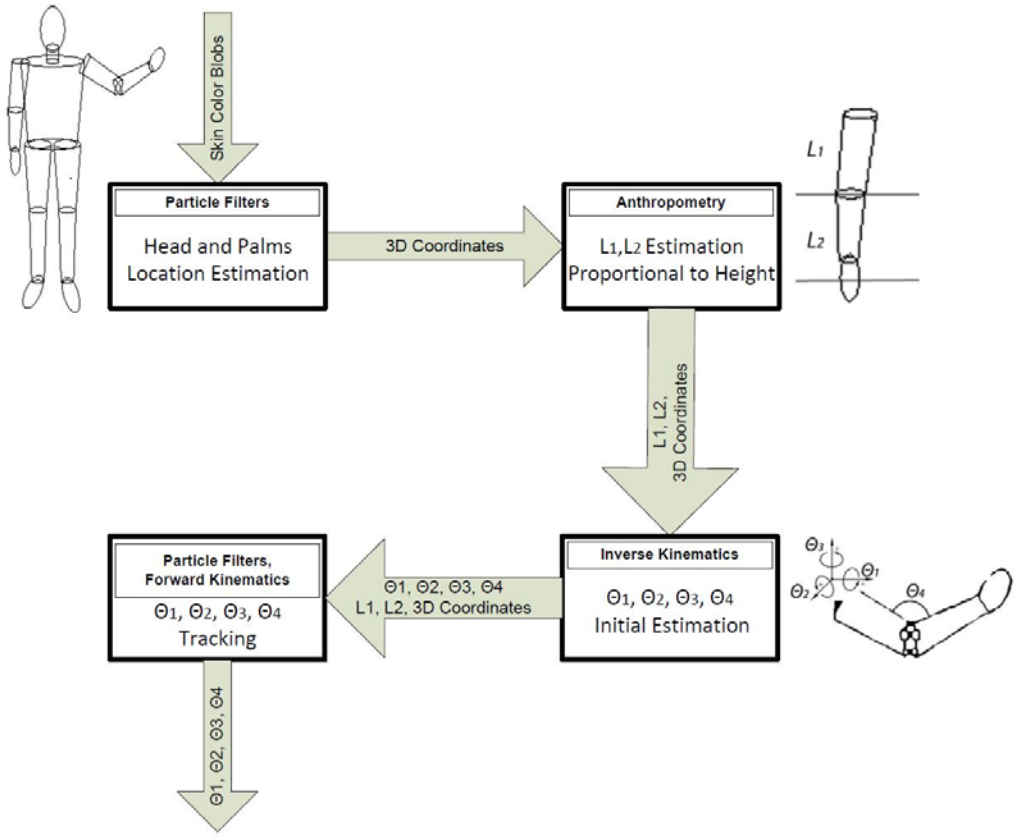


Figure 22: Kinematic Parameters derived from low-level features.

The choice of the model has been based on three factors: *simplicity*, *flexibility* and *efficiency*. The key idea is to extract high-level information by using primitive features of the image, while, at the same time, keeping the processing cost at low levels. In other words, pixel values, provided by the input videos, are transformed into kinematic parameters which fully describe the pose of body and arms.

The skin color tracker –presented in 3.2.2- detects skin-colored objects in the two images. By defining one blob as the head, the height of the user can be easily calculated, according to the extrinsics of the cameras, which finally is used to estimate the lengths of the upper limbs, based on anthropometric measures and proportions.

Having estimated the position and the size of the upper body of the performer, inverse kinematics provide an initial estimation for the kinematic parameters (angles of shoulder and elbow as described in 3.3.3) of the arm. Finally, particle filters track the extracted parameters, and feed them to the gesture recognition module.

Figure 22 illustrates the above described procedure. The input of the algorithm is solely the values of the images pixels. No previous knowledge about the scene or the user (e.g. height) is required. The only information about the setup provided is the intrinsics and extrinsics –extracted from cameras calibration- together with the height where the reference camera (arbitrary choice) has been placed.

4.1 Head and Hands Positions

Triangulation can provide the location of skin-colored objects (detected from skin-color tracker as described in 3.2.2). However, in many cases, the estimation of 3D position with the use of triangulation is subject to considerable error, since it is sensitive to image noise and strictly dependent on the accuracy of the camera calibration process.

Instead, particle filters are being used for estimating the position in space of the hypotheses by back projection of themselves. The first step is the determination of the hypothesis that stands for the head. This issue can be solved by assuming that a user will enter the scene in its normal pose; that is with the body straight (not bent) and with the hands hanging free at the height of thighs. This ensures that head will be the highest skin-colored object and, surely, in the upper part of the image.

Let p=[X, Y, Z] be the position of a particle in space. The above assumption limits the Y-dimension. Z-dimension, namely the depth, is also by definition limited from the room's size, which can be directly defined by the user or even calculated from a laser sensor. In either case, particle filter converges to a satisfactory estimation of

the depth. Note the reference to"estimation" and not to an accurate calculation. As will be described later in this section, accuracy of calculations is not a necessary goal, since the algorithm can sufficiently work with relative positions and proportions.

As soon as at least two hypotheses have been tracked over a time period t_h, particles are deployed uniformly in space as shown in Figure 23(a). t_h is imposed in order to ensure that the hypotheses to be tracked are not accidental. Each particle is projected on each camera as explained in 3.3.1 and the distance of its projection from the corresponding centroid of the hypothesis will determine its weight as follows:

$$w^m = \max\left(\frac{T_W - d(x^m, c)}{T_W}, 0\right), \tag{47}$$

with $1 \le m \le M$ where M is the predefined number of the particles. $d(x^m, c)$ denotes the Euclidean distance of the particle from the hypothesis's centroid ***c***. Finally T_W denotes the window, in pixels, –or equivalently the maximum distance- around the centroid where a particle can lie in order to take part in the resampling step. Although, this way may seem brutal, it works fine since the dimensionality and the nature of the space does not allow ambiguities.

The weights are then normalized so that particles will be resampled accordingly in the next step. Two weight thresholds are imposed: $W_{\min}$ and $W_{\max}$ with $W_{\min} < W_{\max}$. Particles with $w^m < W_{\min}$ are excluded from the resampling step and new ones are generated from the remaining particles (note that sampling is done *with replacement*). As soon as there are particles with $w^m \ge W_{\max}$ the filter is considered to have converged and the centroid of these particles becomes the centroid of the head. Usually three frames are enough for the algorithm to converge. Finally, in order to lower the processing cost, whenever convergence happens, the number of remaining particles is decreased below the half of the original.

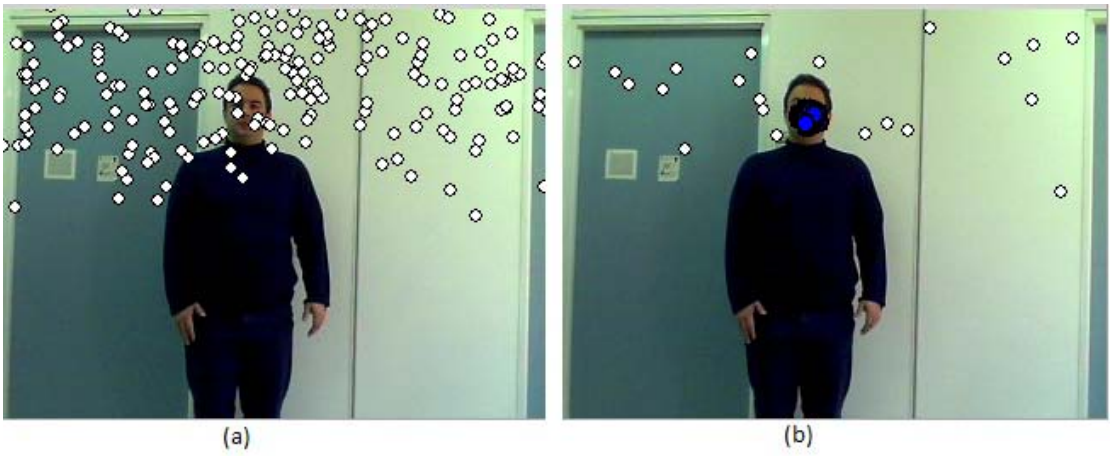


Figure 23: (a) Initial deployment of particles. (b) Particles have converged to the head (blue particles).

Once the position of the head is estimated, separate particles for each hand are deployed. The process is the same as the one described before and is repeated until both hands are detected. The discrimination between left and right hand is of no use and, thus, isn't performed. Finally, no motion model is used, since the framerate of the video ensures that possible movements will have small, or even negligible, effect.

4.2 Shoulder Position Estimation

The next step before the extraction of kinematics parameters is the estimation of the positions of the shoulders. As already stated, there is no need for accurate calculation of the depth, which is the factor that defines the relevant size of the user on the image. Therefore, arms and body size should be estimated according to the depth and height detected so far, since absolute values cannot be calculated.

Art seemed to be very helpful in this issue. Leonardo Da Vinci was the first who studied the ideal proportions of the human body, as depicted in his drawing "*Vitruvian Man*". What is equally interesting is that the length of each part of the human body can be approximated proportionally to the height of that human. More specifically, in "The Physics Hypertext™" [6], an essay on human body proportions analyzes the length of the upper limbs with respect to the height. The results of this essay proved to be very helpful for our work since the height can be easily estimated from the previous step of head detection. The table below presents the results of this essay. All arm parts can be estimated proportionally to the wingspan length, with ratios 0.15 and 0.11 for the forearm and the hand, respectively. Considering that the length of the wingspan is equal to the height of the human, and assuming

that the upper arm is approximately equal to the forearm, both shoulders position and arm length can be estimated.

Wingspan (cm)	Height (cm)	Wingspan-Height proportion	Forearm+Hand (cm)	Forearm (cm)	ForearmHand-Forearm proportion
172,3	172	1,002	44,5	26	1,712
159,4	154,4	1,032	41,4	24,4	1,697
182,1	178,2	1,022	46,1	26,4	1,746
169,6	168,6	1,006	43,2	23,6	1,831
164,5	167,6	0,982	43,1	26	1,658
165,5	169,4	0,977	41,3	23	1,796
172,4	167	1,032	46,8	27,2	1,721
163,2	164,2	0,994	43,6	26	1,677
164,5	162	1,015	42,5	25,5	1,667
162,5	160,5	1,012	43,8	26	1,685
180,5	173,5	1,040	47	27,5	1,709
170,9	161,5	1,058	45,7	26	1,758
150,5	150,5	1,000	39,1	23,8	1,643
165,6	161,4	1,026	42,7	26	1,642
175,1	165,9	1,055	46,6	27,6	1,688
175	166	1,054	46,3	27,2	1,702
192,5	187	1,029	46,7	27	1,730
188,5	179,9	1,048	47,8	28	1,707
184	176,5	1,042	47,5	27,5	1,727
169	168,6	1,002	43,6	25,2	1,730
178,4	174,1	1,025	47,7	28	1,704
167	166,5	1,003	45,2	26	1,738
158	157	1,006	42,4	24,6	1,724
190,7	179	1,065	48,7	27,6	1,764
			48,8	28	1,743

Figure 24: Arm length proportionally to height.

One more issue still persists. There is no knowledge of whether the user looks straight to the camera or there is some angle to its pose; namely the rotation of the body about the Y-axis (vertical). This leads to the existence of several rotation angles which can correspond to the user's pose seen by the camera. For example assume that the original rotation is around 90°, that is when the camera sees the profile of the user. Based on the available information, such a case is exactly the same with a rotation angle of around 180°. It is at least risky, if not pointless, to make any assumption here and exclude some of the cases. Therefore, for now, every rotation is considered –and will be treated- as valid until the dominant ones are decided. The number of rotations is predefined. Testing showed that an amount of 15-20 rotations is adequate.

4.3 Kinematics Tracking

Each of the candidate rotations represents a pair of shoulders, which will eventually be the "starting point" for the arms. This means that the arms are somehow coupled together, since they should *"belong"* to the same rotation. With respect to that, one would treat rotations and arms as one. Unfortunately, this would raise the complexity of the algorithm, since the dimensionality of the parameters space increases dramatically. Therefore, instead of treating the rotations and arms as one, we tried to decouple them, without neglecting the natural limitations imposed.

4.3.1 Hand tracking

Two sets of particles are deployed for every rotation; one for each arm. Each arm is tracked separately to the other, but with respect to the corresponding rotation. Thinking of this abstractly (with no correspondence to particle filters), it would be acceptable to say that every rotation particle "proposes" two arms. The proposed arms determine the weight of each rotation particle. Finally, the dominant rotation is determined by the resulting weights.

Each arm particle keeps track of the kinematic parameters of the correspondent arm; the four angles as discussed in 3.3.3.2. By solving the forward equations, its location in space can be estimated. The projection of particle on the camera will determine its weight. Still, no motion model is necessary.

4.3.1.1 Initialization

Although the dimension of the parameter space is small enough, the number of particles required to cover all cases is prohibitive. Therefore, in order to use only a small number of particles, a good initial estimation of the kinematic parameters is crucial. This can be done by solving the inverse kinematics equations presented in 3.3.3.3, since the positions of both the base (shoulder) and the end effector, together with the length of the links, are known.

The kinematic parameters are estimated for both hands, for every rotation. Moreover, since the rotation angle of the elbow around the shoulder-palm axis is still unknown, particles for every possible pose of the elbow (that is every possible elbow for every hand for every rotation) will be created. Certainly, there will be cases where the kinematic equations give no solution. This is desirable since not every rotation fits to the real one.

4.3.1.2 Hand particles weighting function

The weighting function is similar to the one used for the position estimation in 4.1. Once the kinematic parameters have been estimated –either from inverse kinematics or from previous iterations- the locations of the elbows and the palms can be estimated, by solving the forward kinematics equations. The distance between the projection of these locations and the centroid of the correspondent object hypothesis will determine the weighting factor of each particle.

However, there is still a factor which can lead to poor results: the angle of the elbow rotation φ. It is obvious that not every value of φ is acceptable since it can sometimes lead to unnatural results. In order to cope with that, not only the shoulders and palms are projected, but also the elbow. The projection of the elbow, in order to be "acceptable", should lie inside the foreground area.

The ideal foreground image, on which the elbow should be projected, would be a representation of the actor's skeleton. In that case, every estimated elbow would be limited only to its real position. Unfortunately, this is not yet feasible because of two factors: 1) performing skeletonization on the input image would add an excessive computational cost and 2) the foreground image contains "unnecessary" parts, such as clothing or even the skin.

However, the foreground mask could be very helpful in order to approximate the actor's skeleton. Since accuracy in this task is not one of the main goals, this problem can be solved by simply contracting the foreground mask by some pixels. Therefore, the area of each arm is decreased, approximating the actor's skeleton shape, as depicted in Figure 25(b). The resulting image consists of values equal to 1 for pixels lying in the contracted foreground area, and 0 otherwise.

Figure 25: (a) Original foreground mask. (b) Contracted foreground mask.

The elbow is, therefore, projected on the contracted foreground mask image. The corresponding value will determine the influence of the elbow on the particle's weight. Finally the weighting function for the arm particles is given by:

$$w^m = h\max\left(\frac{T_W - d(x^m, c)}{T_W}, 0\right) + eE_F, \tag{48}$$

where h and e denote the influence of each factor on the weighting function and add up to 1. E_F denotes the value of the projection of the elbow on the enhanced foreground mask image.

Two thresholds are introduced: $W'_{\min}$ and $W'_{\max}$. As in 4.1, particles with $w^m < W'_{\min}$ will be replaced during the resampling step, whereas, particles with $w^m > W'_{\max}$ will determine the centroid of the candidate solution. Additionally, since resampling will take place from only particles with $w^m > W'_{\min}$, the weights of these particles are normalized for the number of particles with $w^m > W'_{\min}$ such as:

$$\tilde{w}^m = w^m / \sum_{j=1}^{M} w^j \tag{49}$$

with $w^m > W'_{\min}$ and $w^j > W'_{\min}$. The normalized weight of each particle represents the probability of that particle to be drawn during resampling phase.

Resampling of arms cannot be done at this moment because of the coupling between arms and rotations. This coupling enforces the resampling of hand particles to depend directly on the rotation particles weights which is discussed in the following section. However, due to this coupling, rotation weights depend on the hand weights as well.

4.3.2 Rotation tracking

The resulting rotations from section 4.2 constitute the initial estimation for the rotation particles. Note that the only parameter tracked by rotation particles is solely the rotation angle. Therefore, as discussed in the previous section, the weight of each rotation particle depends on the weights of the hand particles assigned to each rotation. The weighting function for the rotation particles can be then given by:

$$w^r = 0.5\left(\sum_{m=1}^{M} w_L^m / M\right) + 0.5\left(\sum_{n=1}^{N} w_R^n / N\right),$$

$$w_L^m \geq W'_{\min}, w_R^n \geq W'_{\min} \tag{50}$$

with $1 \le r \le R$ and R the number of rotation particles and $1 \le m \le M$, $1 \le n \le N$ with M and N the number of particles with $w \ge W'_{\min}$ for the left and right hand, respectively. Note that the above equation uses the mean value of the **non-normalized** weights of the particles. Finally, the weight of the rotation particles is normalized. Once more, two thresholds are being used: $W^R_{\min}$ and $W^R_{\max}$ which determine the influence of the particles during resampling as described in 4.1.

What is interesting here is that the weight of each rotation particle with $w^r > W^R_{\min}$ not only represents the probability of this sample to be drawn, but also determines the resampling for the hands. To make this clear consider the following example. Let L1 and L2 be two particles of the left hand with normalized weights of 0.2 and 0.4, respectively. Additionally there are two rotation particles, R1 and R2, with normalized weights equal to 0.4 and 0.1, respectively and $M = 1000$. This means that 400 hand particles have to be drawn from the set of particles assigned to R1 and 100 from those assigned to R2. Assume finally that L1 is assigned to R1, L2 is assigned to R2 and that there are other hand particles as well assigned to both rotations. Then, 80 left hand particles will be drawn from L1 and 40 from L2.

This is reasonable since, as stated earlier, each rotation particle "represents" a set of **pairs** of hands. By doing this, we manage to limit the participation of rotation particles which are not naturally –by means of human joints limitations- acceptable. On the other hand, multiple pose hypotheses are being tracked simultaneously.

4.3.2.1 Resampling

The resampling of the rotation particles is done according to the weight of each particle. The weight represents the probability of each particle to be drawn. The new samples are drawn from a Normal distribution with mean the value of the particle parameter –in this case the rotation angle- and deviation either a predefined value, or a function of the weight. Finally, in order to keep computational cost as low as possible, whenever there is at least a particle with $w^r \ge W^R_{\max}$, which means certainty about the real rotation, the number of rotation particles is reduced as well.

4.3.3 Hand particles resampling

The resampling of the hand parameters particle is based on both the weights of each hand particle and the weight of the correspondent rotation, as explained in the previous example. Each new sample's parameters, namely the four joint angles, is drawn from a –separate for each parameter- normal distribution with mean value the corresponding parameter value of the seed particle and deviation equal to a

predefined value. Like before, whenever there is at least one hand particle with $\tilde{w}^m \geq W_{\max}$, the number of the remaining particles is reduced.

4.4 Tracker's Output - Clustering

The algorithm presented in the previous sections, manages adequately to export high-level information, such as kinematic parameters, with the use of a minimum amount of input. Moreover simplicity and efficiency is preserved, while computational cost is kept low.

The result, however, of previous stages is a set of weighted particles and not a single pose. Note that although multiple poses are being tracked, they are similar to each other[7]. Therefore, they could be represented by a single pose. A very efficient way for representing the output pose is by clustering the hand particles with $w^m > W'_{\max}$ of the most dominant rotation particle. The centroid of the bigger cluster (which contains most of the nodes) will constitute the output of the tracking algorithm. Unfortunately, due to the number of deployed particles (several thousands) and the dimensionality of parameter's space, clustering adds a large computational load to the system, and thus increases the time needed for the processing of each frame –at least 10 seconds per frame, depending on the number of particles.

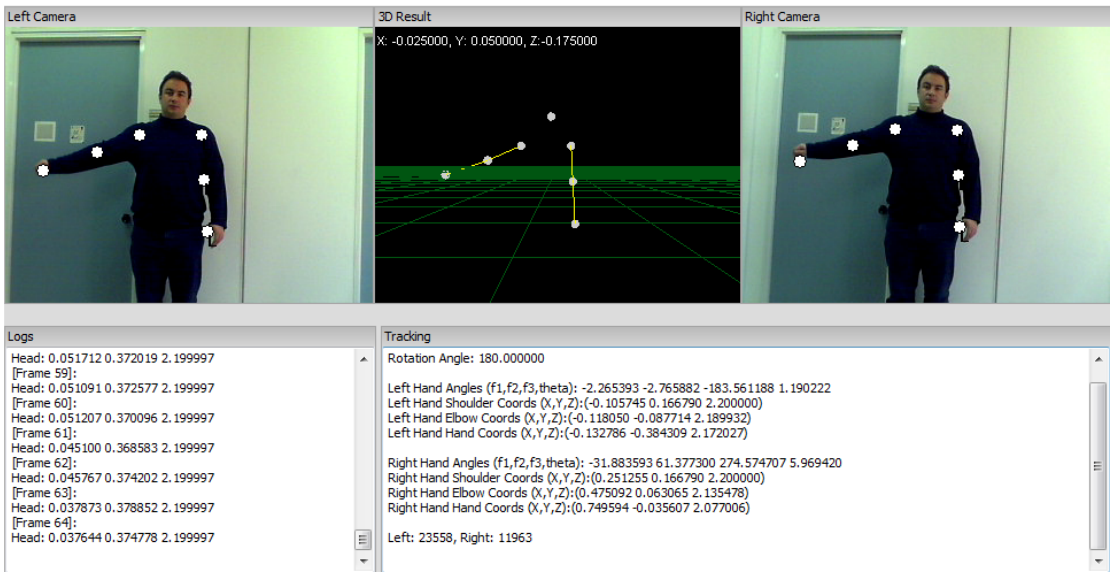


Figure 26: Kinematics Tracker Output.

The extracted kinematic parameters, will be the input for the last stage; the gesture recognition phase. Figure 26 illustrates the output produced by the

[7] Their projections on the camera are more or less the same, despite the actual pose.

kinematics tracker, depicting the extracted kinematic parameters, joint locations in space and their projections on the cameras. It is obvious that the constructed model is accurate enough, although abstract. Therefore, the gesture recognition process becomes even easier, due to the stability of the output.

Chapter 5

Gesture Recognition

In this final phase of the hand gesture recognition system, the extracted kinematic parameters of the hands are being translated into meaningful information about a possible ongoing gesture. This involves a mechanism capable of discovering and recognizing gesture patterns from the input data. Moreover, apart from efficiency, that mechanism should fulfill a crucial requirement: ability of **generalization.**

Here, generalization has both spatial and temporal meaning. Both aspects can be easily understood by considering the variety of each gesture together with the way with which each human carries out a gesture. To clarify this, assume a pointing gesture ("go there") carried out by a person with the left hand. First of all, depending on the desired direction, a pointing gesture can have various spatial representations. Additionally, even when the direction of the gesture is the same, there will always be slight differences in the exact pose of the hand. Finally, when executed by the same or different persons, the duration of each gesture may differ.

A tool which fulfills the above criteria is Neural Networks. More precisely, MLPs (as described in 3.4) proved to be very efficient in pattern classification tasks, having, in the same time, great generalization capabilities. Moreover, supervised backpropagation training of MLPs, seems more likely to adapt to the needs of the gesture recognition nature, since it is most probable that, due to the complexity of the parameter space, error surface will present local minima and plateaus. Before proceeding to the presentation of the neural network used in this phase, we have to cope with two problems produced by the output of the pose tracker: **output interruptions** and **discontinuities**.

Due to corrupted video input –i.e. lost frames- or poor tracking results, there is a possibility for some frames to produce no output. On the other hand, as multiple pose hypotheses are being tracked simultaneously and it is not –and should not be-

guaranteed that the same hypothesis will be the one producing the output over time, hands movement may not be smooth, jumping from a certain pose to a very distant one. Both problems can be solved by filtering the output passed to the neural network. Instead of producing an output at each frame, it will be produced either whenever there is a significant change of the hand pose or after a predefined period of inactivity (with no new output produced).

In order to constrain output discontinuities though and, thus, preserve as much motion smoothness as possible, the change of the pose should not be too intense. In combination with the previous, this implies the use of two threshold values which determine if and when an output set should be produced. An output will be produced only when the observed pose change lies between the two threshold values. The use of these threshold values ensures time invariance of the output, since regardless of the speed with which a gesture is executed, it will be discriminated spatially and not temporally.

5.1 Gesture Recognition Scheme Overview

In the observed scene, the actor is supposed to perform gestures with both hands, possibly simultaneously. Therefore, the output of both hands has to be processed for possible ongoing gestures. Since there is no prior knowledge of whether a hand performs a gesture –and which hand is the performing one- the fact that both hands are being tracked implies that at each frame, nine parameters (4 angles for each hand and the body rotation around the Y-axis) should be fed as input to the neural network. The complexity and the dimensionality of the parameter space, however, restrict any successful training of the neural network.

To cope with this, a first step is to use a separate neural network for each hand. The space dimensionality is now reduced to 5 –the body rotation has to be kept for both hands- which enables an acceptable training of the network. Unfortunately, parameters are vaguely deployed in space resulting to possible ambiguities between different gestures. Despite the fact that the complexity of the dataset is decreased by processing each hand separately, trying to recognize different gestures with a single neural network, will most probably lead to very poor results. Moreover, if each neural network is responsible for the recognition of multiple gestures, the final "translation" of each network's output will not be an easy task.

Consequently, further division of neural networks is compulsory. A distinct neural network will be assigned to each gesture for each hand. This means that if there are N=3 gestures to be recognized, a total number of six neural networks will be employed. The independency of the neural networks guarantees the absence of

recognition ambiguities, since each network will be responsible for a linear 2-D classification; namely ‘0’ if the trained gestured is not being executed and ‘1’ otherwise.

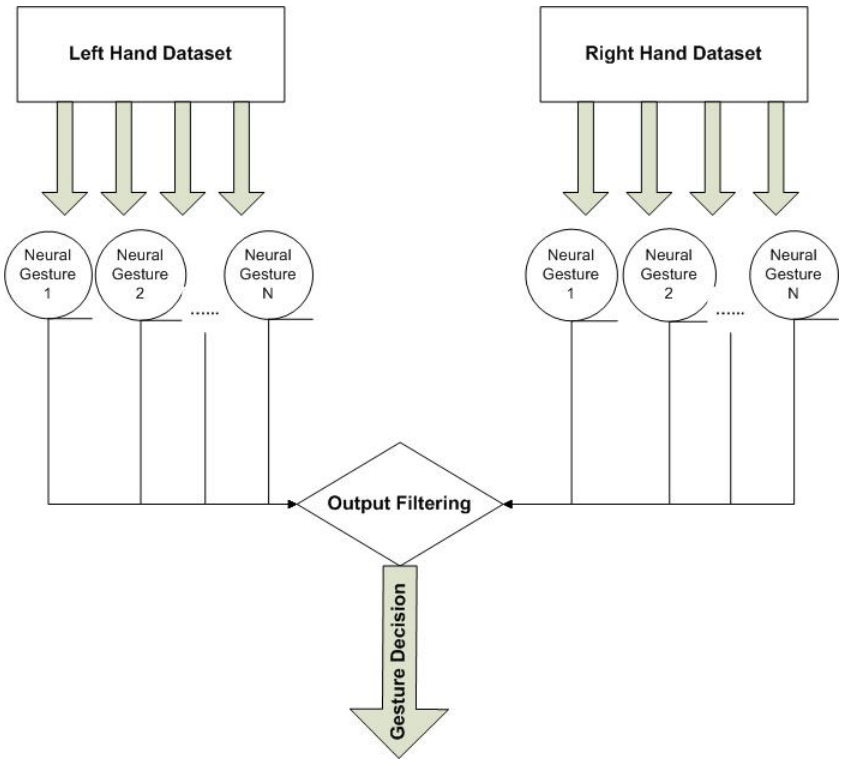


Figure 27: Gesture Recognition Scheme.

However, a case where more than one neural networks claim that a gesture is being carried out is possible. In reality, the output of each network is set in the continuous range of [0, 1], depicting the certainty for the corresponding gesture. Therefore, all outputs can be set and combined together in order to end up with a final decision of the ongoing gestures. Plural is used since both hands might perform gestures simultaneously. Figure 27 illustrates the idea described above. The outputs of the neural networks are being “filtered” by a simple the-winner-takes-it-all gate, which is responsible for the final decision.

5.2 Neural Network Architecture

The architecture of each neural network, should adapt to the needs of the gesture modeling in order to provide accurate results. In this work, each gesture is a dynamic structure, which means that it is formed by a set of sequential arm poses. In other words, a single frame (i.e. a single arm pose) does not provide any information about a possible ongoing gesture. Therefore, each output, provided by the kinematics tracker, should be processed in combination with the preceding outputs.

By considering the way that each output is produced –as described at the beginning of this section- it is guaranteed that a small amount of preceding outputs can depict a significant pose change (or *movement*) of the arm. This fact implies that only a small number of input nodes is needed. The neural network architecture used is illustrated in Figure 28, where p(t) denotes the angle quadruplet (ϑ_1, ϑ_2, ϑ_3 and ϑ_4) at time t.

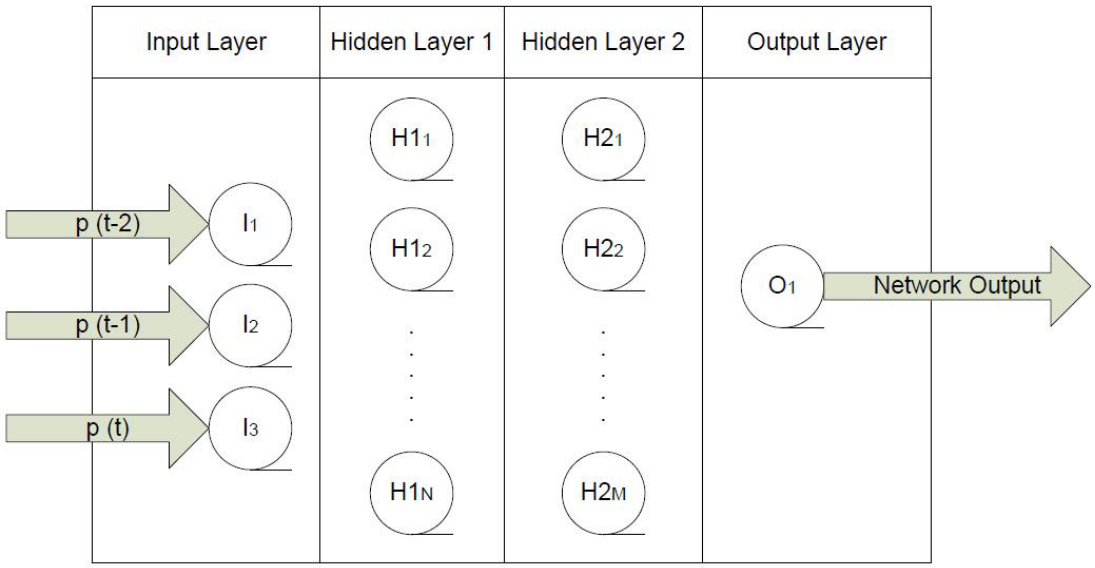


Figure 28: Gesture Recognition Neural Network Architecture.

In the above figure, the connections between nodes of different layers have been omitted for the sake of brevity, while each input node represents, in reality, four separate input nodes. The output produced lies in the range [0, 1], where values close to 0 and 1 depict that the trained gesture isn't, or is, respectively, being performed. Here, the second layer is used for generalization reasons. With M>N the data are spread throughout the network, achieving better representation and relevance between them.

Having determined the architecture of the neural network, we can proceed to the description of the training phase. At first, a closer look on the training datasets needs to be taken.

5.2.1 Training datasets

The datasets used during the training phase have to be both realistic and, if possible, accurate. Obviously, the output of the tracker will be the input dataset for the training phase, in order to maintain relevance between training and actual data. However, even in this case, a great obstacle occurs, imposed by one of the main advantages of the kinematics tracker: *tracking of multiple hypotheses*. As several

body (rotation around the Y-axis) and arm (kinematic parameters) hypotheses are being tracked simultaneously, and given that there is no apparent way to eliminate some of them, multiple possible outputs are extracted. Since hypotheses are irrelevant to each other, severe discontinuities of angle values might appear, making training impractical.

Fortunately, kinematics tracker results for each body rotation seem to be stable. Here, 'stable' means that, given a fixed rotation value, the promoted arm hypothesis –chosen by clustering or by simply comparing the weights- represents the actual scene and preserves relevance from frame to frame. Therefore, it is convenient to use datasets produced with a fixed body rotation value, simplifying network training both in efficiency and time consuming terms.

5.2.2 Network training

Having a realistic representation of each gesture through the training datasets enables fast and efficient training of the neural network. It also gives the possibility of using simple training techniques and error functions, as the nature of the datasets overcomes most of their disadvantages.

As described before, each gesture consists of a sequence of poses (outputs), which imposes that network weights should be updated for every new input. Therefore, the training technique that suits the most to the examined problem is that of *online backpropagation* (as discussed in 3.4.3.2). Additionally, in order to impose pattern continuity, the weight of each node is calculated based on the *momentum* weighting function described in 3.4.3.1 Eq. 44.

For each new input, the current output is calculated based on the estimated node weights. The error which depicts the difference between the current and the expected output is calculated based on the Mean SSE function presented in Eq. 38. The reason of this choice is that input data present many similarities –different gestures may have similar or same subsequences-, which leads to the existence of sub-optimal solutions. Mean SSE copes efficiently with this type of problems.

Since the desired output lies between 0 and 1 (gesture isn't or is performed), a fine representation is the sigmoid function of Eq. 34. Figure 29 presents a snapshot captured during training phase, where current and desired (expected) output together with the average network error are shown.

Figure 29: Neural Network Training.

Learning rate, momentum value and number of epochs should be adjusted according to the specific task. Although momentum value can be the same for all input datasets, the more complex the dataset is, the smaller learning rate and the larger number of epochs is needed in order to cope with discontinuities and suboptimal solutions. For example, in some cases where datasets presented similarities up to 1000 epochs were needed for the network to converge, while in cases where the datasets where simpler (classification of data was more “obvious”) the network converged in only 400-500 epochs. However, the values for these parameters should be chosen carefully in order to avoid over-training of the network, which will eventually prevent generalization.

Finally, the convergence threshold (i.e. the acceptable average error value) is another parameter which should be carefully chosen: too high and the network won’t perform efficiently; too low and the network won’t be able to generalize. An acceptable value for the convergence parameter seemed to be approximately 0.05.

5.3 Gesture Modeling and Network Choice

The above issues concern the sequential nature of each gesture. However, this does not unambiguously distinguish gestures, since parts of their motion paths are similar –if not identical. While different gestures share the same –or similar- paths, it is possible that both training and recognition phase will fail. In order to cope with this problem, the first two rules of the Definition 3, expressed in 2.1.2, seem to be very helpful:

Rule 1: *Gesture interval consists of three phases: preparation, stroke and retraction.*

Rule 2: *Hand configuration during the stroke follows a classifiable path in the parameter space.*

Indeed, most of the hand gestures can be divided into three phases: preparation, stroke and retraction. Additionally, whereas preparation and retraction phases of each gesture are practically identical, the stroke phase is clearly distinguishable between gestures. This fact can be used to minimize possible ambiguities, since it enables a stricter and more robust gesture modeling.

In order to employ these rules into our neural network scheme, each phase is assigned to a separate neural network. Assuming that preparation and retraction phases are the same for every gesture, the additional computational load is limited, since only two neural networks will be added. Moreover, the proposed gesture modeling clearly outlines the "borders" between different gestures and thus, facilitates the classification between both the gestures and the phases they are consisted of. The resulting gesture recognition scheme is illustrated in Figure 30.

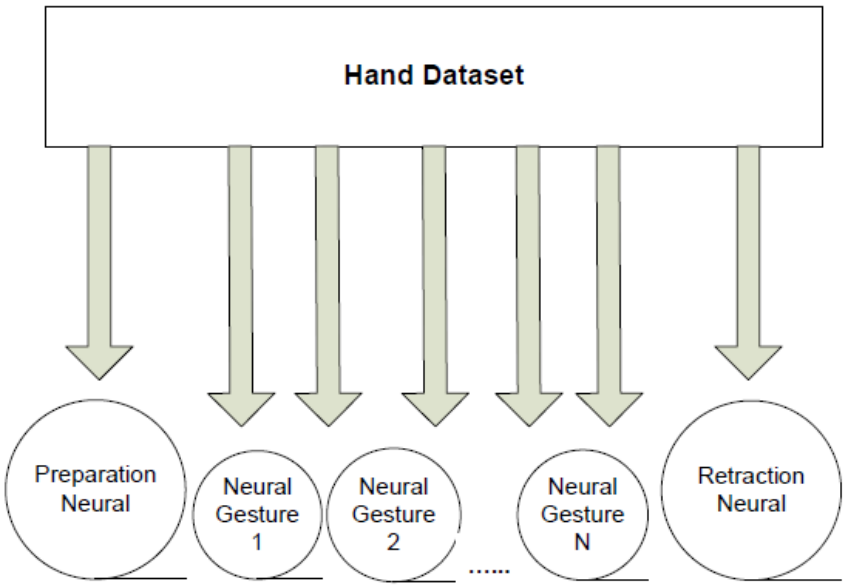


Figure 30: Final Gesture Recognition Scheme.

The above scheme increases the accuracy of the gesture recognition system, since it makes different gestures clearly distinguishable. It eliminates most of the possible ambiguities, since gestures no more share common "paths". Still, some factors have to be tuned in order to end up with a capable gesture recognizer.

5.4 Recognizing Gestures

Although for training purposes it is feasible to have a fixed body rotation, this is not the case when trying to recognize a gesture performed by an actor, in an unknown scene without previous knowledge of its pose. Therefore, gesture recognition might become tricky, since we are not able to promote just one hypothesis. Moreover, kinematic parameter values of different body rotations are completely irrelevant to each other, having large difference, even if the actual pose is the same.

A solution to that is given by clustering the candidate body rotation hypotheses and applying the gesture recognition scheme presented in Figure 27 for each of the clusters. This is derived from the fact that, no matter the explicit angle values, there is direct relevance between the angles of the training fixed rotation and the ones of each rotation extracted from the kinematic tracker.

The decision for a possible ongoing gesture is therefore an easy task. Although there will be several network outputs (one for each cluster), only those corresponding to the cluster representing the actual pose will be valid. However, the number of clusters should be carefully chosen, since a large amount of clusters could increase execution times and might decrease recognition efficiency.

The gesture recognition from the neural network follows more or less the procedure of the training phase. Each new input is propagated through the network (based on the nodes weights calculated during training) in order to have the final output. Output with value close to 0 depicts that no gesture is recognized, whereas output value close to 1 means that the trained gesture is being performed.

Chapter 6

Results

The main target of the current thesis was to develop a robust and efficient gesture recognition system. Moreover, this work focused on the extraction of high-level information –like kinematic parameters-, which will finally facilitate the gesture recognition problem, with minimum a priori information.

Indeed, the developed system performed more than adequately in the examined cases. Evidently, the novelty introduced by this work is the abstraction achieved in gesture modeling. The assumption that common gestures can be expressed, and therefore recognized, without detailed knowledge of the arm configuration proved to be valid. On the other hand though, gestures which contain high level of detail – e.g. where the configuration of the fingers is needed- are excluded from recognition. Still, the number of gestures that can be recognized remains large, including mostly deictic, navigating and manipulative types of gestures (see Chapter 2).

Since hands are being tracked independently, gestures can be recognized on one or both hands. Therefore, combinatorial gestures, where actions from both hands are needed, can be easily recognized. In our test cases, three types of gestures have been examined: *pointing, hello and attention*. A straight raised arm indicates a pointing gesture, while a bouncing forearm might imply a hello gesture. Whilst pointing and hello gestures are being carried out by a single hand (they can also be tracked for both hands), attention gesture is being performed by both hands. In fact, it is recognized whenever both hands perform a hello gesture simultaneously.

Although our main goal is the recognition of hand gestures, the process of hand tracking, which extracts the parameters that are fed to the neural network, is crucial. Accurate hand tracking facilitates hand gesture recognition. Therefore, a closer look at the results of the kinematics tracker needs to be taken.

6.1 Hand Tracking Results

As already discussed, the basic idea behind the hand tracker is the development of a **simple** and **efficient** system. This target has been more or less achieved, since with the use of simple mathematics and algebraic transformations, pixel values lead to the extraction of high-level information, which can adequately model the human arm. Hence, the performance of the hand tracker is accurate in most cases at hand.

6.1.1 Calculation accuracy and prior scene knowledge

One of the main advantages of the proposed methodology is that there is no need for highly accurate calculations. On the contrary, possible errors produced at each stage (skin-color tracking, particle filters, kinematics), do not have an immense effect on the final outcome. A representative example of this is depicted in Figure 31. In this result, the right camera accidentally moved after the calibration process, which means that the extracted extrinsics did not anymore match the actual ones. Although this was noticed during the experiments, it did not have a significant impact on the produced results.

Figure 31: Right Camera moved after calibration.

What's also worth mentioning is that there is no need for prior knowledge about the scene and/or the actor. The anthropometric proportions used ensure the adequate limb length estimation, regardless of the actual proportions of the actor. Therefore, the performance of the hand kinematics tracker is within acceptable limits without prior training with respect to the actor and/or environment.

6.1.2 Initialization procedure

The initial estimation of arms pose is a crucial part of the hand tracking process. Ideally, most of the deployed particles should reflect –or should be close to- the

actual pose. However, the dimensionality of the sampling space –i.e. four degrees of freedom for each hand- presents an obstacle to particle initialization. Interestingly, the use of inverse kinematics for initial pose estimation gave promising results, facilitating the rest of the process.

Figure 32: Initialization using Inverse Kinematics.

In Figure 32, two cases are presented, being appropriate to demonstrate the effect that inverse kinematics have on the initialization process. In Figure 32(a) initialization has been done randomly, while in Figure 32(b), inverse kinematics have been used for obtaining the initial estimation of the kinematic parameters. As a result, more particles reflect the actual arms pose for various body rotations. This fact has a beneficial impact to the tracking process, as convergence is largely facilitated.

6.1.3 Robustness

During our experiments, simple web cameras have been used as input devices. This led to poor video quality and/or corrupted videos with missing frames. Although this constitutes an additional problem, it was easily solved by fine-tuning two of the parameters used in kinematics particle filtering: window margin T_W and resampling deviation value. By increasing T_W, the acceptable distance of the hand projection from the projection of the skin-colored blob increases proportionally. In cases where some frames are lost, particles, which in other cases would have been eliminated, continue to be part of the tracking process, increasing the odds for the tracker to converge. On the other hand, resampling deviation declares the margin inside which a particle can lie. Small deviation value does not allow particles to approach the

actual arm pose, in case of missing frames. Another factor that contributes toward this is the multiple hypotheses tracking. Despite the significantly small number of particles used, tracking multiple possible poses ensures, up to a scale, that, eventually, the tracker will converge to the actual actor pose.

Figure 33: Missing frames do not affect the tracking process.

Figure 33 illustrates an example where some frames are missing from the input video. Although the difference was immense (consider that the framerate of the input video is approximately 25 fps, which normally implies slight changes from frame to frame), the tracker managed to converge in only 4 frames. Additionally, due to the thresholds imposed (discussed in Chapter 5), tracker output remained stable, as transitional outputs might be unpredictable.

Margin window and resampling deviation values, however, should be carefully chosen, in order to be both realistic to human capabilities and able to cope with discontinuities. Excessive increase of these values might lead to unwanted results, which will eventually lead to failure of the tracking process. The next figure depicts a case where the deviation chosen could not represent the actual pose of the actor. Consequently, the tracking process produced false results.

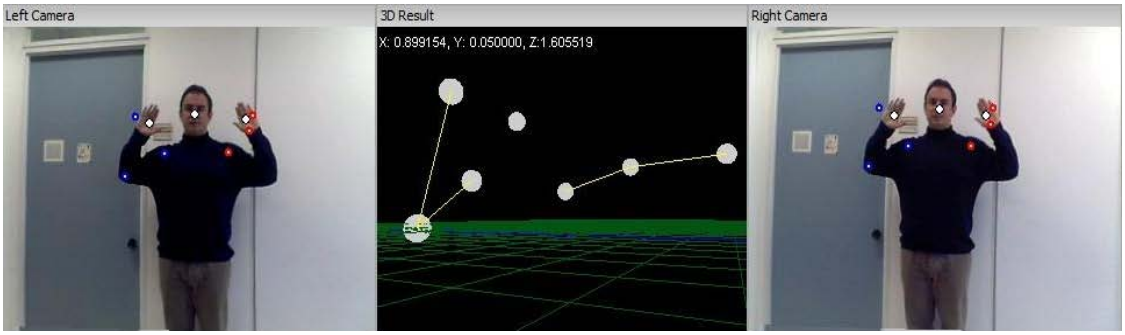


Figure 34: Tracker failure due to unsuitable deviation value.

On the other hand, if window margin and resampling deviation are set so that they reflect the reality, hand tracking performs efficiently in most cases, even ambiguous ones as the one presented in Figure 35. For comparison reasons, the next figure is cited, illustrating a similar scene but with altered deviation value. The results, apart from obvious, are quite accurate.

The resampling deviation can be abstractly thought of as being a measure of change priority of the kinematic angles. In general, resampling deviation of ϑ_4 angle –namely the angle of the elbow- should be kept smaller than the ones for the rest angles. However, large differences of resampling deviations might end up with false outputs (like the one illustrated in Figure 34). An acceptable value for the deviations of ϑ_1, ϑ_2, ϑ_3 seemed to be approximately 0.08 and 0.05 for the ϑ_4 angle.

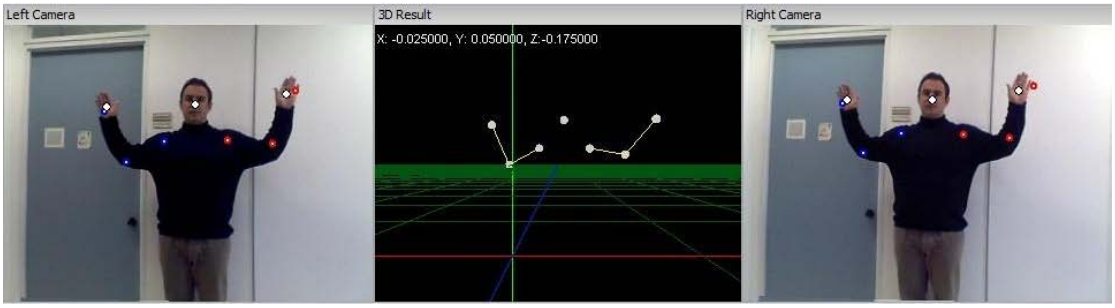


Figure 35: Correct tracker results after parameter fine-tuning.

6.1.4 Time invariance

Due to the rate and the criteria, based on which the output is produced, invariance on the timing that each gesture is performed has been achieved. Having fixed gesture duration, or fixed gesture path, would limit the gesture recognition to the actor that trained the system. Instead, by imposing appropriate rules to the produced output, gesture recognition can be equally efficient for different actors, and consequently different gesture speeds and paths. This, along with the fact that no prior information exists regarding the actor and/or the scene (see section 6.1.1), makes the hand tracker completely independent on the actor and the scene.

6.1.5 Pose and depth ambiguities

Although accurate for the majority of the examined cases, uncertainties concerning the depth, and consequently the actual pose, may arise. This is caused mostly due to the absence of depth calculation (except for the initial estimation), which therefore implies that two completely different poses can have equal weights. Figure 34 can also be an example of failure due to depth ambiguity. In that case, it is possible that the tracker could never resume on the actual pose.

Hopefully, tracking of multiple hypotheses can cope with this problem, as the tracker will eventually converge to the actual pose if the corresponding hypothesis is being tracked. Moreover, since several gesture recognition modules are being executed simultaneously (discussed in **5.4**) –one for each rotation hypothesis cluster- it is likely that depth ambiguities might not be an intractable obstacle.

6.1.6 Execution time

The main drawback of the proposed methodology is the long execution time needed. Despite the efforts to minimize the number of deployed particles, which are the most time consuming modules, this figure cannot be limited significantly, since there should be an adequate number of particles in order to cover several hypotheses. To make things worse, the already long execution time has also some important side-effects: it prevents the clustering of the hypotheses, which would lead to even more accurate (and stable) results.

Without performing hypotheses pose clustering, in order to process each video frame, up to four or five seconds are needed. By performing clustering, this number becomes excessively large. In some cases, the time needed to process a single frame was more than 30 seconds, which is clearly prohibitive.

6.2 Gesture Recognition Results

Having an output as accurate as possible from the hand tracker facilitates both neural network training and gesture recognition processes. However, although gesture recognition performed efficiently in most cases, some small problems may occur. For example, gestures might have similar (or same) subparts, which will lead to a temporal false positive outcome. The term “temporal” is used since, eventually, the network will converge to the desired output, as input data are being fed sequentially. In order to have a better overview of the produced results, both network training and gesture recognition results will be presented.

6.2.1 Neural network training

Various configurations have been tested in order to find a balance between effectiveness and training time. Despite the fact that the chosen architecture itself – the use of two hidden layers- imposes a significant processing load, it should not be altered due to the nature of the input data (similar gesture subparts as discussed earlier). On the other hand, the addition of a third hidden layer does not guarantee increase of effectiveness.

Neural Network training phase is directly dependent on the nature of the input datasets. Consequently, training parameters -such as learning rate, momentum, number of epochs and number of hidden nodes, should be adjusted so that they ensure increased efficiency and ability for generalization (or should constraint network's over-training). After numerous trials, an acceptable number of nodes of the two hidden layers found to be N=25 and M=90, while, in order to keep reasonable training times, the number of epochs was limited to 100.

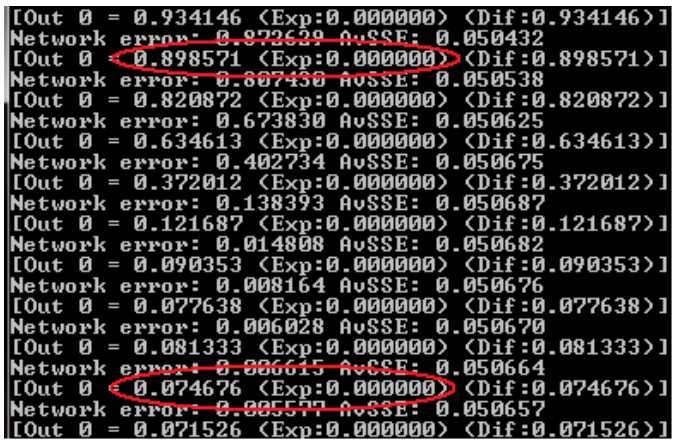


Figure 36: Despite the uncertainty, the network converges to the expected output.

Due to the sequential nature of the datasets, learning rate preferred to be kept in low-levels and approximately around 0.1. Finally, the momentum value was set to 0.8, not deviating significantly from the proposed value in **3.4.3.1**.

In general, network training was successful with the use of the above configuration. The above figure, illustrates the case where the trained gesture presents similarities with another one, leading to false positive outputs. After the weight update, and within a few iterations, the network managed to converge to the desired output (depicted as Exp). Training, however, took long enough for each gesture (approximately two hours), which was the only drawback of the training phase.

6.2.2 Gesture recognition

It is obvious that the success of the gesture recognition does not rely solely on the effectiveness of the training phase. Without a representative gesture modeling, which implies the clear distinction between different parts of each gesture, many false positives might occur due to large overlapping areas. However, temporary ambiguities do not completely vanish, since gestures, naturally, continue to share similar subparts, no matter their modeling. Fortunately, these ambiguities are being eliminated fast enough.

As already mentioned, resampling deviation plays a crucial role to the recognition procedure. Additionally, if the chosen deviation isn't capable of covering the real case, it can even enhance the depth ambiguity problem, leading to unsuccessful tracking results and, consequently, to gesture recognition failure.

During the experiments, three different types of gestures have been tested: *pointing, hello* and *attention* gestures. While pointing and hello gestures need to be trained first, attention gesture does not require any further training since it is formed whenever both hands perform a *hello* gesture. Let's have a closer look to the results produced for each of the examined gestures.

6.2.2.1 Pointing gesture recognition

Pointing gesture can be generally characterized as an "easy" gesture to be recognized. However, there are cases –like the one when the actor points directly to the camera- where depth ambiguities and wrong resampling deviation can prevent a successful recognition.

In the simple case where the real pose can be straightforwardly derived –namely cases where there is no ambiguity concerning the depth- the gesture recognition met almost no obstacles. Figure 37 illustrates a successful recognition of the pointing gesture, with the actor pointing on his right side.

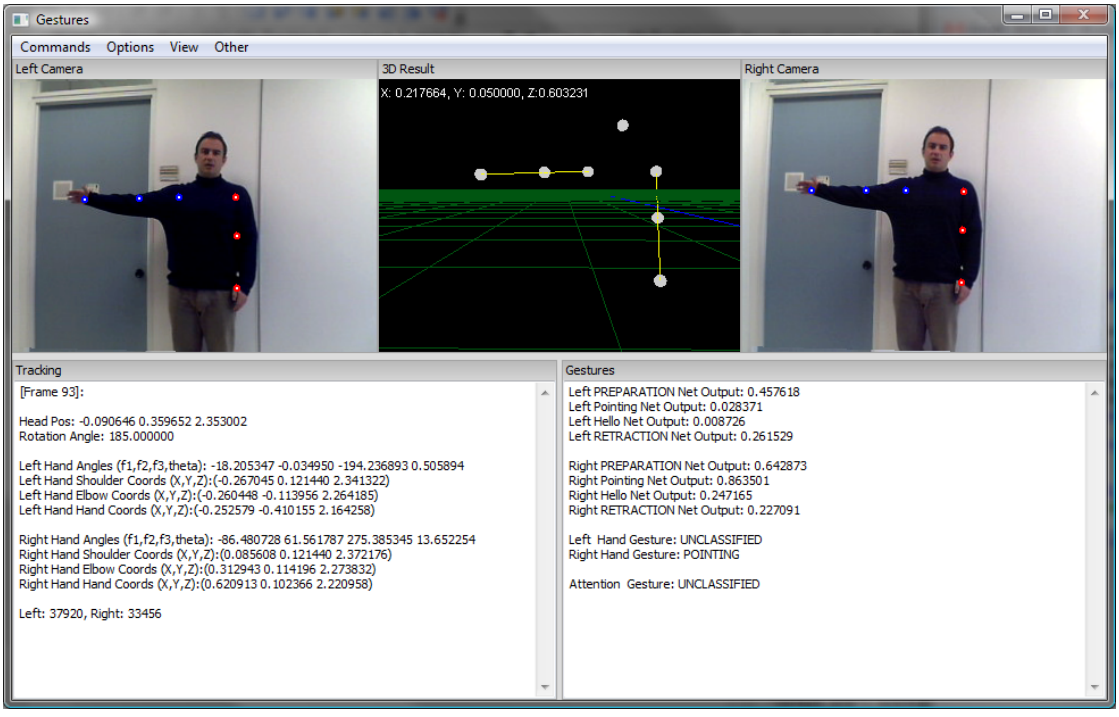


Figure 37: Successful recognition of right pointing gesture.

However, when the actor points directly to the camera, the recognition can sometimes fail; mostly because of the false parameters produced by the tracker.

Figure 38 depicts such a case, where, obviously, the parameters do not match the actual arm pose. Therefore, no matter how effective the network training has been, there is no gesture pattern which could match with the produced parameters.

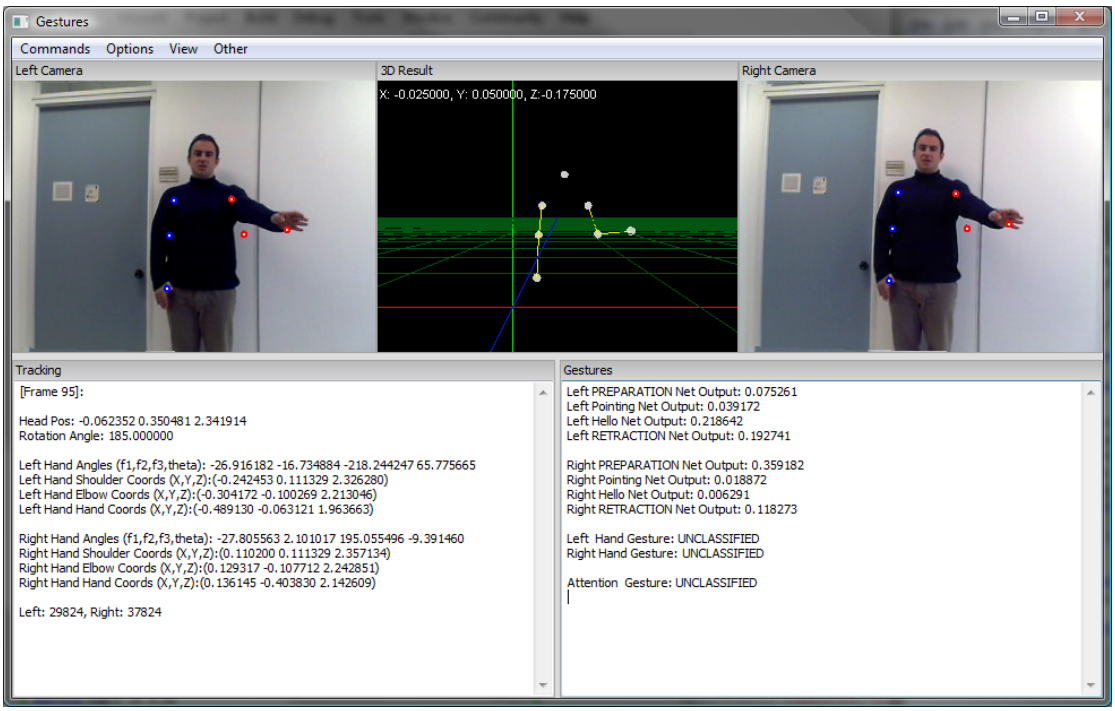


Figure 38: Failure to recognize the performed gesture due to unsuitable resampling deviation.

Figure 39: Successful recognition after resampling deviation fine-tuning.

The above failure occurred because of the fact that the chosen resampling deviation led to wrong pose estimation. Fortunately, a careful choice for the deviation values – as the ones mentioned in 6.1.3- was beneficial for the recognition procedure, having as result the successful recognition of the performed gesture, as illustrated in Figure 39.

6.2.2.2 Hello and attention gestures

As already stated, the attention gesture is being performed whenever both hands perform the hello gesture. The following figures illustrate a representative example of an attention gesture recognition experiment. During this test case, although the preparation step has been successfully recognized, for a short time period after that, two networks were producing high outputs and were competing with each other for the final gesture decision. Although, this case is, unfortunately, inevitable, the appropriate gesture modeling and network training enable the fast recovery of the system.

As the sequence starts, no gesture (or part of a gesture) is recognized. After a few frames of inactivity, the preparation phase is recognized for both hands, as depicted in Figure 40. This does not affect the rest of the neural networks, which keep producing outputs, since all kind of actions can follow. However, notice that the two main gesture networks (responsible for hello and pointing gesture) do produce an average output, around the value of 0.5. This occurs because during the training of the preparation phase, and in order to apply the meaning of “preparation” into the gesture and to preserve smooth state transitions, the main gesture networks were producing an output of 0.5.

With the above fine-tuning in place, the transition from one phase to the other was indeed smooth. However, as depicted in Figure 41, the decision was not the correct immediately. As observed, although the actual gesture of both hands is the “hello” gesture, the recognition of the gesture performed by the left arm failed, due to both tracking misdetection and the fact that the state of the arm fits to both gestures.

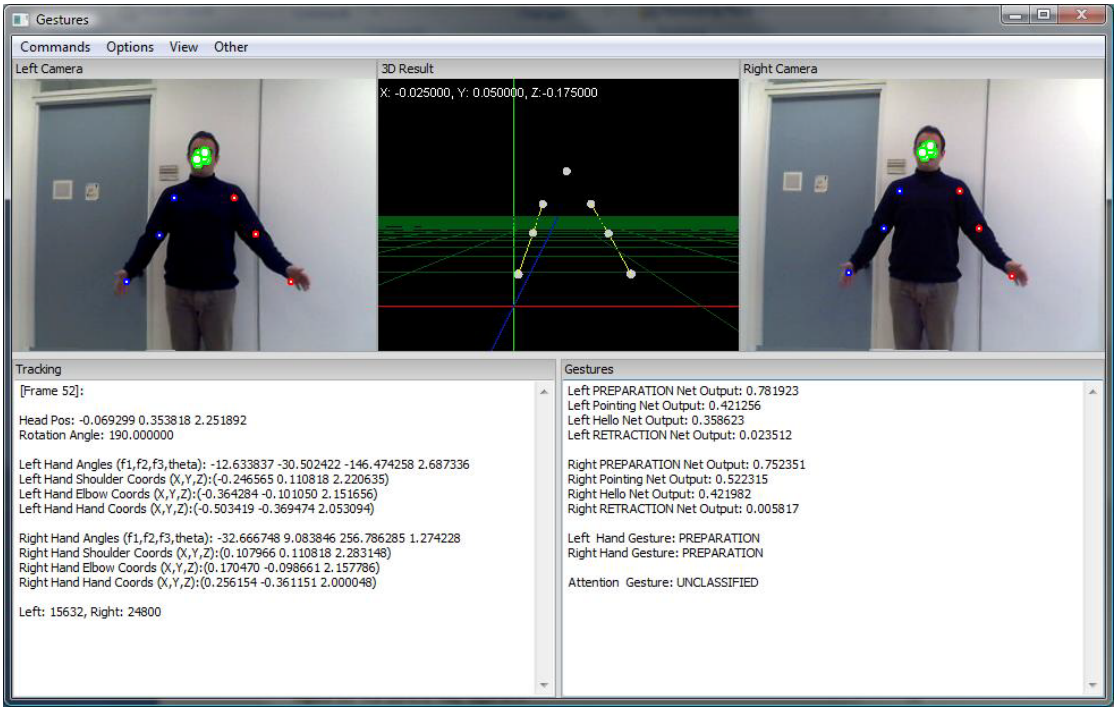


Figure 40: Gesture Preparation.

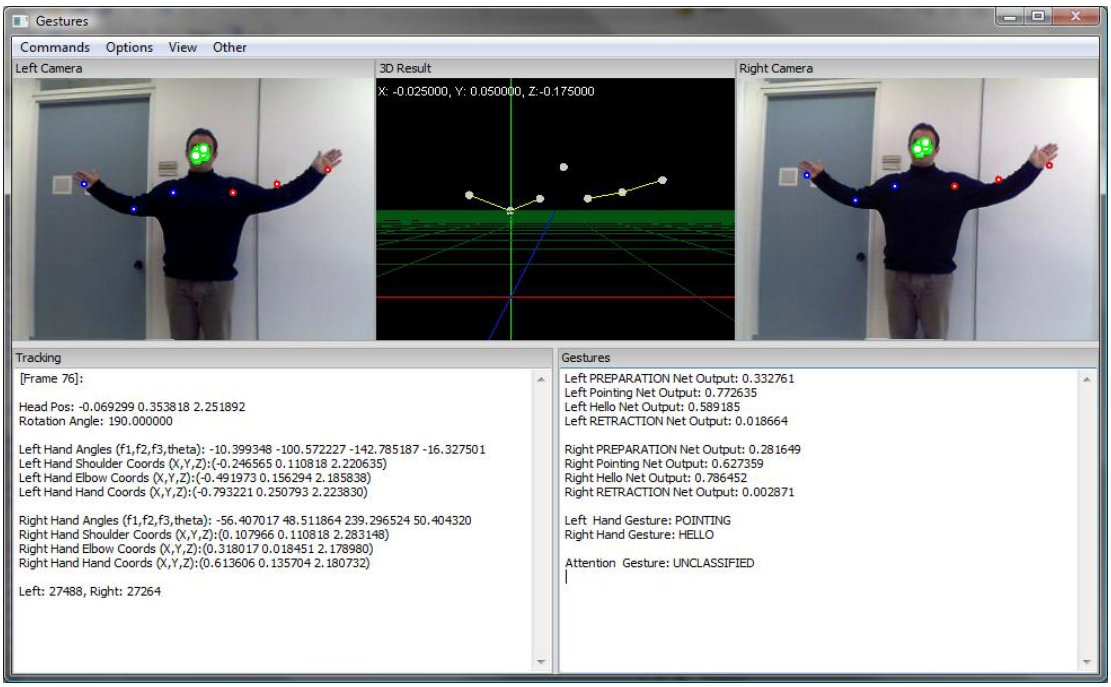


Figure 41: Uncertainty concerning the gesture performed by left arm.

A few frames later, however, things seem to clear up, since the followed path differs completely from that of the pointing gesture (where the arm remains practically stable). Observe the difference of the outputs between these two frames (Figure 40 and Figure 41). The output of the network responsible for the hello gesture has been increased, while the output of the pointing network has been

decreased significantly, allowing the extraction of a safe decision about the ongoing gesture.

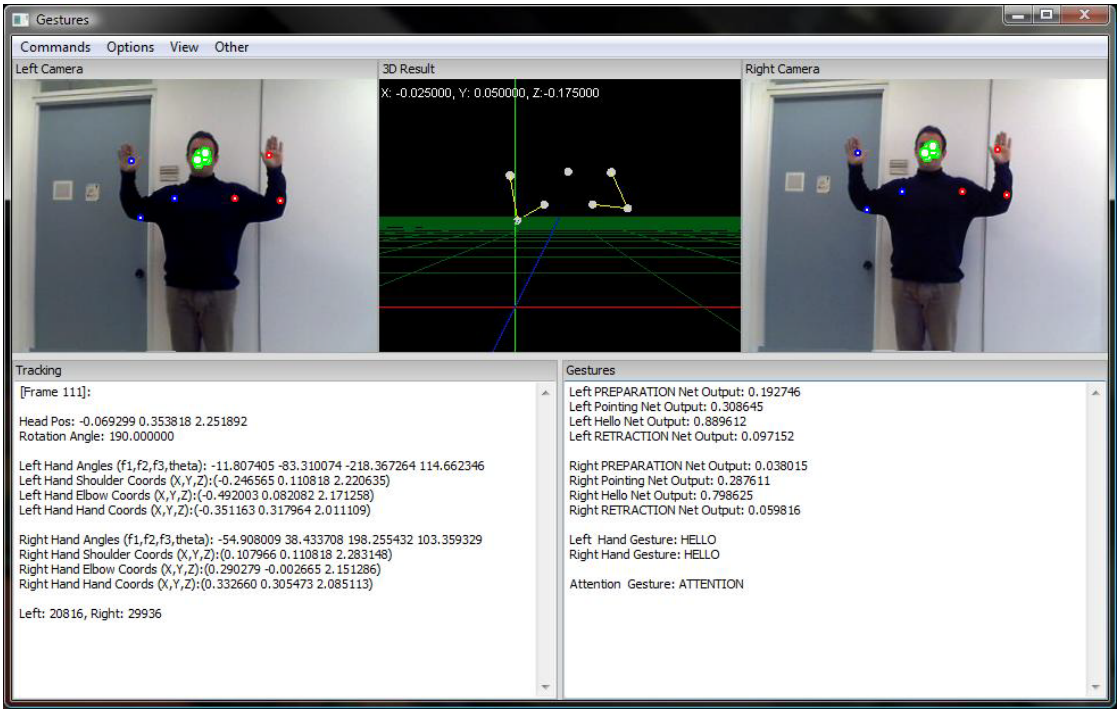


Figure 42: Attention Gesture successfully recognized.

Chapter 7

Discussion

In the current thesis, a probabilistic approach for recognizing hand gestures has been proposed. The main goal was the extraction of high-level information with the minimum knowledge regarding the scene and/or the actor. The core of our work was based on the assumption that many common hand gestures do not require detailed information in order to be recognized. This facilitated the decrease of the space dimensionality and, hence, the complexity of the problem. By doing so, the tracking of arm parameters and, consequently, the task of gesture recognition became more tractable.

One of the main achievements was that, for most of the subtasks involved, accurate calculations are not necessary. This manifests itself in the tasks of camera calibration and the calculation of the size of the limbs. In both cases, the extracted estimations contained a (possibly) large amount of error. Nevertheless, this did not prevent the successful recognition of the examined gestures.

The proposed recognizer may be useful in various applications. Communication between humans and machines can be boosted, since gestures are a very common form of interaction, facilitating tasks such as navigation in virtual environments, object manipulation or control of devices. Moreover, it can serve as a method for interactively action/gesture robotic arm learning, since the kinematic parameters produced can be directly translated and incorporated by a robotic system.

Despite the fact that the performance of our implementation was acceptable in all examined cases, various aspects are amenable to further improvements. These regard ambiguities in depth estimation, excessive execution times and particle filter resampling.

Depth ambiguity remains a limiting factor in gesture recognition accuracy, although it has been addressed in the design and implementation of the system. It

can lead to complete tracking and recognition failure, since the behavior of particles is unpredictable and it is not guaranteed that there will always exist hypotheses which satisfy the real case.

Furthermore, training and execution times are still excessively long. The current implementation requires considerable off-line work (preparatory stages for the video input) and, moreover, the on-line processing of a single frame is prohibitive for a real-time usage of the recognizer. This problem is of major importance since it limits the potential of the proposed approach due to practical reasons.

Finally, by far the most important problem is the direct dependency between system's effectiveness and resampling deviation. As already described, corrupted video sequences or poor image quality can lead to severe pose discontinuities. In that case, the chosen deviation (resampling deviation in the kinematic tracking) should be able to cover the produced "gap". However, deviation is also application dependent. For example, in some cases, deviation values, which worked well for the task of "pointing" gesture recognition, did not provide a good result for the task of "hello" gesture recognition.

7.1 Future Work

Notwithstanding the limitations of the proposed approach and the corresponding implementation, the obtained gesture recognition accuracy is satisfactory for most of the examined cases. Still, there is much room for improvements and enhancements to our system. Apart from coping with the previously discussed problems, new techniques could be used, in combination with the proposed recognizer, in order to increase its effectiveness and expand its capabilities.

At first, use of high-quality video, together with code optimization, might confront with the depth ambiguity problem and/or drastically reduce execution and training times. Still, a significant step would be the ability of automatic deviation adjustment, according to the observed scene and the foreseen gesture. A fine-tuned deviation could possibly guarantee the elimination of parameter discontinuities, which would consequently lead to more accurate recognition results.

The full potential, though, of the gesture recognizer could be revealed when used in combination with other forms of human-computer interaction. Speech and face expression recognition, for example, could boost the communication capabilities provided by the proposed recognizer, since recognition of sequential or more complex gestures would be feasible. In the same context, one can envisage its operation while personalizing a person's habits. In other words, distinct (personal)

ways of performing gestures may be recorded and exploited during recognition, while other modalities (e.g. face recognition) are employed to identify the actor.

It is anticipated that systems based on the above mentioned technologies will gradually come into play in experimental interaction setups. Still, the advent of robust and seamless gesture recognizers in every day applications will be commensurate on advances in technological factors, as the limiting ones mentioned above.

Bibliography

1. Ahmad, S., *A usable real-time 3D hand tracker.* Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on. **2**.
2. Bolic, M. *Assistant Professor, School of Information Technology and Engineering, University of Ottawa*. Available from: http://www.site.uottawa.ca/~mbolic/.
3. Bouguet, J.-Y., *Complete Camera Calibration Toolbox for Matlab®*.
4. Craig, J., *Introduction to robotics*: Addison Wesley.
5. Darken, C., J. Chang, and J. Moody. *Learning rate schedules for faster stochastic gradient search*.
6. Elert, G. *The Physics Factbook™*. Encuclopedia of Scientific Essays]. Available from: http://hypertextbook.com/facts/.
7. Siganos, C.S.a.D. *Neural Networks*. Available from: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.
8. STURMAN, D. *J., ZELTER, D., AND PIEPER, S*.
9. Denavit, J. and R. Hartenberg, *A kinematic notation for lower-pair mechanisms based on matrices.* Journal of Applied Mechanics, 1955. **22**(2): p. 215?221.
10. Thieffry, S., *Hand Gestures.* The Hand, 1981: p. 488-492.
11. Titterington, D., *Recursive Parameter Estimation Using Incomplete Data.* 1982.
12. Korein, J., *A geometric investigation of reach*. 1985: MIT Press Cambridge, MA, USA.
13. Kendon, A., *Current issues in the study of gesture.* The Biological Foundations of Gestures: Motor and Semiotic Aspects, 1986: p. 23-47.
14. Rumelhart, D., G. Hinton, and R. Williams, *Learning internal representations by error propagation, Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. 1986, MIT Press, Cambridge, MA.
15. Zimmerman, T., et al., *A hand gesture interface device.* ACM SIGCHI Bulletin, 1986. **17**: p. 189-192.
16. Solla, S., E. Levin, and M. Fleisher, *Accelerated learning in layered neural networks.* Complex Systems, 1988. **2**(6): p. 625-639.
17. Craig, J., *Introduction to Robotics: Mechanics and Control*. 1989: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

18. Le Cun, Y., *Generalization and network design strategies.* Connectionism in Perspective, 1989: p. 143-155.
19. Eglowstein, H., *Reach Out and Touch Your Data.* Byte, July, 1990. **15**: p. 283-290.
20. Guo, H. and S. Gelfand, *Analysis of gradient descent learning algorithms for multilayerfeedforward neural networks.* Circuits and Systems, IEEE Transactions on, 1991. **38**(8): p. 883-894.
21. Lowe, D., *Fitting parameterized three-dimensional models to images.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1991. **13**(5): p. 441-450.
22. Murakami, K. and H. Taguchi, *Gesture recognition using recurrent neural networks.* Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology, 1991: p. 237-242.
23. Turk, M. and A. Pentland, *Eigenfaces for Recognition.* Journal of Cognitive Neuroscience, 1991. **3**(1): p. 71-86.
24. Hauptmann, A. and P. McAvinney, *Gestures with speech for graphic manipulation.* International Journal of Man-Machine Studies, 1993. **38**(2): p. 231-249.
25. Lee, J. and T. Kunii, *Constraint-based hand animation.* Models and Techniques in Computer Animation, 1993: p. 110–127.
26. Rehg, J. and T. Kanade, *DigitEyes: Vision-Based Human Hand Tracking.* 1993.
27. Segen, J., *Controlling Computers with Gloveless Gestures.* Proceedings of Virtual Reality Systems, 1993.
28. Watson, R., *A Survey of Gesture Recognition Techniques.* 1993.
29. Annema, A., K. Hoen, and H. Wallinga, *Learning behavior and temporary minima of two-layer neural networks.* Neural Networks, 1994. **7**(9): p. 1387-1404.
30. Cho, K. and S. Dunn, *Learning shape classes.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1994. **16**(9): p. 882-888.
31. Davis, J. and M. Shah, *Gesture recognition.* Proc. European Conf. Comp. Vis, 1994: p. 331–340.
32. Quek, F., *Toward a Vision-Based Hand Gesture Interface.* Virtual Reality Software & Technology: Proceedings of the VRST'94 Conference, 23-26 August 1994, Singapore, 1994.
33. Rehg, J. and T. Kanade, *Visual tracking of high DOF articulated structures: an application to human hand tracking.* Proc. European Conference on Computer Vision, 1994. **2**: p. 35–46.
34. Brockl-Fox, U., *Realtime 3-D Interaction with up to 16 Degrees of Freedom from Monocular Video Image Flows.* Proc. of Int. Workshop on Automatic Face and Gesture Recognition, 1995: p. 172–178.
35. Cui, Y. and J. Weng, *Learning-based hand sign recognition.* Proc. of the Intl. Workshop on Automatic Face-and Gesture-Recognition, 1995.
36. Freeman, W. and C. Weissman. *Television control by hand gestures*. 1995.
37. Huang, T. and V. Pavlovic, *Hand gesture modeling, analysis, and synthesis.* Proc. 1995 IEEE International Workshop on Automatic Face and Gesture Recognition, 1995: p. 73-79.

38. Kadous, W., *GRASP: Recognition of Australian sign language using Instrumented gloves.* Unpublished manuscript, University of New South Wales, Sydney, Australia. Retrieved October, 1995. **1**: p. 2002.
39. Lee, J. and T. Kunii, *Model-based analysis of hand posture.* Computer Graphics and Applications, IEEE, 1995. **15**(5): p. 77-86.
40. Moghaddam, B. and A. Pentland, *Maximum Likelihood Detection of Faces and Hands.* International Workshop on Automatic Face-and Gesture-Recognition, 1995: p. 122–128.
41. Quek, F., *Eyes in the interface.* Image and Vision Computing, 1995. **13**(6): p. 511-525.
42. Quek, F., T. Mysliwiec, and M. Zhao, *FingerMouse: A Freehand Computer Pointing Interface.* Proc. of Int'l Conf. on Automatic Face and Gesture Recognition, 1995: p. 372-377.
43. Boulic, R., S. Rezzonico, and D. Thalmann. *Multi-Finger Manipulation of Virtual Objects*. 1996.
44. Bryson, S., *Virtual reality in scientific visualization.* Communications of the ACM, 1996. **39**(5): p. 62-71.
45. Kiyokawa, K., et al. *VLEGO: A Simple Two-handed Modeling Environment Based on Toy Blocks*. 1996.
46. Oviatt, S. and R. VanGent. *Error resolution during multimodal human-computer interaction*. 1996.
47. Quek, F., *Unencumbered gestural interaction.* Multimedia, IEEE, 1996. **3**(4): p. 36-47.
48. Pavlovic, V., R. Sharma, and T. Huang, *Visual interpretation of hand gestures for human-computerinteraction: a review.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1997. **19**(7): p. 677-695.
49. Shepherd, A., *Second-Order Methods for Neural Networks*. 1997: Springer-Verlag New York, Inc. Secaucus, NJ, USA.
50. Starner, T. and A. Pentland, *Real-Time American Sign Language Recognition from Video Using Hidden Markov Models.* COMPUTATIONAL IMAGING AND VISION, 1997. **9**: p. 227-244.
51. Berry, G., V. Pavlovic, and T. Huang. *BattleView: A multimodal HCI research application*. 1998.
52. Black, M. and A. Jepson, *EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation.* International Journal of Computer Vision, 1998. **26**(1): p. 63-84.
53. Black, M. and A. Jepson, *Recognizing temporal trajectories using the condensation algorithm.* Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on, 1998: p. 16-21.
54. Cutler, R. and M. Turk, *View-based interpretation of real-time optical flow for gesturerecognition.* Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on, 1998: p. 416-421.
55. Imagawa, K., S. Lu, and S. Igi, *Color-Based Hands Tracking System for Sign Language Recognition.* Proceedings of the 3rd. International Conference on Face & Gesture Recognition, 1998: p. 462.
56. Starner, T., J. Weaver, and A. Pentland, *Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video.* IEEE

TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 1998: p. 1371-1375.
57. Trucco, E. and A. Verri, *Introductory Techniques for 3-D Computer Vision*. 1998: Prentice Hall PTR Upper Saddle River, NJ, USA.
58. Akamatsu, S., *Computer recognition of human face - A survey.* Systems and Computers in Japan, 1999. **30**(10): p. 76-89.
59. Becker, M., et al., *GripSee: A Gesture-Controlled Robot for Object Perception and Manipulation.* Autonomous Robots, 1999. **6**(2): p. 203-221.
60. LaViola Jr, J., *A Survey of Hand Posture and Gesture Recognition Techniques and Technology*. 1999.
61. Rittscher, J. and A. Blake, *Classification of human body motion.* Proc. Int. Conf. Computer Vision, 1999: p. 634–639.
62. Stauffer, C. and W. Grimson. *Adaptive background mixture models for real-time tracking*. 1999.
63. Wu, Y. and T. Huang, *Human hand modeling, analysis and animation in the context of HCI.* Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on, 1999. **3**.
64. Wu, Y. and T. Huang, *Vision-Based Gesture Recognition: A Review.* Urbana, 1999.
65. *Wikipedia The Free Encyclopedia*. 2001; Available from: http://www.wikipedia.org/.
66. Wu, Y., J. Lin, and T. Huang, *Capturing natural hand articulation.* International Conference on Computer Vision, 2001: p. 426-432.
67. Rogalla, O., et al. *Using Gesture and Speech Control for Command a Robot Assistant*. 2002.
68. Yang, M., N. Ahuja, and M. Tabb, *Extraction of 2D Motion Trajectories and Its Application to Hand Gesture Recognition.* IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 2002: p. 1061-1074.
69. Brewster, S., et al., *Multimodal 'eyes-free' interaction techniques for wearable devices*, in *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2003, ACM: Ft. Lauderdale, Florida, USA.
70. Residents. *Second Life*. 2003; Available from: http://secondlife.com/.
71. Zhao, W., et al., *Face recognition: A literature survey.* ACM Computing Surveys (CSUR), 2003. **35**(4): p. 399-458.
72. Argyros, A. and M. Lourakis, *Real-Time Tracking of Multiple Skin-Colored Objects with a Possibly Moving Camera.* LECTURE NOTES IN COMPUTER SCIENCE, 2004: p. 368-379.
73. Derpanis, K., *A Review of Vision-Based Hand Gestures.* Unpublished. Feb, 2004.
74. Thrun, S., W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. 2005: MIT press, Cambridge, Massachusetts, USA.
75. Zeleznik, R., K. Herndon, and J. Hughes. *SKETCH: an interface for sketching 3D scenes*. 2006: ACM New York, NY, USA.
76. Zivkovic, Z. and F. van der Heijden, *Efficient adaptive density estimation per image pixel for the task of background subtraction.* Pattern Recognition Letters, 2006. **27**(7): p. 773-780.

77. Badler, N. and D. Tolani, *Real-Time Inverse Kinematics of the Human Arm.* 2007.
78. Mitra, S. and T. Acharya, *Gesture Recognition: A Survey.* Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2007. **37**(3): p. 311-324.
79. Tsetserukou, D., et al., *Development of a Whole-Sensitive Teleoperated Robot Arm using Torque Sensing Technique.* Proceedings of the Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2007: p. 476-481.
80. Baltzakis, H., et al., *Tracking of Human Hands and Faces through Probabilistic Fusion of Multiple Visual Cues.* LECTURE NOTES IN COMPUTER SCIENCE, 2008. **5008**: p. 33.
81. Dipietro, L., A. Sabatini, and P. Dario, *A Survey of Glove-Based Systems and Their Applications.* Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2008. **38**(4): p. 461-482.
82. Google. *Google Lively*. 2008; Available from: http://www.lively.com/.