Optimal Hand Gesture Vocabulary Design Methodology for Virtual Robotic Control

Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

by

Juan Wachs

Submitted to the Senate of
Ben-Gurion University of the Negev

October 2006

BEER - SHEVA

Optimal Hand Gesture Vocabulary Design Methodology for Virtual Robotic Control

Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

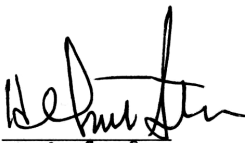
by

Juan Wachs

Submitted to the Senate of
Ben-Gurion University of the Negev

Approved by the advisors: Prof. Helman Stern

Prof. Yael Edan



Approved by the Dean of The Kreitman School of Advanced Graduate Studies _____

October 2006

BEER - SHEVA

This work was carried out under the supervision of

Prof. Helman Stern

Prof. Yael Edan

In the Department of Industrial Engineering and Management

Faculty of Engineering Sciences

For my companion, my friend and my true love, Elizabeth

Acknowledgments

To my advisors, **Prof. Helman Stern and Prof. Yael Edan**: thank you for opening my eyes to the magnificence of science and for being the most human scientists that I have ever met. Thank you for your guidance and support that proved so significant for my work.

Special thanks to **Yoash Hasidim, Shahar Laykin, Peter Bak and Uri Kartoun,** my good friends, with whom I shared great years of discussions, ideas, tennis matches and lots of laughter.

To **Uri Kartoun:** thank you for the help implementing the telerobotic hand gesture system.

To the Open Source Computer Vision Library (OpenCV) generous community, which collaborated with code and discussions throughout this thesis, and specifically to **Mr. Eric Taillard** for the implementation code for the simulated annealing approach for the QAP, thanks.

I would like to thank **Prof. Joachim Meyer and Peter Bak** for their help in developing the experimental evaluation methods, **Dr. Ofer Levi** for helping me to develop the regression procedures and **Prof. Israel David** for his comments, which helped me to complete this thesis.

I want to extend thanks to the members of my Ph.D. committee, **Prof. Joachim Meyer and Prof. Miriam Zacksenhouse** for their comments, which helped improve my research.

This project was partially supported by the Paul Ivanier Center for Robotics Research & Production Management, Ben-Gurion University of the Negev.

Thanks to **Nissim Abuhazira and Yossi Zahavi,** who have worked with me in the Multimedia lab during these wonderful years.

I'd like to thank my family, **Fernando, Samuel** and **Noga Wachs**. They have always provided me with support, encouragement, and hope.

To **Pablo** and **Isabel Kantor** to give confidence on me from the beginning of my Phd studies.

I would like to thank my partner **Elizabeth**, for her patience, unlimited love and sense of humor which were fundamental for me to finish this thesis.

Finally, thanks to **Norah Tabany**, my mother, for showing me the beauty of knowledge and teaching me to believe in my way.

Juan P. Wachs
Ben-Gurion University of the Negev
Beer Sheva, 2006

Table of Contents

List of Figures

List of Tables

List of Appendices

Nomenclature

A	Recognition accuracy
$\mathcal{A}()$	Recognition accuracy function
A_{min}	Minimum recognition accuracy allowed
C	Command Set
CW	Clockwise
CCW	Counter clockwise
$\mathcal{C}_m$	Confusion matrix obtained using the master set of gestures
$\mathcal{C}_n$	Confusion matrix obtained using the G_n gesture subset
CD	Confusion Derived Rutine
CMD	Confusion Matrix Derived Method
CNS	Complete Neighborhood Search
D	Command Duration Matrix
DCM	Disruptive Confusion Matrix Method
DPE	Dual Pair Exchange
EMG	Electromyography
F	Command Transition Matrix
FCM	Fuzzy C-Means algorithm
G	A gesture
G_m	Master set of gestures
G_n	Subset of gestures
G_{HA}	Subset of gestures associated to high accuracy
G_{LA}	Subset of gestures associated to low accuracy
G_z	Large master set of gestures
GV	Gesture Vocabulary
GV*	Optimal hand Gesture Vocabulary
GV_G	Gesture vocabulary obtained from V_G
GV_B	Gesture vocabulary obtained from V_B
I	Direct Intuitiveness matrix
IC	Complementary Intuitiveness matrix
MOP	Multicriteria optimization problem
PNS	Probabilistic Neighborhood Search
QAP	Quadratic Assignment Problem
S	Fatigue matrix
T	Task set
τ	Task completion time
U	Comfort matrix
V	Intuiteveness set
V_G	Good gesture vocabulary set
V_B	Bad gesture vocabulary Set
VMR	Virtual mobile robotic arm
$Z_1(GV)$	Inuitiveness performance measure
$Z_2(GV)$	Comfort performance measure
$Z_3(GV)$	Recognition accuracy performance measure

Abstract

Gesture-based interfaces offer an alternative to traditional teach-pendants, menu, and direct manipulation interfaces. Gesture interfaces can be seen as an alternative to existing interface techniques, offering advantages such as natural interaction and sterile, fast responses. In this thesis we consider only gestures using the hand. The main element in developing a hand gesture interface is the selection of the hand gestures (or postures) themselves. Unfortunately, hand gesture vocabulary (GV) design procedures for human machine interaction have not been extensively researched. The main objectives of this thesis is to formulate the optimal hand GV design problem in a rigorous manner and to develop and validate a solution methodology using mathematical programming, heuristics, image processing and methods to estimate human psycho-physical measures.

The following three factors are currently considered the most important issues affecting the performance of human-machine hand GV design: **1. Fatigue (or Comfort)**: The design of a vocabulary must avoid gestures that require extended, intense muscle tension over long periods of time. A successful procedure will promote natural postures and discourage those that cause excessive muscle strain from repetition. Two types of stress were identifies in this thesis: a) static stress, which is the effort required to hold a static gesture for a defined amount of time, and b) dynamic stress, which is the effort necessary for performing a transition between static gestures. **2. Intuitiveness**: Intuitiveness is the cognitive naturalness of associating a gesture with a command or intent. Intuitiveness is associated with learnability and memorability. Two types of intuitiveness are introduced in this thesis: direct intuitiveness, which is related to the cognitive association between a gesture and a command, and complementary intuitiveness, which concerns the use of complementary gestures to represent complementary commands. **3. Recognition Accuracy**: Recognition accuracy describes the percent of gestures that are classified correctly. To determine the recognition accuracy of a GV, a hand gesture recognition algorithm was developed.

An optimal hand GV is defined as a set of gesture-command pairs that will minimize the time τ required for a user to perform a task T (or tasks). Since the task completion time, as a function of GV, has no known analytical form, three different performance measures are proposed as proxies: intuitiveness $Z_1(GV)$, comfort $Z_2(GV)$, and recognition accuracy $Z_3(GV)$. Maximizing all the objectives simultaneously determines a multiobjective optimization problem (MOP) which can be solved by allowing the decision maker to select the GV from a Pareto frontier according to his own preferences. The Pareto frontier is the set of non-dominated solutions, the solution set of which can be determined through enumeration. However, for even reasonably sized vocabularies, such an enumeration approach is untenable. Hence, two alternative formulations to this problem are presented: a) the three performance measures are mapped into a single measure by using weights w_i to reflect the relative importance of each of the objectives and b) a dual priority objective is used, in which accuracy is the first priority and the human performance objectives are secondary. The optimal hand GV methodology architecture comprises: a) a determination of the human psycho-physiological input factors, b) a search for a feasible gesture subset, each gesture of which satisfies a minimum level of recognition accuracy, and c) a command - gesture matching procedure.

The subjective measures were obtained through a series of experiments by studying human subject responses. The first experiment involved finding intuitive gestures to control a robotic arm and a VMR. The selection of gestures obeyed a 70/30 rule, where 70% of subjects used only 30% of the gestures in a vocabulary. For the stress measure, an ergonomic experiment was conducted which consisted of ranking hand gestures, by the user, from weak to strong on the

Borg scale [Borg, 1982]. Based on the static stress measures and a few measures for transition stress, a model was developed and validated. This model affirms that 90% of the dynamic stress (and its duration) was determined by the final posture in the transition between two postures and only the remaining 10% by the starting posture.

To validate the approach presented in this dissertation, two sets of GVs were created: a) a set of vocabularies that is highly intuitive, comfortable, and easy to recognize (V_G), and b) a set of low intuitive, stressful, and hard to recognize vocabularies (V_B). It was found that when comparing both sets that the use of V_G vocabulary samples resulted in shorter task completion times, higher learning rates, and higher memorability. Therefore, GVs with higher values of the 3 objectives promote decreased performance time, faster learning, and increased memory.

To conclude, a methodology for the design of natural hand gesture vocabularies that considered both the psycho-physiological and the technical aspects in a unified approach was presented. Hence, the main contribution is a rigorous mathematical formulation in which optimization methods are applied, constraints are defined, and the quality of the solution is measured for the GV design problem. There are several advantages to this approach. First, it makes it possible to obtain highly ergonomic and recognizable hand GVs using a rigorous procedure. Second, it offers a data repository of intuitiveness and comfort measures, and an automated methodology for their collection. This approach produces improved task-oriented hand GVs. The framework that is developed makes an important contribution to the development of hand gesture recognition systems for human-robot interaction.

Keywords: hand gesture vocabulary design, machine vision, fuzzy c-means, feature selection, image processing, hand gesture recognition, human-computer interaction, robotic control, human factors, gestures intuitiveness, hand stress

Executive summary

Gesture-based interfaces offer an alternative to traditional teach-pendants, menu, and direct manipulation interfaces. In the context of human-machine interaction, the ability to specify objects, operations, and navigation commands with a single, intuitive gesture appeals to both novice and experienced users. As an example, think about how intuitive it is to tell a robot to "pick" up an object by mimicking a "grasping" gesture, or to select a word by "pointing" at it. Gesture interfaces represent an alternative to existing interface techniques, offering advantages such as natural interaction and sterile, fast responses. One significant contribution gesture interfaces have already made is to provide assistance for people with physical disabilities to access computers and other physical devices. The main problem or challenge in designing a productive hand gesture interface is the selection of the hand gestures (or postures) used to complete task(s). Unfortunately, hand gesture vocabulary (GV) design procedures for human machine interaction have not been extensively researched. The design of a hand GV involves a formidable optimization problem in a large search space and should be based on both human usability and machine recognition factors. The following three factors are considered the most important issues affecting the performance of human-machine hand GV design.

1. Fatigue (or Comfort): Gestural communication involves more muscles than keyboard and mouse based interaction or speech. The wrist, fingers, hand, and arm all contribute to the expression of commands. Gestures must therefore be concise and comfortable while minimizing effort in the hand and arm as a whole. In particular, the design of a vocabulary must avoid gestures that require high levels of muscle tension over long periods of time. Awkward, repetitive postures can strain tissues and cause pressure within the carpal tunnel. A successful procedure will encourage natural postures and discourage those that aggravate the strain of repetition. Two types of stress were identified in this thesis: a) static stress is the effort needed to hold a static gesture for a defined amount of time, and b) dynamic stress refers to the effort necessary to perform a transition between static gestures. A fatigue matrix, S, was created to hold information regarding the stress indices of the gestures used in the current methodology. The comfort matrix U is an inverse function of S. In this study stress is determined based on wrist, finger, and hand positions.

2. Intuitiveness: Intuitiveness is the cognitive naturalness of associating a gesture with a command or intent. This is unrelated to the limitations imposed by hand anatomy. Complex or unnatural gestures are rarely remembered by the user when applied. The gesture should be easy to recall even if it has no cognitively associated action. Intuitiveness is associated with learnability and memorability. Other factors that affect the user's preferred set of gestures are general knowledge, cultural background, and linguistic capabilities. Two types of intuitiveness are presented in this thesis: a) direct intuitiveness is related to the cognitive association between a gesture and a command, and b) complementary intuitiveness is related to the use of complementary gestures to represent complementary commands. The direct intuitiveness matrix I stores information about the direct intuitiveness of the framework. Information about complementary intuitiveness is contained in the matrix of complementary intuitive indices, IC. The intuitiveness V is the set {I, IC}.

3. Recognition accuracy: Recognition accuracy expresses the percent of accepted gestures that are correctly classified. Hand gesture recognition is a very difficult vision task that involves assumptions regarding uniform/complex background, static-dynamic gestures and skin color models. Position, orientation, and finger-palm configuration can be used to emphasize the differences between the gestures, thus yielding a high level of discrimination between them.

Image processing and robust recognition algorithms are crucial factors for classifying hand gestures. To determine the recognition accuracy (A) of a GV, a hand gesture recognition algorithm was developed.

The first two factors, fatigue and intuitiveness, are human-centered while the third factor, accuracy, depends on machine properties (e.g., hardware, software). This thesis deals with the optimal design of a hand GV that addresses the need to improve the user's control experience (intuitiveness and comfort) without affecting the technical aspect (recognition accuracy). The three factors will be used to guide the design of an optimal hand GV.

The main objectives of this thesis are to formulate the optimal hand GV design problem in a rigorous manner, to develop and validate a solution methodology using mathematical programming, heuristics, image processing, and methods to estimate human psycho-physical measures.

Methodology

An optimal hand GV is defined as a set of gesture-command pairs that will minimize the time τ required for a user to perform a task T (or tasks). The number of commands (C) is determined by the task, while the set of gestures is selected from a large set of hand postures, G_z. The performance of the task depends on the recognition accuracy of the subset of gestures (G_n) on human factor measures representing the naturalness of the gesture-command associations and the comfort of the postures.

Definition of the problem and possible solutions

The main problem is to minimize task performance time over a set of all feasible GVs. Since the task completion time, as a function of GV, has no known analytical form, three different performance measures are proposed as proxies: intuitiveness Z_1(GV), comfort Z_2(GV), and recognition accuracy Z_3(GV). Maximizing all the objectives simultaneously determines a multiobjective optimization problem (MOP) that can be solved by allowing the decision maker to select the GV from a Pareto frontier according to his own preferences. The Pareto frontier is the set of non-dominated solutions, the solution set of which can be determined through enumeration. However, for even reasonably sized vocabularies, the enumeration approach is untenable.

Two alternative formulations to this problem are presented: a) three performance measures are mapped into a single measure by using weights w_i to reflect the relative importance of each of the objectives, and b) a dual priority objective is used where accuracy is the first priority and the human performance objectives are secondary.

Architecture

The architecture of the optimal hand GV methodology comprises three serial modules. In Module 1, human psycho-physiological input factors are determined. In Module 2, a search for a feasible gesture subset, subject to machine gesture recognition accuracy, is carried out. Module 3 constitutes a command-gesture matching procedure.

The task set T, the large gesture master set G_z, and the set of commands C are the input parameters for Module 1. The union of all commands used to perform all tasks T constitutes C. The objectives of Module 1 are to establish associations between commands and gestures based on user intuitiveness (direct and complementary), to find the comfort matrix based on command transitions and fatigue measures, and to reduce the large set of gestures to the master set G_m. For Module 2, the necessary inputs are the master set of gestures G_m, and a recognition algorithm to determine the recognition accuracy, A. Module 2 employs an iterative search procedure to find a single feasible gesture subset G_{n^*} (or alternatively, the set of feasible gesture subsets), satisfying a given accuracy level defined by the decision maker. Two metaheuristic approaches were developed for the search procedure. The first approach is referred to as the Disruptive Confusion

Matrix (DCM), and the second is referred to as the Confusion Matrix Derived Solution (CMD). In addition, a case of partial enumeration was demonstrated.

A reconfigurable FCM supervised algorithm was used to obtain the recognition accuracy, A. The parameters of the image processing and clustering algorithm were simultaneously found using neighborhood parameter search routines. Two versions of a local neighborhood search algorithm were designed. These versions were customized for an operational parameter calibration task system, where the number of parameters in the solution vector was dynamically changed. To determine the accuracy of a candidate subset of gestures, it was necessary to train a classifier. Two different approaches were used: one involved retraining the FCM many times for each different candidate G_n, and a second in which the FCM is trained and tuned once for the master set G_m from which the accuracy of candidate G_ns is derived. This second method is an approximate method, but it is very fast.

The inputs to the third module are the matrices intuitiveness V={I,IC}, comfort U, commands C, and the subset of gestures G_{n*}. The goal of this module is to match the set of gestures G_n with the set of given commands C, such that the human measures are maximized. The integer QAP solved the problem of matching gestures to commands. The resulting gesture-command assignment constitutes the GV.

Experiments, analysis and results

The subjective measures were obtained through a series of experiments by studying human subject responses. The first experiment involved finding intuitive gestures to control a robotic arm and a VMR. To collect intuitive data, a sequence of commands (from a robotic arm and a VMR predefined task) was presented to the user, and the user freely associated gestures with these commands. The actual acquisition of gesture responses was done when the subject physically generated a gesture and entered its configuration information. The selection of gestures obeyed a 70/30 rule, where 70% of subjects used only 30% of the gestures in a vocabulary. This refutes the claim that subjects consistently use the same gestures to represent the same commands while performing tasks, as suggested by Hauptmann [Hauptmann and McAvinney, 1993].

For the stress measure, an ergonomic experiment was conducted which consisted of ranking hand gestures, by the user, from weak to strong on the Borg scale [Borg, 1982]. Based on the static stress measures for all the gestures in the master set G_m, and a few measures for the transition stress, a model that describes the transition effort was developed and validated. This model affirms that 90% of the dynamic stress (and its duration) was determined by the final posture in the transition between two postures, and the remaining 10% is decided by the starting posture. Using this relation, prediction of the dynamic stress and its duration is based on the use of only static stress measures. This prediction model saved 197 hours of subject experiments.

To validate the model, two sets of GVs were created: V_G is a set of vocabularies that is highly intuitive, comfortable, and easy to recognize, and V_B is a set of low intuitive, stressful, and hard to recognize vocabularies. GV_G and GV_B are vocabulary samples from V_G and V_B, respectively. Validation of the analytical procedures for finding the optimal hand GVs consisted of testing the following hypotheses: (a) H_1: Min $\tau(GV^*) \propto \max(Z_1)$, $\max(Z_2)$ and $\max(Z_3)$ – task performance time τ can be represented by multiobjective proxy measures. Moreover, the maximization of the multiobjective function causes a minimization in the performance time of the task. (b) H_2: $\tau(GV_G) < \tau(GV_B)$ – using GV_G results in shorter task completion time than for GV_B. (c) H_3: $m(GV_G) > m(GV_B)$ – GV_G is easier to remember than GV_B.

To test the first two hypotheses (H_1, H_2), a t-test was performed between standard completion times for 8 V_G and 8 V_B vocabularies for both a robotic arm and a VMR task. The mean completion time for the tasks using V_G was much shorter than the time using V_B ($\tau(GV_G)$ =87.98

sec < $\tau(GV_B)$=118.95 sec with p=0.0059) and ($\tau(GV_G)$=114.67 sec < $\tau(GV_B)$=153.04 sec with p=0.00031), for the robotic arm and the VMR tasks, respectively. The learning time was expressed in terms of the learning rate of the user's learning curve in the use of certain GV when performing a task. It was found that for V_G the learning rate was lower than for V_B (the robotic arm task 0.785<0.797 and the VMR task 0.827<0.835), which indicate faster learning. The last hypothesis (H_3), suggested that the GV_G is easier to remember than GV_B. Memorability was determined by an experienced user's recall of the gesture-command associations. The average memorability scores for the robotic task using the V_G were higher than when using the V_B (87.5 and 70.83% with p=0.05); however, there was no significant difference in memorability for the VMR task. These results can be restated as: GVs with high values for the 3 performance measures resulted in decreased performance times, faster learning, and increased memory.

Conclusions

This thesis presents a methodology that addresses both psycho-physiological (intuitiveness and comfort) and technical (recognition accuracy) aspects for the design of natural hand gesture vocabularies and that combines both aspects in a unified approach. The main contributions of this research are:

Analytical formulation of the GV design problem: a methodology was developed to find an optimal hand GV using an analytical approach. The main goal of this methodology is to avoid the arbitrary selection of hand gestures when designing human-robotic applications for given tasks and commands. The contribution is a rigorous mathematic formulation in which optimization methods are applied, constraints are defined, and the quality of the solution is measured.

Reconfigurable hand gesture recognition algorithm: the difficult problem of simultaneous calibration of the parameters of an Image Processing - Fuzzy C Means (FCM) hand gesture recognition system was addressed, and an approach to automate the calibration of the parameters was proposed. The hand gesture recognition system design is transferred to an optimization problem.

Two solution methods for solving the GV design problem: two solution methods were developed to solve the optimal design problem: a) a multiobjective decision approach and b) a two stage decomposition procedure. For the first problem, an approximate enumeration of the solutions is performed, and a subset of non-dominated solutions is selected for presentation to the decision maker. The two stage decomposition method is a dual objective problem, where the maximum accuracy objective and human centered objectives (intuitiveness and comfort) are given as first and second priorities, respectively.

Development of intuitiveness and comfort gestural indices, and an automated method for their collection: contributions regarding human psycho-physical factors, comfort, and intuitiveness were introduced in this research. Experiments were developed to find the level of the user's cognitive association (intuitiveness) between command-gesture pairs based on simulating different scenarios and studying how the user decides about the most natural associations between commands and gestures. With respect to intuitiveness, the selection of gestures follows a 70/30 rule, where 70% of subjects use 30% of the gestures in a vocabulary. A complementary intuitiveness measure was defined as the cognitive association between a pair of complementary commands (such us: up-down) and a complementary pair of gestures (such as: thumb up-thumb down). In addition, two types of stress were identified: a) static, and b) dynamic. A model was developed to predict the level and duration of dynamic stress based on static stress measures.

Validation and usability results: GVs with high values of intuitiveness, comfort, and accuracy resulted in shorter task completion times, faster learning, and increased memory.

This thesis is in part based on the following publications:

Journal papers

1. Wachs J., Stern H. and Edan Y. 2005. Cluster labeling and parameter estimation for the automated set up of a hand-gesture recognition system, IEEE Transactions on Systems, Man and Cybernetics, Part A, 35(6): 932-944.

Reviewed conference papers

1. Wachs J., Kartoun U. Edan Y. and Stern H. 2002. Real-time hand gesture telerobotic system using the fuzzy C-means clustering algorithm, Proceedings of the 5th Biannual World Automation Congress, WAC 2002, Orlando, Florida, USA, 13:403–409.
2. Wachs J., Stern H. and Edan Y. 2003. Parameter search for an image processing fuzzy C-means hand gesture recognition system". Proceedings of IEEE International Conference on Image Processing ICIP 2003, Spain, 3: 341-346.
3. Stern H., Wachs J., Edan Y. 2004. Hand gesture vocabulary design: a multicriteria optimization, Proceedings of the IEEE International Conference on Systems, Man & Cybernetics. The Hague, Netherlands.
4. Stern H., Wachs J. and Edan Y. 2006. Optimal hand gesture vocabulary design using psycho-physiological and technical factors, 7th International Conference Automatic Face and Gesture Recognition, FG2006, Southampton, UK, April 10-12.
5. Stern H., Wachs J. and Edan Y. 2006. Human factors for design of hand gesture human - machine interaction, 2006 IEEE International Conference on Systems, Man, and Cybernetics, Oct. 8-11, Taipei, Taiwan.
6. Eliav A., Lavie T., Parmet Y., Stern H., Wachs J. and Edan Y. 2005. KISS human-robot interfaces, Presented in the 18th International Conference on Production Research (ICPR), July, Salerno, Italy.
7. Stern H., Wachs J. and Edan Y. 2004. Parameter calibration for reconfiguration of a hand gesture tele-robotic control system. Proceedings of the U.S.A.-Japan Symposium on Flexible Automation, Denver, Colorado

1 Introduction

1.1 Problem description

There is strong evidence that future human-computer interfaces will enable more natural, intuitive communication between people and devices such as computers and robots. Convenient and efficient methods/styles of interaction with real-world devices are possible using speech and gestures [Abowd and Mynatt, 2000; Segen and Kumar, 2000]. Babies use gestures as a basic form of communication to interact with their environments [Acredolo and Goodwyn, 1996]. People also express themselves using gestures, such as body movements, facial expressions, and finger pointing. However, current interface technology rarely adopts such methods when designing human-machine interfaces and, as a consequence, the expressiveness element embedded in the message is lacking [Card *et al*., 1990]. Most human-machine interfaces are based on joysticks, keyboards, and keypads, but few such interfaces use gestures. Interaction based on hand gestures commonly uses two types of interface, gloved-based and vision-based [Pavlovic *et al.*, 1997]. Vision-based interfaces require powerful image processing algorithms to a) segment the hand from stationary background and lighting conditions [Triesch and Malsburg, 1998; Cui and Weng, 1996b], and b) select features to represent gestures [Wren *et al.,* 1997; Campbell *et al.,* 1996] that enhance gesture classification accuracy. A glove based system requires the user to be tethered to the computer, thus reducing user comfort and constraining the user's work space. Even wireless systems currently offered in the market require the user to wear some kind of glove or marker. Furthermore, accurate devices are expensive and hard to calibrate [LaViola, 1999].

Primarily because of such difficulties, this research focused on developing unencumbered, vision-based gestures. Several issues limit the implementation of a vision-based interface for hand gestures: a) gesture recognition is a highly complex problem, b) large variability naturally exists in user performance of gestures, user physical features, and environmental conditions, and c) there is no consensus about which gestures to use and how to map them into functions.

However, the recognition problem is not intractable, and it has been intensively investigated [Pavlovic *et al.,* 1997; Ng and Ranganath, 2002; Gu and Tjahjadi, 2002; Abe *et al.,* 2002; Yin and Xie, 2003]. Differences between individuals while gesturing and those individuals' physical attributes may be overcome by customizing the recognition system for individual users (**user dependent systems**) [Mäntylä, 2001; Takahashi and Kishino, 1991; Burschka *et al*., 2005] or by incorporating multiple gesture samples from different subjects to create user **independent systems** [Rigoll *et al.,* 1997; Parvini and Shahabi, 2005; Alon *et al*., 2005; Just *et al.*, 2004]. The variability in environmental conditions may be solved by using reconfigurable systems [Stern *et al.,* 2004b].

Natural hand gestures for systems control must entail high learnability, usability, ergonomic design, and comfort [Baudel and Beaudouin-Lafon, 1993]. Unfortunately, technical considerations often supersede human centered aspects, thereby causing frustration among the users of such systems. The selection of hand gestures that are easy to learn, impose minimum stress on the user, and that are cognitively natural and easy to implement is still an open research question. There is no rigorous methodology that formally addresses how to obtain and evaluate gestures that are both highly ergonomic and reliable.

An example of intuitive hand GV selection can be found in Pook and Ballard [Pook and Ballard, 1995]. The value of Pook and Ballard's work is that it allows the user to act more

naturally since no cognitive effort is required in mapping function keys to robotic hand actions. This system, like others based in navigation control, implements deictic gestures[i] to make them intuitive. In this context, gestures are created by a static hand or body pose or by physical motion in two or three dimensions, and it can then be translated by a computer into either symbolic commands or trajectory motion commands. Examples of symbolic command gestures are "stop," "start," and "turn."

Many computer systems can be criticized for their idiosyncratic choice of hand gestures or postures to control or direct computer-mediated tasks [Baudel *et al*., 1992]. However, the choice was probably perfectly natural for the developer of the system, which shows the dependence of gestures on their cultural and social environment. Within a society, gestures have standard meanings, but no body motion or gesture has the same meaning in all societies [Birdwhistell, 1970]. Even in American Sign Language (ASL), few signs are so clearly transparent that a non-signer can guess their meanings without additional clues [Klima, 1974]. Furthermore, gestures can be culturally defined to have a specific meaning. Despite the variability of hand gestures across cultures, there are common gestures that are similar for a wide range of cultures. For instance, the most natural way for every person to choose an object to pick is to point at the object; to stop a vehicle, most people open their palm toward the vehicle; to show that everything is "ok", people often close their fists and extend their thumbs upward.

For specialized, frequent tasks, where the learning of a particular set of gestures and postures is worth the investment, such applications may have value. In everyday life, however, it is highly unlikely that users will be interested in a device for which they have to learn some specific set of gestures and postures, unless there is an obvious increase in efficiency or ease of use over existing methods of hand centered input in the adoption of such a gestural protocol. On the other hand, the economics of the marketplace may dictate the adoption of such a set of gestures, independent of its compatibility with existing cultural and/or social standards, just like the keyboard and mouse have set a standard. Especially when users are allowed to expand or create their own sets, such a protocol may gain some acceptance. For a gesture set to gain major acceptance in the market place, it is advisable to examine the tasks and semiotic functions[ii] most frequently executed, and then choose a hand gesture set that seems to appear natural, at least to a number of different people within a social group or even a culture, when executing those tasks and functions.

Hauptmann and McAvinney [Hauptmann and McAvinney, 1993] found that people consistently used the same gestures for specific commands. In particular, they found that people are also very proficient at learning new arbitrary gestures. Gesturing is natural for humans, and minimal training is required before people can consistently use new gestures to communicate information or control devices [Hauptmann, 1989; Harwin, 1990]. Test subjects used very similar gestures for the same operations [Wolf and Morrel-Samuels, 1987]. Hauptmann also found a high degree of similarity in the gesture types used by different people to perform the same manipulations. Test subjects were not coached beforehand, indicating that there may be intuitive, common principles in gesture communication.

There is growing interest in the adaptability of these common principles in gesture based interfaces and in proposing solution methods that result in highly ergonomic and recognizable

[i] Deictic gestures are gestures that contribute to the identification of an object (or a group of objects) by indicating their location.

[ii] Semiotic function is the action of conveying information to the environment (e.g. Drawing, gesturing and verbal expression)

hand gesture vocabularies. Nonetheless, the focus of hand GV design dictated by usability principles is still a virgin area of research. Most research has dealt with the machine aspects of a GV, focusing on recognition algorithms. A GV is defined as a set of matched pairs of verbal commands and their gestural expressions. Current solution methods of GV design may be classified as ad-hoc and rule-based [Baudel and Beaudouin-Lafon, 1993], [Kjeldsen and Hartman, 2001], [Abe *et al.*, 2002]. The primary methods of determining GVs are ad-hoc, whereby an individual constructs the vocabulary, usually with no rationale for the choices made. Few researchers have considered the human psycho-physiological aspects of gesture design. In Nielsen *et al.* [2003], where human factors are considered, limited attention is given to the technical aspects (e.g. automatic recognition of the hand), and the approach stresses human interpretation of the rules. The matching of gestures to commands is done empirically, via user response queries. Inspired by Nielsen, the "Wizard of Oz experiment" is used in Preston *et al.* [2005] to extract common gestures grouped in classes and then convert them into a vocabulary. No details of this procedure are given. Kohler [1997] suggests mapping every gesture to several similar tasks from different devices (for instance, the + and – volume controls for the CD player and for the TV are mapped to the same pair of gestures).This reduces the number of gestures and the mental load. Nevertheless, no methodology is presented by Kohler for this purpose. A hybrid method between the ad-hoc and rule-based approach is presented in Munk [2001], where the set of gestures of interest has been selected in cooperation with a group of linguists. In spite of this choice, Munk is aware that the gesture set should be validated by testing it with a group of arbitrary users with different gesturing styles, ages, and backgrounds. The same usability rules proposed by Nielsen are used in Cabral *et al*. [2005] while emphasizing the importance of performance evaluation of hand GVs. However, Cabral and his group only compared mouse and gesture interfaces, and they did not evaluate the performance of different hand GVs. Argyros and Lourakis [2006] designed 2D and 3D vocabularies based on intuitiveness, ergonomics, and ease of recognition criteria, although the first two factors only included the authors' own considerations.

This thesis deals with the design of natural hand gesture recognition systems entailing ergonomic (user's desires) and technical (recognition accuracy) aspects that are combined in a unified approach. To achieve such a design, an optimal hand GV methodology is extensively developed, tested in true life scenarios, and discussed in this work. The main performance measure used is the completion time to perform a task. However, since the measurement of task completion time is time consuming, we proposed instead to use other performance measures, intuitiveness, comfort, and recognition accuracy, as proxies for this completion time. Thus, the goal of this thesis is to develop a rigorous methodology for the design of GVs that satisfy both human and technical considerations.

1.2 Research objectives

The main objective is to rigorously formulate GV architecture via a methodology for its optimal design. This will include the development of efficient algorithms to find (a) intuitive associations between command-gesture pairs, (b) comfort indices for gesture poses and inter-gesture transitions, (c) fast setup of gesture recognition system, and d) the ability to select a subset of gestures from a large candidate set. The methodology will be implemented in a system flexible enough to handle single task and multi-task environments. Specific objectives include the development of the following:

1. An analytic formulation of the GV design problem
2. A solution method to solve the optimal GV problem using an heuristic, mathematical programming search approach

3. An automated method for calibrating a joint image processing/gesture recognition system
4. Methods to estimate human psycho-physical measures of hand gesture comfort and intuitiveness and to obtain new insight into human gesture selections
5. Validate the use of proxy measures of intuitiveness, comfort, and accuracy for the task performance time

1.3 Research significance

Examination of the literature reveals random and unstructured approaches to hand GV design for human-machine interfaces[iii] [Baudel and Beaudouin-Lafon, 1993; Kjeldsen and Hartman, 2001; Abe *et al.*, 2002; Nielsen *et al.*, 2003; Preston *et al.,* 2005; Kohler, 1997; Munk, 2001], to cite a few. Current solution methods of hand GV design may be classified as ad-hoc and rule-based [Stern *et al.,* 2006]. In this thesis a systematic methodology for GV design has been developed. The two main components considered in the design of a GV are human (intuitiveness and comfort) and technical (recognition accuracy). Reliable and effective human factors are crucial for the success of hand GV design. The measures "intuitiveness" and "comfort" most reflect the cognitive and physiological states, respectively, of the user population. Due to the large number of possible gestures, the corpus of data for intuitiveness and stress measures is prohibitively large, necessitating the use of automated methods to acquire the human factors data. Using gestures that are highly discriminated by the recognition algorithm embedded in a rapid reconfigurable system will reduce the chance of confusion between gestures, and hence fewer errors will occur while performing the task. The primary need for the recalibration of such a system is due to its frequent relocation to other environments, such as laboratories and remote control stations. A secondary need for recalibration, which is reflected in the method used in this thesis, is due to the custom redesign of the gesture control vocabulary. This occurs for new users, new control tasks, and new vocabularies. Allowing for fast recalibration of system parameters provides the system with the flexibility to respond to a new system setup.

Designed according to human and technical factors, the GV is based on a reliable and logical analytical method. Identification of a good GV impacts the performance of the actions involved in the tasks and is accepted by human robot/computer interface users. Task completion time when using a good GV (intuitive and effortless) was shorter than that when using an unnatural GV. A hand GV designed with human factors in mind invites users to adopt it mainly because it is intuitive, but also based on its comfort and the ease with which it is learned and remembered. As robots enter the human environment and come in contact with users, their interaction should be intuitive. Robots will also be expected to interact with people using voice, face, and hand gestures. The keyboard, mouse, and joystick are no longer acceptable as the only input modalities. People communicate with robots using methods as similar as possible to the concise, rich, and diverse means they use to communicate with one another, such as voice-gesture multi-modal interfaces. This work presents a methodology for constructing highly natural and recognizable GVs for virtual robot control.

1.4 Research contribution and innovations

Hand gesture interfaces usually rely on ad-hoc or rule based selection of the gestures to represent a given set of commands. This thesis used the argument that there are underlying factors that determine the naturalness [Hauptmann, 1989] and comfort of gestures. Following

[iii] Hand GV design means how to select gestures and assign them to actions

these principles, a methodology for optimal hand GV design was developed. The usability principles are low fatigue (effortless) and high intuitiveness. The technical principle is based on optimal image processing of hand gesture features to support high gesture recognition rates. A methodology to find the optimal GV for device control was developed based on the gesture, the object, and the task involved. The specific contributions and innovations of this research are as follows:

1. Analytical formulation of the GV design problem: a methodology to find an optimal hand GV using an analytical approach has been developed. The main goal of this methodology is to avoid the arbitrary selection of hand gestures when designing a human-robot application for given tasks and commands. Most of the works that have approached the optimal hand GV have no objective function to evaluate the quality of a GV, and therefore no mathematical formulations were used to obtain a quantitative measure of the solutions proposed. The main contribution is a rigorous mathematical formulation in which optimization methods are applied, constraints are defined, and the quality of the solution is measured. Two aspects drive the need for such a method: (i) GV design research is presently an ad-hoc procedure, and (ii) gesture interfaces are needed to produce more natural, intuitive communication with devices. We believe this is the first conceptualization of the optimal hand GV design problem in analytical form.

2. Reconfigurable hand gesture recognition algorithm: the difficult problem of simultaneous calibration of the parameters of an Image Processing hand gesture recognition system was addressed. More specifically, a fuzzy clustering approach was used to classify the gestures called " the Fuzzy C Means (FCM) clustering algorithm." The approach taken to automate the calibration of the parameters of such a system is that of the local neighborhood search. Thus, the design of a hand gesture recognition system is transferred to an optimization problem. Two versions of the local neighborhood search algorithm involving a complete and probabilistic neighborhood search were developed. This satisfies the need for an automated procedure for such calibrations.

3. Two solution methods for solving the GV design problem: two solution methods were developed to solve the optimal design problem: a) the first is a multiobjective decision approach, and b) the second is based on a two stage decomposition procedure. For the first problem, an approximate complete enumeration of the solutions is performed, and a subset of non-dominated solutions is selected for presentation to the decision maker so that he can make the final selection according to his own desires. This set of non-dominated solutions is called the Pareto frontier. As an exhaustive search is untenable for high complexity problems where the master set of gestures is large and there are a significant number of commands, the second approach was developed. The two stage decomposition method is a dual objective problem where the maximum accuracy objective and human centered objectives (intuitiveness and comfort) are given as first and second priorities, respectively. Optimal matching is performed only with those solutions that have recognition accuracies above a given threshold. The solutions are obtained by building a tree of solutions, in which the gestures are interchanged according to some implicit rules or by using initial solutions obtained from a large classification problem solved in advance.

4. Development of intuitiveness and gestural comfort indices, and an automated method for their collection: important contributions regarding human psycho-physical factors, i.e., comfort and intuitiveness, were introduced in this research. The first contribution is related to the intuitiveness measure, defined as "direct intuitiveness," which is the strength of the cognitive association between a command and its evoking gesture. The selection of gestures respects a 70/30 rule (similar to the 80/20 rule of inventory theory), where 70% of subjects use 30% of the gestures in a vocabulary. It was also found that the overall rate of agreement regarding the use of certain gestures to represent specific commands was in the range of 18-34%. These results

contradict the claim presented by Hauptmann [1989] that users consistently used the same gestures for specific commands. In the experiments designed by Hauptmann, it was shown that users were highly consistent in the type of gestures that they used for commands such as rotation, translation, and scaling. However, our research implies that the mapping between gestures and commands should be based on the particular social-cultural contexts of the users.

A second contribution is the introduction of the complementary intuitiveness measure. This is defined as the cognitive association between a pair of complementary commands (such us up-down) to a complementary pair of gestures (such as thumb up-thumb down). This cognitive aspect reflects the empirical fact that users prefer complementary gestures (i.e., gestures with opposite appearances) to evoke complementary actions (i.e., actions with opposite outcomes).

A third contribution relates to a stress measure. Two types of stress while gesticulating were identified: a) static stress, which is the effort required to hold a static gesture for a defined amount of time, and b) dynamic stress, which is the effort necessary to transition between static gestures. This thesis shows a clear and simple relation between these two kinds of efforts and reveals how it is possible predict the dynamic stress and its duration based on static stress measures.

Specific experiments were developed to identify the level of cognitive association (intuitive index) that the users gave to the command-gesture pairs based on simulating different scenarios and studying how the user decided about the most natural associations between the functions and gestures. A "bottom-up approach" was adopted to obtain intuitiveness indices, which were a result of collecting gestural responses to command stimulants. The static and dynamic effort of performing gestures was measured using a subjective evaluation experiment. Intuitiveness and comfort indices were measured and stored using an application developed specifically for this purpose.

5. Validation and usability results: The following hypotheses were validated:

$$H_1: \text{Min } \tau(GV^*) \propto \max(Z_1), \max(Z_2) \text{ and } \max(Z_3) \quad \textbf{(1.1)}$$

$$H_2: \tau(GV_G) < \tau(GV_B) \quad \textbf{(1.2)}$$

$$H_3: m(GV_G) > m(GV_B) \quad \textbf{(1.3)}$$

Where;

τ = the task performance time
Z_1 = intuitiveness of the GV
Z_2 = the total comfort of the GV
Z_3 = the recognition accuracy of the GV
GV_G = a vocabulary that is highly intuitive
GV_B = a vocabulary that is low intuitive
m = is the memorability

The first hypothesis states that task performance time τ can be represented by multiobjective proxy measures. Moreover, the maximization of the multiobjective function causes a minimization in the task performance time. This was validated as a second hypothesis, which claims that GV_G will result is shorter task completion time than GV_B. GV_G is a vocabulary that is highly intuitive, comfortable, and easy to recognize, while GV_B is a low intuitive, stressful, and hard to recognize vocabulary. Memorability m is the subject of the third hypothesis, which suggests that GV_G is easier to remember than GV_B. Validation of the aforementioned hypotheses was done by comparing two sets of vocabularies, one dominating the other, at a time. Learning time was expressed in terms of the learning rate of the user's learning curve when using a certain

GV. It was determined that GV with high values of the three objectives resulted in faster learning and increased memory. Memorability was determined by the experienced user's recall of the gesture-command associations.

1.5 Thesis structure

The remainder of this dissertation is outlined in the following chapters. Chapter 2 presents the relevant literature review. Issues concerning hand gesture vocabulary taxonomies, design paradigms, examples, and applications are discussed in the context of human-machine interaction. Chapter 3 describes the methods used in this research. Initially the performance measures, definitions, and notations are introduced. Later, three different formulations of the GV problem are discussed, and the main architecture of the methodology is depicted, including the experimental methodology and the validation of the approaches. The problem of interest is formulated using mathematical programming and analytical methods in Chapter 4. Two examples of use are presented as illustrative cases: (DCM) and (CMD) based on the creation and manipulation of a confusion matrix. The quadratic assignment problem is used to model the problem of optimal matching between commands and the feasible subset of gestures. Chapter 5 describes the vision-based algorithms of a hand gesture recognition system. The Image Processing–Fuzzy C-Means (FCM) components of the hand gesture recognition system are described and the calibration of their operational parameters is presented. In Chapter 6 three groups of experiments were performed: (i) determination of human psycho-physiological input factors, (ii) validation of the multiobjective proxy measures for designing good GVs in terms of task performance time, and (iii) usability measures in terms of learning and memorability rates. Statistical tests were performed to determine the significance of these results. Chapter 7 adopts two tasks as case studies: a robotic arm pick and place, and VMR drive tasks. Candidates of hand GVs are obtained, commands and gestures are matched, and the decision maker selects a single GV. The research is summarized in Chapter 8 and future research directions are suggested.

2 Scientific background

2.1 Gestures and human computer interaction

The increasing number of home computers and other sophisticated gadgets cause researchers to think of advanced methods for improving interaction between people and computers [Norman, 1988]. Users felt more comfortable using computer systems (software and hardware) with such designs that gave them a "natural" feeling of communication with their machines [Shneiderman, 1998]. Recently, developers understood the user's physical and mental requirements for interfaces, and these were a crucial variable in the success or failure of any system [Dix *et al*., 1993].

Gestures are a basic form of communication between human beings. Psychological studies show that young children use gestures to communicate before they learn to talk [Acredolo and Goodwyn, 1996]. Rituals, ceremonies, and dances are clear examples of how gestures are deeply embedded in communication between individuals from different cultures [Huang and Pavlovic, 1995]. Manipulation, as a form of gesticulation, is often used when people speak to each other about some object. All these are good reasons to modify or replace the current interface technology comprising classical devices such as the keyboard, mouse, and joystick with a more natural, human centered interface.

Although the mouse is one of the most common and best pointing devices developed until now, it is still not comparable to natural pointing due to limitations of the device itself, such as its planar platform [Card *et al*., 1990].

Human communication finds expression in various modalities, including speech, gestures, and facial and bodily expressions. A variety of expression forms, such as drama, ceremonies, sign language, imitation, music, religious rituals, and dance, exploit the specific capacities of one or more of these modalities. Even though they are expressed through the whole human body, gestures are still related primarily to the human hand. Hand gestures offer an interface modality that includes control through symbolic commands, as for keyboards, and pointing attributes similar to those of the mouse, but in a more flexible, natural, and expressive form. The following discussion focuses on the design issues involved in implementing hand gestures for human-robot interaction.

2.2 Types of gestures and gesture vocabularies

2.2.1 Types of gestures

There are several ways to characterize human hand gestures. From the psycholinguistic point of view, each gesture comprises four aspects: hand shape (configuration), position, orientation, and movement [Stokoe, 1972]. These aspects are useful for feature extraction in machine vision. Another way to characterize hand gestures is according to the temporal behavior [Pavlovic *et al*., 1997]. A gesture with a fixed position, orientation, and configuration over time is called a **static gesture**, or posture. A **dynamic gesture** is a non-fixed gesture exhibiting variation in position, configuration, or orientation over time [Freeman and Roth, 1995]. Hand gestures can also be classified according to their purposes, such as **communicative, control**, **conversational, and manipulative gestures** [Wu and Huang, 1999]. An example of a communicative gesture is sign language, the most popular being the American Sign Language [Starner and Pentland, 1995], which is also used by disabled people to communicate with computer systems. **Control** gestures are used to control real or virtual objects. **Pointing** gestures, for example, would command a robot to pick up an object [Cipolla and Hollinghurst, 1996]. Another control gesture is the **navigation** gesture, where the orientation of the hand can be used like three dimension

directional input to navigate an object in a real or virtual reality environment. **Conversational** gestures are linguistic gestures that happen during conversation and refer back to the content of the speech. They were traditionally assumed to amplify or modulate the information conveyed by speech, and hence, serve a communicative function. **Manipulative** gestures serve as a natural way to interact with virtual objects and robots where, for example, a digitized glove is used [Balaguer and Mangili, 1991]. Manipulative gestures must be associated with manipulative objects such as a screw. **Communicative** gestures are the basic form of human non-verbal interaction and are related to the psychological aspect of communication. **Communicative** gestures can be decomposed into three-motion phases: preparation, stroke and retraction [Kendon, 1986]. Psycholinguistic studies show that stroke is the richer phase in terms of information content; therefore, most systems capture only this phase to be representative of the gesture [Quek, 1994].

2.2.2 Gestures typologies

Several schemes for gesture classification suggested during recent years originated from the scheme proposed by Efron [1941]. While each scheme has it owns advantages and special uses, most of them are interconvertible. This means that the subject can employ all the schemes or start with one and switch to another and cover the same gestural movements. According to Efron, the two basic uses for gesture are spatio-temporal and linguistic. Spatio-temporal gestures represent pure movement, free from any conversational or referential context. These gestures can be categorized according to five aspects: radius (size of the movement), form (shape of the movement), plane (direction and orientation of the movement), the body part that performs it, and tempo (the degree of abruptness vs. flow). Conversely, linguistic gestures happen during conversation and refer to the content of the speech. Efron divides them into two categories: logical-discursive and objective. Logical-discursive gestures emphasize and inflect the content of the conversations that they accompany, either with baton-like indications of time intervals, or ideographic sketches in the air. Objective gestures have meaning independent of the speech that they accompany, and are divided into three categories: deictic, physiographic, and symbolic. Deictic gestures indicate a visually present object, usually by pointing. Physiographic gestures demonstrate something that is not present, either iconographically, by depicting the form of an object, or kinetographically, by depicting an action. Symbolic gestures represent an object by depicting a form that has no actual relationship with the object, but uses a shared, culturally specific meaning [Marrin, 1999].

The McNeill's scheme [McNeill, 1995] classifies gestures in four major categories: iconic, metaphoric, deictic (pointing), and beat gestures. Iconic gestures are gestures that, by using the shape, location, and movement of your hands, imitate some distinctive features of the referent, such as its form, its typical location, the actions performed to it, and those performed by it. For example, a gesture that expresses the referent "guitar" may use its shape, gestures for "hat" its location and size, gestures for "bird" the flying action, and gestures for "espresso coffee" its size. Metaphoric gestures are like iconic gestures in that they are pictorial, but the pictorial content presents an abstract idea rather a concrete object. The gesture presents an image of the invisible, or an abstraction of an image. The gesture depicts a concrete metaphor for a concept. For example, to refer to the genre of drawings and pictures, and not to a specific picture, the subject will make the concept concrete in the form of an image of a bounded object supported in the hands and presented to the listener. Deictic gestures are used when the referent is in the physical context. The most remarkable feature of the referent is its location, and therefore, the most intuitive action is to point at it. Hands or fingers are used to mark in which direction the subject can find the referent. This is one of the early gestures the can be observed in children.

Deictic gestures can be specific, general, or functional. Specific gestures refer to one object. General gestures refer to a class of objects. Functional gestures represent intentions, such as pointing to a dress, when we have the intention to buy it. Deictic gestures are also useful in gesture language representations. Beats are so named because they resemble the beating out of music time. The hand moves along with the rhythmic pulsation of speech. Unlike iconics and metaphorics, beats tend to have the same form regardless of the content. The typical beat is a simple flick of the hand or fingers up and down, or back and forth.

The scheme defined by McNeill has the goal of identifying the referential values of gestures. The orientation of the scheme is toward the entities, actions, spaces, concepts, or relationships to which the gestures refer. The classification scheme thus requires asking what meanings and functions a gesture possesses.

The Nespoulous scheme [Nespoulous, 1986] uses three categories: mimetic, deictic, and arbitrary. In mimetic gestures, the hand and finger motions describe an object's main shape or representative feature [Wundt, 1973]. For example, a waving hand from the nose can be used to represent an elephant by alluding to its fluttering trunk. His definition of deictic gestures is similar to that of McNeill; he also suggests the use of deictic gestures for language representations. Arbitrary gestures are those whose interpretation must be agreed upon and learned due to their opacity. Although they are uncommon in cultural settings, once learned they can be used and understood without any complementary verbal clue. An example is the set of gestures used for training the proper operation of cranes [Link-Belt, 1987]. Arbitrary gestures are useful because they can be specifically created for use in device control. These gesture types are already arbitrarily defined, learned, and understood without any additional verbal information.

A scheme seemingly more appropriate for human machine interfaces (HCI) was developed by Quek [1994] and slightly modified by Pavlovic et al. [1997]. A first classification divides hand/arm movements into two main classes: gestures and unintentional movements. Unintentional movements are those movements that do not express any meaningful information. Gestures are further classified according to whether they are communicative or manipulative. Manipulative gestures are those used to affect objects in an environment (object movement, rotation, translation). Communicative gestures have an intrinsic communicational purpose. Communicative gestures are usually accompanied by speech, and can be presented by acts or symbols. Symbols are gestures that have a linguistic role. They indicate some referent action (for example, the circular motion of the index finger may be referent for dialing a telephone number), or are used as modalizers, often of speech ("feel the softness of this body cream" and a modalizing gesture depicting the softness of the touch). In the HCI context, these gestures are the most commonly used since they can be performed by static hand gestures. Acts are gestures that are directly related to the meanings of the movements themselves. Such movements are classified as either mimetic (imitate some action) or deictic (pointing gestures). A concise summary of hand gesture classification categories is given in Table 2.1.

Table 2.1: Summary of hand gesture classification categories

Category	References
Iconic – (Features Imitation)	[McNeill, 1995; Efron, 1941]
Metaphoric – (Pictorial Abstraction)	[McNeill, 1995]
Deictic – (Pointing)	[McNeill, 1995; Nespoulous, 1986; Quek, 1994; Efron, 1941]
Beats – (Rhythmic)	[McNeill, 1995]
Spatio-temporal – (Pure movement)	[Efron, 1941]
Logical-discursive - (Conversation content)	[Efron, 1941]
Arbitrary – (No distinction)	[Nespoulous, 1986]
Mimetic – (Features Imitation, Action)	[Nespoulous, 1986] - [Quek, 1994]
Referential – (Indicate Action)	[Quek, 1994]
Modalizing – (Mode Description)	[Quek, 1994]

2.2.3 Gesture vocabularies

A GV is a set of commands (notions or words), each of which has a physical representation in the real world as a gesture, pose, or movement. The signs that are used to carve up complex meanings and then reconstitute the meanings through combinations must also be stable and recallable, and this implies a lexicon [McNeill, 1995]. Here lexicon is used in the context of gestures, although the definition is equivalent to the one for written and spoken language. Hand gesture systems can be divided into three major groups according to lexicon size: small, moderate, and large [Oviatt *et al*., 2000].

2.2.3.1 Large gesture systems (over 1000 gestures)

Finding a suitable and practical approach to design hand gesture systems using a large vocabulary is still an open research problem. In the case of one large vocabulary, continuous Chinese Sign Language (CSL) recognition used phonemes instead of signs as the basic units, with a 92.8% successful recognition rate [Wang *et al.* 2002]. About 2400 phonemes were defined for CSL. Experimental results on a large vocabulary of 5113 signs achieved a recognition rate of 95% using fuzzy decision tree with heterogeneous classifiers [Fang *et al.,* 2004].

2.2.3.2 Moderate gesture systems (25-1000 gestures)

Systems able to recognize a medium set of hand gestures are usually used for hand sign language recognition. The American Sign Language (ASL) recognition system of Starner and Pentland [1995] can recognize a lexicon of 40 words. The Korean Sign Language (KSL) of Lee *et al*. can recognize 51 gestures combining postures and gestures. Cui and Weng [1996a] designed a system able to recognize 40 hand gestures from a hand sign lexicon. A recognition accuracy of 90.19% for 104 mannerism gestures was achieved, where the gestures were modeled by Kahol *et al.* [2004] as a sequence of events that take place within the segments and the joints of the human body.

2.2.3.3 Small sized gesture systems (2-25 gestures)

Most systems able to recognize up to 12 gestures are used for man-machine interfaces. Development including a robotic arm in a pick-and-place scenario used twelve different postures [Triesch and Malsburg, 1998], and a real-time hand gesture recognition system that controls motion of a human avatar used dynamic hand gestures [Kim *et al*., 2000]. This system recognizes 5 kinds of hand motion direction (stop, step, walk-run, turn, rotate and grab) and 4 kinds of hand postures (stop, turn, step, grab). A hand GV for video navigation consisting of 8 gestures was developed [Bradski *et al*., 1999] including the commands up, down, left, right, stop, ok (play), and neutral (null gesture). The gestures were chosen for minimal hand movement and high levels of discrimination between gestures.

A prototype vision-based interface using the input modality of a wearable computer for indoor and outdoor operations was able to track and recognize 5 hand postures [Kolsch *et al.,* 2004]. A hand controlled augmented reality (AR) map navigation system was created by Yao *et al.* [2004]. Two symbolic hand gestures and gesture tracking are defined as the controlling commands. Users can directly move their hands on a real map and their relative geography information is displayed. A human machine interface employs 22 dynamic gestures for the effective operation of a variety of in-car multimedia devices [Zob *et al.,* 2003].

2.3 Hand gestures vocabularies design approaches

The theory of universality of gestures states that some gestures have standard cross-cultural meanings [Aboudan and Beattie, 1996]. This is also true for the most part within societies, but some gestures may have different meanings to different individuals [Archer, 1997]. Device control gestures, however, can be freely chosen to have specific meanings related to the particular device [Cohen, 1999]. For example, there are no universally known gestures for commanding a robotic arm to "go to the home position" or "open the robotic arm gripper." There has been virtually no research concerned with the issue of how to design an optimal gesture-based control vocabulary. The first step is to decide on a task dependent set of commands to be included in the vocabulary, such as "move left" and "increase speed." The second step is to decide how to express the command in gestural form, i.e., what physical expression to use such as waving the hand left to right or making a "V" sign with the first two fingers. The association (matching) of each command with a gestural expression is defined here as a GV, the design of which we distinguish according to the type of designer and the solution method.

2.3.1 Gesture designers

GVs can be overtly or inadvertently designed. The thumbs up and down signs come to us from Roman times, whereas the "OK" sign is more recent. Both can be considered as inadvertently designed or naturally evolved gestures (emblems is the current term). More complete sign vocabularies have appeared in this manner without overt determination of the vocabulary by a designer. For straightforwardly designed vocabularies, the most common practice is for a single individual (usually a system developer) to decide which GV should be used for all users. This can be considered as the "Centrist or Authoritarian Approach" (e.g., Kirishima *et al.* [2005], where a GV including seven gestures is used). Alternatively, we can define a "Consensus Approach," where a group of users, either implicitly or explicitly, decide on a common vocabulary to express a given set of commands (e.g., Munk [2001]). At the lowest level is the "Customized Approach," where each individual defines his/her very own vocabulary (e.g., Kahol *et al.* [2006]). One may hypothesize that the consensus and customized approaches will be more comfortable, easier to remember, and more natural to execute. The disadvantage is that the users will not consider other design factors such as the speed and accuracy of a gesture recognition system. In summary, these three straightforward approaches for designing subjective GVs are (a) authoritarian, the designer decides on the commands and gestures for all users, (b) consensus, multiple users decide jointly on a set of common gestures, and (c) custom, the user selects his/her own set of gestures.

2.3.2 Gesture design solution methods

One of the few works that explores the process of gesture design is that of Long *et al.* [1999]. The application is that of a pen-based user interface where gestures are pen drawn marks or strokes that cause a command to be executed. This work includes a gesture design tool, which advises designers how to improve their pen-based gestures. In a more recent work by Nielsen *et al.* [2003], a procedure and a benchmark for identifying gestures based on nine usability heuristics are presented. However, the important factor of vision recognition was ignored.

2.3.2.1 Ad-hoc methods

Ad-hoc methods are the prime method of determining a GV and many examples prevail in the literature [Kortenkamp *et al*., 1996; Waldherr *et al*., 1998; Becker *et al.*, 1998; Agrawal and Chaudhuri, 2003; Abe *et al.,* 2002; Ng and Ranganath, 2002]. Most are of the centrist type,

whereby an individual constructs the vocabulary, usually with no mention of the method used or the rationale for the choices made.

2.3.2.2 Rule based approach

The work of Baudel and Beaudouin-Lafon [1993] provides an example of the use of design rules. They recommend such guidelines as "favor ease of learning," "use hand tension at the start of a dynamic gesture," and "use relaxed position of the hand at the end." Another is that of Baudel *et al.* [1992], who provide a set of guidelines for designing gestural commands. No mention is made on how these guidelines are implemented to generate the actual vocabulary. The application allows a user to give a lecture by navigating through a set of slides with data glove-based gestural commands. Kjeldsen and Hartman [2001], in a vision-based computer interaction setting, present a set of constraints for control actions defined by the permissible motions users can make to effect control. Stating that "the choice of such control movements is more art than science," they proceed to consider the value of control actions for different types of tasks. Again, the approach is intuitive.

2.3.2.3 Analytical methods

Analytical methods are scientifically based, involving perhaps the use of human factors and other aspects such as ergonomics, hand biomechanics, cognitive science, experimental statistics, machine recognition, and mathematics. Although some researchers have applied these disciplines to portions of the hand gesture design problem (for example, Wagner *et al.* [2003], who used analytical methods to design an ergonomic keyboard), we have not found any work using analytical methods for the design of a complete GV.

2.3.3 Current approaches to measure human factors

Intuitiveness is the cognitive association between a command or intent and its physical gestural expression. Two approaches are used to obtain intuitiveness measures [Nielsen *et al.,* 2003]: (a) bottom-up – takes functions (commands) and finds matching gestures, and (b) top-down – presents gestures and finds which functions are logically matched. An example of the bottom-up approach is used in the Wizard-of-Oz technique [Nielsen *et al.,* 2003; Preston *et al.*, 2005; Höysniemi *et al.,* 2005]. The Wizard-of-Oz experiment has persons respond to commands while under camera surveillance. For this purpose, scripts describing the interaction in specific scenarios, functionalities, and contexts must be prepared. The gestures used in interactions by the users were extracted from the video obtained, and analyzed to find how consistent different users were with gestures. Nielsen *et al.* [2003] exemplify the top-down approach in a benchmark designed to test the user's chosen GV. The final step to test Nielsen's methodology is called "Guess the function," where the testee is presented with a list of gestures and he is asked to guess the commands associated with those gestures.

For stress index measures, experiments vary from subjective questionnaires [Nielsen *et al.,* 2003] to electronic devices, such as EMG, to measure muscle activity [Wheeler, 2003]. Postural comfort is based on a "comfort dimension" along which human feelings are arranged according to states of comfort, discomfort, fatigue, and pain [Kölsch *et al.,* 2003]. Approaches used to evaluate stress (or comfort) include mathematical models (i.e. biomechanical models), physical measurement, and subjective methods. Brook *et al.* [1995] constructed a dynamic model representing the biomechanics of the index finger's flexion-extension and abduction-adduction motion. Yasumuro *et al.* [1999] constructed a biomechanical model of the entire hand comprised

of tendons, muscles, and bones, where physical stress is simulated through the natural constraints of the hand. An *et al.* [1979] developed a three-dimensional normative model of the hand. The authors state that the model can be used to perform force and motion analyses, but they do not extend it to estimate stress. Harling and Edwards [1996] used a rod string model to estimate finger tension although no comparison is made with the perceived tension of the users. The use of EMG measurement is popular, but it tends to measure the activity of only part of the muscles involved in structuring a pose [Shrawan and Anil, 1996]. The model and measurement approaches are prone to errors and have not, for the most part, been satisfactorily validated by user studies.

2.4 Implementations of hand gesture vocabularies

Gestures are interpreted to control computer input/output devices or to control actuated mechanisms. Human-computer interaction (HCI) studies usually focus on the computer input/output interface [Card *et al.,* 1990], and are useful to examine the design of gesture language identification systems. Some examples of applications of computer output devices are large panel display control [Baudel *et al*., 1992], graphic image manipulation [Hauptmann, 1989], video control navigation [Bradski *et al.*, 1999], television control [Freeman, 1994], camera control [Jun-Hyeong *et al*., 2002], home appliance control [Lenman *et al*., 2002], and virtual crane control [Freeman and Roth, 1995].

2.4.1 Types of user oriented systems

Two different types of systems are used to train and test accuracy. User **dependent** (D) and **independent** (I) recognition systems are those systems that are trained and tested with gestures collected from a single or multiple users, respectively. A user dependent hand gesture recognition system based on discrete Hidden Markov models and the Viterbi algorithm was suggested [Mäntylä, 2001]. Thirty four user dependent gestures from the Japanese alphabet were recognized using joint angles and hand orientation from a data glove [Takahashi and Kishino, 1991]. Local based gesture modeling in a 3D interface was developed using a single user skin color distribution model [Burschka *et al*., 2005]. Fourteen different people trained a high performance real-time hand gesture recognition user independent system using Hidden Markov Models [Rigoll *et al.,* 1997]. The system by Parvini and Shahabi [2005] assumes that the range of motion of each joint of a hand participating in making a gesture is a user-independent characteristic of that gesture and provides a unique signature across different users.

Other works allow both user dependent and independent types of control based on the person's desires [Alon *et al.,* 2005; Just *et al.,* 2004; Triesch and Malsburg, 2001]

2.4.2 Existing robotic gesture systems

Some relevant, vision-based hand-gesture robot control systems deal with real world constraints with varying success. In the work of Franklin *et al*. [1996], a robot waiter is controlled by hand gestures using the "Perseus" architecture for gesture recognition. Although the person to be serviced is detected and tracked, his control gestures are very limited. There are two gestures recognized by the system, "empty hand" and "holding hand." Such a system is deficient in language understanding.

The robust system created by Becker *et al*. [1998] allows users to operate a semi-autonomous robot able to learn from its environment and tasks. It can also calibrate itself with respect to

image-to-world coordinates. In this work, recognition takes over 82.2 seconds, and this violates the real-time constraint.

The work presented by Kotenkamp *et al*. [1996] presents a system able to recognize six distinct gestures made by an unadorned human in an unaltered environment, using a coarse, three dimensional model of a human. This system recognizes arm-hand configuration and joint degrees but not hand gestures. The drawback of this system is that the range of possible arm joint configurations is narrow, in that the system uses just six gestures.

Cipolla *et al*. [1994] propose a gesture-based interface for robot guidance based on uncalibrated stereovision and active contours. The robot is guided to a point determined by a hand pointing gesture over a ground plane. Although the main goal of the system is the use of stereo vision without calibration, there are some problems that effect real world constraints: proper vision system function requires a strong contrast between the hand and the background and the setup process consists of marking the corners of the board where the hand points with colors. This violates the complex dynamic backgrounds constraint and the variable lighting constraint.

The research proposed by Guo *et al.* [1998] discusses the creation of intelligent, highly safe vehicles controlled by hand gestures based on segmentation using a color probability distribution model. To segment the hand from the forearm, the center position of the maximum circular region is searched; the assumption is that this region occurs only inside the palm. The problem with this research is that there are some gestures for which the maximum circular area occurs above the beginning of the palm, for example, the fist in a profile position. This would cause confusion in distinguishing between different gestures.

Waldherr *et al*. [1998] propose a vision-based interface designed to instruct a mobile robot through both pose and motion gestures using an adaptive dual-color tracking algorithm. This system deals with pose and motion gestures, and it also has a tracking algorithm able to quickly adapt to different lightning conditions. Despite the robust features, however, the system had problems in tracking a person with a multi-colored skirt, and it lacked the ability to learn new gestures.

Yin and Xie [2001] created a fast and robust system that segments and recognizes hand gestures for human-robot interaction using a novel color segmentation algorithm developed on the basis of a Restricted Coulomb Energy (RCE) neural network. The recognition of hand postures is based on the analysis of topological features of the hand. The drawbacks of this system are the poor recognition of dynamic gestures and the need of a setup process in which the user must enclose the hand region.

A system intended to be particularly robust is that presented by Triesch and Malsburg [1998]. It was designed specifically to deal with real world environmental constraints. The system supports three channels for interaction with the robot: a) gestural command, b) spoken command, and c) imitation learning method. The use of multiple channels conveys redundant information to the robot. In addition, the strength of the recognition is based on the combination of features "cues" such as motion, color, or stereo. The main drawback of the system is the recognition time process. For twelve distinct postures it takes between ten and twenty seconds to recognize them. Most of the systems overviewed rely on the simple idea for detecting and segmenting the gesturing hand from the background, such as motion detection or skin color. They assume that there are no other objects in the near environment that have the same color or motion properties of the hand. When dealing with uncontrolled environments this assumption is rarely true. The robustness attained by feature selection and/or the combination with sophisticated recognition algorithms is the condition of success or failure for the human-robot system using hand gestures.

A tele-robotic arm controlled by twelve hand gestures was developed in the "telegest" project, for pick and place operations [Wachs *et al.,* 2002]. The system operates in real time, and visual feedback from the distant scenario is provided by image views of the task. The classification is performed using a supervised FCM optimized framework, and it relies on static hand poses using a uniform background. Based on the "telegest" project, the KISS system was developed [Eliav *et al.,* 2005] to control a robotic vehicle in real-time, where live video streams were sent back to the user as visual feedback information.

A concise summary of the characteristics of systems using hand gesture robot control is presented in Table 2.2.

Table 2.2. Summary of robotic control systems

Application	**Reference**	**Task**	**Commands**	**Method**	**Speed [fps][b]**
Robot waiter	[Franklin *et al.,* 1996]	Tracking, navigation and grab objects	Tracking plus two metaphors	Feature maps, decision trees and grammar	N.A[a]
Robotic arm control	[Triesch and Malsburg, 1998]	Pick and place	Twelve pointing postures	Elastic graph matching, motion and color cues	1/10
Gripsee	[Becker *et al.*, 1998]	Pick and place	Ten pointing gestures	Elastic graph matching, motion and color cues	1/30
Mobile Robot Control	[Kortenkamp *et al.,* 1996]	Tracking, navigation	Six gestures	Arm and Body 3-D Model, 5 DoF.	N.A
Robotic arm control	[Cipolla and Hollinghurst, 1996]	Pick and Place	One pointing gesture	Active Contour	N.A
Human-Vehicle Interaction	[Guo *et al.,* 1998]	Navigation	Six hand gestures	Template matching, RCE neural network	N.A
Service Robot	[Waldherr *et al.,* 1998]	Tracking, navigation, pick and place	Two motion gestures and a pointing gesture	Temporal template matching, Viterbi Algorithm	N.A
Humanoid Service Robot	[Yin and Xie, 2001]	Navigation, pick and place	Eight gestures	RCE neural network, geometrical parameters	5 fps
TeleGest	[Wachs *et al.,* 2002]	Pick and Place	Twelve gestures	Supervised weighted FCM algorithm	N.A
KISS	[Eliav *et al.,* 2005]	Navigation	Six gestures	Supervised weighted FCM algorithm	N.A

[a] N.A = Not available

[b] fps = Frames per second

3 Research methodology

3.1 Overview

This chapter describes the methods used in this research. The basic definition, notation, and assumptions of this dissertation are presented in the first section. The second section presents three different formulations of the GV problem. The following sections present the performance measures of intuitiveness, comfort, and accuracy as functions of the given GV; afterwards, the architecture of the optimal hand GV methodology and each of the modules in it are described. The last three sections describe the experimental methodology and the validation approaches used in this dissertation.

3.2 Problem definition and notation

The basic research problem is to find an optimal hand GV. An optimal hand GV is defined as a set of gesture-command pairs which minimize the time τ for a given user (or users) to perform a task (or collection of tasks). The number of commands is determined by the task, while the set of gestures is the decision variable selected from a large set of hand postures, called the gesture "master-set." Performance depends on the rate of successful recognition of the subset of gestures by a hand gesture recognition system (technical factor) and on human factor measures representing the naturalness of the gesture-command associations and the comfort of the postures. The main problem is to minimize task performance time over a set of all feasible GVs. Each GV is determined by the task and its associated commands, the gestures considered, the transition between the gestures, the duration of the gestures, the intuitiveness (direct and complementary) of the GV, the stress of the gestures, and the recognition accuracy of the system. This problem is stated in (3.1) :

$$\underset{GV\in\Gamma}{Min}\ \tau(GV \mid T,C,G_Z,F,D,I,IC,S,A)=\Psi(T,C,G_z,F,D,I,IC,S,A) \qquad \textbf{(3.1)}$$

where:

τ(GV) = the task performance time for a given GV

ψ is some function of the following factors:

$T = \{t_1, \ldots, t_n\}$, the set of tasks that can be performed in the current ontology.

$C = \{c_1,..,c_n\}$, the set of commands spanning all tasks in T.

$G_z = \{g_1,...,g_k\}$, the large master set of candidate gestures; from this, a subset of gestures is matched with commands in C.

$F_{n \times n}=\{f_{ij}\}$, the command transition matrix, or after normalization, the stochastic matrix. $P=\{p_{ij}\}$ of commands (where f_{ij} is the frequency of transition from command i to command j, and p_{ij} is the probability of using command i after command j).

$D_{n \times n}=\{d_{ij}\}$, the command duration matrix: for $i \neq j$, it is the duration of the transition from gesture p(i) to p(j), and for i=j, it is the minimum time required for the recognition system to sample the current gesture.

$I_{n \times m}=\{a_{ik}\}$, the intuitiveness matrix, where a_{ik} is a measure of cognitive association between the gesture i and the command k.

$IC_{nm \times nm}=\{a_{ijkl}\}$, the matrix of complementary intuitive indices, where a_{ijkl} is a measure of how natural it is to associate the complementary pair of gestures (g_i, g_j) with the complementary pair of commands (c_k, c_l).

$S_{m \times m}=\{s_{kl}\}$, the stress or fatigue matrix, where s_{kl} is the physical difficulty of a transition between gesture k and gesture l. Note that s_{kk} is the fatigue associated with maintaining the same gesture.

$\bar{S}_{m \times m}$ is the comfort matrix, some inverse function of S.

$A(G_n)$ = the recognition accuracy of a given subset of gestures $G_n \subseteq G_z$ (a scalar).

$GV = \{ (i, p(i) \mid \text{all } g_i \in G_n , p(i) \in C \}$, a GV in terms of a set of gesture-command pairs.

Here a vocabulary GV is described in terms of an assignment function p where p(i)=j indicates that the command i is assigned to gesture j.

Γ = the set of all feasible GVs.

The measurement of task completion time involves the evaluation of (3.1), the analytical form of which is unknown, time consuming, and difficult to estimate from experimental data. Therefore, we propose a set of multiple objective performance measures to act collectively as proxies for (3.1). The proposed methodology will be developed under the following assumptions:

(a) The gestures are static postures. The same methodology can be used with small modifications for dynamic gestures.

(b) Each gesture cannot represent more than one command, and each command must be expressed by exactly one gesture.

(c) Measures of intuitiveness can be collected from a small empirical experiment.

(d) The static stress measures can be determined empirically, and will yield enough information to estimate the dynamic stress measures.

(e) For this problem, recognition accuracy of a set of gestures is determined by a fuzzy means classifier (although any other recognition algorithm can be used).

The weakest assumption in the list is (d), which includes a lack of precision inherent in subject evaluation of the effort necessary for different hand postures. This means that the user's opinion does not always reflect the apparent stress needed to perform different hand postures. This can be alleviated in the future using devices such as EMG to measure muscle activity.

The assumptions described above are used to simplify the problem to one that is tractable. However, for a realistic human machine system, dynamic gestures should be considered. The extension to dynamic gestures involves changes in the experimental methods to obtain the human factor measures, and in the robustness of the recognition algorithms. Nevertheless the analytical approach presented in this thesis remains the same.

3.3 Problem formulation

The main performance measure is the completion time, τ, to perform a task. However, since the task completion time, as a function of GV, has no known analytical form, we propose three different performance measures as proxies for the task completion time. These performance measures are: intuitiveness $Z_1(GV)$, comfort $Z_2(GV)$, and recognition accuracy $Z_3(GV)$. The first two measures are user centered, while the last is machine centered. This multiobjective optimization problem (MOP) may have conflicting solutions where all the objectives can be maximized simultaneously. As with most multiobjective problems, this difficulty is overcome by allowing the decision maker to select the best GV according to his own preferences.

P 3.1 Three priority problem

$$\begin{aligned} &Max\ Z_1(GV), Max\ Z_2(GV), Max\ Z_3(GV) \\ &GV \in \Gamma \end{aligned} \qquad \textbf{(3.2)}$$

where,
Z_1 = intuitiveness of the GV.
Z_2 = the total comfort of the GV.
Z_3 = the recognition accuracy of the GV.

In (3.2), a MOP is defined as maximizing each of the measures over the set of all feasible GVs. Here the set of Pareto solutions can be used to aid the decision maker in selecting the GV. The Pareto frontier solution can be determined through enumeration for small problems and via heuristic methods for large problems. Selecting optimal GV solutions (optimal from the user's point of view) from the complete set is *a posteriori* judgment (derived from experiment and observation rather than theory), which can only be done by examining concrete solutions. Because the enumeration approach is untenable for even reasonably sized vocabularies, we describe two alternative formulations of this problem. The first is to map the three performance measures into a single measure using weights w_i to reflect the relative importance of each of the objectives. Another method of handling the difficulties of the sometimes conflicting multiobjective values is to adapt a goal programming approach. The difficulty with this approach is in the selection of goal values.

P 3.2 Single objective problem

$$Max\, Z(GV) = w_1 Z_1(GV) + w_2 Z_2(GV) + w_3 Z_3(GV)$$
$$GV \in \Gamma \qquad \textbf{(3.3)}$$

where:

w_i = the relative importance of factor Z_i.

The weights in (3.3) can be found empirically by letting the decision maker give the importance of each factor according to his/her needs and preferences. Alternatively, the weights can be varied, and for each unique weighting scheme the corresponding solution can be presented to the user for acceptance or rejection.

As a practical matter, however, it is more convenient to consider a dual priority objective where the technical factor, accuracy, and its impact on performance time, will be considered the most important factors from the user's standpoint. This is due to the fact that if a gesture is not recognized, the command associated with that gesture will not be carried out, and thus the tasks will be interrupted. A GV lacking comfort or naturalness, in contrast, will delay task completion time but will not interrupt it. This is best expressed using dual priority objectives, where recognition accuracy is considered of prime importance, and the human performance objectives are secondary.

P 3.3 The relaxed problem

$$Max \bar{Z}(GV) = w_1 Z_1(GV) + w_2 Z_2(GV) \qquad \textbf{(3.4)}$$
$$GV \in \Gamma$$
$$s.t. \quad Z_3(GV) \geq A_{min} \qquad \textbf{(3.5)}$$

The relaxation of P 3.1 produces P 3.3 by considering recognition accuracy as a constraint (3.5), and combining the human objective measures into one objective function using the

combination weights w_1 and w_2 (3.4). In other words, P 3.3 suggests that the GV solutions considered for maximization are only those GV's which include gestures G_n that are successfully recognized.

This relaxed problem, based on the dual priorities of the objective functions, is the main approach adopted in this thesis and is described in detail in Section 4.3

3.4 Performance measures

Each of the performance measures is described as a function of the given GV. The objective functions $Z_1(GV)$ and $Z_2(GV)$ are human-valued measures (intuitiveness and comfort), while $Z_3(GV)$ is machine valued (accuracy). Each performance measure is described below.

3.4.1 Intuitiveness – Z_1

Intuitiveness is the naturalness of expressing a given command with a gesture. The intuitiveness of a GV is the sum total of the intuitiveness of all gesture-command pairs in the vocabulary, each weighted according to frequency of use.

$$Z_1(GV) = \sum_{i=1}^{n} a_{i,p(i)} + \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,p(i),j,p(j)} \quad \textbf{(3.6)}$$

The value $a_{i,p(i),}$ represents the strength of natural association between command *i* and its matched gesture p(i). The first term in (3.6) represents the sum over all the command-gesture pair intuitive values in the vocabulary. Complementary intuitiveness $a_{i,p(i),j,p(j)}$ is the level of association expressed by the selection of complementary gesture pairs p(i), p(j) for complementary command pairs (i,j). Accordingly, the complementary intuitiveness has a stronger effect than regular intuitiveness, which expresses the tendency to reward vocabularies with complementary gestures selected for complementary commands, and to punish arbitrary mappings. The total complementary intuitiveness for a GV is represented by the second term in (3.6).

3.4.2 Comfort – Z_2

Stress is the internal distribution of force per unit area that balances and reacts to external loads applied to a body. In the context of this thesis, we define stress as the force needed to perform a gesture. The difficulty of composing and holding gestures can be explained by the effects of blood flow restriction on the stressed joints, which, in turn, causes strain and fatigue on the muscles. Obviously, some gestures are easier to perform than others. Even when some of them look comfortable in the beginning, after some time the user may feel fatigue. The amount of fatigue is related to muscle forces that cause finger and palm tensions. Total stress is a scalar value equal to the sum of the individual stress values assigned to maintaining and transitioning between the postures and weighted by the duration and frequency of use.

$$Z_2(GV) = K - \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}\, d_{p(i)p(j)} s_{p(i),p(j)} \quad \textbf{(3.7)}$$

where:
K is a constant obtained empirically
f_{ij} stands for the frequency of transition between commands i and j
p is an assignment function

$d_{p(i)p(j)}$ is the duration between gestures p(i) and p(j)
$s_{p(i)p(j)}$ is transition stress between gestures p(i) and p(j)
p(i)=j indicates that the command i is assigned to gesture j. Let k=p(i) and l=p(j), the value of s_{kl} represents the physical difficulty of a transition between gestures k and l.
The duration of hand reconfiguration between gestures k and l is represented by d_{kl}.

3.4.3 Accuracy – Z_3

Accuracy is the degree of conformity of a measured or calculated quantity to its actual (true) value. In the context of gesture recognition, accuracy is a measure of how well a set of gestures can be recognized. To determine the accuracy of a GV it is only necessary to consider the subset of gesture types G_n and not C. So technically, $Z_3(GV)$ is a function of G_n only. To obtain an estimate of gesture accuracy, a set of sample gestures for each gesture type in G_n is required. These samples are used to train the given hand gesture recognition algorithm. An additional set of samples is used to test the recognition accuracy of the algorithm, designated as T_g .The number of gestures correctly and incorrectly classified in the testing set is usually presented in a confusion matrix. The number of misclassified gestures can be calculated as T_e. The recognition accuracy (in percent) is given below (often represented by the symbol A):

$$Z_3(GV) = \frac{\left(T_g - T_e\right)}{T_g} 100 \qquad \textbf{(3.8)}$$

3.5 System architecture

The architecture for optimal hand GV comprises three modules (Figure 3.1). In Module 1 human psycho-physiological input factors are determined. In Module 2 a search for a feasible gesture subset, subject to machine gesture recognition accuracy, is carried out. Module 3 constitutes a command-gesture matching procedure. The task set T, the large gesture master set G_z, and the set of commands C are the input parameters for Module 1. Note that C is determined by T; where given a set of tasks, the union of all commands used to perform all tasks constitutes C. The objectives of Module 1 are to establish associations between commands and gestures based on user intuitiveness (direct and complementary), to find the comfort matrix based on command transitions and fatigue measures, and to reduce the large set of gestures to the master set G_m, (Figure 3.2). For Module 2, the necessary inputs are the reduced master set of gestures G_m, and a recognition algorithm to determine A. This module employs an iterative search procedure to find a single feasible gesture subset G_{n*} (or alternatively the set of feasible gesture subsets), satisfying a given accuracy level (specified in (3.5), the level is usually determined by the decision maker). Complete enumeration or an heuristic search can be used as a search procedure.

The inputs to the third module are the matrices intuitiveness V, comfort U, command C, and the subset of gestures G_{n*}. The goal of this module is to match the set of gestures G_n with the set of given commands C, such that the human measures are maximized. The resulting gesture-command assignment constitutes the GV.

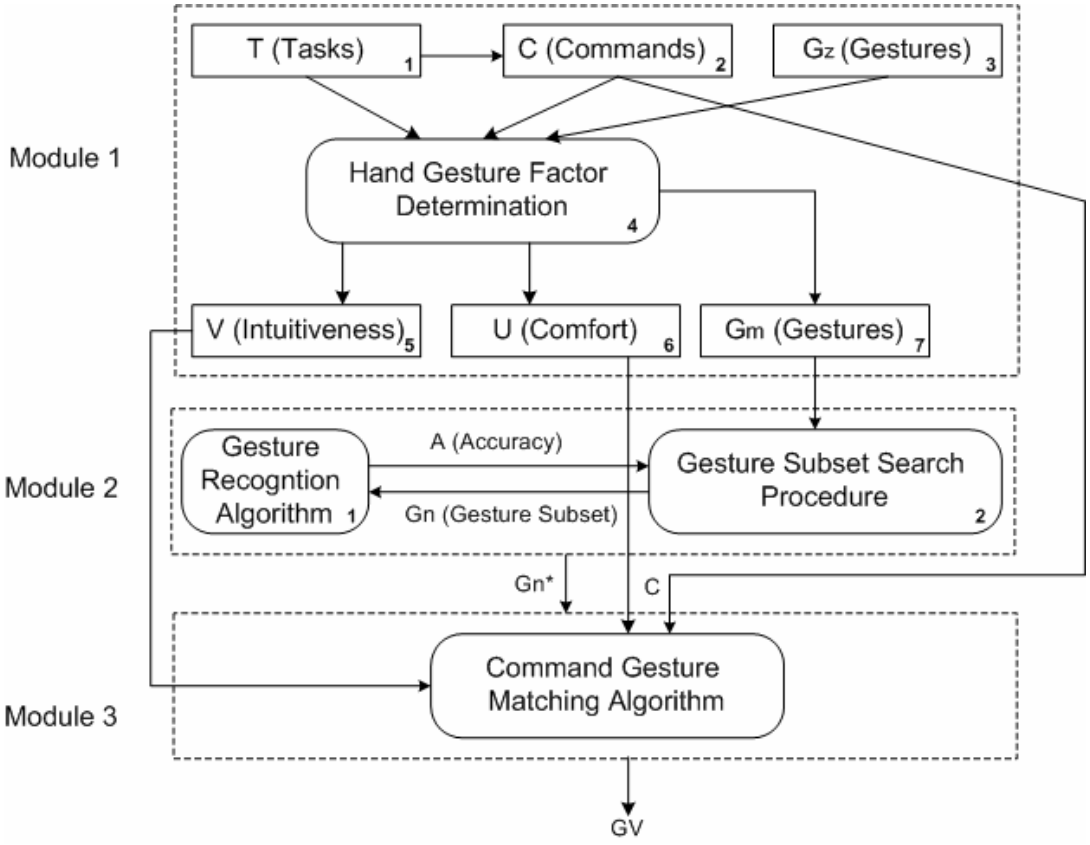


Figure 3.1. Architecture of optimal hand GV methodology

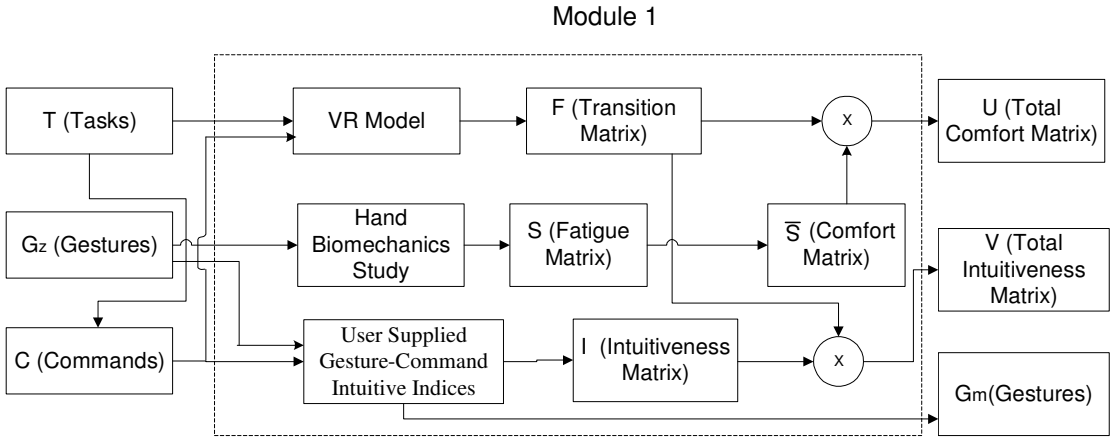


Figure 3.2. Hand gesture factor determination stage

3.5.1 Module 1: Hand gesture factor determination

In this section the inputs to the first module are described, including the methods used to compose the U and V matrices and the gesture master set G_m. In addition, the algorithm to determine the recognition accuracy, A, is described.

3.5.1.1 Module 1.1-1.2: Task and command sets (T, C)

The task set can comprise a single element or multiple element (multi-task) set. For each task t_i, a set of commands C_i is defined. For a multi-task set $T=\{t_1,..,t_n\}$, the command set is the union of the individual task commands.

$$C = \bigcup_{i=1,,n} C_i \tag{3.9}$$

For example for a 'place' task with commands C_1={'left', 'right', 'up', 'down', 'backward', 'forward'} and for a 'pick' task with the commands C_2={'up', 'down', 'backward', 'forward', 'open', 'close'}, a new task (multi-task) 'pick & place' will include the command set C={'left', 'right', 'up', 'down', 'backward', 'forward', 'open', 'close'}

3.5.1.2 Module 1.4: Command transition matrix (F)

To estimate the frequency of command usage for the set of selected tasks T, it is necessary to carry out experiments according to the desired task. Examples include using a real or virtual model of a mechanism[iv] or driving a VMR through a maze. These experiments are discussed in Section 6.2. For a command set C of size n, a matrix F_{nxn} is constructed where f_{ij} represents the frequency that a command c_j is evoked given that the last command was c_i. This measure is significant in the sense that it is hypothesized that (a) an optimal hand GV will pair high frequency commands to gestures that are easy to perform (low fatigue), and (b) the physical ease of movement between gestures will be paired with high frequency command transitions.

3.5.1.3 Module 1.3: Large gesture master set (Gz)

Since the set of all possible gestures is infinite, we first establish a finite set of plausible gesture configurations. To create the finite set of plausible hand gestures there are two possible approaches: (a) visual capture of gesture images, or (b) creation of synthetic gestures. For small hand gesture databases, real hand gesture images may be captured and labeled with the configuration parameters that characterize each particular gesture. The tiresome job of defining a large gesture set (thousands of gestures), however, may be relieved using a synthetic gesture generator. The synthetic generation of gestures has a significant advantage over the capture of real hand gestures, as the hand gestures and the labeling process are done automatically.

One solution is to generate configurations by specifying a collection of gesture primitives such as finger positions (extended, spread), palm orientations (up, down sideways), etc. For additional material on gesture primitives and combining them into whole gestures, including those for dynamic gestures, see Miners *et al.* [2002]. This list should exclude in advance those gestures that are impossible to perform due to inter and intra joint constraints [Lin *et al.,* 2000] (e.g., spreading fingers that are clenched in a fist), and those that are extremely stressful, such as the gestures exclusive to piano players.

[iv] To determine the command transition matrix, we make the assumption that it is independent of the gestures or the process in which commands are executed. Thus, it can be approximated by, for example, a virtual reality model or teach pendant for a robotic task.

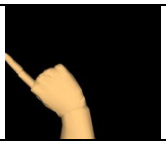
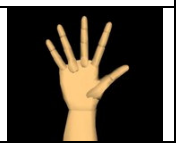
The virtual model used to generate the gestures will be a graphical approximation of a 2D hand gesture view, using a discrete digital coding (with base 2 and 3) to represent each gesture. The string representing each posture consists of 11 bits (Table 3.1), and each is described in the following table. The first bit in the string is the most significant bit MSB (the left most digit) and the last is the least significant bit LSB (the right most digit).

Table 3.1. Configuration of the hand model

Order	Type	Effectors	Description
1	3-State	Palm	0-Palm Down, 1-Palm Up, 2-Side of the Palm
2	3-State	Wrist	0-Straight, 1- Bend to Left, 2-Bend to Right
3	2-State	Thumb	0-Closed, 1-Extended
4	2-State	Index	0-Closed, 1-Extended
5	2-State	Middle	0-Closed, 1-Extended
6	2-State	Ring	0-Closed, 1-Extended
7	2-State	Little	0-Closed, 1-Extended
8	3-State	Thumb-Index	0-Tight, 1-Opened,2-Perpendicular to the Palm
9	2-State	Index-Middle	0- Tight, 1- Opened
10	2-State	Middle-Ring	0- Tight, 1- Opened
11	2-State	Ring-Little	0- Tight, 1- Opened

The first 2 bits control the palm, wrist rotation (0-90-180 degrees), and ulnar deviation (the wrist bent towards the ring finger, in the middle, or towards the thumb finger). The next 5 bits indicate whether the finger is bent (flexed towards the palm) or extended, and the last 4 bits describe whether two adjacent fingers are separated (spread) or not (tight). The thumb (bit 8) has an additional degree of freedom, so it can be opened perpendicular to the palm (out of the palm plane). Several configuration codes are depicted in Table 3.2 with their respective graphical representations.

Table 3.2. Examples of posture encoding

Code	00000000000	10000000000	20000000000	02010000000	10011000100	10111111111
Gesture						

The total number of postures available using the coding described above is $2^8*3^3=6912$. It is still possible to decrease this large gesture set by eliminating those postures that violate inter finger constraints [Lin *et al.,* 2000] and are extremely difficult to perform. Also, gestures that are ambiguous when using only one view of the hand may be eliminated. For example, it is impossible to spread two adjacent fingers when they are bent. It is also extremely difficult to separate two adjacent fingers when one of them is closed and the other is extended. Postures that are ambiguous when using a single camera view are shown in Fig. 3.3 (a)-(c): when they are viewed from the back of the hand (d), they cannot be distinguished.

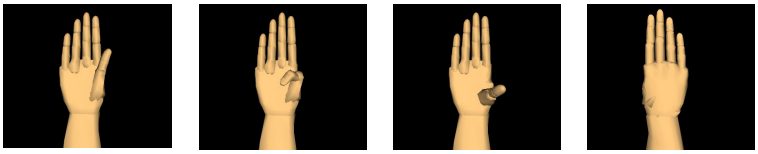


Figure 3.3 Ambiguous postures due to using a single view

To decrease the large gesture set of postures we proposed the following set of rules that reflect the inter and intra finger constraints. The rules considered were a) postures with palm up or palm down cannot have the thumb perpendicular to the palm, b) postures where two adjacent fingers are arranged with one open and the other bent are not allowed to be displayed outspread, c) postures where the fingers are spread out are not allowed to show the palm on its side, d) postures where the palm is on its side are allowed only if all the four fingers (except the thumb) are bent or open, or the index finger is open and the rest are bent, or the index finger is bent and all the rest are open, e) if the thumb is bent, it cannot be extended or perpendicular to the palm, and f) postures where any two adjacent fingers are closed are not allowed to be displayed outspread. Considering these rules, the large set of feasible gestures G_z can be reduced to 648 postures.

3.5.1.4 Module 1.5-1.7: Matrices of intuitiveness and the gesture master set (V, G_m)

The intuitive matrices include the direct intuitive matrix and the complementary intuitive matrix. Both intuitive matrices are obtained by subjective data collection methods. Section 6.3 describes the experiments used to obtain the subjective responses. Based on the popularity of the gestures in the direct intuitive, the master set of gestures is reduced to $G_m \subset G_z$.

3.5.1.5 The direct intuitive matrix, I

The direct intuitive index is a measure of how "natural" it is for a user to express a command with a particular gesture. These indices are determined empirically. For each command c_i a user is prompted to select a posture with the strongest "cognitive" association with the command. Once the user performs the gesture, the next step is to "build" the posture using the virtual model primitives described in section 3.5.1.3. In this way, is possible to encode the user's selection in a simple manner and later on to obtain the distribution of user choices over the different gestures. The process to obtain the intuitive indices is described in Figure 3.4

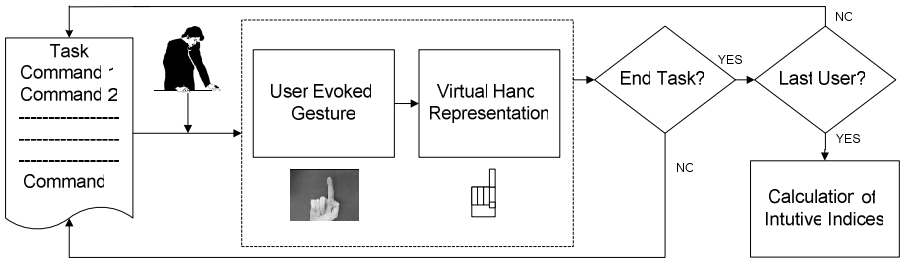


Figure 3.4. Empirically determining the intuitive indices

Using this information, the construction of an intuitiveness matrix, I_{mxn}, is straightforward. The entries for I are represented as a_{ik}.

$$a_{ik} = \sum_{j=1}^{u} a_{ik}^{j}, i = 1,...,z \quad k = 1,..,n \tag{3.10}$$

where,

u = the number of users (j=1,...,u)
z=number of gestures in the master set (i=1,…,z)
n= number of commands (k=1,…,n)

a^{j}_{ik} = is a binary variable to express whether user j cognitively associates gesture i with command k.

a^{j}_{ik} =1 means that user j selects gesture *i* to represent command k, otherwise a^{j}_{ik} =0.

Let w^{j}_{ik} be the level of strength of belief of user j in making the association between the command k and the gesture j then, a weighted intuitive matrix, with common element $\hat{a}_{ik}$, can be generated.

$$\hat{a}_{ik} = \sum_{j=1}^{u} a_{ik}^{j} w_{ik}^{j}, i = 1,...,z \quad , \quad k = 1,...,n \tag{3.11}$$

3.5.1.6 The complementary intuitive matrix, IC

When analyzing the actions in a task, one of the characteristics common to most tasks is the existence of commands that have complementary counter parts. This is especially true for directional commands (left-right, up-down) and two-state action commands (open-close). This mapping is usually expressed by the users by selecting complementary gestures for complementary commands.

A brief study, presented in Chapter 6, revealed that for complementary commands, complementary gestures were selected; however, it was found that there is no single rule to determine the complementary gesture for any given gesture. Moreover, one gesture may have more than one complementary gesture. These gestures can be obtained by flipping the palm, or by closing/extending the fingers. See Figure 3.5 (a), (b), and (c).

The naturalness of matching up a pair of complementary commands (i, j) with a pair of complementary gestures (k,l), is represented by a complementary intuitive index of the form a_{ijkl} . Higher values of complementary intuitive indices will force complementary pairings. The matrix of complementary intuitive indices $IC_{n,m\ x\ n,m}$, can be large, but it can be compacted considerably as most of the entries will be zero. The general intuitive set of matrices V=[I, IC] comprises both the direct and complementary intuitive matrices.

(a) (b) (c)

Figure 3.5. Complementary gestures: (a). Flipping the palm. (b) rotating the wrist. (c) open-closing the fingers

3.5.1.7 The gesture master set, G_m

Each element of the non-weighted intuitive matrix *I* indicates the number of times that a gesture i was used to represent command j. The row sums indicate the popularity of a gesture. The normalized popularity p_i of gesture i is:

$$p_i = \sum_{k=1}^{n} a_{ik} \Big/ \sum_{k=1,}^{n} \sum_{j=1}^{u} a_{jk} \qquad \textbf{(3.12)}$$

Zero values of p_i indicate that some gestures were not selected to represent any command, and low values show that these gestures were rarely used to express commands. Those gestures are not intuitive to the users, and assuming that they are also awkward to perform, they are not "natural" gestures. If these kinds of gestures are not intuitive and are stressful, the master set of postures can be reduced further by removing these postures from the master set. Hence, a reduced master set of gestures can be defined using (3.13).

$$G_m = \{ g_i \mid p_i \geq t \} \qquad \textbf{(3.13)}$$

where,

G_m is the reduced master set of gestures
p_i is the popularity of gesture i
g_i is a gesture
t is the threshold of popularity of the gestures

3.5.1.8 Module 1.6: Fatigue and comfort matrices (S,U)

The fatigue (or comfort) indices were determined via an experimental study (see chapter 6.4). The results are arranged in a matrix $S_{m \times m}$, whose common element s_{ij} represents the physical difficulty of transitioning from gesture i to gesture j. Let the coefficients u_{ijkl} be the entries of a square matrix, $U_{nm \times nm}$. An entry u_{ijkl}=K-f_{ij} x s_{kl} represents the frequency of transition between commands i to j times the transition stress of k to l commands when i and j are paired with gestures k and l, respectively. This product reflects the concept that the total stress measure of the GV depends on the frequency of use of a gesture or a gesture pair transition. The total comfort is the difference between the constant and the total stress detailed above. Note that the diagonal entries s_{ii} represent the total stress associated with using a gesture repeatedly to carry out the same command.

3.5.2 Module 2: A feasible subset search procedure

3.5.2.1 Module 2.1: Gesture recognition algorithm, $A(G_n)$

The goal of the subset search procedure is to evaluate solutions based on recognition accuracy of a hand gesture recognition system. The hand gesture recognition process involves two sequential tasks: (a) extracting relevant features from the raw image of a gesture, and (b) using those image features as inputs into a classifier. Such an algorithm is described in Wachs *et al*. [2002], where the segmentation consists of the extraction of the hand gestures from the background using grayscale cues. For simplicity, gestures are presented with a uniform background. The captured hand image is thresholded to a black/white segmented hand silhouette, and partitioned into block features. These features are used first to train and later to test a classifier. We used a fuzzy c-means clustering algorithm (FCM) as our classifier. When testing

the system (real-time operation), these features are compared to clusters obtained from an FCM algorithm. To obtain an estimate of the recognition accuracy, a set of training samples consisting of images for each gesture type in G_n is used. The classification results are organized as a confusion matrix. From the confusion matrix, the recognition accuracy $Z_3(G_n)$ is computed using (3.8). Note that the recognition accuracy depends only on the gesture set G_n and the commands associated do not play any role in determining $Z_3(GV)$; therefore, $Z_3(GV)= Z_3(G_n)$. Further details may be found in Wachs *et al.* [2003].

To determine the accuracy of a candidate subset of gestures, it is necessary to train a classifier. Two different approaches will be discussed in this thesis. The first involves repeatedly retraining the FCM for each different candidate G_n. The second entails training and tuning the FCM for the master set G_m. For the first method, the recognition accuracy is calculated using (3.8) based on the confusion matrix $\mathcal{C}_n$ obtained from the FCM. For the second approximate method, the confusion matrix obtained from the master set is $\mathcal{C}_m$. For a candidate set of gestures G_n , the recognition accuracy is obtained by creating a new confusion matrix $\mathcal{C}'_n$ which is obtained by dropping all the columns and rows j in $\mathcal{C}_m$ where $j \in \{ i \mid \text{all } g_i \in G_m - G_n \}$. The recognition accuracy (3.8) is then obtained using the confusion matrix $\mathcal{C}'_n$.

Gesture classifiers such as a neural network, Bayesian and boosting methods require large training sets. This thesis used a fast FCM classifier for the gesture recognition algorithm, which requires a relatively small training set. An automated method, based on a parameter search procedure, is used to reconfigure and recalibrate the recognition algorithm for each new set of gestures; more details will be offered in Chapter 5.

3.5.2.2 Module 2.2: Gesture subset search procedure

Consider a subset solution G_n that has recognition accuracy below the minimum desired value. One notes that by observing the indices of the gestures only, it is not possible to predict how to order them in the subset or how to interchange them with new gestures from G_m to obtain improved recognition accuracy. Thus, given a subset solution, G_n, and its neighborhood solutions obtained by some gesture exchange rule, there is no physical reason that the $A(G_n)$ function is well behaved within this neighborhood. Hence, attempting to find a local maximum by the standard search methods of gradient ascent will fail. To overcome this problem two metaheuristic approaches were developed. The first approach is referred to as the Disruptive Confusion Matrix (DCM), and the second is referred to as the Confusion Matrix Derived Solution method (CMD). Under DCM, pairs of gestures are exchanged and maintained in a binary tree. Each of the most confused gestures in the subset is discarded and replaced by a gesture from the remaining gestures in the master set G_m using a MaxMin rule (Figure 3.6). Gesture sets that have associated accuracies below some stipulated value A_{min} are discarded.

The MaxMin rule selects a gesture (i) (where i is the gesture removed from the current subset) from the master set that is least similar to (i.e., farthest away from) all the gestures in G_{n-1}. Like simulated annealing, the method allows moves in the direction of inferior solutions, possibly avoiding pre-convergence to local optima. This method generates a sequence of gesture subsets until a depth of the tree is reached.

The second approach, the CMD, relies on the recognition accuracy obtained from the master set G_m and its associated confusion matrix.

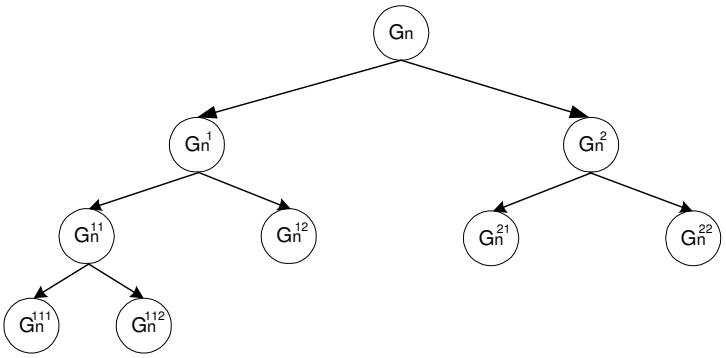


Figure 3.6. Solution tree

Given the confusion matrix, the selection of the n least confused gestures results in the highest recognition accuracy for a subset G_n. Additional solutions with the same or lower recognition accuracies can be obtained by discarding the most confused gesture i over the diagonal of the confusion matrix, and instead, selecting the least confused gesture j in the subset G_m-G_n obtained from the confusion matrix. Each solution is kept in a stack of solutions. When a solution is repeated, i.e., already in the stack, a new solution is generated instead by discarding the most confused gesture $k \neq i$ and selecting a least confused gesture j in the subset G_m-G_n. The feasible solutions obtained while employing one of the methods (DCM and CMD) are then used in the gesture-command matching problem in Module 3 to obtain candidate GVs. Both methods are explained in detail in Chapter 4.

3.5.3 Module 3: Command gesture matching algorithm

In this module every feasible gesture solution $G_n^* \in \{ G_h \mid G_h \in G_z , A(G_h) \geq A_{min} \}$ found using the DCM or the CMD procedures is matched to commands to obtain the final GV set using the integer quadratic assignment problem (QAP) [Koopmans and Beckmann, 1957].This can be done in two ways: if w1 and w2 are known, then a single matching is found directly from the solution; Otherwise, linear combinations of the weights can be used to obtain an associated set of solutions for G_n^*. The integer QAP solves a problem of matching gestures to commands. The 0-1 integer QAP has attracted a lot of attention and many approaches have been proposed for its solution such as (i) find a proper linearization of the objective to obtain a linear program [Finke *et al.,* 1987], and (ii) relax the QAP to a (0, 1) linear integer program by introducing new binary variables and new constraints. A practical approach is used in this work, adopting the simulated annealing algorithm as described in Connolly [1990].

3.6 Experimental methods for estimating intuitiveness and stress

A series of experiments were conducted to obtain subjective measures by studying responses from human subjects. Intuitiveness is the cognitive association between a command or intent and its physical, gestural expression. Two approaches are offered for obtaining intuitiveness measures: (a) bottom-up finds matching gestures for functions (commands) and, (b) top-down presents gestures and finds which functions are logically matched [Nielsen *et al.,* 2003]. To collect intuitive data, we used the "bottom-up approach."

The actual acquisition of gesture responses is not trivial. The following three methods were considered: (a) direct video capture - the subject physically forms the gesture and a photographic image is taken (errors in recognizing similar gestures may occur), (b) use of a database of candidate gesture images (browsing a large database is time consuming, and difficult for the

subject to remember and make comparative judgments), and (c) coded gesture entry – the subject physically generates the gesture and enters configuration information. The coded gesture entry method was selected based on its combination of reasonable time demands and accuracy in gesture labeling. For the stress measure, there are two approaches: (a) EMG based indices (the use of EMG measurement is popular but problematic, as it usually only measures the activity of some of the muscles involved in structuring a hand pose), and (b) the use of ergonomic tests, where the user may rank poses from weak to strong on some scale. Based on the static stress measures for all the gestures in the master set G_m and only a few measures for the transition stress, a model that describes the transition effort was developed and validated.

A test to validate the assumption that task completion times were shorter using V_G than when using V_B vocabulary samples was performed. The V_G vocabularies dominated solutions of the V_B vocabularies, which means that each GV that is from the V_G set of vocabularies had higher associated values for the three objectives (accuracy, intuitiveness, and comfort) than each GV from the V_B set (see Appendix B for an example). Two scenarios were used to study GV performance: robotic arm pick and place, and VMR maneuvering tasks. Beginner users performed those tasks using GVs obtained using either the robotic arm or the VMR framework. The purpose of this test was to measure the time to complete the task, assuming that fatigue effects are introduced in long-term tasks, and an intuitive term remains constant during the completion of the task, see Section 6.6.5. To reduce testing time, a virtual three-dimensional model of the robotic arm was developed for the first task, similar to Ho and Zhang [1999], and a virtual driven VMR was developed for the second task. Learning and memorability usability tests were also performed. The assumption tested was that V_G is easier to learn and remember than V_B vocabulary samples. The learning rate analysis was performed using the task completion time obtained for each trial. The memorability test relied on the user's capability to recall gesture-command associations after performing the tasks.

3.7 Validation methodology

The validation of the analytical procedures for finding the optimal hand GVs consisted of testing the following hypothesis:

(a) The multi-objective function (3.2) is a proxy measure for performance time (3.1) to complete a task.

(b) The analytical performance measure (3.3) is inversely proportional to task execution time as described in (3.1).

(c) The use of GVs from the V_G set will reduce task completion time relative to when using GVs from the V_B set.

(d) The GVs from the V_G set are easier to remember than those GVs from the V_B set.

(e) The GVs from the V_G set are easier to learn than those GVs from the V_B set.

Hypotheses were tested for two different tasks with two sets of hand GVs, a set of n GV's from the V_G and from the V_B sets, obtained using the methodology suggested in this work. Each user performed m trials to complete a task with the same GV. The task completion time τ was saved. From the completion time for each trial, a learning curve was created. The average of the last k trials was used as a representative task completion time for the given GV. The learning curves also allowed obtaining the learning rate (r) for each GV. With this information, the t-test was used to validate the hypothesis that the vocabularies from the set V_G resulted in shorter completion task times, were rapidly learned, and were easier to remember.

3.8 Usability experimental methodology

Two different usability tests, for learnability and memorability, were performed involving the task performance and the quality of the vocabularies of the GVs. The procedure and results of these experiments are given in Section 6.6. The first experiment tested which vocabulary sample, V_G or V_B, was easier to learn using learning curves. The concept of the learning curve is based on the idea that the time required to complete a task decreases as the user gains experience. A learning rate corresponding to a learning curve describes the change in performance time each time the cumulative number of trials doubles. A 0.8 learning rate means that each time the cumulative number of trials doubles, the performance improves by 20%. The model for the learning curve is based on Schwartz [1998]:

$$Y_n = Y_1 n^{-b} \tag{3.14}$$

where Y_n is the estimated value of the completion time, in seconds, of the n^{th} trial, n is the trial number, Y_1 is the time of the first trial, and b is:

$$b = \log r / \log 2 \tag{3.15}$$

where r is the learning rate.

A lower learning rate indicates faster learning. Figure 3.7 shows an example of the learning curves expected from GV_1 that belong to V_G and GV_2 that belong to V_B. From the figure it is possible to see that the task using GV_1 took less time than when using GV_2, and since they started from the same time, the learning rate of GV_1 was lower (i.e., faster learning).

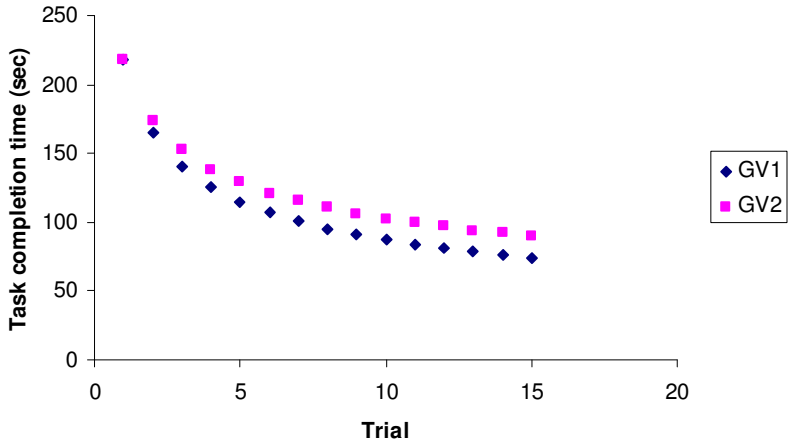


Figure 3.7. Learning curves for two GVs from the V_G and V_B vocabulary set

The second experiment involved testing the memorability aspect by comparing V_G and V_B vocabulary samples. This test was performed immediately after the subject completed the task performance time trials. Given a trained testee presented with a list of commands, the memorability index is obtained through a software application that examines which gesture, selected from a large set of gestures larger than the set of commands, the testee associates with each given command. More specifically, the memorability index is expressed as:

$$m(GV) = (n - n_e)/n \quad \textbf{(3.16)}$$

where,

m(GV) = the memorability index for vocabulary GV

n=total number of commands in the task

n_e= number of wrong command-gestures matched

3.9 Discussion

In this chapter the main methodology of this dissertation was presented briefly, starting from the definition of the problem as a multiobjective optimization problem, where the objectives are performance measures based on technical and psycho-physiological factors. These measures are based on matrices determined by empirical results. Then, the main solution approach is described relative to system architecture, with each module in the architecture being described in detail, thus leading to candidate GV solutions. Reducing the number of postures from thousands to hundreds by respecting the intra-inter finger joint constraints and additional rules simplifies this procedure.

A useful feature introduced in this chapter is the complementary intuitiveness matrix, which expresses the cognitive association between complementary commands and gestures. Finally, the validation methodology was presented to verify the initial hypotheses that motivated this dissertation.

In the next chapter, the MOP will be relaxed to the simultaneous maximization of two objectives, and later reduced to a quadratic assignment problem. Algorithms will be presented to solve each step of the relaxed formulation with detailed examples of the solutions.

4 Optimization approach

4.1 Overview

The multiobjective problem and the dual priority problem presented in Chapter 3 can be formulated using mathematical programming, and hence, computational methods can be used. For the dual priority problem, two methods to find a feasible subset of gestures from the gesture master set will be rigorously described in this chapter; two illustrative examples will be presented. The first method is the disruptive confusion matrix (DCM) to create a metaheuristic search tree. The second method is the confusion matrix derived solutions (CMD), based on the creation of a single confusion matrix for the whole gesture master set. The quadratic assignment problem is used to model the problem of optimal matching between commands and the feasible subset of gestures. To solve this problem an existing simulated annealing scheme was applied, and a number of solutions are presented near the end of the chapter. The MOP was solved using complete enumeration, and the solutions were presented as 3D representations. This gives the user the decision to select the best GV according to his own preferences.

4.2 The multicriteria optimization problem

Before the Dual Priority Problem is discussed, it is useful to formulate a "basic" multicriteria optimization problem (MOP) that contains all three objectives without priorities. The objectives may conflict when solving this problem, i.e., they all cannot be maximized simultaneously. It is then left to the decision maker to provide subjective preferences to select an acceptable solution. For this formulation the master set of gestures g_j j=1,..,m, is provided.

P 4.1 Multicrtieria problem (MOP)

$$\max Z_1 = \sum_{i}^{n}\sum_{j}^{m} v_{ij}x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{l=1}^{m} v_{ijkl}x_{ik}x_{jl} \tag{4.1}$$

$$\max Z_2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{l=1}^{m} u_{ijkl}x_{ik}x_{jl} \tag{4.2}$$

$$\max Z_3 = A(G_n) \tag{4.3}$$

$$G_n = \{ j \mid x_{ij} = 1 \} \tag{4.4}$$

$$s.t \qquad \sum_{j=1}^{m} x_{ij} = 1, \quad i = 1,..,n \tag{4.5}$$

$$\sum_{i=1}^{n} x_{ij} \le 1, \quad j = 1,...,m \tag{4.6}$$

$$x_{ij} \in \{0,1\}, \quad i = 1,...,n \;\; ; \quad j = 1,....,m \tag{4.7}$$

In the P 4.1 formulation, there are i=1,..,n commands and j=1,..,m gestures (n<m). The first and second terms of the intuitiveness objective (4.1) contain intuitiveness indices for the direct and complementary gesture-commands assignments. Higher values of v_{ij} will force gesture-command pairings, which are more intuitive. Similarly, higher values of the complementary intuitive indices, v_{ijkl} will force solutions in which these complementary command gesture pairs are matched. The comfort objective Z_2 (4.2) tends to pair high frequency use commands with less stressful gestures. The accuracy objective Z_3 (4.3) must be determined by a recognition

algorithm. In (4.7) the binary variable x_{ij} =1 represents an assignment of gesture j to command i, and 0 otherwise. Constraints (4.5) and (4.6) ensure that each command i is assigned a unique gesture, and each gesture j is assigned to no more than one command, respectively. To evaluate (4.3), a recognition algorithm must be called and solved for the particular GV represented by the 0-1 assignment variables. When there is more than one non-commensurable objective function to be maximized, solutions exist for which the performance in one cannot be improved without sacrificing performance in at least one other. Such solutions are called Pareto optimal points [Pareto, 1896], and the set of all such points forms the Pareto frontier. A solution x^* is a Pareto point iff there does not exist another solution y such that; $f_d(y) \geq f_d(x^*) \quad \forall\, d=1,..,D,$ and $f_d(y)<f_d(x^*)$ for some $d,$ where f_d is the D^{th} objective function.

Given that the gesture set is of size m and the command set of size n, there are m!/((m-n)!n!) different gesture subsets. For each subset of n gestures the total number of command-gesture matching is n! so that the total solution space for the MOP is m!/(m-n)!. The sub problem P 4.1, formed from (4.1), (4.5), (4.6), and (4.7) , is a quadratic 0-1-integer assignment problem. The P 4.1 was solved by a complete enumeration approach, which appears in Stern *et al.*[2004a]. The results are reproduced in section 0.

4.3 The dual priority problem

We relax P 4.1 by considering the recognition accuracy as a constraint (3.5), while combining the human objective measures into a single objective using (3.4) with the combination weights w_1 and w_2.

Recall that determination of recognition accuracy does not depend on the matched command-gesture pairs in the GV, but only on the subset of gestures, G_n. Thus, it is possible to use a decomposition approach whereby the first stage is to find a feasible solution that satisfies (3.5). In a second stage, this feasible solution is substituted in (3.4), and solved for optimal GV candidates. The first stage problem then is that of finding a gesture subset G_n from the set of all possible G_ns that satisfies a given minimal accuracy A_{min}. This feasible subset problem is stated below as P 4.2.

P 4.2 Stage 1: Feasible subset selection

Find one or all G_n s

s.t

$$A(G_n) \geq A_{min} \tag{4.8}$$

$$G_n \subseteq G_m, n \leq m \tag{4.9}$$

Because the accuracy function is unknown, the search for a feasible solution to P 4.2 is found through the use of two different metaheuristics as described in sections 4.4 and 4.5. Denote the feasible solution found from P 4.2 as G_n^* [v]. Let the gestures in G_n^* be reindexed as $\{g_i^* \mid j =$

[v] There are two versions of this problem: ver1: Find the set of all feasible solutions to P 4.2 called $g^*=\{G_n \mid (4.8)$ and (4.9) are satisfied.$\}$, and ver2: Find one single feasible solution to P 4.2, called G_n^*. In what follows we are finding the second version of the problem.

1,2,…,n}. For simplicity, and when understood, we will represent G_n* by the set of indices {1,2,..i,..,n} , where i represents the j^{th} gesture type in the feasible subset.

Once a single set of gestures G_n* is found, the second stage is initiated. Referring back to P 4.2, and using G_n* found in the first stage, the relaxed problem can be formulated as a quadratic integer assignment problem (QAP). Given a set of n commands (i=1,..,n), n gestures (j=1,..,n) and matrices: F = (f_{ij}), U = (u_{kl}), I = (a_{ik}). Define problem

P 4.3 4.3 below:

P 4.3 Stage 2: Matching Gn* to commands

$$\max_{p \in \Pi_n} \bar{Z}(G_n^*) = w_2 \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}\, \bar{s}_{p(i)p(j)} + w_1\left[\sum_{i=1}^{n} a_{i\,p(i)} + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij\,p(i)p(j)}\right] \tag{4.10}$$

Here, Π_n is the set of all permutations of the set of n integers in G_n*. In the first term, $\bar{s}_{p(i)p(j)}$ represents the comfort cost of the pair of assignments (i, p(i)) and (j, p(j)) (assigning commands i and j to gestures p(i) and p(j), respectively, scaled by the frequency of transition between commands i to j, f_{ij}). In the second term, $a_{ip(i)}$ is the direct intuitiveness of the assignment (i, p(i)) and $a_{ijp(i)p(j)}$ is the complementary intuitiveness of matching complementary commands (i, j) to complementary gestures (p(i), p(j)).

By defining a set of integer 0,1 decision variables {x_{ij}}, a quadratic assignment problem QAP(G_n*) can be formulated as P 4.4, which is equivalent to P 4.3. A network representation of the problem is shown in Figure 4.1.

P 4.4 QAP(G_n*)

$$\max \bar{Z}(G_n^*) = w_2 \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} u_{ijkl} x_{ik} x_{jl} + w_1\left[\sum_{i}^{n}\sum_{j}^{n} v_{ij} x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} v_{ijkl} x_{ik} x_{jl}\right] \tag{4.11}$$

$$s.t. \quad \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,..,n, \tag{4.12}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,...,n, \tag{4.13}$$

$$x_{ij} \in \{0,1\}, \quad i = 1,...,n, \quad j = 1,...,n, \tag{4.14}$$

Here, the x_{ij} binary assignment variable equals 1 if command i is assigned to gesture j and zero otherwise. Constraint (4.12) ensures that each command is paired with exactly one gesture. Constraint (4.13) ensures that each gesture is paired with exactly one command.

Many approaches have been proposed for the solution of the 0-1 integer QAP such as (i) find a proper linearization of the objective to obtain a linear program [Finke *et al.,* 1987] and (ii) relax the QAP to a (0, 1) linear integer program by introducing new binary variables $y_{ijkl} = x_{ij} x_{kl}$

and new constraints. A simulated annealing approach [Connolly, 1990] was adopted in this thesis to solve the QAP.

As a side note, one may start with the master set of gestures G_m, which corresponds to using m gestures and n nodes in the network of Figure 4.1 to create a giant problem QAP (G_m) with (4.11),(4.12),(4.13) replaced by (4.1), (4.2),(4.5) and (4.6).

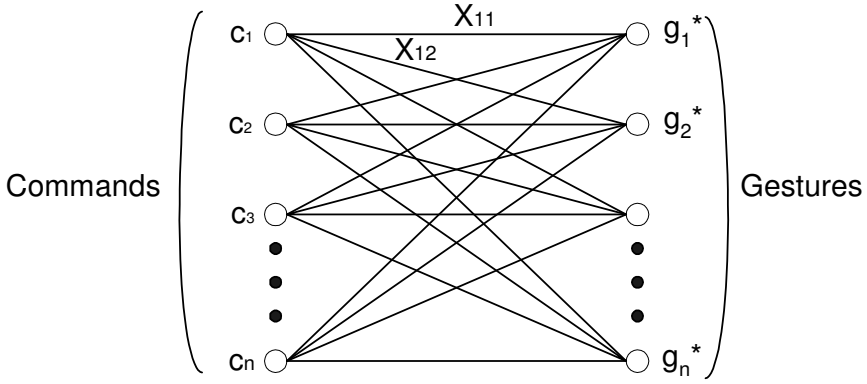


Figure 4.1. Representation underlying the quadratic assignment problem

The solution will determine simultaneously the subset of gestures G_n, and the command-gesture assignment. Hence, the subset G_n* is not given in advance, but is selected directly through constraint satisfaction. Since the decision variables are binary, a subset of exactly n gestures will be selected from the master set of size m. If a procedure is written to rank the solutions according to (4.10), then each solution can be tested to determine if it satisfies the accuracy constraint (3.5), and the first to do so is selected as the best.

4.4 Disruptive confusion matrix (DCM)

The procedure DCM starts with an initial solution for P 4.2 and searches for improved solutions moving in the direction of those with higher accuracies using a gesture interchange method, thus avoiding local optima traps. The procedure is initiated by the construction of an initial feasible solution. The confusion matrix corresponding to the accuracy associated with the current solution is used to provide clues for the disambiguation of confused gesture pairs. To aid in the resolution of ambiguities, the confusion matrix, $\mathcal{C}_n$, is disrupted by switching out (exchanging) the most confused gesture pairs (i',j') with others that have more discriminating power from the master set. We refer to this as a dual pair exchange (DPE). The most confused pair of gestures is found by the max c_{ij} rule:

$$\underset{i,j=1,..n}{argmax}\ c_{ij} = (i', j') \qquad \textbf{(4.15)}$$

where,

i', j' = the pair of the most confused gestures.

$c_{ij} = n_{ij}/n$ = level of confusion between gesture i and j.

n_{ij} = the number of times gestures i is recognized as gesture j.

n = the total number of gesture samples.

This generates two new gesture subsets, G_n^1, G_n^2, which must be evaluated. These sets are constructed from G_n as follows: For G_n^1, gesture j' is discarded and replaced by the most dissimilar unused gesture j" found in the master set. For G_n^2, gesture i' is discarded, and replaced by the second most dissimilar unused gesture i" found in the master set. The rules for determining the replacement gestures $g_{j"}$ and $g_{i"}$ are described later in this thesis.

A record of repeated operations of this type is maintained in a binary tree. The nodes of the tree represent gesture subsets. Associated with each node is a triplet (G_n, $A(G_n)$, δ), where $\delta = A(G_n) - A_{min}$. If the solution is not feasible with respect to A_{min}, δ will be negative. The initial node of the tree is associated with the initial solution, obtained in the construction phase. Branching is conducted after the two most confused gestures associated with the tree node are identified and replaced to create two new descendent nodes, each associated with a new gesture subset. Figure 4.2 shows a flow chart of the DCM method.

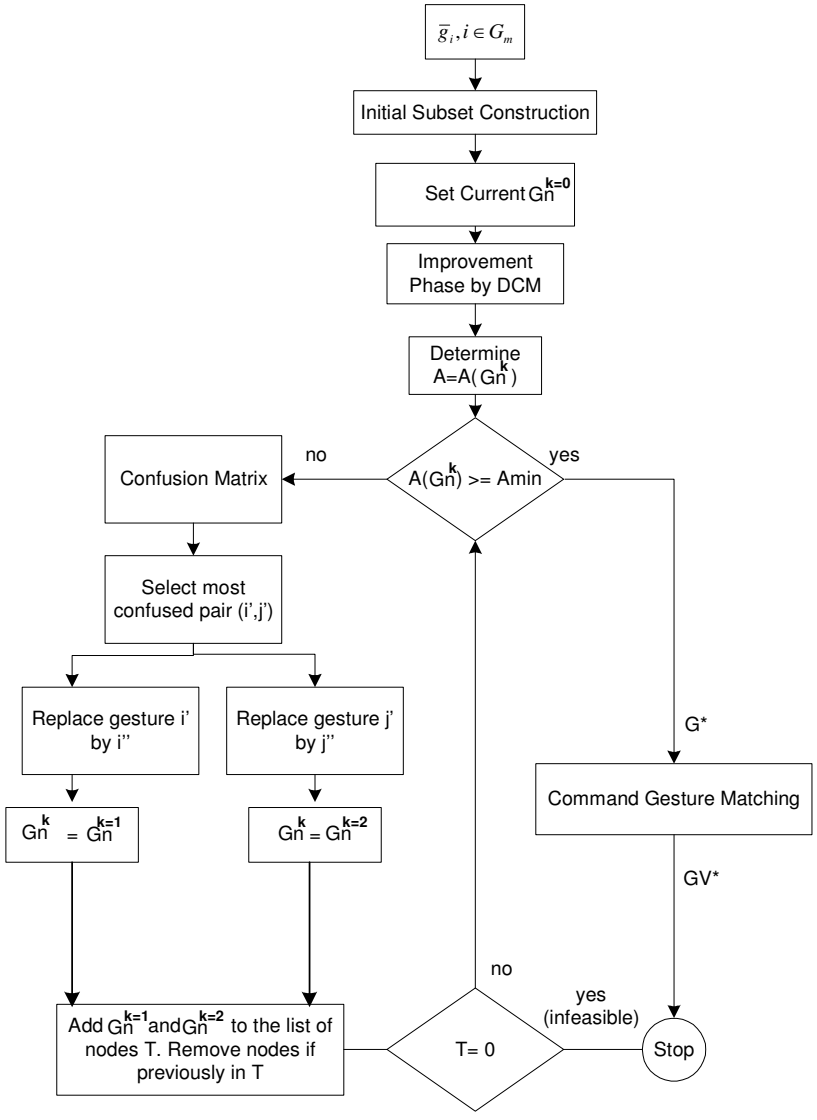


Figure 4.2. Flow chart of the DCM method

4.4.1 The subset search tree

Assume that now node k, and δ is negative. We have determined the most confused pair of gestures, in the current set G_n^k, g_i' and g_j'. We create and branch to two new descendant nodes corresponding to G_n^{k1} and G_n^{k2}. These nodes are placed in a list T, where T is the list of all unevaluated nodes. There are two possible branching rules to be considered: "Depth First" and "Flooding."

All unevaluated nodes in the tree are placed in a list T. If any of these nodes are already in T, they are removed. Thus, a node in the tree can be terminated if its two offspring have previously been generated (otherwise, cycling will occur). Offspring nodes have been generated by an exchange of its elements to create neighbor solutions in such a way as to disrupt the confusion between gestures; thus, they have a potential for greater accuracy. This, however, cannot be guaranteed to be successful at all times, and it may happen that offspring nodes at the next lower level have accuracy levels higher than those from which it has been descended. Thus, there is really no rationale or verified advantage of pursuing an exploitive path through a depth first branching rule of selecting the current best solution. This leads us to use a flooding branching policy.

In flooding, the nodes are evaluated one level at a time from left to right, until the desired accuracy is attained, all nodes are terminated or the maximum number of prespecified levels are reached. Using this method the current node evaluated may exhibit a lower accuracy than those previously examined. Thus, we have increased exploration of the solution space by examining nodes with lower accuracy values. Here again a node in the tree is terminated if its two offspring have previously been generated. (Otherwise, cycling will occur). Although this method appears to be inefficient, the strength of the generation of neighborhood solutions by pair-wise exchanges has been shown to result in a small number of evaluations compared to pure enumeration

Given the binary tree generated by the flood branching rules, index its nodes as i = 0,1,2… where node 0 is the root node. Let the levels of the tree be indexed k = 0,1,2 where the nodes at level k are indexed from (2k -1) to 2(2k-1), and the left son LS(i) and right son RS(i) of node i are 2i+1 and 2i+2 , respectively.

Let K be the deepest level of the tree we designate (based on how much computational power we want to use). Note that the root of the tree corresponds to the initial feasible solution, G_n^0. The stopping rule is (a) $\delta(\iota)>0$, or (b) current level of the tree is K. The exit condition occurs when the recognition accuracy of one of the sub-problems is higher than the specified accuracy level A_{min}. Then a feasible solution to P 4.2 has been found. Alternatively, one can continue with the tree search to find as many solutions G_n as possible that have $A(G_n) \geq A_{min}$ .To avoid cycling, each new node is checked to see if it has been generated earlier, in which case the node is terminated. If all nodes are terminated the problem is found to be infeasible. It is also possible to place a limit on the number of levels generated in the tree to avoid the possibility of excessive computation time. Once the feasible solution is found (or all feasible solutions are found) the gesture-command mapping P 4.4 is then solved to obtain GV*.

4.4.2 Phase A: Initial subset construction

To find the initial G_n from the master set G_m, two heuristic methods are proffered. The first is based on maximizing the inter-gesture distances and is formulated as a quadratic 0-1-integer problem. The second is based on the Max-Min inter-gesture distance, and it is solved by a simple algorithm. Both require the construction of a square matrix D with common element d_{ij}, (the smaller d_{ij}, the more similar the gestures).

$d_{ij} = d(\bar{g}_i, \bar{g}_j)$

$\bar{g}_i, \bar{g}_j$ = the prototype vector of gesture type i, j.

This prototype vector is obtained by finding the centroid of the feature vectors of a training set of the same gesture type.

4.4.2.1 First initial G_n selection method: Max 0-1 integer quadratic problem

A 1-0 integer quadratic program, P 4.5, is designed to select an initial solution, represented by the subset G_n^0, from G_m. This program selects G_n such that the total inter-gesture distance of all gestures in G_n is maximized.

P 4.5 Initial solution G_n (max inter-gesture distances)

$$V = \max \sum_{i}^{m} \sum_{j}^{m} x_i d_{ij} x_j \tag{4.16}$$

s.t.

$$\sum_{i=1}^{m} x_i = n \tag{4.17}$$

$$x_i \in \{0,1\}, \qquad i = 1,...,m$$

The objective, V, represents the total intergesture distances in G_n, and x_i is a binary selection variable equal to 1, if gesture i is selected, and 0 otherwise. The constraint assures that only n gestures are selected from the master set of size m. The initial subset G_n^0 = {all g_i such that x_i^0= 1} where { x_i^0 }, is the optimal solution to P 4.5.

4.4.2.2 Second method for initial G_n selection: MaxMin 0-1 integer program

This method of generating an initial feasible G_n is very simple. The objective is to find a G_n among all possible G_n's from G_m, such that the least discriminating pair of gestures (those two gestures in G_n that have minimal similarity between them) is maximized over all subsets G_n. The problem is stated below as P 4.6.

P 4.6 Initial solution Gn (max min intergesture distance)

$$G_n^0 = MaxMin\{d_{ij} \mid (g_i, g_j) \in G_n\} \tag{4.18}$$

$$G_n \subseteq G_m$$

This can be solved simply by a threshold type operation, where the closest pair of gestures in G_m is removed until the number of gestures remaining is n. The remaining gestures is then the initial subset G_n^0

The threshold algorithm

1. Let the number of gestures be $N = G_m$
   (a) If m is even, repeat steps 2 and 3, 4 (m-n)/2 times and go to 7.
   (b) If m is odd, repeat steps 4 and 5 [(m-n)/2]-1 times then go to step 6.
2. Find min d_{ij} in the matrix D
3. Remove the corresponding column and row and update D. Place i and j in the set of nodes N
4. Find min d_{ij} in the matrix D.
5. Remove the corresponding column and row and update D.
6. Find the min d_{ij} in the matrix D. In row i and column i without d_{ij}, find the next smallest d_{ij}, say $d_{i'j'}$
   If it is found in a row i' =i then remove column j' and place j' in N.
   Otherwise, if it is found in a column j'=j, then remove row i' and place i' in N.
7. Stop. Set $G_n^0 = N$

Algorithm 4.1 The threshold algorithm

4.4.2.3 $A(G_n)$: Accuracy of the set of gestures, G_n

Once a new subset of gestures G_n is obtained, its accuracy $A(G_n)$ is determined by calling the recognition algorithm. Removing one gesture and replacing it by a new one affects the partition obtained by the FCM algorithm. Thus, the FCM classifier must be retrained. As a side note, to speed up the computations in training the FCM clustering algorithm, it is wise to choose as the initial cluster centers those close to the last optimal positions. The result of the training session is a confusion matrix, which is used to guide the next DPE and subsequently, the branching of the search tree.

4.4.3 Phase B: Improvement by DCM

Given an initial solution constructed by one of the methods described above, a gesture pair exchange method is used to find an improved solution. In this dual pair exchange (DPE) method, the two most confused gestures in the confusion matrix are exchanged with two gestures from the master set (in this way we "disrupt the confusion matrix"). A record of repeated operations of this type is maintained through the construction of a binary tree. The nodes of the tree represent gesture subsets. The initial node of the tree is associated with the initial solution. Corresponding to the gesture subset of each node the two most confused gestures are identified and replaced to create two new descendent nodes.

4.4.3.1 Dual pair exchange (DPE)

Here a double set of pairs of gestures is exchanged. A binary tree is constructed to keep track of the exchanges. The information stored at each node of the tree is the subset of gestures, its accuracy measure, and the corresponding confusion matrix. Branching takes place from a father node to two new offspring nodes. The left and right offspring correspond to new gesture subsets obtained by the replacement of its confused gesture with that of a new gesture selected from the master set. The new gesture must be selected so that it can be easily discriminated from the remaining gestures in the gesture subset. Two rules for selecting the gestures to be removed from the current subset of gestures, plus a rule for selecting new gestures form the master set to replace those discarded, are provided. The following notation will be helpful.

$G_{n-1}(i) = G_n - \{ g_i \}$, the reduced set after removing g_i from G_n (the gestures in $G_{n-1}(i)$ reindexed as $i = 1,2,\ldots,n-1$ when convenient)

$G_{m-n} = G_m - G_n$ is the set difference, with gestures reindexed as k =1,2,..,m-n when convenient.

4.4.3.2 Discard rule D_1

The two gestures for which it is hardest to discriminate are intuitively those gestures that have the largest confusion (off diagonal) value in $\mathcal{C}_n$. For example, for the confusion matrix in Table 4.1, the two most confused gestures can be found as 3 and 2 by:

$$\arg Max\left(c_{ij} : \forall ij \neq ii\right) = i'\ j' = 3,2 \qquad \textbf{(4.19)}$$

where, c_{ij} is the number of samples of gesture *i* that was classified as gesture *j*.

Table 4.1. Sample confusion matrix

gi-gj	1	2	3	4	5	sum
1	4	0	0	1	0	**1**
2	0	4	0	1	0	**1**
3	0	2	3	1	0	**2**
4	1	0	1	2	1	**3**
5	1	1	1	1	1	**4**
sum	**2**	**3**	**2**	**4**	**1**	

4.4.3.3 Discard rule D_2

Although D1 is intuitive and simple to implement, it has its disadvantages when the misclassified gestures are distributed evenly over the confusion matrix. This is the case in Table 4.1 where the off diagonal totals of the matrix are recorded in the last row and column. One sees from the maximum of the row totals of the matrix that gesture 5 is the most confused as it was misclassified 4 out of 5 trials. Also, gesture 4 is problematic as it is the gesture that attracted the most (4 of them) misclassified gestures. This may be due to the fact that samples for this gesture may not have been sufficiently compact and dense. This suggests Algorithm 4.2 for selecting the two most confused gestures:

1.Let NR_i = be the total number of off diagonal positive entries in row i. Let NR_j = be the total number of off diagonal positive entries in column j.
2. Find the gesture v' with the maximum number of off diagonal entries, i.e.;
$arg\ Max\left(NR_i, NR_j : \forall ij\right) = v$
3. Find the gesture with the second maximum off diagonal totals.
$arg\ Max\left(NR_i, NR_j : \forall ij \neq v'\right) = w$

Algorithm 4.2. Two most confused gestures

Gestures v and w are selected as the first and second most confused gestures. Any ties are broken arbitrarily. For the above example, gestures 5 and 4 were selected. We can define the best replacement gesture to enter G_{n-1} (i) as the most discriminated gesture in comparison with the gestures that remain after using the discard rule, over all the 'free' gestures in G_{m-n}. Calculate the centroid, c, of all the feature vectors in G_{n-1} (i). Find the gesture g_k in G_{m-n} that is the farthest from c. This rule is problematic since there may still be a very bad gesture in $G_n(-g_i)$ that is very dissimilar to g_k, but is not considered as such since it is averaged out with very good gestures in $G_n(-g_j)$. Instead we use a MaxMin replacement rule.

4.4.3.4 Replacement rule

The MaxMin replacement rule selects a gesture from the master set that is least similar (farthest away) from all the gestures in G_{n-1} (i). To clarify this notion we use the following notation.

Let g_i and g_j be the selected pair of most confusing gestures in G_n. Suppose we want to replace gesture g_j with a gesture g_k from the master set G_{m-n}. Use a reduced distance matrix $\overline{D}$ of size (m-n) x (n-1). The following MaxMin replacement rule finds the replacement gesture k*:

$$arg\, MaxMin\{d_{ki}\} = k^* \qquad \textbf{(4.20)}$$

$$k \in G_{m-n}, i \in G_{n-1}(i)'_n$$

This can be easily carried out using the distance matrix $\overline{D}$ as follows:

1. For a given gesture *k* in the master set, find i', the most similar gesture in the current gesture subset G_{n-1} (i)

$$arg\ \ Min\ \ \{d_{ki} \mid i \in G_{n-1}(i)\} = i';\quad \forall k \in G_{m-n} \qquad \textbf{(4.21)}$$

2. Now find the gesture k* from the master set that is farthest away from the nearest gesture in G_n'

$$\arg Max(d_{ki'} \mid k \in G_{m-n}) = k^* \qquad \textbf{(4.22)}$$

This rule has complexity O(nm) as it takes (n-1)(m-n)log(m-n) to find the row min values and (n-1)log((n-1) to find the max of these.

4.4.4 Phase C: Command-gesture matching

In this phase the feasible gesture G_n* found from the DCM procedure is matched to commands, using P 4.4, to obtain the final GV.

4.5 Confusion matrix derived solution (CMD)

An alternative metaheuristic approach to find the feasible gesture set for P 4.2 is the Confusion Matrix Derived Solution Method (CMD) method. This method is initiated by finding the accuracy of the gesture master set G_m. A confusion matrix $\mathcal{C}_m$ is created for the G_m problem from which the recognition accuracies associated with various gesture subsets G_n are estimated. The set of gestures G_n with accuracies above A_{min} are feasible solutions that can be approximated from the general confusion matrix built for the G_m set of gestures. The CMD method consists of three phases: (i) create a confusion matrix for the G_m subset of gestures, $\mathcal{C}_m$, (ii) solve P 4.2 by finding a subset of gestures G_n with the highest recognition accuracy that meets the minimum accuracy constraint, (iii) repeat the previous steps until a given number of solutions are found or all the solutions that meet the minimum accuracy constraint are found, and (iv) solve P 4.4 (ver1). This is done for all the solutions obtained, and the procedure for it is described in section 0. The confusion matrix is obtained directly from the sample partition result using the supervised FCM optimization procedure.

To solve P 4.2, one may either search for all the feasible G_n's that have accuracies $A \geq A_{min}$ or terminate the search after a fixed given number of solutions have been found. The confusion

derived routine (CD) is used to find the subset of least confused gestures. It receives the parameters G_n, which is a subset of gestures (of any size); j indicates the current number of the solution, and A_{min} is the minimum recognition accuracy accepted.

$$\underset{i=1,..n}{argmax} \quad \{C_{ii}\} = C_{i'i'} \tag{4.23}$$

where,

i' = the least confused gesture.

$C_{ii} = n_{ii} / n$ = rate of gesture i being recognized correctly as gesture type i.

n_{ii} = the number of times gestures i was recognized correctly.

n = the total number of gesture samples.

Let j be the current solution number and A_{min} be the minimum recognition accuracy accepted. The first time that this algorithm is called, $G_n=\emptyset$ and j=0. If (4.23) returns more than one solution, ties are broken arbitrarily. This algorithm is similar to the greedy algorithm used to solve the knapsack problem [Martello and Toth 1990]. A set of N feasible solutions G_n can be obtained, using the following steps in the Confusion Matrix Derived Solution method (CMD algorithm), Algorithm 4.4.

The CD routine (G_n , j, A_{min})

1. Let the number of gestures $n=|G_n|$. Let c be the number of commands
2. Repeat (c-n) times.
3. Find the least confused gesture *i* in the confusion matrix $\mathcal{C}_m$ -G_n using (4.23).
4. $G_n = G_n \cup i$
5. Remove the corresponding column and row *i* from $\mathcal{C}_m$.
6. Go to step 2
7. Calculate A using $\mathcal{C}_n$
8. Stop. If $A \geq A_{min}$, then $G_n^j = G_n$ is a feasible solution and G_n^j is kept a feasible solution subset.

Algorithm 4.3. The confusion derived routine (CD)

The CMD algorithm(N, A_{min})

1. $G_n = \phi$
2. j=0
3. G_n^j=CD (G_n , j, A_{min})
4. Calculate A using $\mathcal{C}_n$
5. If $A \geq A_{min}$ then Add G_n^j to the feasible solution subset, else Exit
6. Take out the highest confused gesture *i* from G_n^j.
7. Remove the corresponding column and row *i* from $\mathcal{C}_m$.
8. G_n^{j+1}=CD(G_n^j,j)
9. If G_n^{j+1}belong to the feasible solution subset:
   9.1 . Take out the highest confused gesture *k, k!=i,* from G_n^j
   9.2 Restore the corresponding column and row *i* from $\mathcal{C}_m$.
   9.3. Go to 8.
10. If $A \geq A_{min}$, then add G_n^{j+1} to the feasible solution subset
11. Restore $\mathcal{C}_m$ to the original
12. If j<N and $A \geq A_{min}$, return to 6.

Algorithm 4.4. Confusion matrix derived solution (CMD)

Let N be the number of solutions requested. The CMD algorithm obtains N solutions, or all the solutions with associated accuracies above a given minimum allowed A_{min}. In every iteration of the CMD algorithm, a solution is created each time by excluding a different gesture from the subset of gestures of the current solution and adding a new gesture from the master set. Figure 4.3 shows a flowchart of the DCM method.

The following example shows how the CMD Algorithm can be used to obtain a subset of three solutions. Let $\mathcal{C}_m$ be the following matrix

Let m be $|G_m| = 12$
Let n be $|G_n| = 8$

After applying the supervised FCM optimization procedure, the recognition accuracy associated with G_m is A=93.54%.

Applying the CMD Algorithm with N=5 and A_{min}=96.56% (Table 4.2), we get the following five solutions:

G_n^0={4,6,7,8,9,10,11,12} A=96.87%
G_n^1={2,4,7,8,9,10,11,12} A=96.87%
G_n^2={2,4,6,7,9,10,11,12} A=96.87%
G_n^3={2,4,6,7,8,9,11,12} A=96.56%
G_n^4={2,4,6,7,8,9,10,11} A=96.56%

In the example above the algorithm CMD is finished when the number of solutions is five.

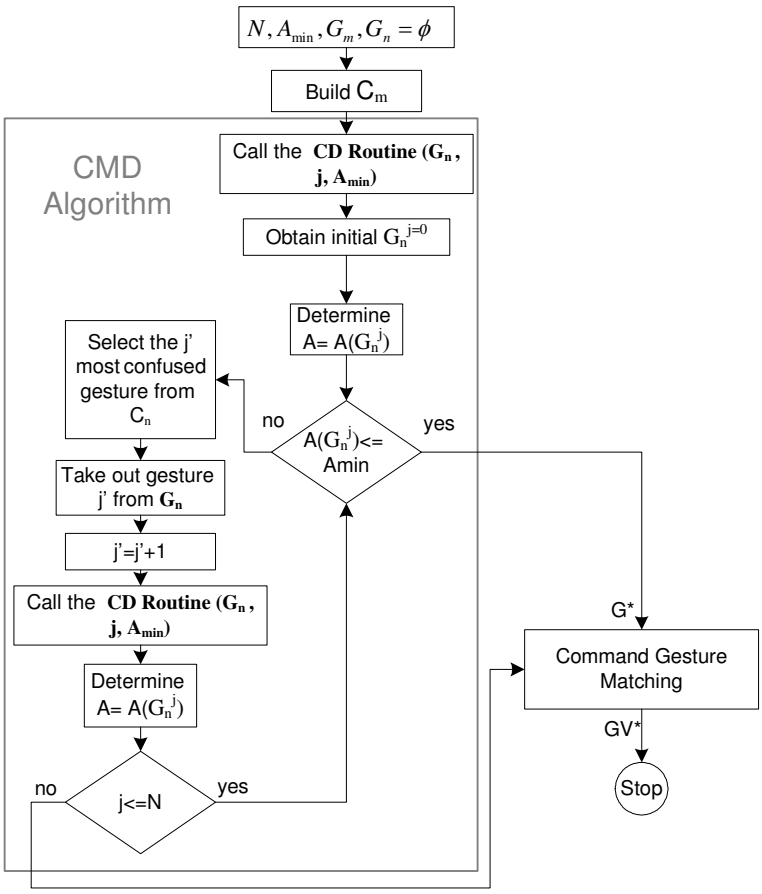


Figure 4.3. Flowchart of the CMD method

Table 4.2. Sample confusion matrix II

gi \ gj	1	2	3	4	5	6	7	8	9	10	11	12
1	33	0	0	0	0	0	0	0	0	0	0	0
2	0	37	0	0	0	0	0	0	0	0	0	0
3	0	0	34	0	0	5	0	1	0	0	0	0
4	0	0	0	40	0	0	0	0	0	0	0	0
5	0	0	0	0	35	0	0	0	0	0	0	0
6	0	0	0	0	0	37	0	3	0	0	0	0
7	0	0	0	0	0	0	40	0	0	0	0	0
8	0	0	0	0	0	5	0	37	0	0	0	0
9	0	0	0	0	0	0	0	0	40	0	0	0
10	0	0	0	0	0	0	0	0	0	38	0	0
11	0	0	0	0	0	0	0	0	0	0	40	0
12	1	0	0	0	0	0	0	0	0	0	0	38

4.6 Illustrative examples

The DCM method is illustrated using a small example with twelve gestures in the master set, and eight commands as shown in Figure 4.4.

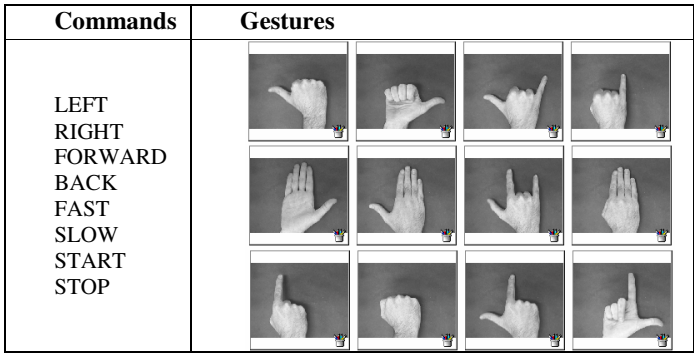


Figure 4.4. Hand GV

Note that at this point, the gestures have no labels associated with them and are only represented as gesture types g_0, g_1,g_2,..,g_i,.. g_{11}. The solution space for this small example is 495 (m!/((m-n)!n!). Two examples are solved using the DPE for two different initial solution methods, Max Obj (P 4.5) and MaxMin Obj (P 4.6), called Ex 1 and Ex 2, respectively. For both examples, we use the maximum value in the confusion matrix to find the gestures to drop (discard rule D1), and the MaxMin rule to select the replacement gestures.

The command-command transition frequencies (Table 4.3(a)) were obtained by an experiment to maneuver a VMR through a maze (Module 1.1-1.3). The experiment was repeated seven times, and the totals of each command transition were recorded in the matrix f. The orders of the columns and rows, indexed from 0 to 7, correspond to the commands listed in Figure 4.4

Table 4.3. Matrices. a) Frequency F, b) Intuitiveness Z_1 c) Comfort Z_2 matrices

$$f = \begin{bmatrix} 302 & 1 & 11 & 0 & 3 & 1 & 0 & 0 \\ 0 & 200 & 8 & 0 & 2 & 0 & 0 & 0 \\ 15 & 8 & 88 & 2 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a)

$$Z_1 = \begin{bmatrix} 100 & 0 & 0 & 27 & 36 & 73 & 45 & 20 \\ 0 & 100 & 9 & 9 & 9 & 64 & 36 & 20 \\ 9 & 91 & 55 & 0 & 18 & 0 & 73 & 70 \\ 82 & 9 & 64 & 82 & 0 & 9 & 100 & 80 \\ 36 & 36 & 36 & 46 & 27 & 27 & 64 & 40 \\ 55 & 45 & 46 & 91 & 9 & 100 & 91 & 83 \\ 73 & 27 & 100 & 73 & 45 & 82 & 55 & 15 \\ 46 & 55 & 18 & 100 & 100 & 91 & 82 & 30 \\ 91 & 18 & 82 & 36 & 55 & 55 & 27 & 23 \\ 18 & 82 & 91 & 18 & 64 & 45 & 18 & 23 \\ 64 & 64 & 27 & 55 & 82 & 18 & 0 & 13 \\ 27 & 73 & 73 & 64 & 73 & 36 & 9 & 15 \end{bmatrix}$$

(b)

$$Z_2 = \begin{bmatrix} 40 & 35 & 45 & 45 & 64 & 78 & 96 & 96 & 44 & 40 & 81 & 56 \\ 53 & 47 & 50 & 50 & 77 & 72 & 90 & 90 & 57 & 53 & 94 & 49 \\ 45 & 50 & 43 & 43 & 70 & 76 & 94 & 94 & 49 & 45 & 86 & 53 \\ 45 & 50 & 43 & 43 & 70 & 76 & 94 & 94 & 49 & 45 & 86 & 53 \\ 64 & 77 & 70 & 70 & 16 & 59 & 68 & 68 & 37 & 40 & 33 & 48 \\ 78 & 72 & 77 & 76 & 59 & 59 & 77 & 77 & 82 & 78 & 81 & 71 \\ 96 & 90 & 94 & 94 & 68 & 77 & 68 & 68 & 100 & 96 & 63 & 87 \\ 96 & 90 & 94 & 94 & 68 & 77 & 68 & 68 & 100 & 96 & 63 & 87 \\ 44 & 57 & 49 & 49 & 37 & 82 & 100 & 100 & 37 & 44 & 73 & 59 \\ 40 & 53 & 45 & 45 & 40 & 78 & 96 & 96 & 44 & 40 & 81 & 56 \\ 81 & 94 & 86 & 86 & 33 & 81 & 63 & 63 & 73 & 81 & 0 & 96 \\ 56 & 49 & 53 & 53 & 48 & 71 & 89 & 89 & 59 & 56 & 96 & 48 \end{bmatrix}$$

(c)

Table 4.3(b) shows intuitive indices for each gesture-command (row-column) pair. The values are normalized in the range of zero to 100 with 100 representing the most intuitive. These indices are the collective subjective assessments obtained from subject queries. Table 4.3(c) contains stress indices for individual gestures and the transition movements between them. They were assessed from a hand biomechanics study [Natan *et al.,* 2003], (Module 1.4). In addition, twelve complementary intuitiveness indices a_{ijkl} are set to 100 for complementary command pairs (i,j) = (0,1), (2,3), (4,5), (6,7) and complementary gesture pairs (k,l)= (0,1), (2,3), (6,9). Command pairs are represented as follows: (left, right), (forward, back), (fast, slow), (start, stop); gesture pairs use the following notation: (g_o, g_1), (g_2, g_3), (g_8, g_9), respectively. All other a_{ijkl} are set to zero. All gestures are right handed. Complementary gestures are obtained by flipping the hand at the wrist to create mirrored images. Thirty images of each gesture type, collected from six subjects, are used to train the FCM recognition algorithm (see Wachs *et al.* [2003] for further details) (Module 2.1) The recognition system is said to be independent since in practice it can be used by multiple subjects.

4.6.1 Example 1 (using max rule for initial solution) (Module 2.2-3)

The Initial subset of eight gestures found by using the Max 0-1 Integer Quadratic P 4.5 is G_8^0={1,2,3,4,7,8,9,11}. The accuracy associated with this subset is 97.08%. The seven misclassified gestures can be shown in the confusion matrix in Table 4.4, where it can be seen that the most confused pair of gestures is 4 and 8.

Table 4.4. Confusion matrix showing the most confused pair

gi \ gj	1	2	3	4	7	8	9	11
1	30	0	0	0	0	0	0	0
2	0	30	0	0	0	0	0	0
3	0	0	28	2	0	0	0	0
4	0	0	0	27	0	**3**	0	0
7	0	0	0	0	30	0	0	0
8	0	0	0	0	0	30	0	0
9	0	1	0	0	0	0	28	1
11	0	0	0	0	0	0	0	30

This confusion matrix is disrupted by a DPE using the MinMax replacement rule. The new subsets are G_8^1 = {0,1,2,3,7,8,9,11}, and G_8^2 = {0,1,2,4,7,9,11}. The G_8^1 subset is found by dropping gesture type 4 and exchanging it for gesture type 0 from the master set. Table 4.5 shows that gesture 0 as the most dissimilar gesture of all the gestures in G_8^1 - {g_4}, according to the MaxMin replacement rule.

Table 4.5. Exchanging gestures 4 and 0 using the MinMax replacement rule

	1	2	3	4 (OUT)	7	8	9	11
0	469493	193243	28276	-	313311	45042	78541	51671
5	436592	138656	27527	-	315620	39741	36418	23568
6	151254	19566	178675	-	121280	168717	148794	137238
10	552455	214477	10084	-	415043	8532	33681	16565

Figure 4.5 shows the improvement tree. The search is terminated at node 4, with G_8^4= {0,1,2,3,5,7,8,11}, which has an accuracy of 100 percent. Note that using this metaheuristic, the best solution was found after creating and evaluating only four solutions out of a solution set of 495.

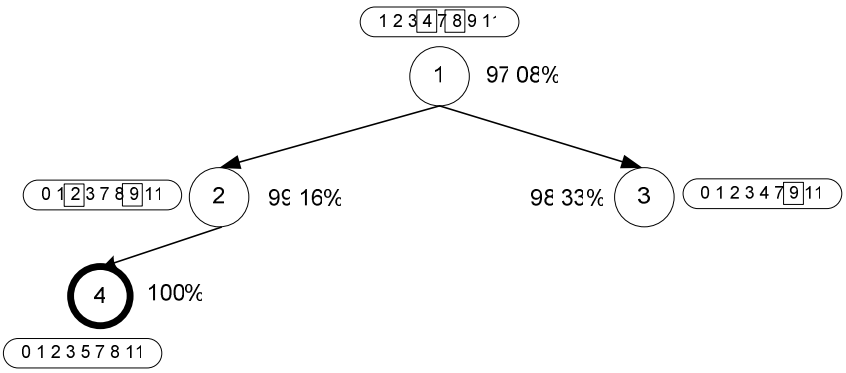


Figure 4.5. Improvement tree for Ex. 1

Using the best subset of gestures found after the improvement tree, these gestures are matched to commands by solving the binary integer quadratic assignment problem QAP(G_n) P 4.4, with $G_n = G_8^4$. Here, intuitiveness and comfort are assigned equal weights, such that $w_1 = w_2 = 1.0$. The resulting values of solving the matching problem were 1258032 and 29303 for the intuitiveness of representing each command by its associated gesture, Z_1(GV) and for the total comfort to perform the gesture, Z_2(GV), respectively. The final GV is shown by the matching of gestures to

commands in Figure 4.6. The two complementary gesture pairs (g_o, g_1) and (g_2, g_3) appearing in the subset were successfully matched with the complementary command pairs (left, right) and (forward, back), respectively. These gesture pairs were not matched with (start, stop) and (fast, slow) as these commands contained low frequency of use weights.

4.6.2 Example 2 (DPE with MaxMin rule for the initial solution) (Module 2.2-3)

Here we use the initial solution $G_n^0 = \{1,2,3,6,7,8,10,12\}$ found by solving P 4.6. The solution tree is shown in Figure 4.7. Note that previously generated nodes have been terminated. Such nodes can be identified in the graph by "cyclic arcs" (those without arrowheads) which emanate from them and connect to previously generated nodes at a higher level (e.g., (10,7)), or to nodes at the same level (e.g., (15,11).

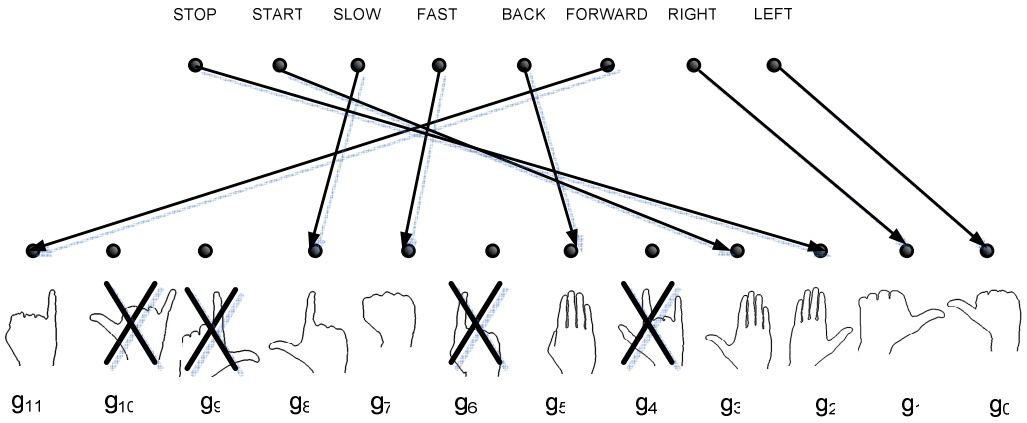


Figure 4.6. Ex 1 command – gesture matching found by solving the QAP(G_n)

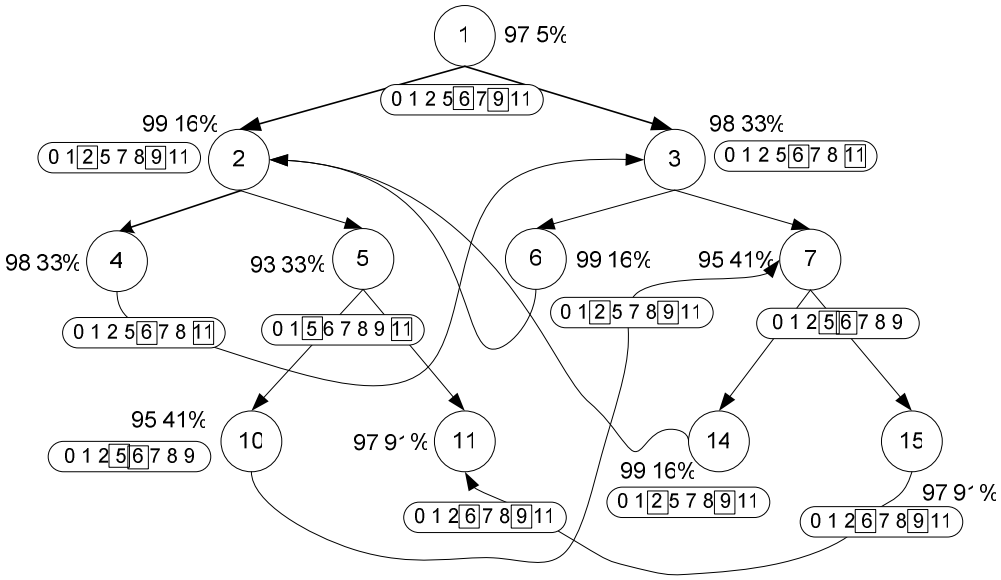


Figure 4.7. Improvement tree for ex. 2

The best solution found is (0,1,2,5,7,8,9,11) at node 14, with an accuracy of 99.16%. The best-matched commands for this gesture subset are depicted in Figure 4.8. Note again that the two complementary gesture pairs, which appeared in the selected subset, are matched with complementary command pairs.

4.6.3 Example 3 Solution to the multi-objective problem

The same small example of 12 gestures and 8 robotic arm commands is also considered in this case. For this problem, the size of the GV solution space is 495. Considering a GV has 8! possible matchings, the solution space is ~ 20×106. By examining each gesture subset in turn we select the best command-gesture matching by solving a quadratic program comprising a quadratic objective (4.1)+(4.2) subject to the constraints (4.5), (4.6), and (4.7). This assumes the human factor weights w_1 and w_2 are initially 1; however, in Section 6.5.5, different combinations of weighs will be used to regulate the impact of each of the human factors[vi]. Here the indices i, j, k, l are placed in correspondence to the n gestures selected in the subset. The optimal assignment variables are used to obtain the intuitiveness, $Z_1(GV)$, and comfort, $Z_2(GV)$, performance values. To evaluate $Z_3(GV)$, a recognition algorithm must be called and solved for the particular gesture subset under consideration. Each result can be viewed as a point in 3D space whose coordinates are intuitiveness, comfort, and accuracy, thereby allowing the decision maker to select the desired solution based on his internalized priorities (Figure 4.9). To aid the decision maker, we also provide the Pareto optimal points shown in the same image and in Table 4.6.

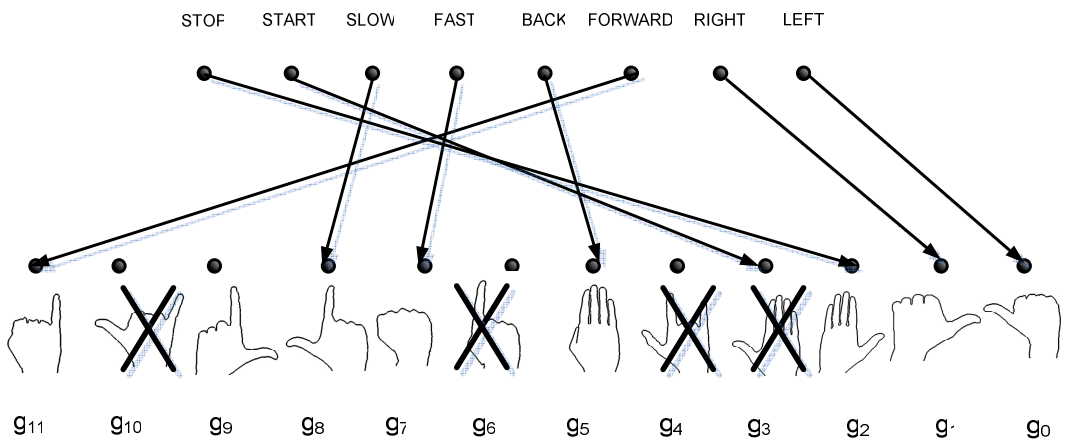


Figure 4.8. Ex 2 command–gesture matching found by solving the QAP(G_n)

Table 4.6. Pareto points for the MOP example

Pareto Pts	Accuracy(%)	Intuitiveness (%)	Comfort(%)
1	100	66.88	99.56
2	98.33	100	18.57
3	99.16	64.47	100

[vi] The empirical values of w_1 and w_2 can be determined by multiple linear regression between the task completion time variable and (4.11) for several GVs.

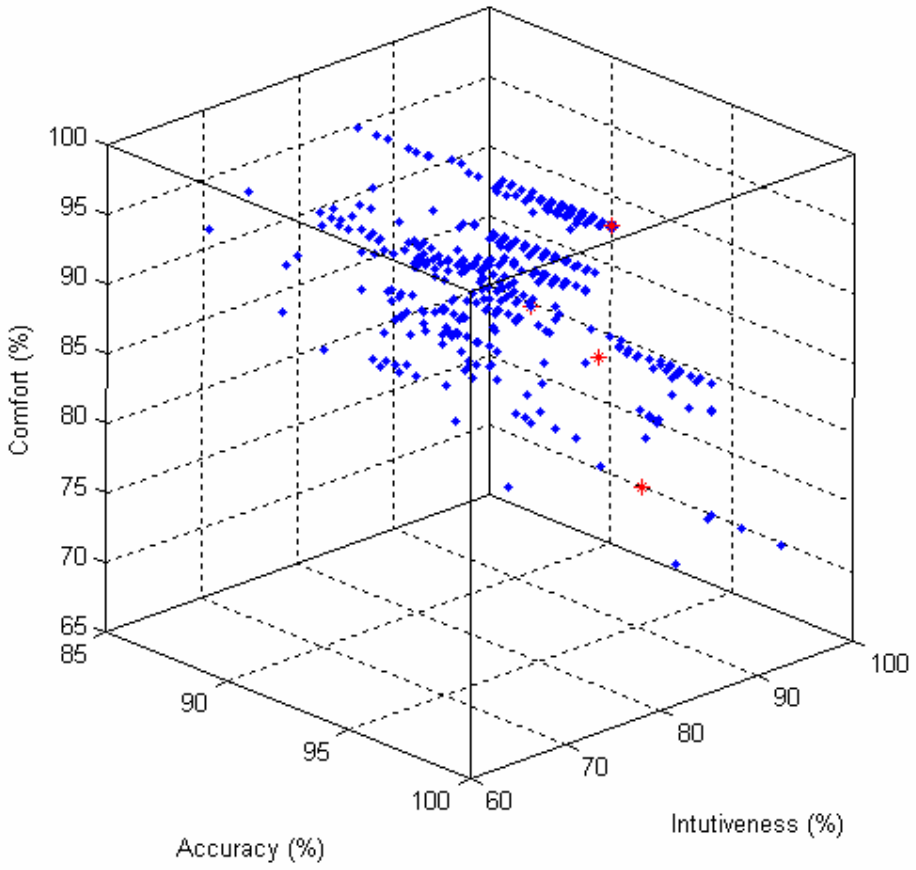


Figure 4.9. 3D plot of GV solutions

4.7 Discussion

In this chapter we discussed the mathematical programming formulation and solution approaches for three analytical methods: MOP, a tree based exchange search metaheuristic (DCM), and a search based on the gesture master set associated confusion matrix (CMD). All methods reflect the ergonomic and technical performance measures upon which a GV control system is judged. A useful feature included in the formulation is the ability to match opposing complementary pairs of gestures to complementary commands.

By posing the optimal GV design problem as a MOP, solutions can be presented as 3D representations, including Pareto optimal ones. This allows the designer to have an overview of possible solutions and select one based on his/her preferences. Calculating the entire Pareto set for larger problems is computationally prohibitive and requires an approach such as an evolutionary multicriteria procedure.

The metaheuristic approaches, the main topic developed in this chapter, are a variation of the MOP in which the objectives are given priorities. The first objective, maximum accuracy, is given the first priority and must be satisfied at some given, acceptable level. The human centered measures of intuitiveness and comfort are given second priority. The metaheuristic for the dual priority problem is based on a two-stage decomposition approach. In the first stage, a feasible gesture subset (or set of feasible subsets) is found which satisfies a minimum acceptable accuracy level (see the left box inside Module 2 in Figure 3.1). Two methods have been developed: the first is a disruptive confusion matrix method (DCM) to create a tree search metaheuristic. To address the problem of repeated training and parameter calibration of a recognition system for

each candidate subset of gestures in the tree, a second method was introduced: the confusion matrix derived solutions method (CMD). In the CMD, the FCM parameter calibration functionality is used only once for the master set of gestures. Using the confusion matrix corresponding to the gesture master set, gesture subsets are extracted, and their approximate recognition accuracies are derived. The second stage uses a QAP to assign the selected gestures to commands such that the human centered measures are optimized. Three examples are solved to illustrate the procedure. The first two use the DCM method, with two different strategies for obtaining the initial solutions. The last example uses a complete enumeration policy to address the solution of the MOP problem.

Examples presented in this section are based in a simple task using 8 commands and a master set of 12 gestures. The next chapter will present in detail the Gesture Recognition Algorithm (left block inside of Module 2 in the system architecture, Figure 3.1). This algorithm is based on an FCM parameter calibration functionality combined with image processing operations. The input of this algorithm is a subset of gestures obtained using one of the methods described in this chapter, and the output is the recognition accuracy, A (the technical factor).

Methods of determining fatigue and intuitiveness indices based on human ergonomic and cognitive experiments will be presented in Chapter 6. A case study using the strategies presented in this chapter will be depicted in Chapter 7.

5 Algorithms

5.1 Overview

In this chapter the vision-based algorithms of a hand gesture recognition system are introduced (Module 2.1). The Image Processing-Fuzzy C-Means (FCM) components of the hand gesture recognition system are described and the calibration of their operational parameters is performed using a neighborhood search algorithm. Two neighborhood search strategies are presented to achieve close to optimal recognition, the first based in a classical neighborhood search and the second based in evolutionary strategy search. User-dependent and user-independent systems using a database of 13 gestures are compared.

5.2 Hand gesture recognition system

The hand gesture recognition system comprises an image processing feature extraction operation followed by an FCM gesture classifier. The FCM clustering algorithm [Bezdek, 1973] is a popular method for image recognition tasks [Wachs *et al.,* 2002]. Although the speed of artificial neural network classifiers allows real-time operation and comparable accuracy, an FCM is used because it requires smaller training sets and shorter training times. The classical FCM algorithm is modified to handle feature weighted clustering, and is supervised using a cluster labeling algorithm [Wachs *et al.,* 2005].

5.2.1 Feature extraction

A database, denoted as BGU-R-DB, consisting of 13 static hand gestures, was constructed for training and testing purposes (Figure 5.1). Preprocessing of the image starts with segmentation of the hand from the background using a threshold value, τ, to obtain a black and white image. The threshold value used is found through a parameter search algorithm (discussed in Section 5.3). Using a component-labeling algorithm, the largest component (assumed a priori to be the hand posture), is identified, and a bounding box is constructed around it to represent the segmented hand. The box is then partitioned into blocks. Although the backgrounds of the gesture images are simple, more complicated backgrounds can be handled by other methods such as color segmentation.

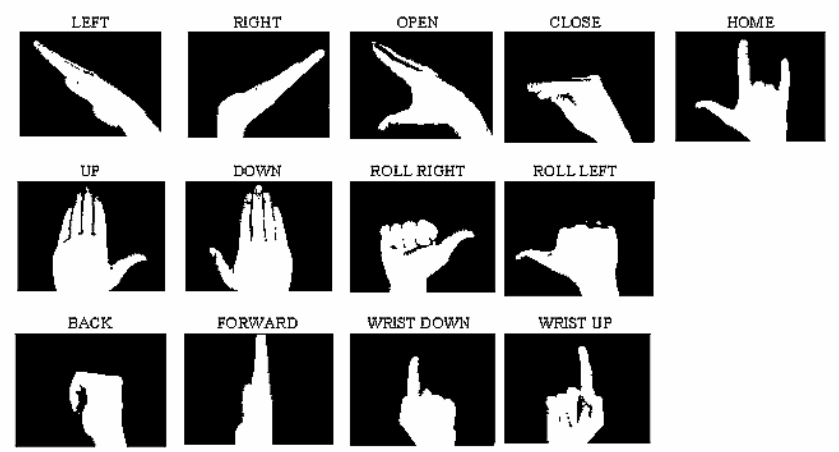


Figure 5.1. Set of static hand gestures

The bounding box and a restriction on the height position of the posture makes the gesture position invariant in translation and size. A feature vector of the image comprises the aspect ratio

of the bounding box and the average intensity of each block (fraction of white pixels). Let R_b and C_b represent the number of rows and columns, respectively, of the block partition. This results in a feature vector of length $v = 1+R_bC_b$, denoted as $f = (f_1,...., f_i,\ldots, f_v)$. The first feature represents the aspect ratio of the bounding box, the remaining represent block averages indexed row-wise from left to right (Figure 5.2).

(a) (b)

Figure 5.2. Feature extraction (a) bounding box of hand gesture (b) 3x4 block partition

For example, the resultant feature vector in Figure 5.2(a) is: f = (102 176 52 2 2 68 249 171 16 3 13 253 188). All feature values are scaled to lie in the range [0, 255]. One can see that the aspect ratio is 102, and blocks 3 and 4 are close to zero (black) while blocks 6 and 11 are close to 255 (white). Let $w = (w_1, \ldots, w_i,\ldots, w_v)$ represent the weight vector where w_i is the weight attributed to feature i. The weights are normalized to sum to one.

$$\sum_{i=1}^{v} w_i = 1 \ , \ 0 \le w_i \le 1 \qquad \textbf{(5.1)}$$

Let $x = (w_1 f_1, \ldots , w_i f_i ,\ldots, w_v f_v)$ be a weighted feature vector (also referred to as a data pattern).

5.2.2 Feature weighted fuzzy c-means gesture classifier

In the weighted feature FCM algorithm, a weighted feature vector represents each gesture. The set of weighted feature vectors is clustered for subsequent use in a recognition system. Note that the particular clustering obtained depends on the number of clusters and the respective values of the feature weights. Let x_k be the weighted feature vector of the k^{th} exemplar in a training set of gestures. Given q data patterns $X=\{ x_1 ,...., x_k ,..., x_n\}$ and a fixed number of clusters c, the FCM algorithm finds: v_i (the prototype weighted feature vector of cluster i), and μ_{ik} (the degree of membership of x_k in the i^{th} cluster). This is done by minimizing a membership weighted within-group sum of squared errors objective function, where m is a weighting exponent on each fuzzy membership value. In this application the number of clusters should be set greater or equal to the number of gestures in the set G_n.

After convergence of the FCM algorithm, each weighted feature vector x_k is assigned to a cluster by finding: $\mu_{i'k} = \text{Max} \{\mu_{ik} , i=1,\ldots,c\}$. This method is selected to reduce computational complexity for real-time operation and to reduce the time taken for large-scale validation studies. Once clusters are labeled by gesture class, a new gesture may be classified by selecting the cluster for which its membership value is maximal.

5.2.3 Parameter estimation

Gestures performed by a user are recognized using the highest membership value. System performance is evaluated using a confusion matrix that contains information about actual and classified gestures. Recognition accuracy, as defined by (3.8), is determined as a function of a set of input parameters of the system. The process of searching for optimal parameters for the combined image processing/supervised FCM system is shown in the flowchart of Figure 5.3

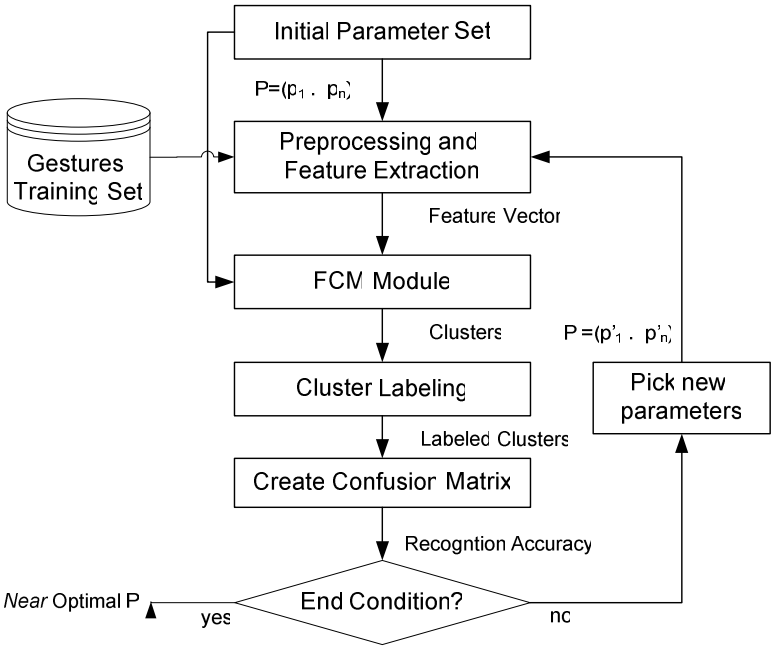


Figure 5.3. Supervised FCM gesture recognition algorithm with parameter search

The output of the process is a near optimal set of parameters achieved by maximizing the recognition accuracy. The procedures used are a complete neighborhood search (CNS) algorithm and a probabilistic neighborhood search algorithm (PNS).

5.2.3.1 Input parameter vector, p

Denote the vector $p \in R^n$ as the set of input parameters in Table 5.1. Two types of input parameters are used: image processing features (block partition size, b/w threshold, feature weights of the aspect ratio, and grayscale block features) and FCM parameters (number of clusters and weighting exponent).

Table 5.1. Parameter definition

Parameter	Meaning	Values
p_1	Number of Clusters, c	$p_1 = g, g+1, c_{max}$
p_2	Weighting Exponent, m	$p_2 = 1.5, 1.75, \ldots, 4$
p_3	b/w threshold, τ	$p_3 = 0, 1, \ldots, 255$
p_4	Number of rows for image partition, R_b	$p_4 = 2, 3, .., 8$
p_5	Number of columns for image partition, C_b	$p_5 = 2, 3, .., 8$
p_6	Weight of the aspect ratio, w_1	$p_6 = 0, 0.1, .., 1$
$p_7, .., p_i \ldots, p_n$	Weights of the image block features, w_i	$p_i = 0, 0.1, .., 1$

5.2.3.2 Neighborhood solutions, N(p)

For any feasible solution $p=(p_1, \ldots, p_n)$ for the recognition system, define a set N(p) of neighboring solutions of the vector p. The number of neighbors of p is 2n as each parameter is incremented up and down (wrap around is used when boundary values are exceeded). The set of feature weight parameters are updated in a special way because of their inter-dependence through equation (5.1). Moreover, the number of feature weights depends on the block partition parameter values. Given a block partition of R_b rows and C_b columns, the number of feature weight parameters is the same as the number of features, $v = 1 + R_bC_b$. The dimensionality, n, of our pattern space is variable and depends on the minimum and maximum block partition values. For block partition values ranging from 2 to 8, the number of feature weight parameters can vary from 5 to 65 resulting in pattern spaces of dimension 10 to 70. To handle such variable length parameter vectors, p_4 and p_5 are taken as control parameters, which turn the appropriate weight parameters 'off' and 'on' according to the following rule: whenever p_4 or p_5 changes, the length of p is set to $n = 5 + v$.

Let $\{w_i : i = 1,\ldots,v\}$ be the current set of weights. To find the neighbor values of a feature weight w_j , increment w_j up and down by the discrete gradient Δj. Since feature weight normalization is necessary to ensure that (5.1) is satisfied, the new neighbor feature weights are:

$$w_i(\pm\Delta j) = \begin{cases} \dfrac{w_i}{1 \mp \Delta j}, & i \neq j \\ \dfrac{w_j \pm \Delta j}{1 \mp \Delta j}, & i = j \end{cases} \qquad \textbf{(5.2)}$$

5.3 Local neighborhood search algorithms

5.3.1 Complete neighborhood search algorithm (CNS)

The CNS algorithm (Algorithm 5.1) starts with an initial solution p_0. To determine the accuracy, A, associated with p, define a mapping $\mathcal{A}$: $p \rightarrow A$. Determination of the functional value of $\mathcal{A}$, for a given solution p, requires extraction of a new set of image features, executing the FCM algorithm, cluster label assignments, gesture classification, and analysis of the confusion matrix to determine the recognition accuracy (Figure 5.3). Cluster labeling assignments are done using the Alg-L algorithm [Wachs *et al.,* 2005].

Algorithm neighborhood search;
1. **Begin**
2. Create an initial feasible solution $p^0=(p^0_1, \ldots, p^0_n)$
3. **local_maxima=false**
4. **Repeat**
5. **Begin**
6. Find $\mathcal{A}$(p') for all p' in N(p)
7. Let p''= Argmax{ $\mathcal{A}$(p') | N(p)},
8. **If** there are ties:
9. Pick the last p''
10. **If** p'' not in Stack, **Push** p'' into Stack
11. **Else local_maxima=true**
12. **End if**
13. **If** p'' = p then **local_maxima=true**
14. Replace p by p''

15. **End**
16. **Until** $\mathcal{A}$ (p'') = 100% **or local_maxima=true do**
17. Output p'', the local or global max solution
18. **End**

Algorithm 5.1. The classic neighborhood search (CNS).

The main idea behind the CNS algorithm is to continuously find a better solution by advancing in the parameter space in one coordinate direction each time. Define an iteration as one cycle starting from the current solution p until the best neighbor solution p'' is selected. Note that each iteration consists of an evaluation of all 2n neighborhood solutions in N(p). If the accuracy in the iteration did not increase, i.e., p'' equals p, then a local maximal was found and the algorithm stops. However, if there are ties, a plateau has been reached. In case of plateau, the algorithm will try to find a better solution by advancing in the parameter space along the direction of a tied solution. However, if the best neighbor p'' has been visited before (kept in a stack) and no improvement is made in the entire plateau, an expansion of the neighborhood is made by doubling the step size for the next five iterations, in the hope of escaping from the local maximal.

The recognition accuracy function $\mathcal{A}$ is a non-decreasing function of the number of iterations k, i.e.; $\mathcal{A}(p^k) \geq \mathcal{A}(p^{k-1})$ where p^k is the parameter vector at iteration k. The algorithm stops when two successive iterations give the same accuracy value after exploring all neighbor solutions, if a plateau is reached. A plateau is the case where at least one neighborhood solution has the same value as p''. Since A is bounded above by 100 percent termination in a finite number of steps is guaranteed. Detailed proofs may be found in [Wachs *et al.,* 2003] and in (Appendix I).

5.3.2 Probabilistic neighborhood search algorithm (PNS)

Unlike the CNS algorithm where the entire neighborhood is examined before a move is made, in the PNS algorithm, solutions in the neighborhood N(p) are randomly sampled and evaluated. A move is made to the first improved solution found. If no improvement is made after K evaluations, the neighborhood is expanded and a probability distribution is sampled to generate a new solution. For any parameter p_j the following is defined:

Δp_j = the smallest step size increment taken in any coordinate direction j = 1,…,n in the solution space.

s = the number of steps made in either the positive or negative coordinate direction j

Δj = the discrete gradient in the j^{th} coordinate direction, where

$\Delta j \in \{s\Delta pj : s = 0, \pm 1, \pm 2, \pm 3, .. \pm S\}$

$\nabla p = \{\ \Delta j\ \}$ = the gradient direction of a move from p (an n dimensional vector) with common element Δj.

$\hat{p} = \Psi(p, \nabla p)$, = the updated solution, where Ψ is a special operator mapping a vector p of size n_p to a vector $\hat{p}$ of size $n_{\hat{p}}$ ($n_p = or \neq n_{\hat{p}}$).

Identical neighborhood sampling probability distributions are defined for each coordinate parameter. Discrete Gaussians or equivalent binomial approximations (using probability of success = 0.5) have the property that an increased standard deviation not only spreads out the distribution but also reduces the peak value. To control the proportion of parameter changes, a special mixture type point distribution model was designed. The characteristics of this distribution are that the tails can be spread out while the peak probability remains constant.

The notation is

S = maximum number of step increments.

h = probability of no change

x_j = a random variable representing the signed (positive or negative coordinate direction) number of step size changes for parameter p_j.

$\Omega = \{0,\pm1,\pm2,...,\pm S\}$ = the universe of the random variable x of size 2S +1.

$P_{S(x|h)} = P_r(x = s)$ the probability of step size s , given h.

The probability distribution $P_{S(x|h)}$ is characterized by the two parameters, h and S.

$$PS(x \mid h) = \begin{cases} h, & x = 0 \\ h((1-h)^{|x|})/2, & x = \pm1,\pm2,,,,\pm(S-1) \\ ((1-h)^{|x|})/2, & x = \pm S \end{cases} \quad \textbf{(5.3)}$$

In (5.3) |x| is the absolute value operation. For example, if h=0.9 and S =3, the probability x = -2 is 0.0045.The probability mass function is symmetric, with a peak at x = 0, which represents the probability that the parameter remains unchanged. The range of the distribution is a linear function of S. For a fixed h, the probability of x = 0 remains the same. Also, the distribution range increases linearly in S. This expands the neighborhood, allowing larger steps while the probability of not moving remains constant. This will have the effect of, on average, allowing a same number of parameters to be changed, but by larger possible step sizes, increasing the chance of escaping from a local extrema. If, for example, h=0.9, in the long run 10 percent of the parameters in the parameter string will change and 90 percent will remain the same.

Algorithm PNS

Given a solution, its updated value is determined by:

$$\hat{p} = \Psi(p, \nabla p) \quad \textbf{(5.4)}$$

where,

$\Delta j = s\Delta p_j$

s = x with probability $P_S(x|\ h)$

$\hat{p}_j = p_j + \Delta_j, \forall p_j$

p and p_j are same size vectors

If one or more weight parameter changes occur, then a repair operation is applied to ensure that the weights are normalized (sum to one). The new neighbor feature weights are updated using formula (5.5) below.

$$w_i' = \left\{ w_i + \Delta j \Big/ 1 - \sum_{k=1}^{v} \Delta_k \quad , i = 1,..,v \right\} \quad \textbf{(5.5)}$$

A run starts with an initial solution. Once an improved solution p is found, a new iteration commences with the improved solution. At the start of the iteration the neighborhood size is set to S=1. If an improvement is found before K evaluations, the current solution is updated according to (5.5). If no improvement is found, the neighborhood is further expanded to S = 2

and then to S=3. If after three-neighborhood expansions no further improvement is found, the algorithm terminates. As each evaluation of p, determining the accuracy value of the functional $\mathcal{A}$ is time consuming; therefore, to reduce computation time, a list of prior solution vectors p is maintained. After generating a new parameter vector by sampling from the neighborhood distribution, all previous solutions on the list are checked. If the new solution appears in the list, it is dismissed; otherwise, an evaluation is made. Although searching the list of previously generated solutions involves additional computational effort, time wise it is several orders of magnitude less than the time spent during accuracy evaluation. Recall that determination of the functional value $\mathcal{A}$ for a given solution p requires extraction of a new set of image features, executing the FCM algorithm, cluster label assignments, gesture classification, and analysis of the confusion matrix.

5.3.3 Comparison of CNS and PNS algorithms

An example test was conducted to illustrate the performance of the CNS and PSN algorithms using 35 samples per gesture for 13 gestures to obtain a training set of 455 samples. Using nine heuristics described in Wachs *et al.* [2005], the following starting solution was generated.

Table 5.2. Initial solutions used for CNS and PNS runs

Run	c	m	τ	R_b	C_b	w_i	$A(\%)$
1	13	2	146	2	2	0.73, 0.075, 0.074, 0.058, 0.063	84.84
2	16	2	146	2	2		96.04
3	20	2	146	2	2		95.60
4	13	2	146	5	5	0.343, 0.03, 0.024, 0.023, 0.025, 0.04, 0.025, 0.022, 0.027, ...	77.80
5	16	2	146	5	5		88.35
6	20	2	146	5	5		77.80
7	13	2	146	8	8	0.195, 0.016, 0.015, 0.012, 0.012, 0.013, 0.013, 0.018, 0.02, ...	83.30
8	16	2	146	8	8		85.93
9	20	2	146	8	8		90.33

Both the CNS and PNS algorithms were tested with the same starting solutions. Table 5.3 shows the numerical results obtained for both algorithms. The accuracies obtained from the two algorithms are shown in columns 6 and 7. Both algorithms obtained the same best accuracies of 99.78 (bold). Columns 4 and 5 contain the total number of iterations and the number of accuracy evaluations up to the start of the last iteration, which then runs for 3K more evaluations. This sum was added to column 4. For all PSN runs, the values of h = 0. 9, and K = 30 were used. Figure 5.4 shows the convergence for run 5.

For the test run, the PNS algorithm found the same best solution as the CNS but in 48.5 percent shorter computational time (1,935 vs. 3,754 evaluations). The reduction, however, is a conservative estimate as it is based on equal time evaluations for the PNS algorithm runs. This is because the times to evaluate a solution are not equal as assumed, but are directly proportional to the size of the solution vector p, which is dynamic. Thus, a weighted evaluation time function should be used, whereby given t(n) as the time to evaluate a solution vector p of length n, an adjusted evaluation time can be determined as t(n)*r(n), where r(n) is the number of times a solution vector of size n is evaluated.

Table 5.3. Comparison of CNS and PNS algorithms on the basis of computational steps and accuracy

Initial Solution	Number of Iterations		Evaluations to Best Sol [1]		Accuracy, A(%)	
Run	PNS	CNS	PNS[2]	CNS[3]	PNS	CNS
1	17	3	359	60	97.08	97.8
2	5	1	124	20	99.12	97.8
3	3	2	20	40	98.02	98.02
4	2	7	69	434	97.14	99.12
5	16	12	376	744	**99.78**	**99.78**
6	1	8	1	496	96.04	99.34
7	8	4	115	560	95.6	99.12
8	2	3	16	420	96.92	98.46
9	1	7	45	980	99.34	99.56
Total			1,935	3,754		

[1] Evaluations to last parameter change for the run

[2] 3K=90 evaluations added after the last parameter change for each run (1,125 + 810 = 1,935)

[3] (Number of iterations)*No of evaluations /iteration)= total evaluations, where no of evaluations /iteration is fixed at 20,62 and 140 for sol 1-3, 4-6 and 7-9,respectively.

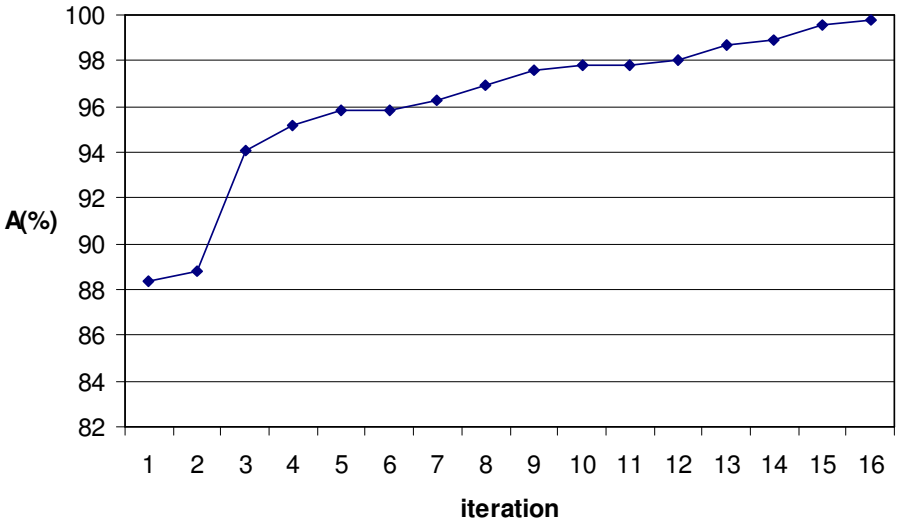


Figure 5.4. Convergence curve for PNS (run 5)

5.4 Performance testing and results

A sample run was conducted to illustrate the performance of the NS algorithm using the data from a data set called BGU-R DB. For this database we extracted 35 samples for each of 13 gestures to obtain a training set of 455 samples. Two different types of systems were used to train and test user, **dependent** (D) and **independent** (I) recognition systems. The D and I systems were defined as the systems that were trained by single and multiple users, respectively. Three types of subjects were used in the experiments: **Owners** (O), **Experienced Users** (E), and **Novice Users** (N). **Owners** trained all I and D systems and were also used to test these systems. **Experienced Users** are users that test systems that were trained by others. These users were previously trained

owners who played the roles of **experienced users** at this later stage. **Novice Users** were new users who had never seen, trained, or tested a system.

Seven **Owners** and twelve **Novice Users** tested the D and I systems. For each of the user-system combinations, the mean recognition accuracies were calculated from the results of the k-fold cross validation runs (for k=4). The mean recognition accuracies between systems were compared using a two-tailed t-test. Table 5.4 shows the hypothesis formulated, the population used to compare each side of the hypothesis, and the recognition accuracy, variance, hypothesis result, and significance level. Recognition accuracy of system x tested with user y is represented by R (x, y). The number of gesture instances is n_i , the recognition accuracy is x_i, and the variance is S_i^2. A summary of the important results is shown in Table 5.5. When the systems were tested using their own trainers, mean accuracy of D was better than for I (98.9% over 98.2%). This was expected, since any learning system should have better performance when tested with its trainer. For **O users**, the opposite was true, testing recognition accuracies where better for I than D systems (98.2% over 96.0%). This was also expected, as **E users** tested systems trained by others. Here, the I system was trained with a wide variation of hand gesture samples, and as a result, it had better generalization properties. These results were statistically significant. Similarly, **N user's** testing accuracy was also better for I than D systems, resulting in values of 95.7% and 94.6%, respectively.

Table 5.4. Performance comparison between systems

No.	Hypothesis	n1	n2	x1 (%)	x2 (%)	S_1^2	S_2^2	Answer	Signif. (%)
1	A(D,E)>A(D,N)	21840	18200	96.01	95.19	0.0383	0.0458	TRUE	0.0032
2	A(I,O)>A(I,N)	3640	2600	98.21	96.19	0.0175	0.0366	TRUE	0
3	A(D,O)>A(D,E)	3640	21840	98.90	96.01	0.0109	0.0383	TRUE	0
4	A(D,O)>A(I,O)	3640	3640	98.90	98.21	0.0109	0.0175	TRUE	0.69
5	A(I,E)>A(D,E)	3640	21840	98.21	96.01	0.0175	0.0383	TRUE	0
6	A(I,N)>A(D,N)	2600	18200	96.19	95.19	0.0366	0.4580	TRUE	1.2

Table 5.5. System recognition accuracy

Type of User	Type of System	
	Dependent (*D*)	Independent (*I*)
Owners (*O*)	**98.9%**	98.2%
Experienced (*E*)	96.0%	**98.2%**
Novice (*N*)	94.6%	**95.7%**

Compared to previous runs using 5 novice users, the results were slightly better, as expected (96.1% and 95.1%). These values had a statistical significance at the .005 level. Again, the results are for novice users who have neither trained systems nor have had experience using them. Previous research [Wachs *et al.,* 2002] indicates that novice users can reach 98-99 % accuracy after several trials.

5.5 Discussion

This chapter described a hand gesture recognition system using an optimized Image Processing-Fuzzy C-Means (FCM) algorithm. The parameters of the image processing and clustering algorithm were simultaneously found using two neighborhood parameter search routines, resulting in solutions within 1-2% of optimal. Two versions of a local neighborhood search algorithm were designed. These versions were customized for a system operational parameter calibration task, where the number of parameters in the solution vector is dynamically

changing. The first and second methods perform complete and incomplete probabilistic neighborhood searches, respectively. The primary need for recalibration of such systems is frequent relocation to other environments such as laboratories and remote control stations. A secondary need for recalibration occurs due to demands for custom redesign of the gesture control language. This occurs for new users, new control tasks, and new vocabularies. Allowing for a fast recalibration of system parameters provides the system with the flexibility needed to respond to such new system setups. The two proposed methods were compared using a test case of 13 gesture commands, and a recognition accuracy of 99.78% was obtained. However, the probabilistic version performed 48.5% fewer solution evaluations.

A comparison of user dependent and user independent systems based on a database of 13 gestures was made. When the system was tested with its own trainers, recognition accuracies of 98.9% and 98.2% were revealed for the dependent and independent systems, respectively. These results are statistically significant at the .007 level. For experienced users testing systems they did not train, testing recognition accuracies were better for user independent than user dependent systems (98.2% over 96.0%). These results are statistically significant at the .00 level. The near optimal parameter search procedure is easily extended to systems with larger parameters and more complex hand gesture recognition systems. The problem is not unique to hand gesture recognition systems, but is shared by other human-machine systems. Thus, the methodology presented here for automating system setup has far wider application.

The next Chapter includes the experiments performed to determine human psycho-physiological factors, the validation of the measures as a function of task completion time, and conclusions with regard to learning and memorability.

Chapter 7 presents two tasks as case studies to obtain candidates of hand GVs for use in a multiobjective criterion problem. The feasible solutions (GVs) are presented to the decision maker together with an approximate set of Pareto points GV' to aid the decision maker in the selection of a single GV.

6 Experiments

6.1 Overview

Three groups of experiments were performed: (i) determination of human psycho-physiological input factors (Module 1.4), (ii) validation of the multiobjective proxy measures for designing good GVs in terms of task performance time, and (iii) usability measures in terms of learning and memorability rates. Human psycho-physiological input factors were found through a series of empirical experiments to obtain the intuitiveness V, comfort U, command C, and gesture G_m matrices (Modules 1.5-1.7). More specifically, the frequency of commands, and direct and complementary intuitiveness were tested, and stress experiments were conducted. These empirical measures were used in operator task control experiments to perform a validation test showing the connection between GVs and task completion time. This validation experiment used a significant group of subjects in the context of two tasks: (i) a robotic arm pick and place task, and (ii) a VMR drive task. In addition, two usability tests were performed: (i) the learning rate, and (ii) memorability. Statistical tests were performed to determine the significance of these results.

6.2 Command frequency experiment

6.2.1 Overview

Robotic arm and VMR tasks were used in the experiments, to determine the human factors measures. Both tasks contain 'navigational' (directional) commands to control the direction of movement of an object, its speed, and additional functions to interact with other objects in the environment. An experiment was set up to determine the frequency of each command from typical command sequences. The sequence of commands depends on the type of task. In addition, duration of each command and the durations of breaks (intentional and unintentional) between commands were obtained. For this purpose, two virtual reality environments, in which the user has to conduct a task, were developed: (i) a robotic arm and (ii) a maneuverable vehicle.

6.2.2 General Set Up

To avoid long delays in the task completion times resulting from the latency inherent in communications ports and the slow movement response of the physical robotic arm and the vehicle robot, virtual reality models were developed. The virtual models were designed to simulate the functionality of the real robots as accurately as possible. For the 5 DOF robotic arm, the close form of inverse kinematics equations were solved to mimic the world coordinate operation. The effect of gravity on the object and the use of collision detection to avoid impacts were considered. The VMR was simulated in a straightforward fashion, such that only collision detection rules were included. These assumptions helped me to carry out the experiments successfully, without suffering from the electromechanical problems that occur frequently when dealing with real robots.

The virtual mechanical devices (robotic arm and VMR) were controlled by the user to convey actions in the form of a basic set of commands. The basic set of commands were: C_1={'start', 'finish', 'up', 'down', 'forward', 'backward', 'left', 'right'} and C_2={'start', 'finish', 'up', 'down', 'left', 'right', 'forward', 'backward', 'wrist cw', 'wrist ccw', 'wrist up', 'wrist down' 'open gripper', 'close gripper', 'home'} for the VMR and robotic arm tasks, respectively.

6.2.3 Software applications

Both virtual models were developed using MS Visual C++ and OpenGL (Appendix K) to create a realistic scenario of the tasks. Of course, there is a limit to the virtual model's match of the real world. For example, texture is an attribute of real objects only, constant robot velocity is not possible in reality, and the effects of wind and other external forces were not considered in the virtual models.

In the VMR task application, the scenario consists of a maze-like road surrounded by a garden. The path to be traversed by the vehicle is composed of nine straight linear segments (Figure 6.1). At every junction there is only one possible way to turn. The terminal ends of the road are marked with the start and stop lines, and the VMR is initially parked at the start line. Also, an item, which must be visited by the vehicle, is placed in the middle of the path. The item is a teapot, but can be any object of small size (surface of 170 pixels). The following parameters are displayed across the top of the screen: time, impacts, score, speed, and engine status.

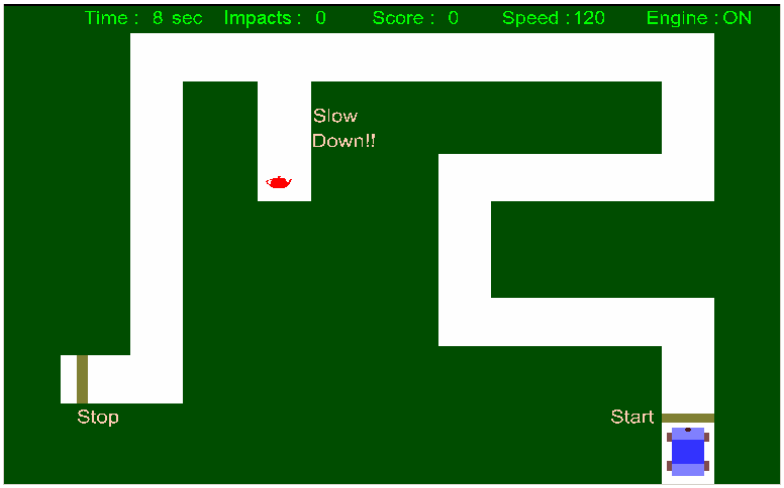


Figure 6.1. VMR maze application

The time indicates the task duration time. Impacts represent the number of times that the VMR hits the side of a road segment and ventures into the garden. The score is based on the task completion time and the number of impacts. Speed shows the current selected velocity of the VMR: (120 is fast, 60 is slow). The engine of the VMR can be 'on' or 'off'.

In the robotic arm application, a five degrees of freedom virtual robotic arm placed over a table was displayed (Figure 6.2). On the table were three wooden boxes. On one side, a blue box was standing on a red box (Point A). At a fixed distance from them, another red box was laying on the table (Point B). The parameters presented in the top of this screen were time and engine status, and their meanings were the same as those described for the VMR application.

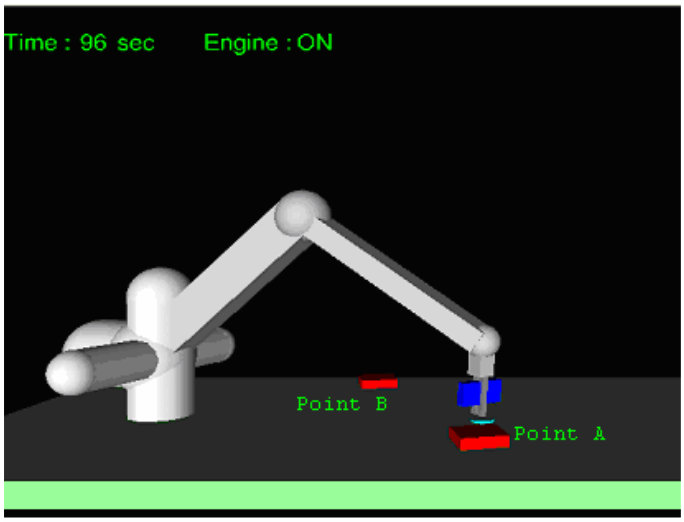


Figure 6.2. Robotic arm application

6.2.4 Procedure

In the VMR pilot study, the user has to control and guide a VMR from the starting line to the finishing line within the shortest time and without hitting the sides of the road. Every impact added a fixed time penalty to the total time. The goal was to finish the course as quickly as possible, and therefore, the user was encouraged to drive at high speed. The user had to respect the speed limit posted at some point along the road. The VMR also had to run over a teapot lying on the road for full completion of the task. A standard Qwerty keyboard (ISO9995) was used to control the VMR, and the sequence of keystrokes was recorded during the whole task. The mapping between the keys and the commands was totally arbitrary (the same for each subject) with the purpose of obtaining the sequence of keystrokes and pauses (mapped to commands).. Negligible delays were ignored from the command sequence, and hence, delays resulting from finger movement between keys did not show up in the sequence. This implies that the particular mapping between commands and the keyboard does not affect the resulting sequence of commands. Table 6.1 shows the mapping used between commands to the keyboard for the VMR task.

Table 6.1. Commands for the VMR task

Index	Commands	Key press
0	Rest	No key
1	Start	VK_LSHIFT
2	Finish	VK_RSHIFT
3	Forward	VK_UP
4	Backward	VK_DOWN
5	Turn Left	VK_LEFT
6	Turn Right	VK_RIGHT
7	Speed=Fast	VK_PRIOR
8	Speed=Slow	VK_NEXT

In the robotic arm task, the robotic arm's gripper had to reach a blue wooden box standing over a red wooden box (Point A), over a flat table, pick up the box, move it to a different location, and release it over another red wooden box (Point B). To pick the blue box, the gripper

had to be at a certain angle with respect to the arm; otherwise, the task of gripping was impossible. To release the box, from a vertical position over the target object, without dropping it, the gripper must be in a specific position as well. This operation must be performed by the user in the shortest time. The user controls the Cartesian coordinates of the arm (world coordinates), a two degree of freedom gripper, and open and close (grasp-release) operations using the computer keyboard (Table 6.2). Also, in this experiment the sequence of key-presses was saved since this information is necessary to build the frequency matrix.

Table 6.2. Commands for the robotic arm task

Index	Commands	Key press
0	Rest	No key
1	Start	VK_LSHIFT
2	Finish	VK_RSHIFT
3	Up	VK_UP
4	Down	VK_DOWN
5	Left	VK_LCONTROL
6	Right	VK_RCONTROL
7	Forward	VK_RIGHT
8	Backward	VK_LEFT
9	Wrist Up	VK_PRIOR
10	Wrist Down	VK_NEXT
11	Wrist CW	VK_INSERT
12	Wrist CCW	VK_HOME
13	Open Gripper	VK_END
14	Close Gripper	VK_DELETE
15	Home	VK_BACK

For each task 30 trials were conducted. The sequence of keystrokes, representing the sequence of commands evoked for each trial, was stored as an ordered vector S for each type of task. The keyboard was sampled every 14 ms, and if a key was pressed, its corresponding index (based on the values in the first column of Tables 6.1 and 6.2) was added to the sequence vector; otherwise, the value 0 was inserted. Note that for the robotic task, the average completion time over the 30 trials was 42 seconds, and hence, a typical length of the sequence vector in that task, was 42/0.014=3000 commands long. A frequency matrix F (of size n × n) was created by parsing the sequence of commands. The entries in F represent the appearance frequencies of the same command (diagonal), and the transition between commands (off diagonal) over all the sequences. Algorithm 6.1 shows pseudo-code to perform the extraction along with an example.

Create frequency matrix (F)
F = 0
from_A= S(1)
for index=1 to length(S)
to_B=S(index)
F(from_A, to_B)= F(from_A, to_B)+1
from_A= S(index)
next
End

Algorithm 6.1 Frequency matrix creation

Example 1

Given a sequence S_1=(1,1,1,1,2,2,2,1,1,3,3,3,2,1,3), the frequency matrix found is:

$$F = \begin{bmatrix} 4 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

6.2.5 Results and analysis

Each experiment was repeated 30 times by an **experienced user**. For each task, a frequency matrix was constructed as the sum of the frequency matrices obtained in the 30 trials. The column and row headings of each of the frequency matrix tables (Tables D.23 and D.24) are indices representing command names according to the coded lists in Tables 6.1 and 6.2 for the robotic arm and VMR task, respectively. The mean task completion time was 42.3 (σ=4.5040) and 67.1 (σ=4.1312) seconds, for the robotic arm and VMR, respectively. Tables D.21 and D.22 show the total frequency matrix for the robotic and VMR tasks including the 'rest' command, respectively.

When the 'rest' command was not included in the command set, and there was no need to have a gesture for the 'rest' action, the appearances of the index representing the pause (0) were ignored when parsing S (Tables D.23 and D.24). There was a significant difference between the matrices including the 'rest' command and the ones excluding it. Besides the additional row and column for the 'rest' posture, the values of the frequencies were also different. For example, F1,7=30 in Table D.23 was F1,7=0 in Table D.21. The reason is that the command '1' always switched to 'rest' (30 times) and from 'rest' to '7' (96 times) instead of changing straight from '1' to '7' (30 times).

Examination of the sequences showed inevitable short pauses between different keystrokes. However, these tiny pauses did not represent intentional 'rests', but rather delays when moving/relocating the finger to the correct key. Every pause shorter than 42ms was considered an unintentional pause. This period of time was obtained empirically by observing the maximum time that takes to process each stroke in a consecutive sequence of keystrokes. To keep only intentional pauses (rests), the sequence of indices was parsed to eliminate each single, double or triple consecutive appearance of the index representing the pause (0). For example, the sequence: S_1=[1,2,0,1,2,0,5,6,2,0,0,5,0,0,0] was converted to S_1=[1,2,1,2,5,6,2,5] and S_2=[4,2,0,0,0,0,6,5,0,0,0,0,0,0] was converted to S_2=[4,2,0,0,0,0,6,5,0,0,0,0,0,0] (no changes). After all the unintentional pauses were discarded, the sequence was analyzed to determine the frequency of use of each command and the transition frequencies between them. These sequences seemed to be totally dependent on the type of task addressed by the user, possibility changing according to the representation of the environment. The more realistic was the virtual model, the closer the sequence of commands was to the real task.

6.2.6 Discussion

Results indicate the importance of experimental analysis of specific tasks. Observing the frequency matrix for the robotic arm and VMR tasks using a 'rest' command, the 'rest' command occurred at a far higher frequency than any other command in the task (42761 and 101462 times, respectively). This does not mean that there were long pauses while performing the task, but the pauses were very frequent. The matrix shows that between the execution of any two commands, there was a short rest. Except for the 'start' command, and the 'finish' command, which occurred only at the beginning and end of the task, there were no transitions either to 'start' or from 'finish' registered in the sequence.

Excluding ‘rest,’ the most popular command for the VMR task was ‘forward’ because the sequence to complete the VMR task was very rigid. There was only one path to follow to complete the task successfully. The user had to continuously correct the direction of the VMR to avoid hitting the sides of the road. This explains why the ‘forward’ command occurs more frequently than the other commands. For the robotic arm task, the most frequent command was ‘right,’ because the object to be picked up was placed on the far left side of the robotic arm while the place to release the object was to the far right of the robotic arm. Therefore, the distance necessary to travel to reach point B after reaching point A was larger than the distance to reach point A (assuming the gripper is somewhere between both points).

The above scenario also demonstrates that it was incorrect to assume that complementary commands were used with the same frequency. The use of the two different command types relied on the nature of the task and its topology. The (forward, backward) command pair is another example of complementary commands that exhibited different frequencies. The ‘backward’ command was evoked after the VMR picked the teapot up off the road, and had to return to the main road traveling in reverse since the road there was not wide enough to turn around. This specific transition was repeated in each of the 30 trials, hence the value 30 for the ‘forward’ to ‘backward’ transition. Theoretically, the frequency of transition between ‘backward’ to ‘forward’ should be zero, since such a transition was not necessary to complete the task. However, the 7 ‘backward’ to ‘forward’ transitions may be explained as corrections in the direction of the VMR while turning left or right to avoid driving off the road and into the garden.

6.3 Intuitiveness experiments

6.3.1 Overview

The experiment used to obtain the cognitive association between commands and gestures considered several approaches: (a) present a large database of images and the user will select the image that reminds him the most of the given command (restrictive); (b) let the user gesticulate with one and/or both hands and take a picture of the gesture associated with the given command (unrestrictive); and (c) allow the user to manipulate a rigid hand gesture model, where the intra and inter joints, rotation, and other features are constrained. While the restrictive approach is advisable when working with a small gesture dataset, it is prohibitive for larger datasets. For the unrestrictive approach, the dataset obtained by capturing user gestures can be prohibitively large. Therefore, the approach used here is to represent the postures by configuring a number of hand segment primitives. Through the use of an application, a random sequence of commands was presented to the user after which the user manipulated a hand model until it was configured to represent the desired gesture. Each command was displayed to a cohort of users, and the gesture-command associations were ranked accordingly to popularity based on the number of times they were selected. The most highly ranked gestures (most popular) were assigned to the gesture master set. Complementary intuitiveness indices were obtained by extracting the number of times that the subjects chose the same pairs of complementary gestures to represent the same pair of complementary commands.

6.3.2 General set up

A WE-160 Panasonic Video Imager (Appendix F) included a platform on which the users displayed their postures (Figure 6.3). While viewing their posture they configured the virtual hand model to replicate it. The video was not connected to the computer; the video imager served only as a comfortable flat surface where the users could lay their hands and get an idea of the camera’s view of their hands.

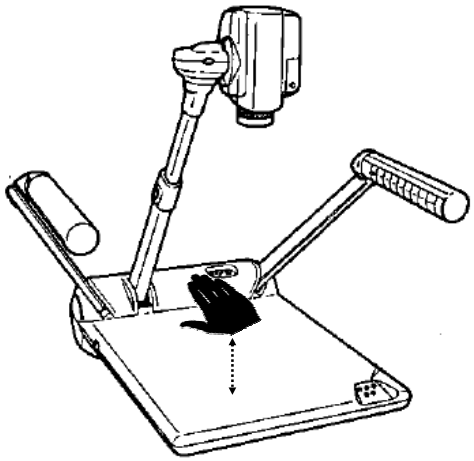


Figure 6.3. User hand over the WE 160 Panasonic video imager

6.3.3 Software applications

An intuitive assessment application was developed to intuitively map between commands and gestures (Figures 6.4 and 6.5). The interface window of the application was divided into four sub areas: A, B, C, and D. The first (A) contained query fields for user details. The second (B) contained the name of the current command with a picture or animation below it that corresponded to this command. The third sub area (C) had an image of the virtual model of the hand posture, and below it, a set of check boxes and combo boxes. The fourth sub area (D) showed two rows of thumbnails, each under a command label.

In the third sub area, the first combo box from the left controls the palm position of the hand, from three possible positions: 1-Down, 2-Up, or 3-Side. The second combo box was for the wrist position, with three options: 1-Middle, 2-Left, 3-Right. The next five check boxes determined whether the finger was flexed towards the palm (not checked) or extended (checked).

Under the five check boxes there was a combo box and three other check boxes. They controlled whether there was the fingers were separated (checked) or not (not checked). The combo box allowed a third state only for the thumb: Separated and perpendicular from the extended palm. Checking the boxes and selecting from the combo boxes updated the virtual model of the hand. The radio buttons to one side of the check boxes were used to express the strength of the association between the command and the current gesture (Figure 6.6). There are three options to choose from: “Weak”, “Medium,” or “Strong”.

In the fourth sub area the thumbnails presented a miniaturized version of the virtual hand model as selected by the user for the command prompted, which appears over the respective thumbnail. Thus, every command-gesture association appears in this area of the screen.

Every selection of a command-gesture pair was added to the intuitiveness database that stored all the associations of the subjects. This database had a table including the following fields: First name, Last name, SSN, Action, Gesture, and Level. The first three fields were for user details, while the last two were the command of the task, the encoded gesture (Table 3.1, in Chapter 3), and the strength of association.

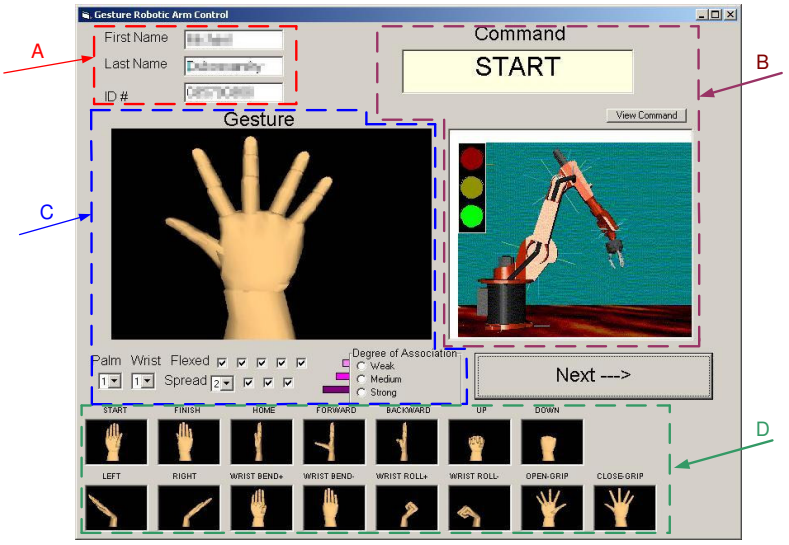


Figure 6.4. Intuitive assessment application for the robotic arm

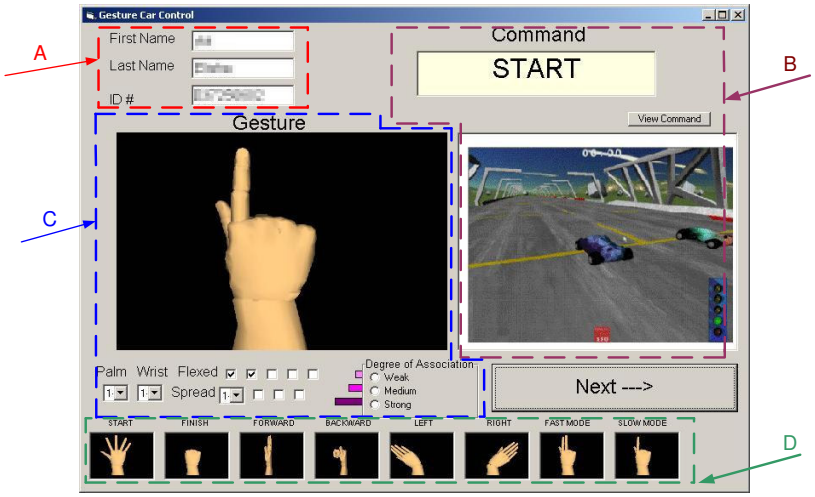


Figure 6.5. Intuitive assessment application for the VMR

Figure 6.6. Interfaces for (a) Hand virtual model and (b) Strength of association

6.3.4 Procedure

To find the intuitive mapping between commands and gestures, the user needed to see the entire set of commands required, and the action that each command represents, to fulfill a task. At the start of the experiment each subject was instructed that devices were controlled using hand gestures only. Also, at the start of the experiment, the effects of the commands were demonstrated to the users, using animated models of the robotic arm and VMR that simulated the actions that they can accomplish. The intuitive assessment application automates the collection of each user's choice of gesture in response to each command stimulant. The commands are

presented to the user in random order. The user holds the posture for only 2 seconds on the camera capture field. The palm of the hand can be in one of the three states: flat down, flat up, or on its side. The rotation of the wrist can be to the left, to the right, or in the middle. Each finger can be closed or extended, and separated from its neighbor fingers or tight. Immediately after the subject removed the hand from under the camera, the user was required to 'build' a hand posture model resembling the posture he held using an interactive virtual model of the hand embedded on the interface. The user set the configuration of all parameters of the virtual model of the hand by ticking off check boxes and selecting from the combination lists. In addition to this, the user selected the 'strength' of the association using 3 options: weak, medium or strong (Figure 6.6.).

The gestures and degrees of association were collected from 35 students, from the Industrial Engineering Department at Ben Gurion-University. In the intuitive matrix, each row was a gesture type, from the constrained set of 648 gestures, each column was a command, and each entry represented the number of subjects who selected that gesture to represent that command. This matrix can be reduced by eliminating all the gestures that no subject picked. A total of 114 and 59 gestures were selected for the robotic arm and VMR tasks, respectively. These gestures were selected at least once to represent a command. The intuitive matrix was converted to a weighted intuitive matrix, where each entry represents the number of subjects who selected that gesture for a given command, multiplied by the strength of association as stated by the user.

6.3.5 Results and analysis

6.3.5.1 Direct intuitiveness matrices

To be able to approach the GV design problem, it was necessary to reduce the number of gestures to a small master set. This was accomplished by reducing the intuitive matrix for the robotic arm experiment to a subset of the most popular gestures. The most popular gestures were selected according to those selected by a) at least five subjects, or b) at least four subjects who selected the same gesture-command association. For the VMR experiment, only gestures according to those selected by a) at least 4 subjects or b) at least 4 subjects who selected the same gesture-command association were considered. This operation reduced the master set to 23 and 22 gestures for the robotic arm and VMR tasks, respectively. The union of both master sets resulted in 27 unique gestures (Figure 6.7). The intuitive matrices for both the robotic and VMR tasks are presented in Tables D.1 and D.2 in Appendix D.

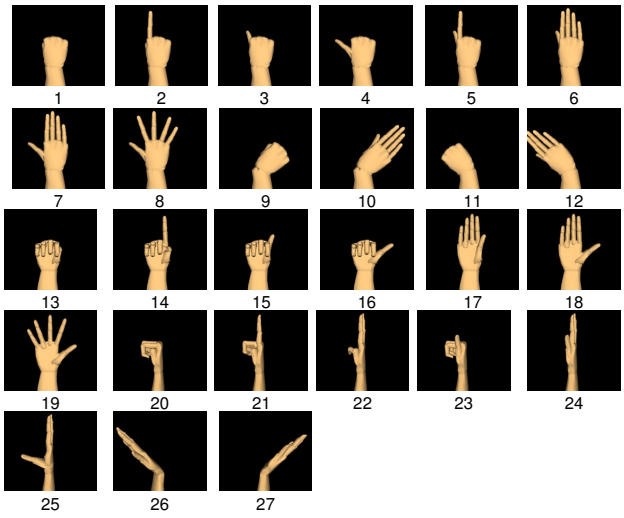


Figure 6.7. Common master set of gestures

An agreement measure S_i was used for determining the proportions of overall and specific gesture-command agreements, and was defined as follows:

$$S_i = \frac{\alpha_i}{\alpha_i^{poss}} = \sum_{k=1}^{n} a_{ik}(a_{ik} - 1) \Big/ \left(\sum_{k=1}^{n} a_{ik} \right) \left(\sum_{k=1}^{n} a_{ik} - 1 \right) \quad \textbf{(6.1)}$$

where,
α_i = specific agreement between subjects for gesture i
α^{poss}_i = maximum possible agreement between subjects for gesture i
S_i = ratio of agreement for gesture i

Then,

$$\Phi = \sum_{i=1}^{m} S_i p_i \quad \textbf{(6.2)}$$

p_i=popularity of gesture i (in terms of probability)
Φ= mean overall agreement

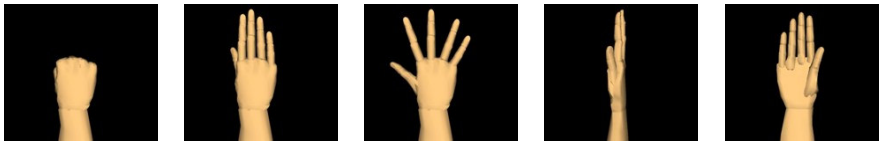
The maximum possible pair-wise agreements between those subjects who selected certain gestures gives an indication of the measure of agreement of the group studied. Table D.7 (in Appendix D) shows p_i (the probability of selecting gesture i), α_i (specific agreement between subjects for gesture i), α^{poss}_i (maximum possible agreement between subjects for gesture i), S_i (the ratio of agreement for gesture i), and the overall agreement measure (Φ).

The three most popular gestures for the VMR task were selected by 24, 21, and 15 respondents. The next two were tied with 14 each (Figure 6.8).

(a) 24 (b) 21 (c) 15 (d) 14 (e) 14

Figure 6.8. Most popular gestures (number of users) for the VMR study

As expected, these gestures were easy to compose. Gestures 6, 7, and 24 of (a) in Figure E.1, were strongly associated with the "Stop" (Finish) command (column 2) in Table D.7(a). Gestures 10, 16, and 27 were associated with the "Right" command, having been selected with ratios of 10/11, 9/11, and 8/9, respectively. These gestures are highly intuitive for this command, as they all tilt or point to the right (Figure E.1 (a)). For the robotic arm task, the most popular gestures were 1, 6, 8, 24 and 17 selected by 26,23,19,19 and 18 subjects, respectively (Figure 6.9). These gestures are natural, with gestures 6, 8 and 24 the most common and highly popular as identified by the VMR task.



(a) 26 (b) 23 (c) 19 (d) 19 (e) 18

Figure 6.9. Most popular gestures (number of users) for the robotic arm study

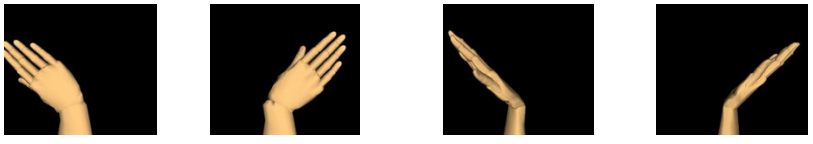
Gesture 19 was strongly associated with the “open gripper” command [column 13 in Table D.7(b)], and was selected with a ratio of 10/13. (Opening the palm of the hand is a natural depiction of the opening command.) Another highly associated gesture was number 12, which was associated with the “left” command, with a selection ratio of 10/14. The leftward orientation of the hand reflects another natural portrayal (Figure E.1 (a)). Maximum agreement was recorded for gesture 12: 100% of the subjects who selected gesture 12 (39% of the testees) assigned it the same command association in the VMR task. For the robotic arm task, 59% agreed on the command association for gesture 19 (20% of the testees). The mean total gesture-command agreements were 34% and 18% for the VMR and robotic arm tasks, respectively. Similar to the 80:20 rule of inventory theory [Juran, 1975], we found rules of 72:31 and 71:29, in which 72% and 71% of respondents selected 31% and 29% of all the gesture types, respectively, for the VMR and robotic arm task.

6.3.5.2 Complementary intuitiveness matrices

For the VMR task, the following commands are complementary: start-finish, left-right, forward-backward, and fast-slow. In the robotic arm test, the complementary commands are start-finish, left-right, forward-backward, up-down, wrist CW, wrist CCW, wrist up, wrist down, and open-close. All commands for VMR tasks had complementary commands, while for the robotic arm task, the ‘home’ command had no complementary command. The complementary commands were straightforward for the user familiar with the task. The complementary gestures g_i and g_j are represented by the pairs (g_i, g_j) (first two columns in Tables D.10 and D.11). Examples of complementary gestures appear in Figure 3.5. To avoid any assumptions in advance about whether a pair of gestures was complementary or not, all the pair-wise combinations between the gestures in the master set were initially included in the matrix. Each cell in the complementary intuitive matrix shows the number of subjects that for a given pair of complementary commands selected the complementary pair of gestures. Several possible pairs of gestures from the master sets were discarded since no respondent selected them to match a pair of complementary commands. For each task, a complementary intuitive matrix was created (Tables D.10 and D.11). The first two columns stand for a pair of complementary gestures, with indices g_1 and g_2 (the indices are the numbers over the gesture images in Figure E.1. The remaining columns represent pairs of complementing commands, and each row is a combination of complementary postures of the master set. Combinations that no participants selected are not part of the matrix.

Figure 6.10 illustrates a number of complementary commands-gesture pairs that appeared in the complementary intuitiveness matrix. The complementary commands (left-right), and the first pair of complementary gestures (a) were selected by 10 participants, while the second (b) were selected by 7. In both cases, this matching was considered highly popular. For the complementary commands up-down, the strength values were lower, only 4 participants chose (c) and a single subject chose (d). The matching was ordered from the highest (a) to the lowest (d) strength association.

Left-Right



(a) 10 (b) 7

Up-Down

(c) 4 (d) 1

Figure 6.10. Complementary commands and the matching complementary gestures

6.3.6 Discussion

The direct intuitiveness results shed light on the level of agreement of a certain population regarding the use of a set of gestures to accomplish certain tasks. Even though agreements for gesture-command associations ranged from 59%-100% for the VMR and robotic arm tasks, the overall agreement was only 34% and 18% for the VMR and robotic arm tasks, respectively. This seems to refute the claim that subjects consistently use the same gestures to represent the same commands while performing tasks, as suggested by Hauptmann [Hauptmann and McAvinney, 1993].

Regarding complementary intuitiveness, the master set for the VMR task presents an interesting case. Five gestures in the master set for the robotic arm task are missing in the master set for the VMR task, and four gestures in the master set for the VMR task do not appear in the master set for the robotic arm task. From the gestures missing in the VMR task, 9 (fist right) and 11 (fist left) were paired as complementary gestures in the robotic task for 'wrist CW/ and wrist CCW,' which are commands unique to the robotic arm gripper control, and the use of the fist is naturally suited to these actions. Some concepts regarding the choice of complementary matching can be explained using part of the data from the complementary intuitiveness matrices in Tables D.10 and D.11.

The complementary commands (left-right) (a-b) are highly popular, and one reason being the high intuitiveness between the wrist movements and their correlations with the direction of the command (for the left command, the wrist is turned left, and the same for the right direction). The postures based on palm down are more popular since they are more natural to hold (cause less strain). For the complementary commands up-down (c-d), the strength of belief values were lower, probably due to the fact that it is more difficult to resemble the (up-down) direction, when holding the hand in 2D and when the camera is located above the hand (Figure 6.3). Nevertheless, it is somehow more natural to match the "palm-up" gesture to the "up" command, and the same regarding the "down" command (Figure 6.10 (c)). Also, for the up-down commands, the last pair of gestures (Figure 6.10 (d)) is complementary because of the extension/closing of the index finger. The complementary relation is not apparent, and the connection to the commands can be explained by the index pointing out for the "up" command, and the finger retracted to show a lack of the "up" action.

6.4 Stress experiments

6.4.1 Overview

The experiments aimed to assess the static stress of the postures, the transition stress between different postures, and the duration of static and posture transitions, using subjective evaluations of the users. A predictive model was developed to predict most of the values for the transition stress and duration of transition. This was due to the large number of experiments required to

assess these measures for all possible transition gesture pairs. In this alternative approach the model was built based on empirical data obtained from the static stress experiment.

6.4.2 General set up

The experiments involved holding postures required in a work environment similar to the real environment in which a user controls robots. The WE-160 Panasonic Video Imager connected to the Matrox Meteor Standard frame grabber was used to capture gesture poses. The device included a flat plate. The user held the gestures while he was sitting with his hand extended over the surface or suspended in air at a fixed height (Figure 6.3). The camera captured the upper view of the hand, from the wrist to the end of the hand. Because of the physical set up of the gesture capture system, the application used in the experiments was designed to reflect this view of the hand (Figure 6.11).

Figure 6.11. Plan view of the user's hand

6.4.3 Software applications

The screen layout (Figure 6.12) of the application developed to collect static stress responses was divided into several areas. There was an area for the user to enter personal details including first name, last name, and ID number, and instructions in a large text box telling the user how to proceed. Below it was an image with the virtual hand posture. A time scale shows the current lapse of time. To the right of the time scale there was a scale to show the progress of the experiment, in percents. Thumbnails of the master set of gestures appear at the bottom of the display. The thumbnail of the current gesture is highlighted to show the current posture selected by the application. A vertical group of radio buttons ordered according to the "Borg scale for rating perceived exertion" [Borg, 1982] was presented in the right side of the interface. The scale had 10 levels of fatigue: 0-Nothing at all, 1-Very Weak, 2-Weak, 3-5 Moderate, 5-6 Strong, 7-9 Very Strong, and 10-Extremely Strong.

All the gestures appear randomly on the image window, and per appearance, the user rates the gesture according to the effort invested. Every selection was inserted into a static stress table, which is part of the stress database. Every record in the table includes the user details, the code of the posture, and a stress level from the Borg scale. Twenty seven gestures, which represent the master set of gestures for the robotic and VMR tasks together (union of sets of gestures of both tasks), were presented in the application. Two additional gestures were also included to present extreme conditions of fatigue. The database information was later used to find the average level of effort that the users assigned to every posture in the master set.

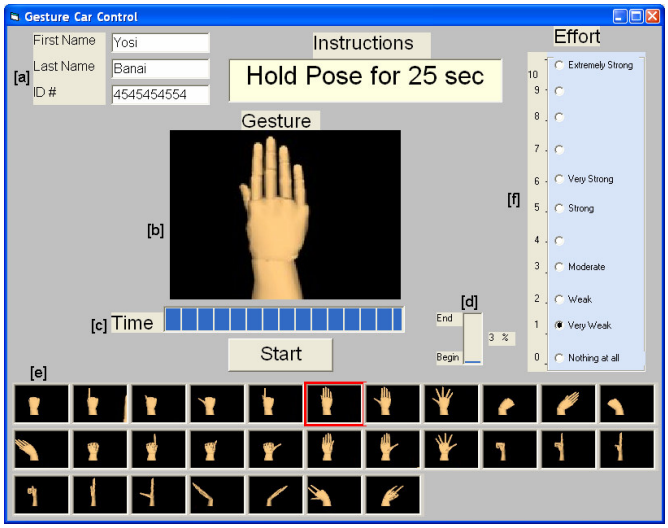


Figure 6.12. Interface for static stress experiment

The application (Figure 6.13) to measure the transition stress was similar to that used for the static stress, with the following differences: 1) there were two images with the virtual hand on them, the left image was for the beginning gesture and the right image was for the ending gesture; 2) there is no time scale; 3) there was a button under the virtual hand images. This button had three different labels: (1) "start " for the starting gesture, (2) "stop" for the ending gesture, and (3) "Finish", was written, after which the user ranked the transition between gestures.

Both start and end gestures appeared randomly on the image windows, but the same pair was never repeated. The user ranks the transition using the Borg scale, which reflects the effort to change from one posture to another. This value, user details, and the code of both postures in the gesture master set were inserted in a record entry in the stress database. The total number of posture pairs presented in the interface was 60, obtained from two subsets of six postures each from the robotic arm and VMR master sets of gestures. This information was used to find the average transition effort between gestures and enabled the dynamic stress to be partially completed.

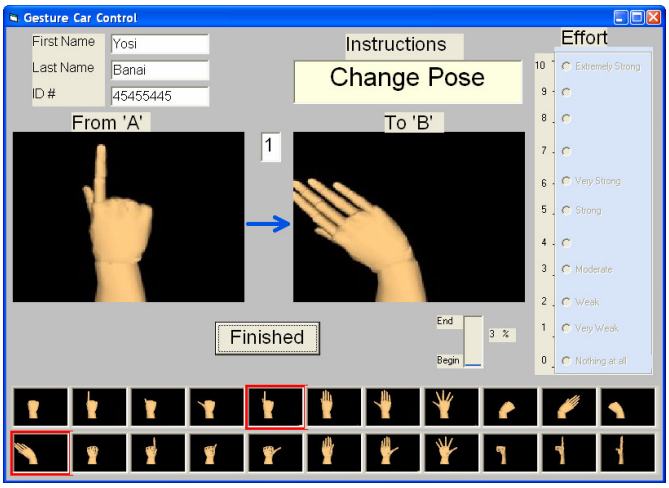


Figure 6.13. Interface for the dynamic stress experiment

6.4.4 Procedure

To measure static stress, an experiment was conducted using an application developed for this purpose. The gestures appeared in random order on the screen, and the user was asked to imitate the gesture and hold it suspended in mid-air for 25 seconds. The user's hand was placed over the flat surface of the video imager, so the upper view of his hand appeared similar to the gesture proposed in the interface. When the 25 seconds had elapsed, the user rated the effort in holding the pose on a scale of 1 to 10, with 1 being the least stressful and 10 the most. The Borg scale for rating perceived exertion was used for the rating process. Although the stress measure was relative to every subject, the experiment revealed that there were universally difficult to repeat gestures. However, there were also completely effortless gestures. For example, the 'rest' posture, in which the hand is completely relaxed, is one of these gestures. At the start of the experiment, the user was asked to experiment holding a very difficult posture and a relaxed one, to have a clear idea of both extremes of the scale. The 27 gestures discovered in the previous section plus two additional, very stressful gestures (Figure 6.14) were added to the testing set.

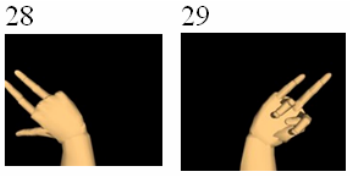


Figure 6.14. Extremely difficult postures

Static stress measures were especially useful to develop a predictive model that expressed the values of dynamic stress as functions of static stress. The model was developed using static stress measures from 29 hand gestures obtained from 19 students from both the Industrial Engineering and Management Department and the Communications Department at Ben-Gurion University of the Negev. The dynamic stress values and their durations were postulated to be a function of the starting and ending posture stress values. The current experiment captured the transition duration together with the subjective assessment of stress in carrying out the transition. In the dynamic stress application, the user controlled the flow of the transition. When the user was ready, he/she requested the 'begin' gesture of the transition, imitated it with his hand, and then requested the 'end' gesture. He changed the configuration of his hand to imitate the 'end' gesture, after which he finished the cycle by pressing the 'finish' button. The time required by the user to imitate the 'end' gesture and press the finish button was the transition time for this pair of gestures. Once the cycle was over, the participant graded the transition (from the appearance of the end gesture to the end of the cycle) with the same scale used to rate static gestures. Similar to the static gesture rating, the users were instructed to consider both physical and mental stress. Both the stress and the duration time for the transition were recorded in a database.

Without the prediction stress model, it would have been necessary to find 506 entries (23*23-23). For the VMR task, there were a total of 22 postures; however, some of the stress transitions revealed themselves when identifying stress motions and positions for the robotic arm, since there were 18 common postures between the different task vocabularies for the robotic arm tasks. Four additional postures and their transitions must therefore be added to the calculation. This would make 506 + 4 (new postures) * 22 (remainder of the postures) * 2 (transition 'to' and 'from' the posture) = 682 observations necessary. Since we already found 60 observations, it was only necessary to make an additional 622 observations. To find 30 transitions, however, requires the duration of the experiment to be 30 minutes with at least 19 students. For 622 observations, 622*19=11,818 minutes were required, and therefore a total of 197 hours were saved.

This number is prohibitive, and hence I formulated a method to gather this data based on the partial information that was already acquired in the static stress experiment. The basic intuition

that guided this reasoning was that transition stress is a function of the static stress of each of the two gestures participating in the transition. Assuming a linear relationship between the transition stress and the static stress, it is possible to create a linear regression function. The assumption of linearity was the initial approximation to model the dynamic stress, and can be confirmed empirically using subjective data. An additional route is to create a biomechanical model that can be validated with physiological results using an EMG.

To validate the linear assumption, a short experiment that utilized a small subset of the master set was performed. The training subset used all the transitions between the gestures (1,7,25,27,28,29), and transitions between all the gestures (4,6,8,10,16,27). The sample set used all the transitions between the gestures: (30,31,32,33,34,35). Twelve, seven, and seven participants took part using the first, second, and validation sets, respectively.

6.4.5 Results and analysis

The average stress and standard deviation of holding each of the master set gestures (a total of 27 originals plus 2 additional = 29), is shown in Table D.16. The results for the average transition stress using two subsets of six gestures each is presented in Tables D.17 and D.18.

Collectively, both subsets used for the transition stress experiment result in a total of 60 observations (2*6)(6-1). To establish a linear regression function, define S_{Gi} and S_{Gj} as independent variables where here S_{Gi} and S_{Gj} represent the stress of holding gesture G_i and G_j, respectively. Let the dependent variable S_{Gij} represent the transition stress from changing gesture G_i to G_j. The regression function can now be stated as

$S_{Gij}=a_1*S_{Gi} + a_2*S_{Gj}$

Note that this regression function goes through the origin. This is because a transition stress between two equal postures requires zero effort to hold (the relax posture, for example must also be zero). The regression analysis yielded an R^2 of 0.977 and regression coefficients of a_1=0.091 and a_2=0.905 with the significance levels of 0.01 and 0.0, respectively. The standardized mean squared error is a good indication of how close the initial assumption was to the reality. A scatter plot of the data and the regression line along with the detailed results of the regression are presented in Appendix H.

Similar to the model to predict the transition stress, a function to predict the transition time from a start posture to an end posture was pursued. It was reasonable to assume that changing the posture to a stressful gesture, takes longer than changing to a comfortable posture. For highly unpleasant gestures, it took several minutes to imitate it correctly (when possible). Alternatively, releasing a stressful gesture from tension occurred almost instantly.

By using all the 60 observations, including transition times from transitions between different gestures, a prediction model using linear regression through the origin (without constant term) is constructed. Let the independent variable be T_{Gij}, the transition time (duration) for changing gesture G_i to G_j. The hypothesed regression function then becomes:

$T_{Gij}=b_1*S_{Gi} + b_2*S_{Gj}$

In this case, R^2= 0.95, and b_1=0.104 and b_2=0.973 with the significance of the coefficients 0.063 and 0.000 respectively. Results of the regression run and a scatter plot can be found in Appendix H.

A subset of 6 out-of-sample gestures (taken from the large master set G_z , not from the master set G_m), labeled 30-35, was used to validate the resultant regression functions. Seven participants (not those that participated in the training) took part in the validation experiments. Figures 6.15

and 6.16 show the regression results for the transition stress and transition duration experiments, respectively.

6.4.6 Discussion

From Table D.16 gestures 29 and 28 received high stress values, and hence, corroborated the initial assumption that those gestures were stressful to hold (Figure 6.14). They also received higher standard deviations, probably since difficult gestures were perceived slightly different by different people (depending on tendon flexibility and the skill of the participant), while there was a wide consensus regarding the stress levels of medium and low stress level gestures. Another highly stressful gesture, number 12 (Figure 6.17), received similarly high stress values; it is the next most difficult gesture after 28 and 29 with a high standard deviation. For gesture 12 the wrist is bent toward the thumb, which is less stressful than bending the wrist toward the ring finger. However, this posture is considered a special case of ulnar deviation, and indeed, it is one of the most difficult cases described above.

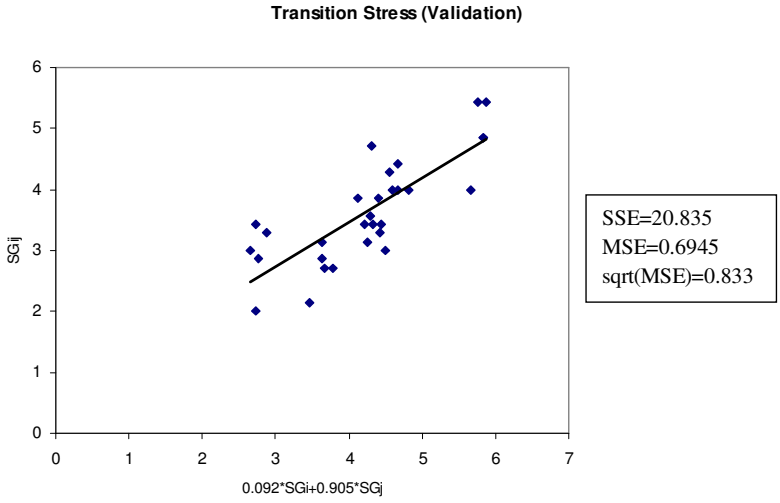


Figure 6.15. Plot between real and predicted transition stress (validation)

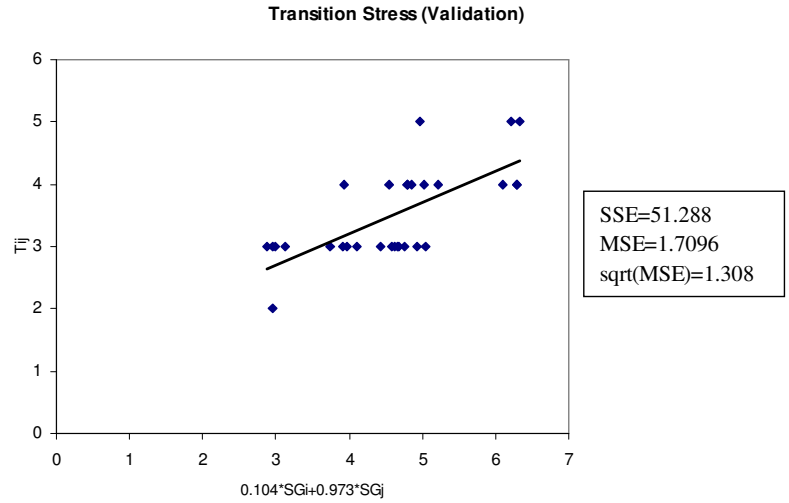
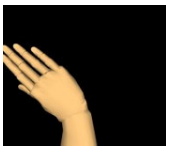


Figure 6.16. Plot between the actual and predicted duration time (validation)



12

Figure 6.17. Difficult gesture caused by ulnar deviation

Gesture difficulty levels were determined based on the effects of blood flow restriction to the stressed joints during static load, which caused strain and fatigue on the muscles. The higher the strain, the greater was the level of difficulty. In terms of their limbs, people preferred being in motion rather than maintaining static postures. For example, when holding a stressful posture for an extended period, we extend and retract the finger joints to relieve the stiffness. Frequently, fatigue is concentrated on the wrist as a result of its posture and as a direct result of repetitive finger and hand movements. Several general guidelines are used regarding wrist postures [Griffins, 2001]:

1) Avoid Extension: Bending the hand upward at the wrist
2) Avoid Flexion: Bending the hand downward at the wrist
3) Ulnar Deviation: Bending the wrist toward the ring (little) finger.

Regarding the transition stress, interesting results were obtained from the experiments. The regression analysis conducted with both subsets of gestures indicates that the transition stress is affected primarily by the ending posture. Specifically, 90% of the static stress of the final posture was present in the transition stress between the starting and ending postures (the coefficients of the regression were a_1=0.091 and a_2=0.905, respectively). The same was observed for the duration of the transition, which showed 90% dependence on the ending posture stress (the coefficients of regression were b_1= 0.104 and b_2=0.973). Without the prediction stress model, it would have been necessary to find 30 transitions, involving a 30 minute experiment with at least 19 students. For 622 observations, 622*19=11818 minutes were required; therefore, a total of 197 hours were saved.

6.5 Validation experiment (task completion time performance)

6.5.1 Overview

These experiments were designed to test the claim presented in the beginning of this thesis that the analytical performance measures Z_1, Z_2, Z_3 may act collectively as proxies for task completion time. This implies that good vocabularies, indicated by high intuitiveness, comfort, and accuracy, correspond with reduced task completion time; bad vocabularies, on the other hand, with low intuitiveness, high stress (low comfort), and low recognition accuracies, correspond to longer task times. We state this in terms of the hypothesis (1.2) (one for each task). GV_G and GV_B are sets of GVs where $GV_G \succ GV_B$ i.e. $Z^G_i > Z^B_i$ (i=1,2,3). With respect to the performance of a task, we will obtain learning rate curves for repeated trials of a given vocabulary. The learning curve showed steady improvements in performance as the task was repeated [Asher, 1956; Boston Consulting Group, 1970; Wright, 1936]. We selected the standard times of the learning curve to represent the run time performance of a given GV. Thirty two users participated in this experiment. Each user tried one different GV for 15 trials. Previous experiments showed that 15 trials were enough to reach standard times. [Wachs *et al.,* 2002].

6.5.2 General set up

The following experiments use the hand gesture recognition system, with a setup similar to that in Section 6.4. The only difference was that the video capture stream was activated, using the

WE-160 Panasonic Video Imager connected to the Matrox Meteor Standard frame grabber. The user controlled the actions in the applications by evoking commands using hand gestures. Figure 6.11 shows a top view of the gestures that were captured from the wrist to the finger tips.

6.5.3 Software application

Three applications were used in this experiment. The first was an interface that lets the user select the type of task to complete and the type of vocabulary to be used in the task. The user's selection must respect a guideline in which a task and vocabulary are assigned to each user. Two types were presented to the user: the VMR and the robotic arm tasks, and the user must select one. The vocabularies were indexed from 1 to 16 for each type of task. The first eight were considered "Good GV's" and the last eight were "Bad GV's", to be described in Section 6.5.5.1; however, no indication was given to the users about this determination. On the right side of the screen the number of times that each task was performed by the same user was displayed, (Figure 6.18).

Figure 6.18. Main application for task and vocabulary selection

The main interface launched either the hand gesture robotic arm control system (Figure 6.19) or the hand gesture VMR control system (Figure 6.20) depending on user selection.

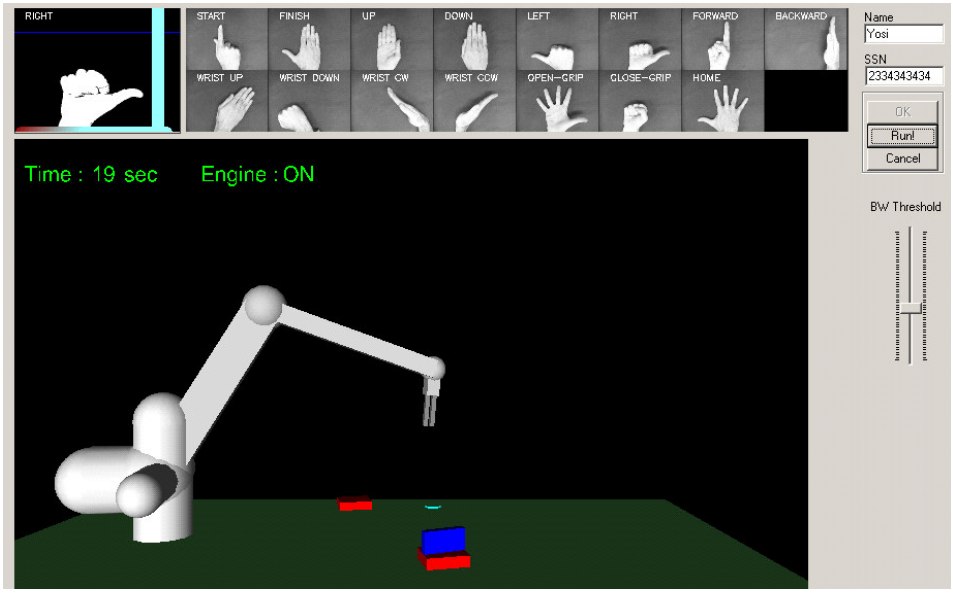


Figure 6.19. Hand gesture robotic arm control system

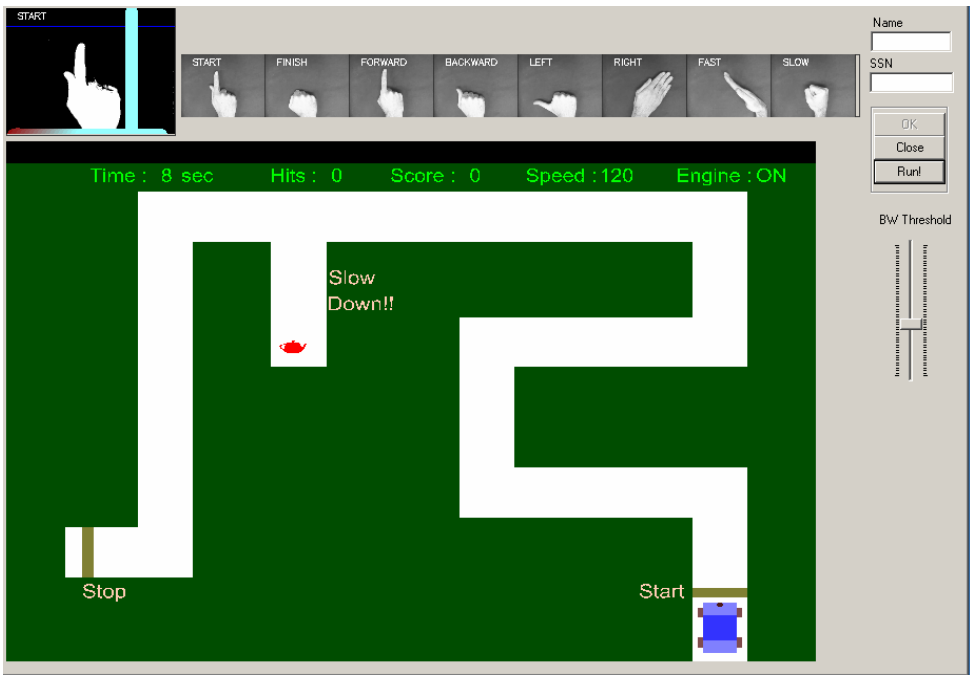


Figure 6.20. Hand gesture VMR control system

Each system had a similar layout. The top left side of the screen showed the capture window, which presented continuous video images acquired with the Panasonic Video Imager. The hand gesture was displayed when the user held his hand under the camera and appeared black and white as a result of the preprocessing stage. A small label with the name of the command appeared in the top left of this window when the gesture was recognized; otherwise, "Unrecognized" appeared. On the top center of the screen, a row of vocabulary images were shown. The row of thumbnail images of gestures in the GV was displayed as a reminder. Above each gesture a command name associated with the gesture was presented. This way the user knew which command was evoked while the user was holding a posture. Below the capture window, in the main area of the screen, the virtual 3D model of a 5 DOF robotic arm or a VMR were presented according to the type of task. On the right of the main screen there were two buttons: "Run" was used to start the video capture process, and "Close" was used to close the application.

6.5.4 Procedure

Sixteen subjects participated in this experiment. One type of task (Robotic arm or VMR) and a vocabulary (from the sixteen GVs) were assigned to each subject. The assignment (user, task type, and vocabulary) was given to the user by the tutor. The first eight vocabularies were from the V_G set ($GV_{G(i)}$, i=1,..,8), and the last eight were from the V_B set ($GV_{B(i)}$, i=1,..,8). Each subject tested one vocabulary. Once the subject selected the task and vocabulary from the main interface (Figure 6.18), the application with the appropriate task was launched. The controlling commands were configured to work with the hand GV selected by the user. The task was explained to the user, following the description in Sections 6.1.1 and 6.1.2. The procedure to complete the task remained the same, but the user must control the devices using only hand gestures. To initiate the user on proper posture position, the user was allowed some trial exercises. The user was asked to try each gesture sequentially until every gesture was recognized. Recognition was achieved when the command assigned to the gesture was presented in the top

left of the capture window. This process was necessary to ensure the user knew how to hold the posture correctly. Of course, this did not guarantee that the testee formed and held the gesture correctly during the actual execution of the task. Moreover, successful learning of the gesture required repeated practice.

Once the user knew the commands using hand gestures, he was allowed to start the task. In both applications there was a "start" command to begin the task, and once this action was evoked, the completion task time was initialized and displayed in the task view. The completion time was stopped and recorded after the user evoked the "stop" command upon task completion. In addition to the completion time, the first and last names and the identification number of the user were stored. For the VMR task, impacts of the sides of the road were stored as well. After the first trial the subjects received feedback on task performance. The subjects could also raise questions about the task. In subsequent trials, the user completed the task and no help was provided to him. Each subject repeated the experiment 15 times for each assigned task-GV.

6.5.5 Results and analysis

6.5.5.1 Generation of good and bad vocabularies (V_G and V_B)

To validate the statement that there was a relation between the GV selected and the corresponding task completion time, it was necessary to find eight V_G and V_B vocabularies per task type. The V_G vocabularies were dominating solutions of the V_B vocabularies, which means that each GV that was from the V_G set of vocabularies had higher associated values for the three indices (accuracy, intuitiveness, and comfort) than each GV from the V_B set. Both vocabularies, V_G and V_B, were obtained from a series of solutions generated using an initial subset of solutions and a combination of weights. Using the CMD (see Chapter 4.5), the initial subset of the solution was obtained. The V_G and V_B sets were obtained from initial solutions with high and low recognition accuracy, respectively. To acquire solutions with high recognition accuracy, the CMD can be used as presented in Chapter 4.5. However, to get low recognition accuracy solutions, instead of looking for the Max argument in (4.11), the Min argument was necessary. Eight high recognition candidate gesture sets using A_{min}=96.25%, and an additional eight low recognition candidate gesture sets using A_{max}=87.81%, were selected. The subsets of gestures that yielded high recognition accuracy were called "G_{HA}," and the subsets that yielded low recognition accuracy were called "G_{LA}." The same procedure was used for the robotic arm case, where the subset of necessary gestures was ten. The upper and lower bounds were A_{min}= 98.33% and A_{max}=90.667% for the G_{HA} and G_{LA}, respectively. Tables 6.3 and 6.4 show the 16 subsets of gestures for the VMR and 10 subsets of gestures for the robotic arm cases, respectively. The top half of the table included the solutions for the G_{HA} and the bottom half those for the G_{LA}.

From the initial solutions obtained from Tables 6.3 and 6.4, a series of associated solutions were generated. To obtain a set of associated solutions for a given GV from G_{HA} or G_{LA}, the weights w_1 and w_2, given to the intuitiveness and comfort objectives in the QAP(G_n) (see P 4.4), were varied. Each of the weights w_1 and w_2 were varied from 1 to 10 in steps of 1, such that $w_1+w_2=10$. Hence, a total of 11 combinations of weights for each of the 16 solutions for the robotic arm case yielded 176 solution points. For the VMR case, the 11 weight combinations for each of the 10 solutions, yielded 110 solution points. These solutions appear in Figures 6.21 and 6.22, where the solutions generated using the same G_{HA} or G_{LA} (same accuracy value) are represented with the same icon

Table 6.3. Initial subset of gestures for the VMR case

id	G_{HA+LA}	Acc(%)
9	6 7 8 10 12 13 17 21	99.38
10	6 7 8 10 12 17 20 21	99.38
11	6 7 8 10 12 17 21 23	96.25
12	6 7 8 10 12 17 21 24	99.06
13	6 7 8 10 12 17 18 21	99.69
14	6 7 8 10 12 17 22 24	97.50
15	6 7 8 10 12 17 18 20	99.38
16	6 7 8 10 17 21 26 27	99.69
1	1 2 3 4 5 10 20 26	88.13
2	1 2 3 4 5 10 20 23	84.69
3	1 2 3 4 5 10 22 26	86.88
4	1 2 3 4 5 10 17 22	86.56
5	1 2 3 4 5 10 13 17	87.81
6	1 2 3 4 5 10 13 23	84.69
7	1 2 3 4 5 10 13 22	86.56
8	1 2 3 4 5 10 18 22	86.88

Table 6.4. Initial subset of gestures for the robotic arm case

id	G_{HA+LA}	Acc(%)
1	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	98.5
2	4 5 6 7 8 10 11 14 16 17 19 20 24 26 27	98.5
3	5 6 7 8 10 11 13 14 16 17 19 20 24 26 27	98.33
4	4 5 6 7 8 10 11 13 14 16 19 20 24 26 27	98.33
5	4 5 6 7 8 10 11 13 14 16 17 19 24 26 27	98.33
6	1 2 3 4 5 9 12 13 15 16 17 19 20 23 27	90.67
7	1 2 3 4 5 6 9 12 13 15 16 17 19 20 23	90.67
8	1 2 3 4 5 7 9 12 13 15 16 17 19 20 23	90.67
9	1 2 3 4 5 8 9 12 13 15 16 17 19 20 23	90.67
10	1 2 3 4 5 9 10 12 13 15 16 17 19 20 23	90.67

.

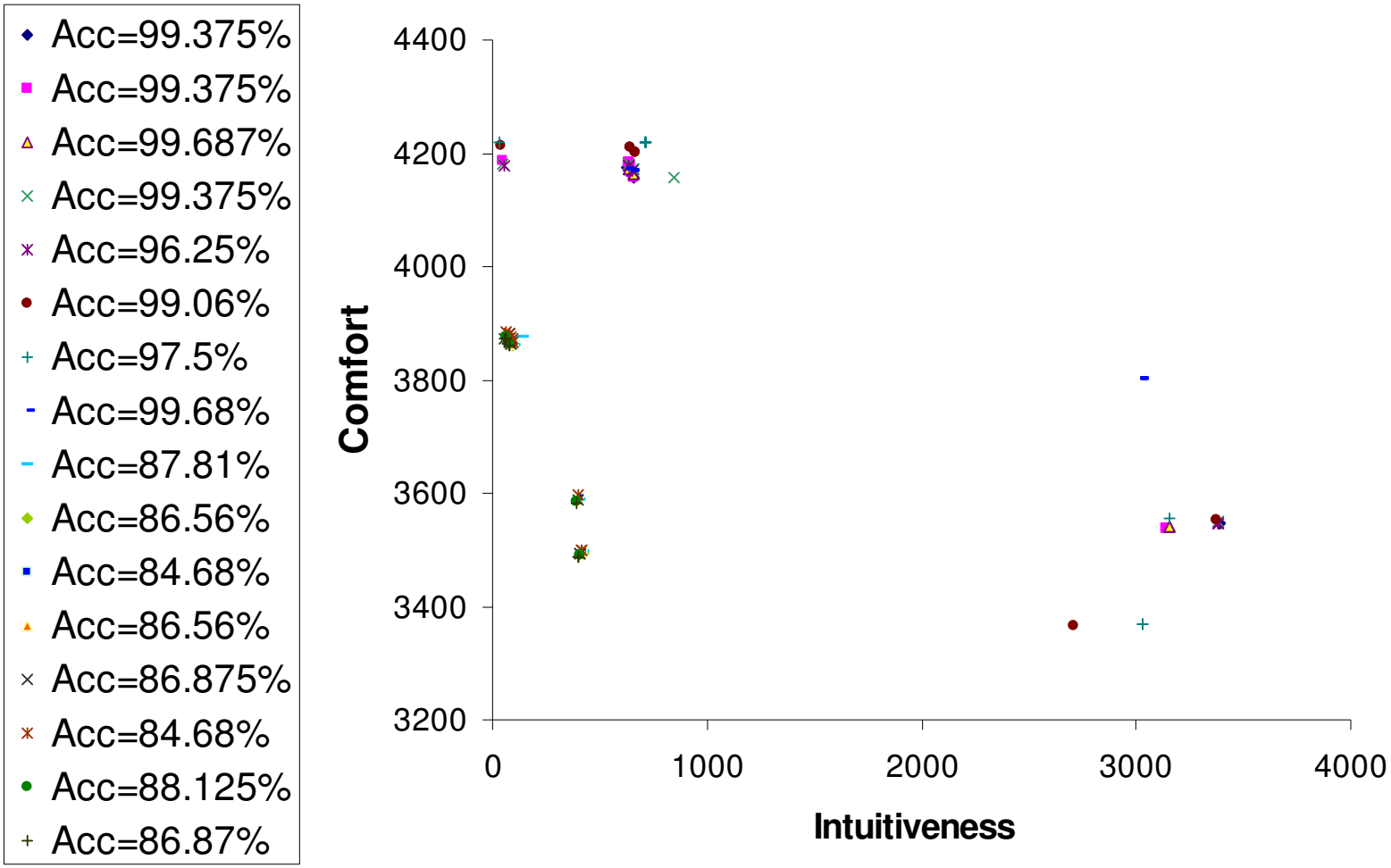


Figure 6.21. Intuitiveness vs. comfort for 16 gesture subsets for the VMR study

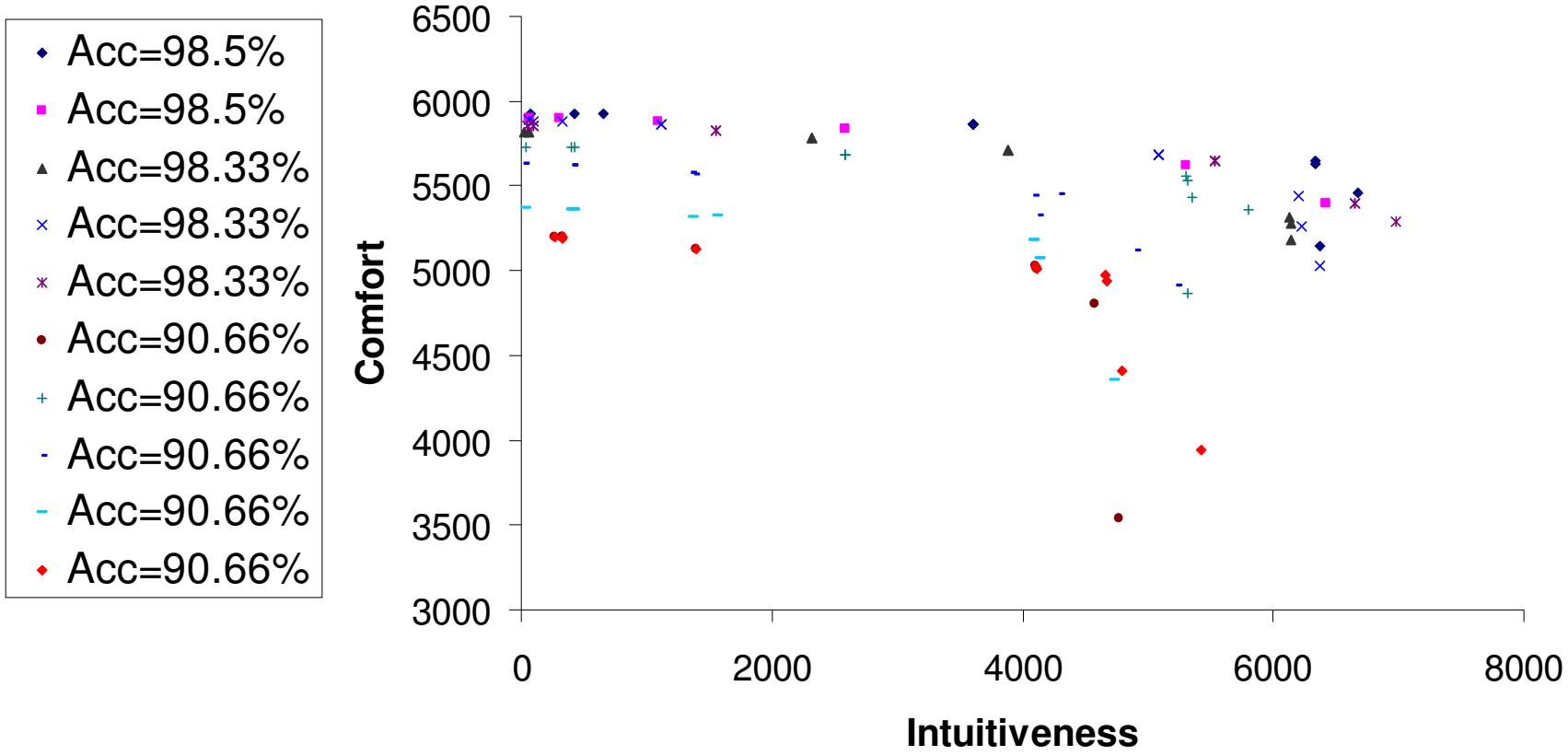


Figure 6.22. Intuitiveness vs. comfort for 16 gesture subsets for the robotic arm study

It should be noted that a few equal solutions were obtained for different values of the weights (w_1,w_2), and thus appear as a single point in the graph. It is also noted that these tradeoff curves are mostly piecewise convex [a few points do not follow this pattern due to the non-exact solutions obtained from the simulated annealing approach used to solve the integer QAP(G_n) (P 4.4)].

Eight dominating and dominated solutions were selected from the 176 and 110 solutions for the robotic arm and VMR studies, respectively. The selection can be done visually by picking the points that are placed on the upper right side of the plot for the V_G set, and by selecting points on the bottom left side of the plot for the V_B set. The points for the V_B set must be dominated by the points selected for the V_G set. For an explanation of dominating and dominated solutions, see Appendix B. Table B.1 presents the 16 GV solutions. The first 8 rows are for the V_B solutions and the last 8 are for the V_G solutions. Table B.2 presents the results for the robotic arm study. To see the images of the gestures associated with the commands for each of the V_G and V_B sets, see Appendix C.

6.5.5.2 The user learning curves

The task completion times (for the robotic arm and VMR tasks) for the 16 users in the 15 trials, (total of 240 trials) are presented in the Tables 6.5 and 6.6, respectively.

Table 6.5. Completion time (in seconds) for the robotic arm task

GV \ Trials		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AVG	AVG
	1	283	177	151	153	213	125	96	107	104	107	95	88	86	95	99	93	
	2	171	229	177	144	165	197	94	99	126	85	96	85	81	99	84	88	
	3	240	232	150	138	128	115	129	141	111	108	98	88	89	80	85	85	
V_G	4	207	144	123	92	116	104	93	75	77	80	95	92	71	83	67	74	88
	5	208	139	99	112	106	84	88	83	88	159	86	121	80	87	88	85	
	6	223	183	160	136	162	242	118	97	92	112	100	102	89	90	81	87	
	7	244	236	292	155	164	147	100	97	109	102	101	97	104	122	96	107	
	8	167	121	98	81	107	90	87	78	83	115	78	89	84	62	109	85	
	9	260	300	129	135	150	91	136	133	125	126	131	134	87	103	98	96	
	10	255	131	118	136	140	142	136	101	96	218	96	83	110	107	80	99	
	11	300	293	300	236	243	236	300	162	208	157	198	127	100	125	132	119	
V_B	12	300	229	221	165	123	130	155	169	125	111	111	114	91	115	114	107	119
	13	300	300	237	255	300	209	271	250	134	121	191	124	139	164	125	143	
	14	300	282	296	242	294	257	160	173	187	179	166	122	115	126	155	132	
	15	300	260	184	300	148	178	137	148	124	115	98	107	102	85	91	93	
	16	190	196	223	135	226	128	236	265	266	135	267	184	141	182	168	164	

.

Table 6.6. Completion time (in seconds) for the VMR task

GV \ Trials		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AVG	AVG
	9	300	278	180	188	179	162	149	143	144	114	103	112	121	103	123	116	
	10	294	174	148	136	151	138	134	130	138	115	120	117	118	126	118	121	
	11	229	207	186	137	142	133	147	133	145	140	132	129	116	117	119	117	
V_G	12	177	142	158	170	133	129	119	119	114	115	112	121	113	111	104	109	115
	13	300	251	222	181	255	168	163	160	135	133	139	145	121	132	134	129	
	14	275	182	135	141	127	115	110	110	101	97	97	103	98	104	95	99	
	15	261	148	145	224	136	160	149	159	143	156	143	130	131	142	113	129	
	16	160	138	122	130	106	110	111	100	98	105	106	103	96	103	94	98	
	1	300	283	226	182	173	175	194		170	170	167	178	180	130	147	152	
	2	255	255	239	186	189	222	214	151	169	169	157	151	144	128	200	157	
	3	300	300	300	190	183	147	171	147	188	127	180	132	127	153	155	145	
V_B	4	300	300	265	249	231	233	192	167	177	180	137	126	156	155	143	151	153
	5	300	252	212	196	195	171	169	155	173	149	141	130	146	135	143	141	
	6	300	265	242	233	158	189	144	162	127	147	119	125	113	134	112	120	
	7	300	326	229	251	243	253	241	202	214	200	192	223	214	157	169	180	
	8	251	194	173	205	182	161	204	205	300	159	211	228	171	191	170	177	

In these tables, the rows stand for the different GV's and the columns for the trial numbers. Each GV was tested by a different participant. The last column of the tables shows the average of the last 3 trials, for which we consider the standard task completion time. The first eight rows used "good GV's" (V_G) while the last eight rows used "bad GV's" (V_B) for the robotic arm task; the opposite was true for the VMR task. Figure 6.23 shows the robotic arm task learning curves for the V_G and V_B vocabularies, respectively. The learning curves for V_G and V_B obtained in the VMR task are presented in Figure 6.24. Scatter plots can be found in Appendix G.

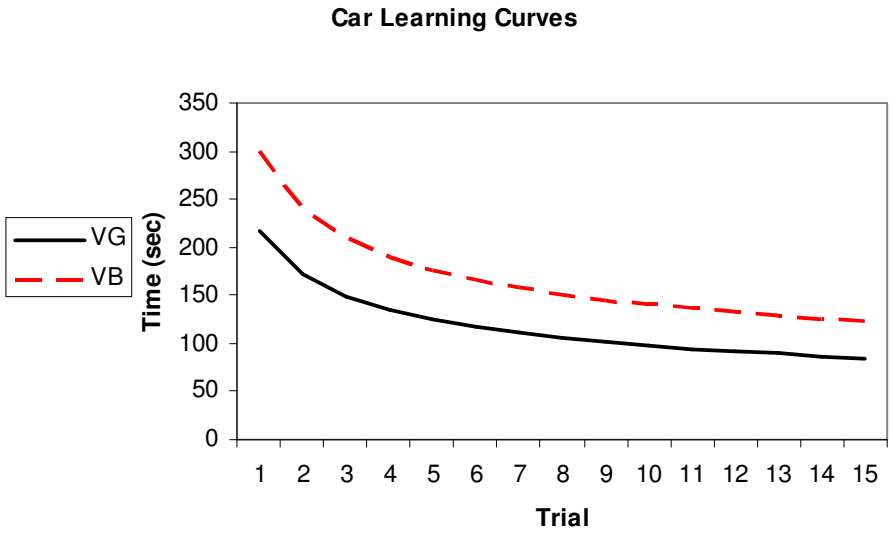


Figure 6.23. Learning curve for the V_G and V_B vocabularies used for the robotic arm task

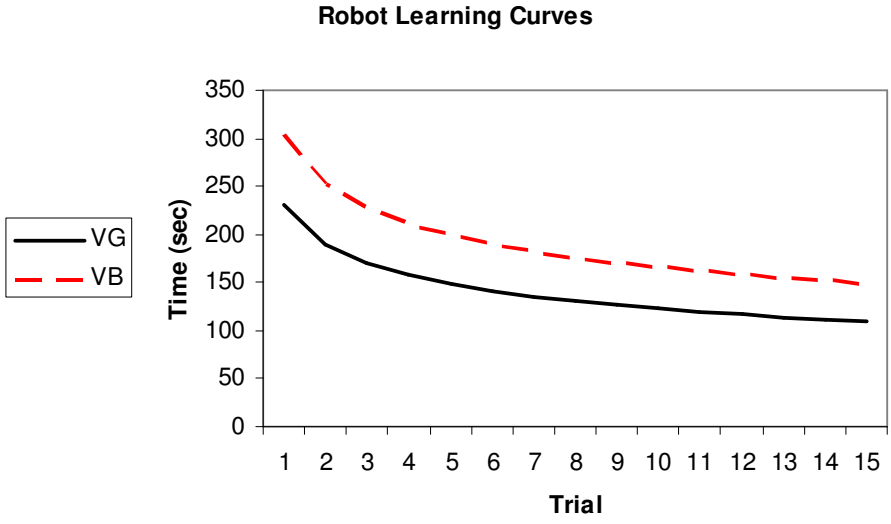


Figure 6.24. Learning curve for the V_G and V_B vocabularies used for the VMR task

The best fit learning curve equations (3.14) for the V_G and V_B vocabularies for the robotic arm task are $Y_n=217.89\ n^{-0.348}$ and $Y_n=298.27\ n^{-0.327}$, respectively. The learning rate, using (3.15) for the V_G and V_B are r=0.785 and r=0.797, respectively. In the VMR task the learning equations for the V_G and V_B vocabularies are $Y_n=229.86\ n^{-0.273}$ and $Y_n=302.16\ n^{-0.260}$, and the learning rates, for the V_G and V_B are r=0.827 and r=0.835, respectively. A lower learning rate means faster learning.

To test the hypothesis in (1.2), a t-test was performed between standard completion times for the V_G and V_B sets for both the robotic arm and VMR. To determine the standard times, an average of the last three trials of the learning curve was taken. Table 6.5 shows that the mean completion time for the robotic arm task using V_G was much shorter than the time using V_B ($\tau(GV_G)$ =87.98 sec< $\tau(GV_B)$=118.95 sec). This was true with a p=0.0059 at the 0.5% level of significance. For the VMR task (Table 6.6) the task completion time using V_G was shorter than using V_B ($\tau(GV_G)$=114.67 sec< $\tau(GV_B)$=153.04 sec). This was true with p=0.00031 at a 0.03 percent level of significance. The complete t-test runs are in Appendix H

6.5.6 Discussion

The experimental results indicate the connection between the selected GVs and task completion and learning times. For the robotic arm task, the learning curves showed that standard times were reached after 13 trials. When using V_G, the average standard task completion times during the last three trials a were shorter than when using the V_B (p= 0.0059). For the VMR task, in which the averaged completion time was compared between the V_G and V_B vocabularies, the time to complete the task using V_G was also significantly shorter than when using V_B (p=0.000313). The regression results, two exponential learning curves, incorporated learning rates of r=0.785 and r=0.827 for V_G and V_B, respectively. For the VMR and robotic arm tasks, the results were r=0.827 and r=0.835 for the V_G and V_B, respectively.

In both tasks, the use of the V_G vocabulary yielded shorter standard task completion times (the robotic arm task $\tau(GV_G)$ =88 sec< $\tau(GV_B)$ =119 sec, and the VMR task $\tau(GV_G)$ =115 sec < $\tau(GV_B)$ =153 sec). Therefore, the main hypothesis (1.2) is true, indicating that the use of more natural vocabularies have a positive direct impact on the performance of the task, by reducing its completion time. In the case of V_G, the first trial is much shorter (the robotic arm task 218 sec< 298 sec, and the VMR task 230 sec < 302 sec) than when using V_B, which corroborates that V_G is easier to use for a beginner than V_B. Regarding the learning rate, it was lower for V_G than for V_B (the robotic arm task 0.785<0.797, the VMR task 0.827<0.835). A smaller learning rate represents faster learning, supporting the posit that beginner users will learn faster, and thus achieve shorter standard times more quickly, when using V_G than when using V_B.

The process of learning is related strongly to the intuitiveness of the vocabulary but the performance time is also affected by vocabulary stress factors.

6.6 The memorability test experiment

6.6.1 Overview

To establish whether there is a relation between the naturalness of a GV and the memorability of the subject when using that GV, a post-validity experiment was conducted. This experiment was performed immediately after finishing the task completion time experiment (Chapter 6.5). Through a software application, the users matched commands to gestures, reflecting the associations existing in the GV that were assigned to them in the previous experiment. The goal of this chapter is to validate the hypothesis (1.3).

6.6.2 General set up

The memorability experiment required only a computer station placed next to the computer with the Panasonic Video Imager, and hence, the user was not able to see the previous setup to avoid clues in the memory testing process. In this PC station, an application was executed for the subject's use.

6.6.3 Software application

The memorability application (Appendix A) was based on a computer screen display where on the left side there was a list of all the commands necessary to complete the task. Close to each command there was an arrow icon pointing to the opposite side to the command. On the bottom part of the form there was a collection of thumbnails representing the common set of gestures. Each thumbnail could be dragged to the right side of an arrow to match it with a command.

6.6.4 Procedure

At the completion of each of the 15 trials for each task completion time experiment, the subject was presented with the memorability application discussed above. For the robotic task the list included 15 commands, while for the VMR it only included 8. At the bottom of the form there was a group of 27 gesture posture thumbnails. The user was instructed for each command to select and drag a thumbnail adjacent to it. This represented the gesture the subject remembered as being associated with the command during the experiments. It was explained to the user that he would remain with extra thumbnails that were not selected to be associated with the commands at the end of the test. In case the user does not remember an association, he/she can choose to leave the command without a paired thumbnail. When the memorability test was finished, the user filled out a hardcopy feedback form where he/she could, for example, express any problems encountered and suggest improvements. Additional information about gender, handedness, or coordination problems was also collected in this form. Figure A.3 shows a copy of the feedback form.

6.6.5 Results and analysis

The score for this task was a measure of memorability based on the percent of correct associations. For the robotic task, Tables 6.7 and 6.8 show the result of the memorability test for the robotic and VMR tasks, respectively. The success column stands for the memorability score in percent. The last column indicates the type of vocabulary (V_G for Good GV, and V_B for Bad GV). The average memorability scores for the robotic task were found to be 87.5 and 70.83% for the V_G and V_B, respectively. To confirm whether this was significantly different, a t-test was conducted. The t-test results are shown in Table H.9 in Appendix H. The result showed that the mean scores were significantly different (0.053 ~= 0.05) at the 5 percent level.

For the VMR task, the average percent memorability scores were high at 96.66 and 95.00%. The t-test showed that the difference was far from being significant (0.58>>0.05) at the 5% level. The t-test results are shown in Table H.10 in Appendix H.

6.6.6 Discussion

The robotic V_G vocabulary memorability test performed immediately after the 15 repetitions of the task trials shows that two individuals found all matchings between commands and gestures with no errors. The worst test was done by one individual who confused four matchings. With the same task, however, using the V_B, only one subject succeeded in matching all command-gestures

correctly. The worst performance included nine matching mistakes. It can be concluded that the more natural vocabulary was easier to remember than the less natural (p=0.053).

Table 6.7. Memorability score test for the robotic arm task

Subj N	GV	Gender	Errors	Success (%)	Type
1	1	F	4	73.33	GG
2	2	M	0	100	GG
3	3	M	0	100	GG
4	4	M	2	86.67	GG
5	5	F	2	86.67	GG
6	6	M	3	80.00	GG
7	7	F	3	80.00	GG
8	8	M	1	93.33	GG
9	9	M	5	66.67	GB
10	10	M	3	80.00	GB
11	11	F	8	46.67	GB
12	12	F	9	40.00	GB
13	13	M	2	86.67	GB
14	14	F	5	66.67	GB
15	15	F	3	80.00	GB
16	16	F	0	100	GB

Table 6.8. Memorability score test for the VMR task

Subj N	GV	Gender	Errors	Success (%)	Type
1	1	M	0	100	GB
2	2	M	0	100	GB
3	3	M	1	93.33	GB
4	4	F	0	100	GB
5	5	M	1	93.33	GB
6	6	M	3	80.00	GB
7	7	M	1	93.33	GB
8	8	M	0	100	GB
9	9	M	1	93.33	GG
10	10	M	1	93.33	GG
11	11	F	0	100	GG
12	12	M	0	100	GG
13	13	F	2	86.67	GG
14	14	M	0	100	GG
15	15	F	0	100	GG
16	16	M	0	100	GG

The same test was performed on the VMR task. It was found that using the V_G, 5 subjects matched all commands to the correct gestures. In the worst case, two mismatchings were done by one subject. Using V_B, 4 subjects found all the associations (0 mistakes), and in the worst performance one subject made 3 mistakes. However the results for the VMR task were not statistically significant. It seems that the reason that there was no significant difference in memorability for good and bad vocabularies is that the VMR task included only eight commands-gestures associations. It was not difficult to remember a limited number of associations even when there was no correlation at all between the objects to be associated. When the number of associations grows, any clue that may help to find a correct association is highly valuable. The naturalness of a vocabulary is a considerable clue for large vocabularies as shown in the robotic arm memorability test.

7 Case studies

7.1 Overview

The previous chapter described all the steps required to obtain the human psycho-physiological input factors (Figure 3.1) for the robotic arm pick and place and a VMR drive task. The present chapter adopts the same two tasks as case studies to obtain candidates of hand GVs for use in a multiobjective criterion problem (3.2). Specifically, this chapter shows how the input factor matrices obtained in module 1 of the architecture (Figure 3.1) were applied to modules 1 and 2 of the vocabulary methodology. This means finding a feasible subset of gestures using either of two decomposition methods (DCM or CMD), and then using a complete enumeration method. Only a limited enumeration run is presented due to the large search space of the problem. For module 3, the command matching algorithm is used for each subset of gestures. The feasible solutions (GVs) are presented to the decision maker together with an approximate set of Pareto points GV' to aid the decision maker in the selection of a single GV. Denote the set of feasible GV solutions found as Γ. A solution GV'$\in \Gamma$ is said to be Pareto optimal (or a non-dominated solution for the MOP), if and only if there is no other GV$\in \Gamma$ such that $Z_i(GV) \geq Z_i(GV')$ for all i=1,2,3, with at least one strict inequality.

7.2 Determination of input matrices – module 1

As a result of the human factors experiments, the frequency, direct and complementary intuitiveness, and the stress and duration matrices were obtained. A normalization step was necessary to have all the matrices in the same range of values. Let b_{ij} be elements of any arbitrary matrix B that we want to normalize. Let Q_{tot} be the sum of all the elements in the matrix. Let ℓ be the scaling factor, and $\bar{b}_{ij}$ the elements of normalized matrix B. The values of $\bar{b}_{ij}$ are obtained by applying (7.1) and (7.2). The scaling factor $\boldsymbol{\ell}$ was 1,000 for the frequency, direct and complementary matrices, and was 100000 for the stress and duration matrices to get normalized values from 0-999.

$$Q_{tot} = \sum_{i}^{m} \sum_{j}^{m} f_{ij} \tag{7.1}$$

$$\bar{b}_{ij} = \frac{b_{ij}}{Q} \times \ell \tag{7.2}$$

Normalized matrices for the robotic arm and VMR task, respectively, were calculated using (7.1) and (7.2). In Tables D.25 and D.26 the frequency matrices have rows and column indices corresponding to commands. For the intuitive matrices (Tables D.5 and D.6), the indices of the gestures appear in the first column, and the indices for the commands appear in the first row. The commands and their respective index are presented in Tables 6.1 and 6.2.

Tables D.13 and D.14 show the complementary matrix with the indices of each gesture in the first and second columns, the rest of the columns are for the indices of pairs of complementary commands. For example, the index '1' represents the first two commands in Tables 6.1 or 6.2, the index '2' represents the second pair of commands in those tables, and so on. Both the direct and complementary intuitiveness matrices used were weighted. The stress and duration matrices show values for the union of gestures used in both tasks. There are 22 and 23 gestures in the VMR and robotic arm gesture master set, respectively (Appendix E). The process to obtain these master sets was detailed in section 6.3.5.1. There are 27 gestures in common between both master sets; therefore, only one matrix for the stress and duration values was necessary.

7.3 Finding the recognition accuracy for G_n using the calibrated FCM – module 2

The proposed methodology explained in chapter 5 was used to calibrate the FCM algorithm. This procedure requires several iterations to converge for a given subset of gesture instances. Its complexity grows with the number of gestures used in the training set. Since it is not known which subset yields the best accuracy, it was proposed earlier to find this subset through the construction of a tree of solutions where each child of the tree is a subset of gestures obtained with the help of a Disruptive Confusion Matrix (DCM). For every node, the supervised FCM must find the best set of operational parameters, and hence, the best recognition accuracy.

This method is useful for systems with a small number of gestures in the master set (<20). However for larger master sets of gestures or larger subset sizes, this process may take several days[vii] to find a candidate solution GV.

7.3.1 Approximate accuracy method

A different approach suggests that instead of running the optimization procedure for every "small" subset of gestures, run it only once for the master set and then derive the candidate subset of gestures from this single run. Gesture classification was achieved through a fuzzy clustering algorithm, where each cluster signifies a gesture class. Assume that for a data set of m gestures, the supervised FCM was optimized to achieve optimal recognition accuracy. When the supervised FCM procedure was run again for a subset of n<m gestures, it was discovered that centroids of each of the n clusters were very close to the centroids found the first time the supervised FCM procedure was used. Therefore, to find the accuracy of subsets of gestures of size n from the master set of gestures of size m, an approximation retains all the n centroids for the selected subsets of gestures of size n and removes the remaining m gestures. The main advantage of this approach is that the supervised FCM optimization procedure was run only once for the master set, and the recognition accuracy for any smaller gesture set was deduced from the original partition. The recognition accuracy of any subset of gestures smaller than the master set can be obtained using the confusion matrix created from the original partition $\mathcal{C}_m$. The confusion matrix for the smaller subset of gestures $\mathcal{C}_n$ will include only the rows and columns for the gestures in G_n.

7.3.2 Training the FCM classifier thru parameter search

The gestures used to train the FCM classifier with the parameter neighborhood search were those in the master set for the robotic arm and the VMR tasks. Each master set was used to train a different, independent system. Both systems were trained by eight participants. For each gesture in the master set, five images were acquired from each participant, and therefore, 40 samples per gesture. The first system was used for the VMR task, with n=22 gestures, and therefore a total of 880 samples were used to train it. In the robotic arm system, m=23 gestures, and therefore, 920 samples were used.

The supervised FCM optimization procedure was applied first on the independent system for the VMR master set. To find a good initial solution of the parameter vector for the optimization

[vii] To quantify this assessment, to find the optimal set of parameters for the independent FCM with a master set of 23 gestures was necessary 235 iterations. Each iteration takes 140 sec on average for a Pentium IV, 2.4Mhz with 1 Gb of memory. For a tree with 100 nodes, 913 hours are necessary for the whole computation.

of the supervised FCM, nine solutions were generated using the five heuristic rules explained in Wachs *et al.* [2005]. In Table J.1 the nine starting solutions are presented.

Using initial cluster-value limits of 15 and 25, the nine starting solutions in Table J.1 were used to start the NS algorithm for the supervised FCM. Using each starting solution, the corresponding final solutions were compared. The initial solution that yielded the best accuracy was number 5. The sequence of solutions starting from initial solution 5 is shown in Table J.2 and Figure 7.1.

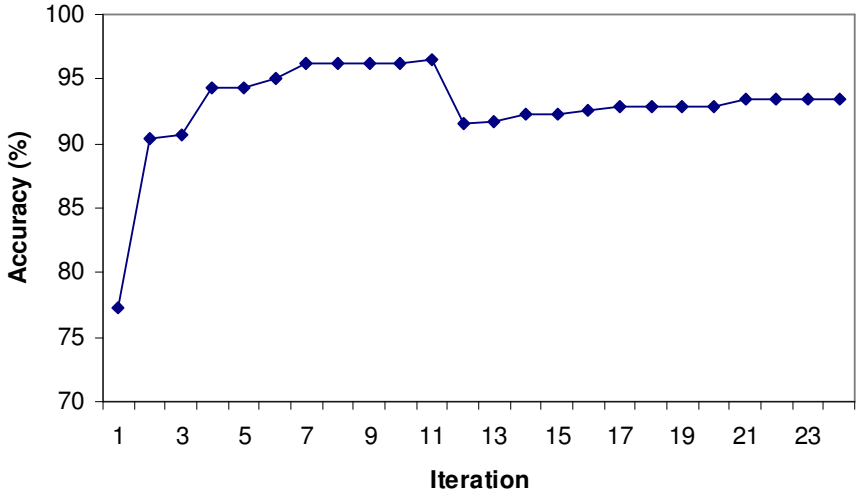


Figure 7.1. Recognition accuracy versus iterations for solution 5 – VMR gesture set

From this run, the parameter changed throughout the convergence profile. To speed up the convergence process, in the beginning only 30 samples per user for each gesture were used. Once it converged in iteration 11, another 10 instances per user were added to each sample set for each gesture. This caused a decrease in the accuracy, since the total instances grew from 690 to 880. However it took another 13 iterations to achieve the near optimal accuracy.

To find the near optimal parameter vector for the robotic arm master set of gestures, the near optimal parameter vector found for the VMR gesture set was used as a good initial solution. Here the assumption was that this initial guess was probably better than any of the nine solutions obtained using the five heuristic rules. The reason is that the VMR master set had four gestures not existing in the robotic arm master set and the robotic arm master set had five gestures that were not in the VMR master set. Therefore, most of the centroids (representing the gestures) remained equal, five new centroids were added, and four were discarded from the partition in the VMR example. The parameter optimal solution for the robotic arm case appears in Table J.3 and Figure 7.2 shows that convergence was reached in only three iterations. The final confusion matrices for the optimal parameter vector obtained for the VMR and robotic arm cases are presented in Tables J.4 and J.5.

The confusion matrices reveal the possible reasons for the recognition accuracy lower than 100%. For the VMR and robotic arm cases, the most significant confusion happened between gestures 1 and 3. Only 42.5% of the instances of gesture 1 were satisfactorily recognized for the VMR case. In the robotic arm case, 72.5% instances of gesture 3 were recognized. Visual inspection of both gestures shows a high similarity between them, i.e., both are very close to a fist in form, and therefore, the block features used might not be robust enough to discriminate between these gestures (Figure 7.3).

The best recognition accuracy 93.41% for the VMR task was obtained after 24 iterations (Table J.2). For the robotic arm task, the best recognition accuracy of 93.91% was obtained only after 4 iterations, using the optimal solution for the robotic arm task as the initial solution for the run (Table J.3).

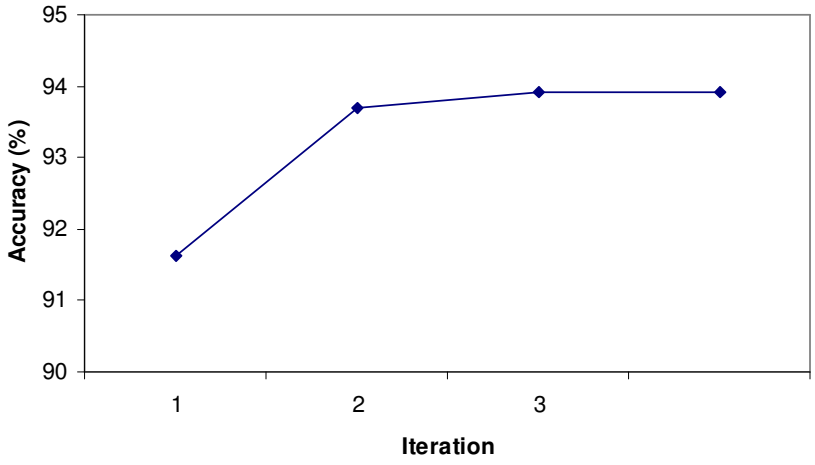


Figure 7.2. Recognition accuracy versus iterations - robotic arm gesture set

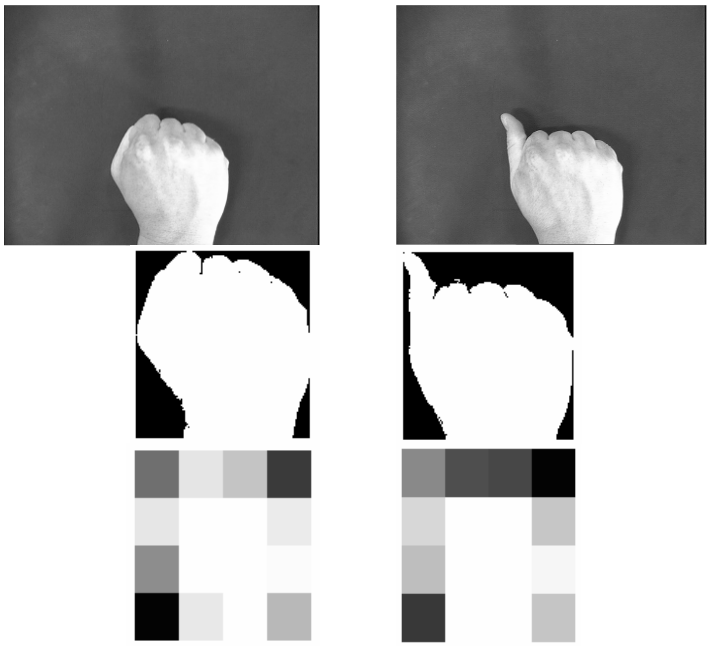


Figure 7.3. Gesture 1 and 3 highly confused, in the VMR case

7.4 Solution by two stage decomposition method – (modules 2 and 3)

The two stage decomposition approach suggests relaxing the multiobjective problem to two sub problems. The first stage finds a feasible subset of gestures from the master set given some recognition accuracy threshold, A_{min} (module 2). The second stage uses the human factors matrices values for the subset of gestures found in the first stage. The solution of the second stage

is a set of GVs, each obtained by finding the best match between n commands and gestures so the sum of the total intuitiveness and comfort are maximized (module 3). The matching solution also depends on the weights assigned to each of the intuitiveness and comfort components, as expressed by (3.4). Section 4.3 discussed two different methods for the subset selection. The first is the Disruptive Confusion Matrix (DCM) and the second is the Confusion Matrix Derived Solution (CMD). The CMD method was used here because of the large size of the master set of gestures G_m (>20), and hence, the computational time to compute the recognition accuracy for each was untenable. The CMD method is an approximation method for determining subsets of gestures and their associated accuracies. It requires using the supervised FCM optimization procedure only once and values from the confusion matrix to approximate the recognition accuracy of the subset. The feasible solutions of gesture subsets were obtained using the confusion matrix derived solution method algorithm (CMD), as described in Section 4.5

The CMD algorithm was used to generate five solutions each for the robotic arm and the VMR cases. The value of the minimum accuracy accepted was A_{min}=100% and A_{min}=98.33% for the VMR and robotic arm studies, respectively (Tables 7.1 and 7.2).

Once the subset of gestures that meet the constraint of the minimal recognition accuracy were found, it was possible to proceed to Stage 2. This stage matched the commands to gestures in such a way that the psycho-physiological measures were maximized by solving the binary integer quadratic assignment problem QAP(G_n). The intuitiveness and the comfort measures were scaled by weights that reflect the importance of each factor on the solution. A set of candidate solutions associated with each subset G_n, selected in Module 2, was determined. These were obtained by changing each weight w_1,w_2 from 0 to 10 in steps of 1, such that w_1+w_2=10. The generated solutions in general reflect the gradual effect of intuitiveness over comfort, but several reflect the opposite. The enhanced simulated annealing algorithm was used to solve the quadratic assignment problem QAP(G_n).

Table 7.1. The subset of gestures for the VMR case

i	Gn	Acc(%)
1	6 7 8 10 12 16 18 21	100
2	6 7 8 10 12 16 18 25	100
3	6 7 8 10 12 16 18 26	100
4	6 7 8 10 12 16 18 27	100
5	6 7 8 10 12 16 21 25	100

Table 7.2. The subset of gestures for the robotic arm case

id	Gn	Acc(%)
1	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	98.5
2	4 5 6 7 8 10 11 14 16 17 19 20 24 26 27	98.5
3	5 6 7 8 10 11 13 14 16 17 19 20 24 26 27	98.33
4	4 5 6 7 8 10 11 13 14 16 19 20 24 26 27	98.33
5	4 5 6 7 8 10 11 13 14 16 17 19 24 26 27	98.33

For each pair of weights (w_1,w_2) and a subset of gestures G_n, a solution was obtained in terms of a gesture-command matching and values of Z_2 and Z_3. Since there were 5 different subsets of gestures, and 11 combinations of weights, a total of 55 solutions were obtained for the VMR. For the robotic arm case, 5 subsets of gestures were used, and hence a total of 55 solutions were obtained. The plots in Figures 7.4 and 7.5 show the intuitiveness versus the comfort for each

solution G_n. The views from the plots are orthogonal to the recognition accuracy. From this set of solutions, it is possible to find the Pareto set of GVs.

Tables B.3 and B.4 show the Pareto points (non dominated solutions) generated using the different weights for the VMR and robotic arm, respectively. The number of Pareto points were 11 and 13 shown in Tables B.3 and B.4 for the VMR and robotic arm, respectively. The first column shows the solution number from which the curve was generated, the second shows the point number from 155 points, and the third shows the subset of gestures that used G_n. The fourth column shows the solution, where the place of each gesture index means the command that was matched with that gesture. The next three columns are for the intuitiveness, comfort, and accuracy indices. The last two columns are the weights for the intuitiveness and comfort, respectively. Figures 7.6 and 7.7 show the GV solutions and the Pareto points, for the VMR and robotic arm study respectively, plotted in the 3D coordinate system of the three multiobjectives. Each point represents a solution in terms of intuitiveness, comfort, and accuracy values.

7.5 Solution by multiobjective method

Due to the large computer run times this method was used in the VMR study only[viii]. The two stage decomposition method does not assure finding the best GV, since only a promising subset of solutions was investigated. Those solutions were the subset of gestures G_n, with high accuracy (over the A_{min} specified minimal accuracy estimated acceptable by the user), from the reduced master set G_m. However, it was not possible to say whether there were other subsets of gestures in the solution space that will yield Pareto points. The only way to find all the Pareto solutions GV is through a complete exhaustive search of the solution space or the use of heuristic multiobjectives like the evolutionary (GA) multiobjective method [Deb *et al.,* 2000]. Given a gesture set of size m and a command set of size n, there are m!/((m-n)!n!) different possible subsets. Each subset of gestures can be matched with commands in n! different ways, hence the total number of subsets is m!/(m-n)!. Hence, the search space was $1.2*10^{10}$ and $6.4*10^{17}$ for the VMR (n=8 and m=22) and the robotic arm (n=15 and m=23), respectively.

Alternatively, a limited search around an initially high recognition accuracy solution will reduce the solution space (a single initially high recognition accuracy associated solution was obtained using the CMD method with A_{min}=100%). For each of the 600 sets of gestures G_n, a set of associated GV solutions was generated by changing each weight w_1,w_2 associated to Z_1 and Z_2 from 0 to 10, in steps of 1, such that $w_1+w_2=10$ and solving the integer QAP problem (see the P 4.4). The set of basic solutions (before the extension due to the changes in the weights) was obtained using the following pseudo code based on a limited complete enumeration. The parameters used were g^1_L=6 and g^1_H=20 and N=600. The lower and upper bounds were selected such that in the first iteration, the first solution inspected was GV_1={6,7,8,10,12,13,16,17}. This solution was obtained with the CMD method, and the associated recognition accuracy was 100%. A search starting from such high recognition accuracy assures solution with correspondingly high recognition accuracy.

The gestures g_4, g_6, g_9 and g_{10} were not included in GV_j . Some gestures were missing in the gesture master set for the VMR (the gestures are 9,11,14,15 and 19), since they are in the robotic arm master set. The set of non dominated solutions (Pareto front) can be determined from this limited search (Table B.5). For the VMR study 6600 solutions were generated, including 98 Pareto solutions. This process lasted for 48 hours using a Pentium IV 2.4Mhz with 1Gb memory. Each GV was represented as a point in a 3D space whose coordinates were intuitiveness, comfort, and accuracy (Figure 7.8).

[viii] The partial enumeration took 8 hours and 30 minutes using an Intel IV, 2.4 Mhz and 1Mb memory.

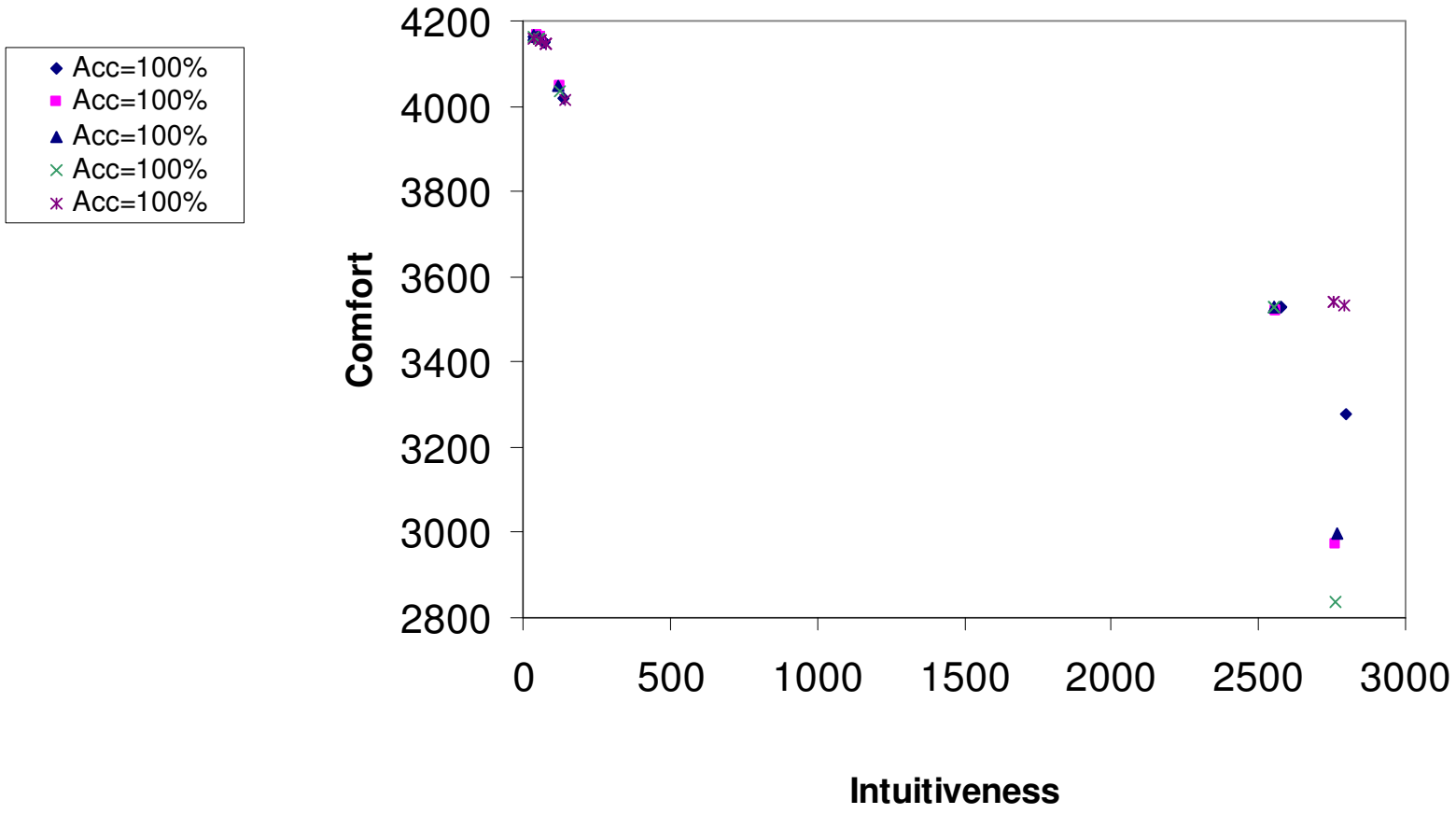


Figure 7.4. Intuitiveness vs. comfort plots for 5 gesture subsets for the VMR study

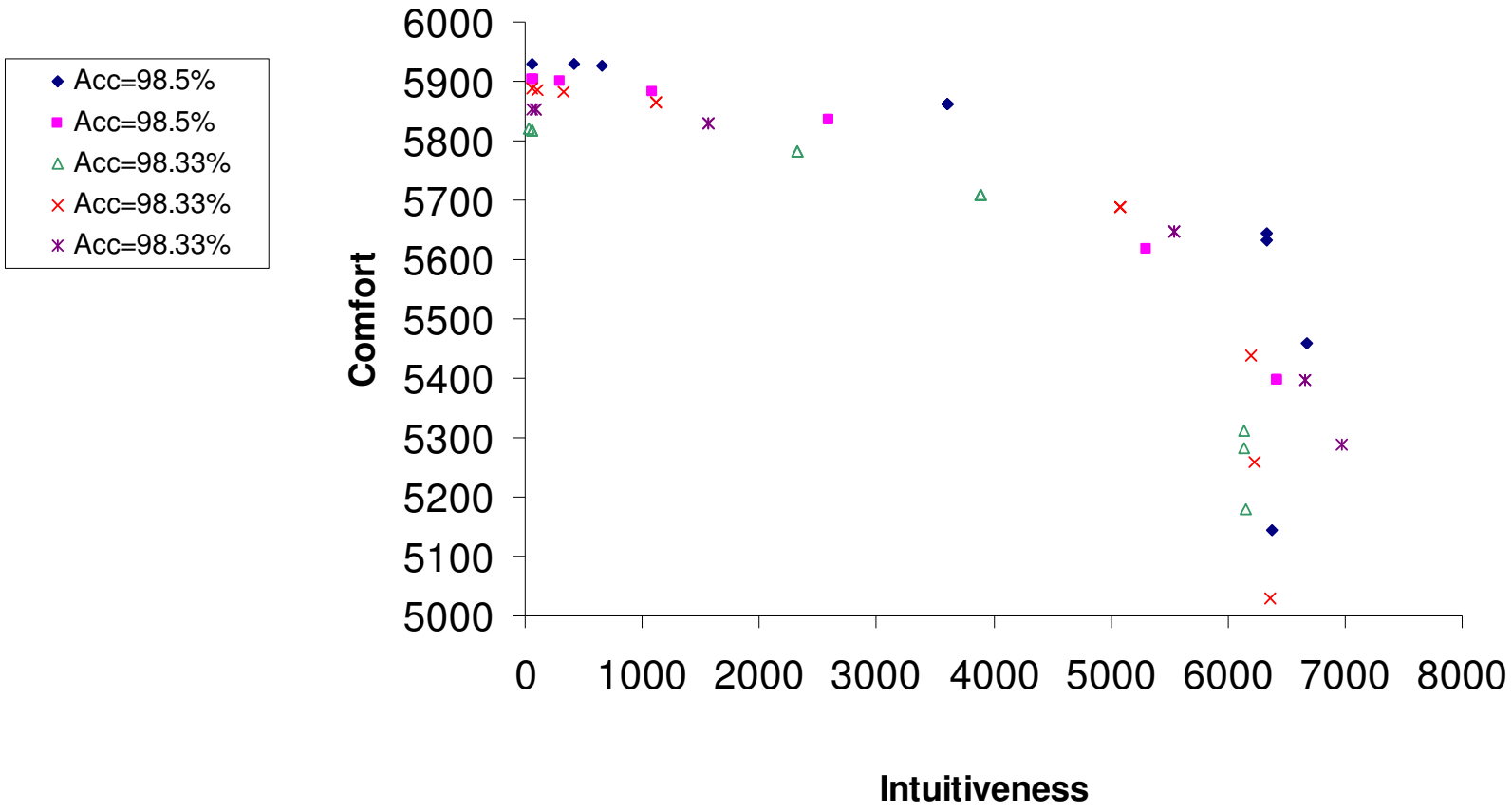


Figure 7.5. Intuitiveness vs. comfort plots for 5 gestures subset for the robotic arm study

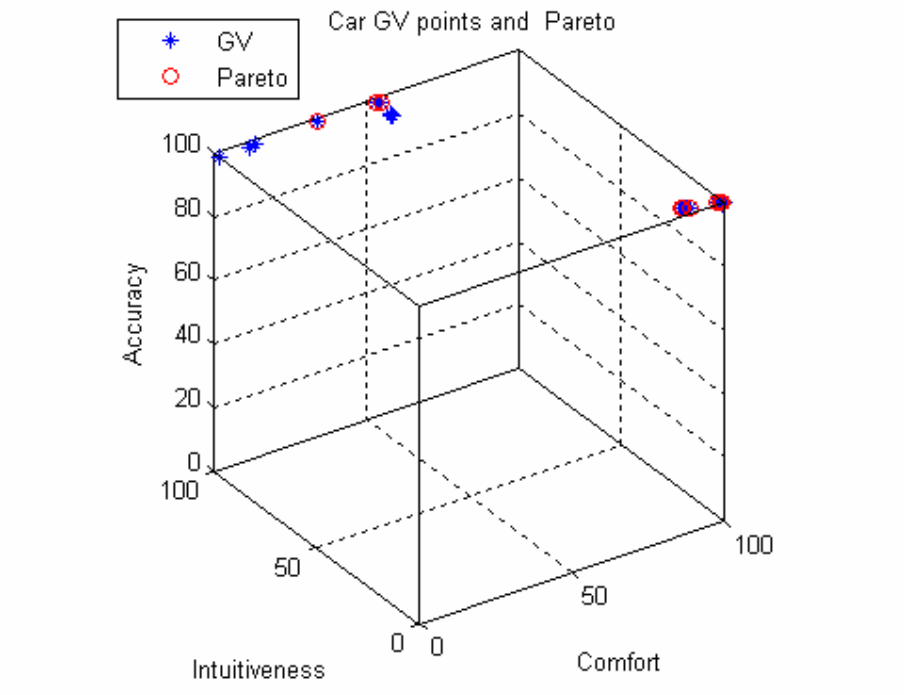


Figure 7.6. 3D plot for the solutions generated with 5 GV for the VMR study

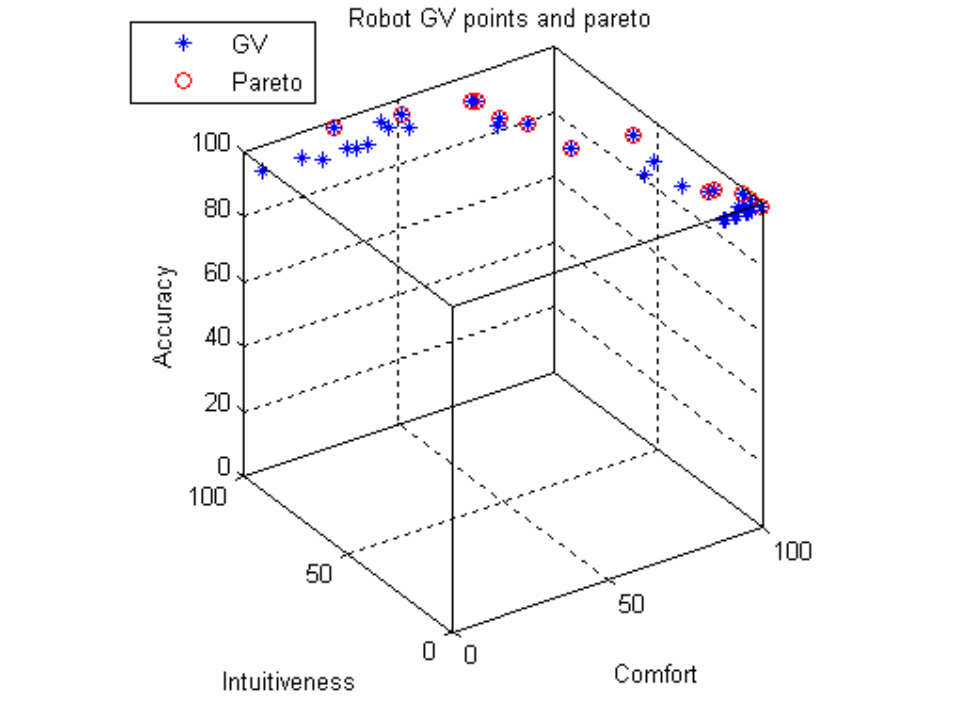


Figure 7.7. 3D plot for the solutions generated with 5 GV for the robotic arm study

The semi complete enumeration algorithm(N, g^1_L, g^1_H)

For $g_1=g^1_L$ to g^1_H,
  For $g_{1+1}=(g_1+1)$ to (g_H+1)
    ……
      For $g_{1+k}=(g_1+k)$ to (g_H+k)
……..
        For $g_{1+h}=(g_1+h)$ to (g_H+h)
          $GV_j=\{ g_1,.., g_{1+k},.., g_{1+h} \}$
          If (j>N) exit
        End
    End
End

Algorithm 7.1 The semi complete enumeration algorithm

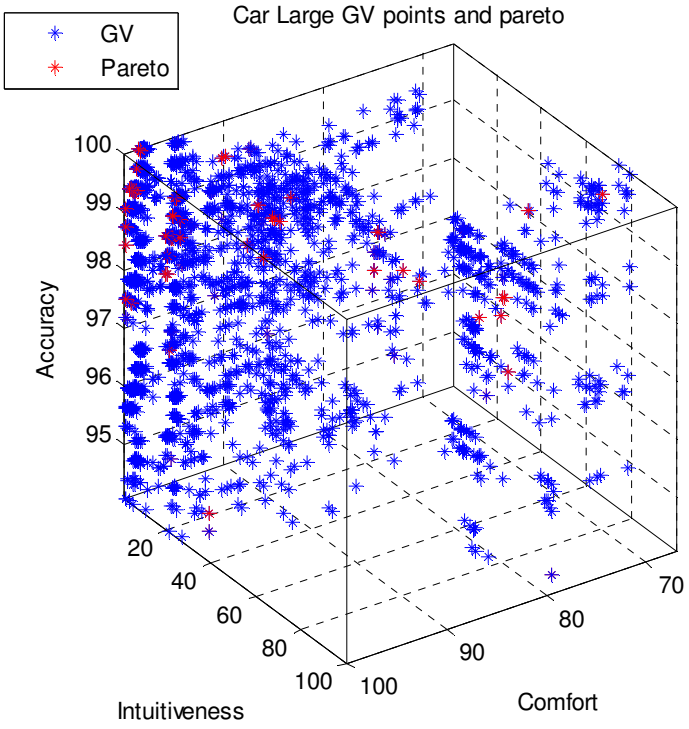


Figure 7.8. 3D plot of the GV solutions obtained using a semi-complete search for the VMR study

7.6 Discussion

Two case studies were conducted in which the two stage decomposition approach and the multiobjective solution methods were demonstrated. The two stage decomposition procedure included two different metaheuristic approaches to obtain G_n from G_m. Those were the Disruptive Confusion Matrix Method (DCM) and the Confusion Matrix Derived Solution Method (CMD). The CMD method was used in this chapter because its main advantages are the short running times required for the approximated accuracy calculations. The supervised FCM optimization method was only run once for the whole reduced master set G_m. The main disadvantage of the CMD method is that the solutions are based on a larger (m classes) clustering problem, instead of being obtained from a smaller problem (n classes). In such a case, it is likely that the recognition

accuracy will be higher for fewer classes than for a problem with more classes, when using the supervised FCM optimization method. Moreover, using the DCM method, the FCM algorithm is required for every subset of solutions G_n, and optimized according to this subset. This is in contrast to the CMD method, in which the FCM is called only once, and the optimization is based on the general master set of solutions G_m.

The CMD was used to obtain five initial gesture subsets G_n with A_{min}=100% and A_{min}=98.33% for the VMR and for the robotic arm, respectively. For each G_n a set of associated GVs and their objective values were obtained. For the VMR study, eleven Pareto points belonged to only curves generated from solutions 1,3, and 5. For the robotic arm study thirteen Pareto points were obtained using all the 5 solutions; however, more than 50% of the points in the Pareto set were generated using the first solution. These solutions were shown in a 3D plot, with axes of intuitiveness, comfort, and accuracy. For the multiobjective decision approach, a reduced complete search was adopted. Instead of inspecting $1.2*10^{10}$ and $6.4*10^{17}$ solutions for the VMR (n=8 and m=22) and the robotic arm (n=15 and m=23), respectively, an approximation method was employed for the VMR task. A total of 6600 solutions were generated. Using these solutions, 98 Pareto solutions were obtained. These solutions can be offered to the decision maker to select the GV according to his/her own preferences. The decision maker may wish to prioritize the objectives such that the accuracy Z_3 is 1st priority, comfort Z_2 is 2nd priority and intuitiveness Z_1 is 3nd priority. Using this criteria, the following solution was obtained: $GV_{(i=4744)}$={21,16,6,18,7,8,25,10}(row 42 in Table B.5). The associated indices to this solution are Z_1=72, Z_2=4167 and Z_3=100%. If, alternatively, the decision maker is willing to accept a lower comfort in turn for higher intuitiveness, he may pick the GV with intuitiveness of 2907 which has a comfort of 2992, without affecting the recognition accuracy, $GV_{(i=1804)}$={8,6,26,27,12,10,18,7}(row 6 in Table B.5). Images of the solutions $GV_{(i=4744)}$ and $GV_{(i=1804)}$ are presented in Figure 7.9(a) and (b), respectively.

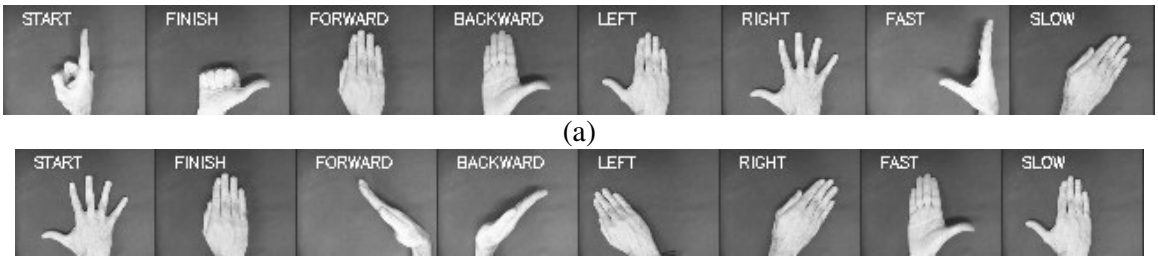


Figure 7.9. Two different GV selected by the decision maker. (a) First priority is accuracy. (b) First priority is intuitiveness

Examining the solutions obtained, clearly the solution in Figure 7.9(a) was less intuitive compared to the solution in Figure 7.9(b). For example, note the lack of complementary intuitiveness in gesture-command pairings in Figure 7.9(a) and their presence in Figure 7.9(b). However, the comfort decreased significantly in Figure 7.9(b). Slanted gestures cause ulnar deviation, extension and flexion at the wrist, and therefore, are harder to perform [Griffins, 2001].

8 Conclusions and future work

8.1 Conclusions

This thesis presented an optimal hand GV design methodology that considers both human factors and technical aspects (the recognition accuracy). The first aspect includes the intuitiveness and comfort attributes of GVs, while the last is related to the development of a hand recognition algorithm. The most salient advantage of the approach presented is a structured formulation of the GV design problem in a rigorous manner so human psycho-physiological and technical aspects are combined in a unified approach.

The methodology developed consists of an analytical formulation of the GV design problem, a reconfigurable hand gesture recognition algorithm, two quantitative solution methods for solving the GV design problem, and methods for quantifying and automating the collection of intuitiveness and comfort gestural indices.

This research suggests an analytical approach that includes quantitative methods to measure and to compare the different aspects of a GV by mapping psycho-physiological indices in one objective function. Previous works dealt with selection of hand gesture vocabularies using rule based [Baudel and Beaudouin-Lafon, 1993] and ad-hoc methods [Kjeldsen and Hartman, 2001]. A computational approach allows one to consider a multitude of performance measures in an objective function, and hence to establish a quantitative method to measure the naturalness of a GV and to compare the performances of different GVs. The contributions of this disseration correspond to previous research conclusions as stated in the following. Munk suggested [2001] highly ergonomic vocabularies; however, he was not able to perform a comparison among them. He suggested that a future implementation of his methodology should provide a benchmark for the exploration of different gestures from two standpoints: computer recognizability and subjective naturalness of those gestures experienced by the user. Nielsen [Nielsen *et al.,* 2003] recommended as future work to extend his benchmark procedure to include technical aspects (such as automatic gesture recognition based on machine vision).

The analytical formulation presented in this dissertation considered both the ergonomic and technical factors in contrast to Wagner *et al.* [2003], where only the ergonomic factor was considered in a mathematical function.

The unified methodology presented in this thesis is a clear demonstration of the future need, defined by Pavlovic *et al.* [1997], that "*substantial research effort that connects advances in computer vision with the basic study of human-computer interaction will be needed in the future to develop an effective and natural hand gesture interface*." The methodology presented here is based only on vision, and hence, no devices were required to be attached to the hand (unencumbered interface). Guidelines for defining gestural command sets from an ergonomics stand point were presented by Baudel and Beaudouin-Lafon V1993] who expressed the need for a similar procedure for unencumbered interfaces.

While this thesis is a breakthrough in the hard problem of GV design, several limiting assumptions were made regarding the hand GV design problem. To reduce the complexity of the GV design problem several assumptions were made: a) Each gesture in the GV is associated with one command, and each command is associated to only one gesture. b) Gestures are static poses. In real life, gestures are dynamic, and their trajectories and configurations over time usually express additional information. Future recommendations address this assumption. c) The strain of maintaining a gesture increases linearly with the duration of the pose and with the frequency of use. Further experimentation is required to model the increase of effort with time; in fact, the effort may increase quadratically with time. d) Task completion time was considered the only

performance index of a GV. Other indices can be used jointly to assess the performance of the GV, such us number of errors while performing the task. e) Intuitiveness and comfort were considered the only human factors in the analytical formulation. Other human factors such us learnability, memorability, efficiency, and mental load should be included in the model even if there exists a correlation between them.

Recognition accuracy, intuitiveness, and comfort were used in this thesis as proxy performance measures for task completion time. The reason for this is twofold: (a) measuring the task completion time empirically for a large set of vocabularies is untenable, and (b) these proxy measures are easier to quantify than the determination of an analytical function for task completion time. One of the problems faced in this work was identifying the appropriate experimental methods to obtain reliable psycho-physiological measures. This aspect was not addressed in the past by other researchers; for the technical factor, many hand gesture recognition algorithms are available.

In this work, experimental methods to automate the collection of intuitiveness and comfort indices were developed. For the collection of intuitiveness measures, an automated tool was developed that simulated different scenarios, and through user interactions, it collected data on the cognitive associations between commands and gestures. Even though the functions (commands), the context, and the available master set of gestures are defined in advance, the collection of user responses and the computation of the intuitive index are fully automatic. Previous research [Nielsen *et al.,* 2003; Preston *et al.,* 2005; Höysniemi *et al.,* 2005] used Wizard-of-Oz techniques to collect data regarding cognitive associations between command-gestures pairs. The Wizard-of-Oz experiment had people respond to commands stimulated under camera surveillance. However, this video extraction method is rather time consuming, and the scenarios must be carefully written in order to stimulate the user to evoke the gestures in the scope of the task, as expressed by Nielsen. As for the stress index measures, in this thesis, a subjective evaluation tool was used to obtain the static and dynamic stress of performing gestures. This tool associated static and dynamic stress indices for each gesture and for inter-gesture transitions and stored automatically all data. Previous works collected stress measures through experiments that vary from subjective questioners [Nielsen *et al.*, 2003] to electronic devices, such as EMG, to measure muscle activity [Wheeler, 2003].

Once the collection of intuitiveness and stress indices were obtained, it was possible to answer the following questions presented by Wolf and Rhyne [1987]: a) how consistent are people in their use of gestures, b) what are the most common gestures used in a given domain, and how easily are they recalled, c) do gestures contain identifiable spatial components that correspond to the functional components of command (the action to be performed), scope (the object to which the command is applied), and target (the location where the object is moved, inserted, copied, etc.). Analysis of the experimental results provides some answers to the questions asked by Wolf and Rhyne.

When examining a pair of complementary commands it was found that the response was often a pair of complementary gestures. Complementary gestures possess the property of "mirrored gestures" or "present-absent", such as when flipping the palm of the hand, closing/opening fingers, spreading or keeping the fingers together, etc. This is evidence that there is a type of intuitiveness related to pairing complementary commands to complementary gestures. This type of intuitiveness is called “complementary intuitiveness,” while the intuitiveness of a single command-gesture matching is called the “direct intuitiveness.” Therefore, a finding with respect to the third question presented by Wolf and Rhyne [1987] is that there are spatial components that the users identify in gestures, and moreover they are used as “complementary” gestures to match complementary commands.

With regards to the second question by Wolf and Rhyne [1987] (not including the recall factor), the results in this thesis indicate that the selection of gestures followed a 70/30 rule (similar to the 80/20 rule of inventory theory), where 70% of subjects' gesture responses were composed from only 30% of all the gestures used by the respondents. Even though agreements for a gesture-command association ranged from 59%-100% for the VMR and robotic arm tasks, respectively, the overall agreement was only 34% and 18% for the VMR and robotic arm tasks. This refutes the claim that subjects consistently use the same gestures to represent the same commands while performing tasks, as suggested by Hauptmann [Hauptmann and McAvinney, 1993; Hauptmann, 1989].

A question not addressed by Wolf and Rhyne [1987] is related to selection of the most comfortable gestures for which the effects of strain and fatigue on the muscles are minimized while performing the task. Two types of stress were defined: a) static stress, which is the effort that takes to hold a static gesture for a defined amount of time, and b) dynamic stress, which is the effort that is necessary for performing a transition between static gestures. It was found that 90% of the dynamic stress (and its duration) was determined by the final posture in the transition between two postures, and only 10% by the starting posture. This relation allows us to predict the dynamic stress and its duration based on the use of only static stress measures.

The tools used in the methodology can be used to design high recognition, easy to learn and remember, hand GVs, answering the need expressed by Long *et al.* *that it is important that designers will not only be "able to design gesture sets that are easy for the computer to recognize, but also for humans to learn and remember.*" They also concluded "*To perform this difficult task, designers will require significantly better gesture design tools than are currently available*"[ix]. Although this methodology does require effort to obtain human ergonomic and cognitive indices, it provides a rigorous structure for replacement and expansion. More accurate fatigue or intuitiveness indices can easily replace old data by updating the gesture knowledge database. This effort will not be lost as it can provide a database for subsequent methodologies.

Another problem addressed in this thesis is the issue of reconfigurability and calibration of the recognition system. The primary need for recalibrations of a hand gesture recognition system is its frequent relocation to other environments such as laboratories and remote control stations. A secondary need for recalibration occurs due to demands for custom redesign of the gesture control language. This occurs for new users, new control tasks, and new vocabularies. A fast recalibration of system parameters provides the system flexibility to respond to such new system setups. To address this issue of reconfigurability or flexibility, a stand alone methodology was developed for simultaneous calibration of the parameters of an Image Processing-Fuzzy C Means (FCM) hand gesture recognition system. Local neighborhood search was used to automate the calibration of the parameters of the system. Thus, the design of a hand gesture recognition system is recast as an optimization problem and the proposed solutions were compared using a reduced master set of gestures. Kray and Strohbach [2004] provided an application with the ability to create and dynamically reconfigure a vision-based user interface that recognizes basic interaction gestures. Their configuration used a weight table (a standard table enhanced with load sensors), in contrast to my system that does not require any additional hardware. Kjeldsen *et al.* [2003] presented an interface that can be dynamically reconfigured, changing both form and location on the fly. A device that combines a steerable projector/camera system, dynamic correction for oblique distortion, is required to use this interface.

[ix] Results of the learning and memorability tests appear later in this chapter

The following hypotheses were validated: task performance time τ can be represented by multiobjective proxy measures, and the maximization of the multiobjective function causes a minimization in the performance time of the tasks. Both hypotheses were validated through an additional hypothesis which claimed that vocabularies from the V_G set result in shorter task completion time than the ones from the V_B set. Mean completion time for the robotic arm task using V_G was much shorter than the time using V_B at the 0.5% level of significance (p=0.0059). For the VMR task, task completion time using V_G also was shorter than using V_B at a 0.03% level of significance (p=0.00031). Regarding learning rates, it was found that for the V_G the learning rate was lower than for V_B. The last hypothesis suggested that the GV_G is easier to remember than GV_B. Memorability was determined by experienced user's recall of the gesture-command associations during the task performance trials. The average memorability scores for the robotic task were found to be higher for the V_G than for the V_B at the 5% level (p=0.05). For the VMR task, the difference was not significant at the 5% level (p=0.58). Summarizing the results, the use of the V_G compared to the V_B vocabulary samples resulted in shorter task completion time, higher learning rates and higher memorability. Therefore, GVs with higher values of the 3 objectives, result in decreased performance time, faster learning and increased memory.

To summarize, a methodology for the design of natural hand gesture vocabularies, which considered both the psycho-physiological and the technical aspects in a unified approach was presented. This provides several advantages. First, it makes it possible to obtain highly ergonomic and recognizable hand GVs using a rigorous procedure. Second, it offers a data repository of intuitiveness and comfort measures, and an automated methodology for their collection. This approach results in improved task oriented hand GVs. The developed framework is an important contribution to the development of hand gesture recognition systems for human-robot interaction.

Regarding the applicability of this dissertation to real world problems, it is necessary to consider that the number of commands to control a robotic arm from the teach-pendant is more than 20 (depending on the DOF of the robot, and additional factors). The main limiting factor is that the user has difficulty remembering more than 12 gestures. This constraint impacts mostly the chances of a hand gesture driven robot to be applied to industrial tasks although will repeated use and reminder cues this difficulty may be overcome. Nevertheless, simple common tasks of the "pick and place" type, where the precision is not an important requirement, are appropriate to be performed by a hand gesture recognition driven robot.

8.2 Future work

Future research should address the following issues:

8.2.1 Algorithms

The hand gesture recognition algorithm is an image processing based-Fuzzy C-Means (FCM) algorithm that is capable of classifying static gestures in a uniform background. Future research should implement robust image processing algorithms for the detection and classification of static/dynamic hand gestures in an unconstrained environment [such as detailed in Just *et al.,* 2006; Zhou *et al.,* 2004; Zhenyao and Neumann, 2006]. The hand gesture recognition algorithm developed in this work included a feature that allowed fast recalibration of system parameters providing system flexibility to respond to demands for custom redesign of the GVs, new users, and new control tasks. Future work should investigate the effect of various dynamic strategies for expanding and contracting the neighborhood size.

For the metaheuristic two stage decomposition algorithm, two approaches were presented in this thesis, the DCM and the CMD methods. Future research should investigate extended

comparative testing between the DCM and CMD algorithms, such as complexity, calculation times, simplicity and optimality of their solutions. This includes evaluation times of each algorithm, the dynamic nature of the size of the vocabulary, and the maximum number of solutions required. Both algorithms perform the search by changing a gesture from the solution (adding and discarding a gesture) using some interchange rule. The DMC looks to improve the accuracy at every stage, where the CMD decreases it. Both cases do not consider in the interchange rule the possibility that one of the gestures from a complementary pair will be discarded while the other is part of the solution. Further work should modify the rules so complementary pairs of gestures always remain together or are changed by a new few complementary gestures.

The feasible gesture solutions found using the DCM or the CMD procedures are matched to commands to obtain the final set of GV using the integer quadratic assignment problem (QAP) (P 3.3). In future work the matching problem may be solved using a GA approach [Holland, 1975]. In this case each individual can be encoded as a chromosome of length n which represents the associations between commands $C = \{c_1,\ldots,c_n\}$ and a subset of gestures $G_n = \{g_1,\ldots, g_n\}$ [Wachs *et al.,* 2004]. The simulated annealing algorithm [Connolly, 1990] was implemented in this thesis to solve the QAP. Other techniques, such as the particle swarm approach [Liu *et al.,* 2006] or using grid computing optimization systems [Goux *et al.,* 2000], may be exploited in future work to solve the QAP.

8.2.2 Problem modeling

Different modeling and representations can be used to solve the optimal hand GV problem. For example, in this thesis, one of the main assumptions is that the number of commands and gestures in the GV is the same. Each command is mapped to only one gesture, and each gesture is associated to one command. However, other types of GV can be defined such that, for example two commands are associated to the same gesture to alleviate memory load. When the gesture is made, the right command is called according to the context of the task and the operation mode [Kohler, 1997]. In addition, it is possible to extend the work to consider several gestures as representations of a single command. Thus, for example, a closed fist with the left thumb out, as well as, an open hand with the left thumb out can both be used to represent a "left command".

8.2.3 Performance measures

One of the formulations presented in this thesis consists of mapping the three performance measures into a single measure by using weights w_i to reflect the relative importance of each of the objectives in Eq. (3.3). These weights were varied, and for each unique weighting scheme a corresponding solution was presented to the decision maker for acceptance or rejection. An alternative method to find these weights is through empirical tests. For this, it is necessary to generate vocabularies where each of the objectives is dominating in turn, and then use these vocabularies in experiments, where the task completion time is recorded. A linear/non linear regression can then be performed to obtain these weights.

Three main objectives (accuracy, intuitiveness, comfort) were included as proxies of task performance using a GV. Additional psycho-physiological indices may be included to the methodology presented in this thesis, such as mental load and mental stress, user satisfaction, learnability, memorability and efficiency.

8.2.4 Psycho-physiological methods

The stress measure can be obtained using two approaches: EMG based indices and the use of ergonomic tests. In this thesis, the ergonomics approach was adopted, where the user may rank poses from weak to strong on some scale. Future work may include the use of EMG to record the electrical activity of muscles thereby obtaining the static and dynamic stress measures [Natan *et al.,* 2003]. These can be used to validate the data obtained using the ergonomic test approach and to confirm the prediction model to obtain dynamic stress proposed in this work. Preliminary work in this area [Ronen *et al.,* 2005] indicated many research problems with this approach. The next step may be the development of a bio-mechanical model to determine the hand effort based on its kinematics and kinetics. The hand can be represented by the primitives described in Table 3.1 in Chapter 3. An interesting question to solve could be whether there are functions f_s and f_d such that the static and dynamic stress are described by $S_i=f_s(p_i^1, \ldots, p_i^k,.., p_i^n)$ and the dynamic stress is $S_{ij}= f_d\{(p_i^1, \ldots, p_i^k,.., p_i^n) , (p_j^1, \ldots, p_j^k,.., p_j^n)\}$, respectively.

Results were presented regarding the consistency of a population selecting certain gestures to given commands. An interesting observation is the one regarding the use of complementary gestures to represent complementary commands. In this thesis we mentioned that complementary gestures can be obtained usually by flipping the palm, rotating the hand to opposite sides, or by closing or opening the palm or fingers. The finding in this thesis may suggest that gestures contain identifiable spatial components which correspond to the functional components of command (the action to be performed). Complementary gestures can be obtained by reversing/mirroring or rotating these spatial components. Future research can focus on finding the identifiable spatial components of the gestures, and therefore examining whether people are consistent while associating similar commands to gestures with similar spatial components. This work was done for the handheld stylus writing symbols [Wolf and Rhyne, 1987; Long *et al.,* 1999] but not invested in for the hand gesture domain.

8.2.5 Dynamic hand gestures and other types of gestures

A more flexible methodology should apply the principles presented in this thesis to dynamic hand gestures. In this case, the experimental methods presented here can be applied to obtain the intuitive measure. However, different hand gesture recognition capable of recognizing dynamic gestures will be necessary. To obtain the stress measure, the principles presented in Kölsch [2003] can be used. Starting and ending positions may be identified by the tension required for issuing the gesture, and a relaxed position of the hand will indicate the end of the gesture [Baudel and Beaudouin-Lafon, 1993]. Additionally, the rest position between gestures may be included in the methodology as a 'rest' command.

An interesting extension for this methodology will be to include face gestures for intelligent communication between humans and service robots. In this context, face gestures are used to convey emotions to the robot. When combined with voice and hand gesture control, these emotions serve as a feedback to the learning process of the robot (e.g. a smile encourages certain behaviors).

8.2.6 Expanded experimentation

By posing the optimal GV design problem as a MOP, solutions can be presented as 3D representations, including Pareto optimal ones. Calculating the entire Pareto set for the large problems (the search space is $1.2*10^{10}$ and $6.4*10^{17}$ for the VMR and the robotic arm, respectively) presented in the case study, Chapter 7, was computationally limiting using the

presented procedures. However, future work may overcome the complexity problem by a) running a complete enumeration over the solution space using parallel computing so the complexity of the problem is approachable, b) using the CMD algorithm and allowing a significant amount of feasible solutions (>10,000), and c) calculating the Pareto frontier using an evolutionary multicriteria procedure.

To validate the hypothesis that highly accurate, intuitive, and comfortable vocabularies are available, eight vocabularies of the V_G and V_B set were used. In future work more vocabularies should be compared. Moreover, larger vocabularies (n>20) should be evaluated and additional tasks should be investigated. The current methodology was applied to the control of virtual robots. The virtual models were designed to simulate as accurately as possible the functionality of the real robots. For the 5 DOF robotic arm, the close form of inverse kinematics equations were solved to mimic the world coordinate operation. The effect of gravity on the object and collision detection to avoid impacts were considered. The VMR was simulated in a straightforward fashion where only collision detection rules were included. These assumptions were helpful to carry out the experiments without facing electro-mechanical problems and movement delays, which occur frequently when dealing with real robots.

The implementation of real robots in this framework will be a natural extension to this work. In this case, the main factor that should be considered is how to eliminate the delay between the gesture recognition (the command) and the robot's actual performance of the action. The value of the delay is not fixed and depends on factors such as communication protocols between the robot and the PC, the controller's stack memory, type of interpreter, and others. Therefore, it is very difficult to predict the specific time "contribution" to task performance time, and hence, to eliminate it from the total time τ. Experiments with real robots were not possible in the framework of this thesis due to time constraints.

Further validation experiments should include an increased number of users to cover cultural diversity and allow a better generalization.

8.2.7 Applications

This work focused on the use of hand gestures for human-robot interfaces. Two kinds of virtual robots were used as case studies: a robotic arm and a virtual mobile vehicle. The immediate application for this methodology will be to test the optimal hand gesture methodology on these types of robots. For each robot the commands can be obtained from the selection in the teach pendant of the robot. They can be further reduced to the commands that are most frequently used. Furthermore, the methodology presented can be applied to novel types of interfaces, such as tabletop devices, where hand gestures are the main form of control. This kind of novel interface may require a larger set of gestures than those presented in this work. An obstacle that we faced is that the complexity of the problem grows exponentially with the number of gestures. We believe that as the computers become faster, and have higher computational power, a larger set of hand gesture vocabularies can be used.

Long-term applications may include the development of prosthetic devices for amputees and paralyzed individuals. Control of such devices is possible through the real-time classification of (EMG) signals recorded from intact muscles. In this case, the master set of gestures is replaced by a master set of forearm signals that the amputee can generate. These signals are similar to those obtained from healthy subjects to perform the gestures.

9 References

[1] Abe K., Saito H., and Ozawa S. 2002. Virtual 3-D interface system via hand motion recognition from two cameras, IEEE Transactions on Systems, Man and Cybernetics, Part A, 32(4): 536-540.

[2] Aboudan R. and Beattie G. 1996. Cross-cultural similarities in gestures. The deep relationship between gestures and speech which transcends language barriers, Semiotica, 111(3-4): 269-294

[3] Abowd, G.D. and Mynatt, E.D. 2000. Charting past, present, and future research in ubiquitous computing, ACM ToCHI, 7(1): 29-58.

[4] Acredolo L. and Goodwyn S. 1996. Baby signs: How to talk with your baby before your baby can talk, Contemporary Books, Chicago, IL.

[5] Agrawal T. and Chaudhuri S. 2003. Gesture recognition using position and appearance features, Proceedings of IEEE International Conference on Image Processing, Barcelona, Spain, September 14-17, 2:109-112.

[6] Alon J., Athitsos V., Yuan Quan, and Sclaroff S. 2005. Simultaneous localization and recognition of dynamic hand gestures, Proceedings of IEEE Workshop on Motion and Video Computing: 254-260

[7] An K. N., Chao E. Y. Cooney W. P.and Linsheid R. L. 1979. Normative model of human hand for biomechanical analysis, Journal of Biomechanics, 12: 775-788.

[8] Archer D. 1997. Unspoken diversity: cultural differences in gestures, Special Issue on Visual Sociology, Qualitative Sociology, 20(1): 3-137.

[9] Argyros A.A., Lourakis M.I.A. 2006. Vision-based Interpretation of hand gestures for remote control of a computer mouse, Proceedings of the HCI'06 workshop (in conjunction with ECCV'06), LNCS 3979, Springer Verlag, Graz, Austria, May 13: 40-51.

[10] Asher, H. 1956. Cost quantity relationships in the airframe industry, Rand Corporation, Report 291, Santa Monica, CA.

[11] Balaguer F. and Mangili A. 1991. Virtual environments, D. Thalmann N. Magnenat-Thalmann, editor, Wiley, New Trends in Animation and Visualization, (6): 91-106.

[12] Baudel T., Beaudouin-Lafon M, Braffort A. and Teil D. 1992. An interaction model designed for hand gesture input, Technical Report No. 772, LRI, Université de Paris-Sud, France.

[13] Baudel T., and Beaudouin-Lafon M. 1993. Charade: remote control of objects using freehand gestures, Communications of the ACM, 36(7): 28-35.

[14] Becker M., Kefalea E., Mael E., Von der Malsburg C., Pagel M., Triesch J., Vorbruggen J. C., Wurtz R. P. and Zadel S. 1998. GripSee: a gesture-controlled robot for object perception and manipulation, Autonomous Robots, 6(2): 203-221.

[15] Bezdek J. C. 1973. Cluster Validity with fuzzy sets, Cybernetics, 3 (3): 58-73.

[16] Birdwhistell R. L. 1970. Kinesics and Context; essays on body motion communication, University of Pennsylvania Press, Philadelphia.

[17] Boston Consulting Group. 1970. Perspectives on experience, Boston, MA.

[18] Borg G. 1982. Psycho-physical bases of perceived exertion, Medicine & Science in Sports & Exercise, 14(5): 377-381.

[19] Bradski, G., Yeo B.L. and Yeung M. 1999. Gesture for video content navigation, Proceedings of International Society for Optical Engineering SPIE'99 3656: 230- 242.

[20] Brook N., Mizrahi J., Shoham M. and Dayan J. 1995. A biomechanical model of index finger dynamics, Medical Engineering & Physics, 17(1): 54-63.

[21] Burschka D., Ye G., Corso J. J., and Hager G. D. 2005. A Practical approach for integrating vision-based methods into interactive 2D/3D applications, Technical report, The Johns Hopkins University, CIRL Lab Technical Report CIRL-TR-05-01, Baltimore, US.

[22] Cabral M. C., Morimoto C. H. and Zuffo M. K. 2005. On the usability of gesture interfaces in virtual reality environments, Proceedings of the 2005 Latin American conference on Human-computer Interaction, Mexico: 100-108.

[23] Campbell L W, Becker D A, Azarbayejani A, Bobick A F, and Pentland A. 1996. Invariant features for 3-D gesture recognition, Proceedings of Automatic Face and Gesture Recognition FG'96: 157-162.

[24] Card S. K., Mackinlay J. D., and Robinson G. G. 1990. The design space of input devices, Proceedings of the CHI '90 Conference on Human Factors in Computing Systems: 117-124.

[25] Cipolla R., Hadfield P. A. and Hollinghurst N. J.1994. Uncalibrated stereo vision with pointing for a man-machine interface, IAPR workshop on machine vision applications MVA'94. Tokyo, December.

[26] Cipolla R. and Hollinghurst N. J. 1996. Human--robot interface by pointing with uncalibrated stereo vision, Image and Vision Computing, 14(2):171-178.

[27] Cohen C. 1999. A brief overview of gesture recognition, [Online]. Available: http://www.dai.ed.ac.uk/CVonline/local_copies/cohen/gesture_overview.html

[28] Connolly D. T. 1990. An improved annealing scheme for the QAP, European Journal of Operational Research, 46: 93-100.

[29] Cui Y. and Weng J. J. 1996a. Hand segmentation using learning based prediction and verification for hand sign recognition, Proceedings of the Second International Conference on Automatic Face and Gesture Recognition: 88-93.

[30] Cui Y. and Weng J. J. 1996b. View-based hand segmentation and hand-sequence recognition with complex backgrounds. 13th International Conference on Pattern Recognition, Vienna, Germany, 3:617-621.

[31] Deb, K., Agrawal, S., Pratab, A., and Meyarivan, T. 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii, Proceedings of PPSN-6, Schoenauer, M., editor, Springer-Verlag: 849-858.

[32] Dix A., Finlay J., Abowd G. and Beale R. 1993. Human-computer interaction, Prentice Hall (UK), Hertfordshire, UK.

[33] Efron D. 1941. Gesture and environment, King's Crown Press, Morningside Heights, NY.

[34] Eliav A., Lavie T., Parmet Y., Stern H., Wachs J. and Edan Y. 2005. KISS Human-Robot Interfaces, 18th International Conference on Production Research (ICPR), July, Salerno, Italy.

[35] Fang G. Gao W. and Zhao D. 2004. Large vocabulary sign language recognition based on fuzzy decision trees, IEEE Systems, Man and Cybernetics, Part A., 34(3): 305-314.

[36] Finke G., Burkard R.E. and Rendl F. 1987. Quadratic assignment problems. Annals of Discrete Mathematics, 31: 61-82.

[37] Franklin D., Kahn R. E., Swain M. J., and Firby R. J. 1996. Happy patrons make better tippers --- creating a robot waiter using Perseus and the animate agent architecture, Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, Killington, Vermont, USA: 14-16.

[38] Freeman W. 1994. Television control by hand gestures, IEEE International Workshop on Automatic Face and Gesture Recognition, Zurich.

[39] Freeman W. and Roth M. 1995. Orientation histograms for hand gesture recognition, International Workshop on Automatic Face and Gesture Recognition, Zurich, June.

[40] Goux J. P., Kulkani S., Linderoth J. and Yoder M. 2000. An enabling framework for master-worker applications on the computational grid, Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania: 43-50.

[41] Griffins T. 2001. Usability testing in input device design. On the web: http://tim.griffins.ca/writings/usability_body.html
[42] Gu Y. I. and Tjahjadi T. 2002. Multiresolution feature detection using a family of isotropic bandpass filters, IEEE Transactions on Systems, Man and Cybernetics, Part B, 32(4): 443–454.
[43] Guo D, Yan Y H and Xie M. 1998. Vision-based hand gesture recognition for human-vehicle interaction. Fifth International Conference on Control, Automation, Robotics and Vision, Singapore, (1):151-155.
[44] Harling P. A. and Edwards A. D. N. 1996. Hand tension as a gesture segmentation cue. Proceedings of Gesture Workshop on Progress in Gestural Interaction: 75–88. March 19th 1996, University of York, U, ISBN: 3-540-76094-6.
[45] Harwin W. S. and Jackson R. D. 1990. Analysis of intentional head gestures to assist computer access by physically disabled people. Journal of Biomedical Engineering, 12:193-198.
[46] Hauptmann, A.G. 1989. Speech and gestures for graphic image manipulation, Proceedings of Computer Human Interaction, ACM Press, Austin , TX, 1: 241-245.
[47] Hauptmann, A.G. and McAvinney, P. 1993. Gestures with speech for graphic manipulation, International Journal of Man-Machine Studies, 38(2): 231-49.
[48] Ho T. T., Zhang H. 1999. Internet-based tele-manipulation, Proceedings of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering, Edmonton, Canada, 3: 1425-1430.
[49] Holland J. 1975. Adaptation in natural and artificial systems, University of Michigan Press, Cambridge, MA, USA.
[50] Höysniemi J., Hämäläinen P., Turkki L., Rouvi T. 2005. Children's intuitive gestures in vision-based action games, Communications of the ACM, 48(1): 44-50.
[51] Huang T. S. and Pavlovic V. 1995. Hand gesture modeling, analysis and synthesis, Proceedings, International Conference on Automatic Face and Gesture Recognition. Zurich, Switzerland.
[52] Jun-Hyeong D., Jung-Bae K., Kwang-Hyun P., Won-Chul B. and Zenn B. 2002. Soft remote control system using hand pointing gesture, International Journal of Human-friendly Welfare Robotic Systems, 3(1): 27-30.
[53] Juran, J. M. 1975. The non-Pareto principle: mea culpa, Quality Progress, 8(5):8–9.
[54] Just A., Marcel S., and Bernier O. 2004. Recognition of isolated complex mono- and bi-manual 3D hand gestures using discrete IOHMM, Sixth IEEE International Conference on Automatic Face and Gesture Recognition, Seoul, Corée, May 17-19.
[55] Just A., Rodriguez Y., and Marcel S. 2006. Hand posture classification and recognition using the modified census transform, Proceedings of International. Conference on Automatic Face and Gesture Recognition, April 10-12: 351- 356
[56] Kahol, K., Tripathi, P. and Panchanathan, S. 2004. Computational analysis of mannerism gestures, Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004, 3:946-949.
[57] Kahol K., Tripathi K. and Panchanathan S. 2006. Documenting motion sequences with a personalized annotation system, IEEE Multimedia, 13(1): 37-45.
[58] Kendon A. 1986. Current issues in the study of gesture, The biological foundations of gestures: motor and semiotic aspects, Lawrence Erlbaum Associates, Hillsdale, NJ,: 23-47.
[59] Kim J. S., Park K. H., Kim J. B., Do J. H., Song K. J. Bien Z. 2000. Study on intelligent autonomous navigation of avatar using hand gesture recognition, Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Nasheville, USA: 846-851.

[60] Kirishima T., Sato K. and Chihara K. 2005. Real-time gesture recognition by learning and selective control of visual interest points, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(3): 351-364.

[61] Kjeldsen R. and Hartman J. 2001. Design issues for vision-based computer interaction systems, Proceedings of the Workshop on Perceptual User Interfaces, Orlando, Florida, USA.

[62] Kjeldsen, R., Levas, A. and Pinhanez, C. 2003. Dynamically reconfigurable vision-based user interfaces, 3rd International Conference on Computer Vision Systems (ICVS'03), Graz, Austria: 323-332

[63] Klima E. S. and Bellugi U. 1974. Language in another mode. Language and brain: developmental aspects, Neurosciences research program bulletin. 12(4): 539-550.

[64] Kohler, M.R.J. 1997. System architecture and techniques for gesture recognition in unconstraint environments, Proceedings of the 1997 International Conference on Virtual Systems and MultiMedia (VSMM'97):137-146.

[65] Kölsch M, Beall A. C., and Turk M. 2003. An objective measure for postural comfort, HFES Annual Meeting Notes, October 13-17, Denver, Colorado.

[66] Kolsch M. Turk M. and Hollerer T. 2004. Vision-based interfaces for mobility, The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, August 22-26: 86- 94.

[67] Koopmans T. C. and Beckmann M. J. 1957. Assignment problems and the location of economic activities, Econometrica, 25:53-76.

[68] Kortenkamp D., Huber E. and Bonasso R. P. 1996. Recognizing and interpreting gestures on a mobile robot, Proceedings of the National Conference on Artificial Intelligence (AAAI), 2: 915-921

[69] Kray, C. and Strohbach, M. 2004. Gesture-based interface reconfiguration, Workshop on Artificial Intelligence in Mobile Systems (AIMS), at Ubicomp 2004, Seattle, USA.

[70] LaViola J. J. 1999. Whole-hand and speech input in virtual environments, Master's Thesis, CS-99-15, Brown University, Department of Computer Science, Providence, RI.

[71] Lenman S.,Bretzner L. and Thuresson B. 2002. Computer vision based recognition of hand gestures for human-computer interaction, Technical report TRITANA-D0209,CID-report, Stockhom, Sweden.

[72] Lin J., Wu Y., and Huang T. S. 2000. Modeling human hand constraints, Proceedings Workshop on Human Motion (Humo2000), Austin, TX, December 7-8: 121-126

[73] Link-Belt Construction Equipment Company. 1987. Operating safety: crane and excavators.

[74] Liu H., Abraham A. and Zhang. J. 2006. A particle swarm approach to quadratic assignment problems, To appear in the Proceeding of 11th Online World Conference on Soft Computing in Industrial Applications. September 18 - October 6.

[75] Long, A.C., Landay, J.A., Rowe, L.A. J. 1999. Implications for a gesture design tool, Proceedings of Conference on Human Factors in Computing Systems CHI'99, Pittsburgh, Pennsylvania, United States: 40-47.

[76] Mäntylä, V.-M. 2001. Discrete hidden markov models with application to isolated user-dependent hand gesture recognition. VTT Electronics, Espoo Publisher. 104 p. VTT Publications ISBN 951-38-5875-8; 951-38-5876-6: 449.

[77] Martello S. and Toth P. 1990. Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons. ISBN 0-471-92420-2.

[78] Marrin, T. 1999. Inside the conductor's jacket: analysis, interpretation, and musical synthesis of expressive gesture, Massachusetts Institute of Technology PhD, Thesis, Media Arts and Sciences, Cambridge, MA

[79] McNeill D. 1995. Hand and mind – what gestures reveal about thought, The University of Chicago Press. Paperback Edition. Chicago and London, ISBN 0-226-56132-1: 416

[80] Miners B. W., Basir O. A., and Kamel M. 2002. Knowledge-based disambiguation of hand gestures, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, October 6-9, 5:201-206.

[81] Munk K. 2001. Development of a gesture plug-in for natural dialogue interfaces, Gesture and Sign Languages in Human-Computer Interaction, International Gesture Workshop, GW 2001, London, UK

[82] Natan R., Stern H., Wachs J. (supervisors). 2003. Fourth Year Project Report, Bardenstein D. and Ben Yair T., Department of Mechanical Engineering, Ben Gurion University of the Negev, Be'er Sheva, Israel.

[83] Ng C. W. and Ranganath S. 2002. Real-time gesture recognition system and application, Image and Vision Computing, 20(13–14): 993–1007.

[84] Nespoulous J., Perron P. and Lecours A. 1986. The biological foundation of gestures: motor and semiotic aspects, Lawrence Erlbaum Associates, Hillsdale, MJ.

[85] Nielsen M., Storring M., Moeslund T. B., and Granum E. 2003. A procedure for developing intuitive and ergonomic gesture interfaces for man-machine interaction, Technical Report CVMT 03-01, CVMT, Aalborg University, March, 2003.

[86] Norman D. 1988 .The psychology of everyday things. New York, Basic Books, ISBN: 0465067093.

[87] Oviatt S.L., Cohen P.R., Wu L. ,Vergo J., Duncan L., Suhm B., Bers J., Holzman T., Winograd T., Landay J., Larson J. and Ferro D. 2000. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions, Human Computer Interaction, 2000, 15(4): 263-322.

[88] Pareto V. 1896. Cours d'economic politique. Rouge, Laussanne, Switzerland, 1896.

[89] Parvini F. and Shahabi C. 2005. UTGeR: A user-independent technique for gesture recognition, 11th International Conference on Human-Computer Interaction, July 22-27, Las Vegas, Nevada, USA

[90] Pavlovic V., Sharma R., and Huang T. 1997. Visual interpretation of hand gestures for human-computer interaction: A review, IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7): 677-695.

[91] Pook P.K, Ballard D.H. 1995. Teleassistance: A gestural sign language for teleoperation, Proceedings of Workshop on Gesture at the User Interface, International Conference on Computer-Human Interaction CHI 95, Denver, CO, USA.

[92] Preston S, Matshoba L., and Chang M. C. 2005. A gesture driven 3D interface, Technical Report CS05-15-00, Department of Computer Science, University of Cape Town, South Africa.

[93] Quek F. K. H. 1994. Toward a vision based hand gesture interface, Proceedings of the conference on Virtual reality software and technology: 17-31.

[94] Rigoll G., Kosmala A., and Eickeler S. 1997. High performance real-time gesture recognition using hidden markov models, Gesture Workshop, Bielefeld, Germany: 69-80.

[95] Ronen A., Edan Y., Eliav A. (supervisors). 2005. Experiment with selected hand gestures for robotic vehicle navigation, Fourth Year Project Report, Karkash Y. and Mei Zahav H., Department of Mechanical Engineering, Ben Gurion University of the Negev, Be'er Sheva, Israel.

[96] Segen, J. and Kumar, S. 2000. Look, ma, no mouse!, Communications of the ACM, ACM Press, New York, USA, 43(7): 102-109.

[97] Shneiderman, B. 1998. Designing the user interface. Addison Wesley (3nd Ed.), Reading, MA.

[98] Shrawan K. and Anil M. 1996. Electromyography in ergonomics, Taylor and Francis Ed., Taylor & Francis, Bristol, PA.

[99] Starner T. and Pentland A. 1995. Visual recognition of american sign language using hidden markov models, International Workshop on Automatic Face and Gesture Recognition, Zurich, Switzerland:189-194.

[100] Stern H., Wachs J., and Edan Y. 2004a. Hand gesture vocabulary design: a multicriteria optimization, Proceedings of the IEEE International Conference on Systems, Man & Cybernetics. The Hague, Netherlands,1: 19- 23.

[101] Stern H., Wachs J., and Edan Y. 2004b. Parameter calibration for reconfiguration of a hand gesture tele-robotic control system, Japan-USA Symposium on Flexible Automation, July 1921, Denver, CO.

[102] Stern H., Wachs J., and Edan Y. 2006. Optimal hand gesture vocabulary design using psycho-physiological and technical factors, 7th International Conference Automatic Face and Gesture Recognition, FG2006, April 10-12, Southampton, UK.

[103] Stokoe W. C. 1972. Semiotics and human sign languages, Approaches to semiotics series, C. Baker and R. Battison edition, Mouton, The Hague, 21, ISBN 90-279-2096-6.

[104] Schwartz D. 1998. FAA pricing handbook, On the web: http://fast.faa.gov/pricing/index.htm

[105] Takahashi T. and Kishino F. 1991. Gesture coding based in experiments with a hand gesture interface device, SIGCHI Bulletin, April, 23(2):67-73.

[106] Triesch J, Malsburg C.1998. Robotic gesture recognition by cue combination, Proceedings of the Informatik'98, 28th Annual Meeting of the Gesellschaft. Magdeburg, Germany: 223-232.

[107] Triesch J, Malsburg C. 2001. A system for person-independent hand posture recognition against complex backgrounds, IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(12): 1449-1453.

[108] Wachs J., Kartoun U. Stern H. and Edan Y. 2002. Real-time hand gesture using the fuzzy-c means algorithm, Proceedings of WAC 2002, June 9-13, Orlando, Florida, 13: 403–409.

[109] Wachs J., Stern H. and Edan Y. 2003. Parameter search for an image processing fuzzy c-means hand gesture recognition system, Proceedings of IEEE International Conference on Image Processing, Barcelona, Spain, 3: 341-345.

[110] Wachs J., Stern H. and Edan Y. 2004. Optimization of hand gesture command vocabularies - A multiobjective quadratic assignment approach, X ELAVIO Latin American Association of Operations Research Societies Summer School for Young Scholars, Montevideo, Uruguay.

[111] Wachs J., Stern H. and Edan Y. 2005. Cluster labeling and parameter estimation for automated set up of a hand gesture recognition system, IEEE Transactions on Systems, Man and Cybernetics, Part A, 35(6): 932-944.

[112] Waldherr S., Thrun S., Romero R., and Margaritis D. 1998. Template-based recognition of pose and motion gestures on a mobile robot, Proceedings of the AAAI Fifteenth National Conference on Artificial Intelligence: 977-982.

[113] Wagner M. O., Yannou B., Kehl S., Feillet D., and Eggers J. 2003. Ergonomic modeling and optimization of the keyboard arrangement with an ant colony algorithm, Journal of Engineering Design, 14(2): 187-208.

[114] Wang C., Gao W. and Shan S. 2002. An approach based on phonemes to large vocabulary Chinese sign language recognition, Proceedings of Fifth IEEE International Conference on Automatic Face and Gesture Recognition, Washington, DC, USA:393-398

[115] Wheeler, K.R. 2003. Device control using gestures sensed from EMG, Proceedings of the 2003 IEEE International Workshop on Soft Computing in Industrial Applications: 21-26

[116] Wolf C. G. and Morrel-Samuels P. 1987. The use of hand-drawn gestures for text editing, International Journal of Man-Machine Studies, 27: 91-102.
[117] Wolf C. G. and Rhyne J. R. 1987. A taxonomic approach to understanding direct manipulation, Journal of the Human Factors Society 31th Annual Meeting: 576-780.
[118] Wren C., Azarbayejani A., Darrell T., and Pentland A.1997. Pfinder: Real-time tracking of the human body, IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7):780-785.
[119] Wright, T.P.1936. Factors affecting the cost of airplanes. Journal of Aeronautical Sciences, 3(4):122 -128.
[120] Wu Y. and Huang T .1999. Vision-based gesture recognition: a review, Proceedings of the International Gesture Recognition Workshop:103-115.
[121] Wundt W. 1973. The language of gestures. Translated from Wundt, W. Völkerpsychologie: Eine Untersuchung der Entwicklungsgesetze von Sprache, Mythus und Sitte. (Stuttgart: Alfred Kröner Verlag) by Thayer, J. S., Greenleaf, C. M. and Silberman, M. D. The Hague: Mouton, 1(1)2.
[122] Yao Y. Zhu M. Jiang Y. and Lu G. 2004. A bare hand controlled AR map navigation system, IEEE International Conference on Systems, Man and Cybernetics. 3:2635- 2639.
[123] Yasumuro Y. Chen Q. and Chihira K. 1999. Three dimensional modeling of the human hand with motion constraints, Image and Vision Computing, 17:149-156.
[124] Yin X. M. and Xie M. 2001. Finger identification in hand gesture based human-robot interaction, Journal of Robotics and Autonomous Systems, 34(4): 235-250.
[125] Yin X. and Xie M. 2003. Estimation of the fundamental matrix from uncalibrated stereo hand images for 3-D hand gesture recognition, Pattern Recognition, 36(3): 567–584.
[126] Zhenyao M. and Neumann, U. 2006. Lexical gesture interface, Proceedings of the IEEE International Conference on Computer Vision Systems, ICVS '06: 7.
[127] Zhou H., Lin D.J. and Huang T.S. 2004 .Static hand gesture recognition based on local orientation histogram feature distribution model, Proceedings of Conference on Computer Vision and Pattern Recognition Workshop: 161-161.
[128] Zob M. Geiger M. Schuller B. Lang M. and Rigoll G. 2003. A real-time system for hand gesture controlled operation of in-VMR devices, Proceedings of International Conference on Multimedia and Expo, 2003. ICME '03, 3:541-544.

10 Appendices

Appendix A Memorability test application and queries

This appendix presents two forms used to measure the memorability indices, and two queries used for user feedback

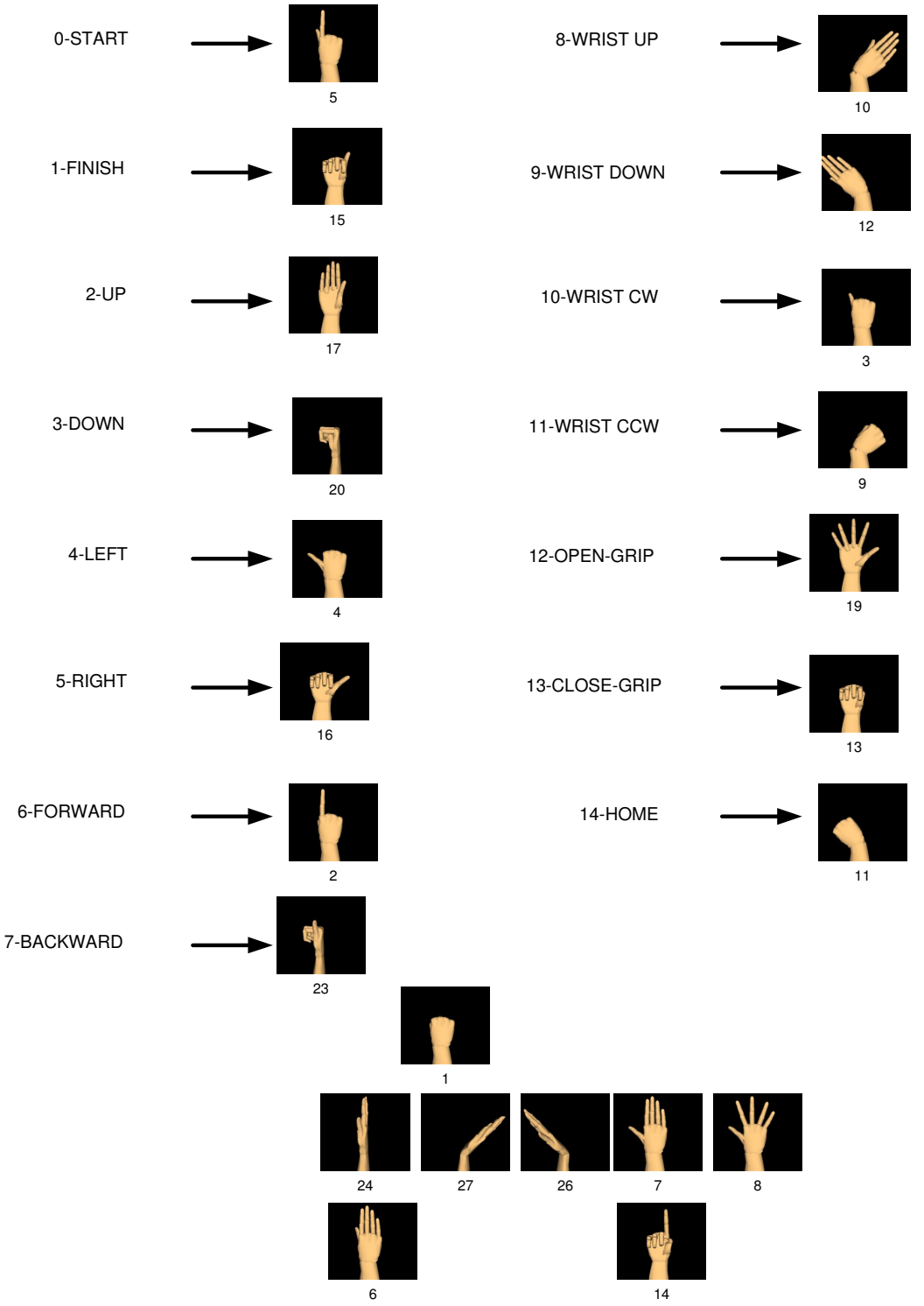


Figure A.1. Memorability test application for the robot task

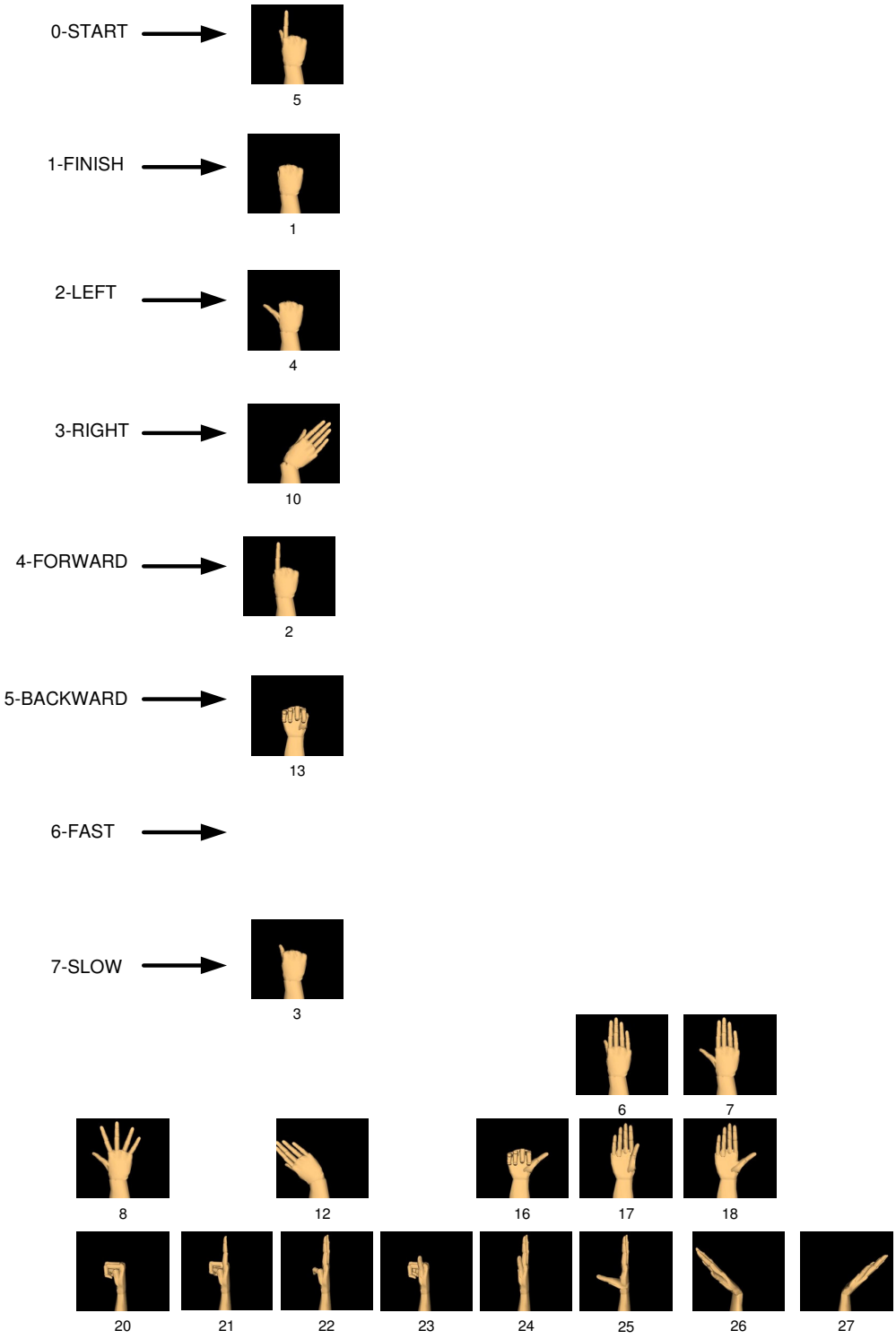


Figure A.2. Memorability test application for the VMR task

שם: **ת"ה**

סיכום הניסויים

ניסוי עם הדבוט

1) מהן הבעיות שנתקלת במהלך הניסוי?

---

2) באיזה דרכים ניתן לשפר את הממשק?

ניסוי עם הרכב

3) מהן הבעיות שנתקלת במהלך הניסוי?

---

4) באיזה דרכים ניתן לשפר את הממשק?

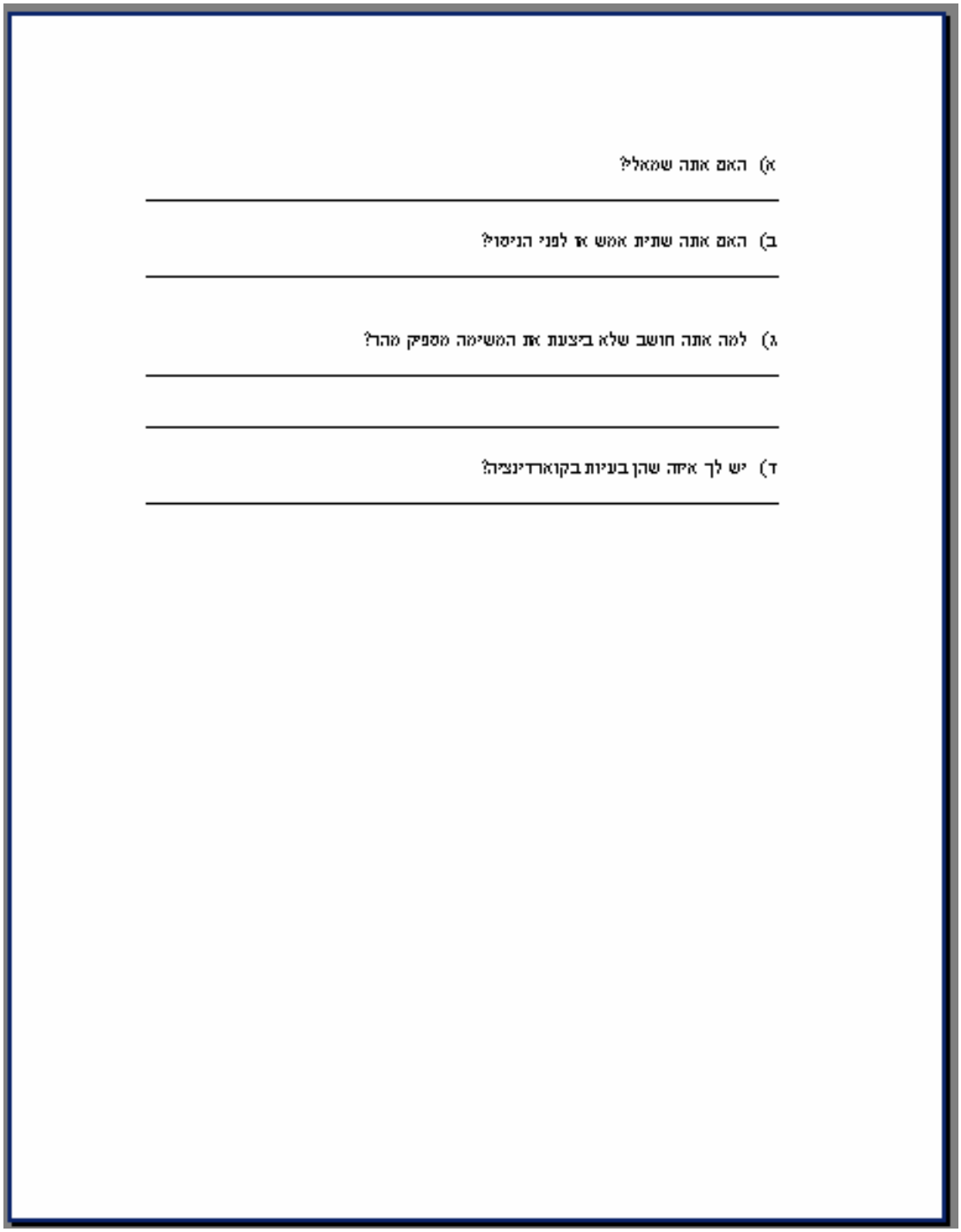


Figure A.3. Feedback form for the VMR and robot tasks

Appendix B. Dominate set partition: good and bad GV solutions

This appendix introduces the definitions of dominating and dominated solutions, presents an example of how these solutions can be found, and shows their relation to the good and bad GVs.

Let the i^{th} Multiobjective solution be Z (i) = [z(i,1),z(i,2),z(i,3)] Let N be the set of multiobjective solutions, such that N = {1,..i,..,n}

Definition: Dominate solution pairs
For any pair i,j of solutions from N we say solution i dominates solution j, iff
z(i,k) > z(j,k), k=1,2,3
Let

- $i \succ j$ (B.1)

Denote the relation solution i dominates solution j
An example of a pair of solutions where i dominates j is given below:
Z(i) = [98.33, 5086, 5687] and Z(j) = [90.66, 4739, 4359]
It may be that a pair of solutions do not satisfy the dominance relation for example.
Then we write

- $i \not\succ j$ (B.1)

An example of a pair of solution where one does not dominate the other is
Z(i) = [98.33, 5086, 56] and Z(j) = [90.66, 47, 4359]
[D, D'] is a dominant pair partition of n solutions, if the following holds:
(i) [D, D '] is a partition of the n solutions, where D intersection D' = empty set, and D union D' = N
(ii) for any two solutions (i,j), i is an element of D, and j is an element of D' iff
$i \succ j$

Example:
Let D = V_G and D' = V_B then for the robotic arm a DSP is given by the following two tables:

D= V_G

I	Z(i,1) Acc	Z(i,2) Int	Z(i,3) Conf
1	98.33	5086(min)	5687
2	98.33	6203	5439
3	98.33	6224	5259 (min)
4	98.33	6658	5393
5	98.33	5541	5647
6	98.5	6335	5633
7	98.5	6677	5458
8	98.5	6421	5396

D'= V_B

I	Z(i,1) Acc	Z(i,2) Int	Z(i,3) Conf
1	90.66	4798(max)	4405
2	90.66	4766	3535
3	90.66	4739	4359
4	90.66	4136	5075
5	90.66	4115	5007
6	90.66	1390	5128
7	90.66	331	5192
8	90.66	262	5196 (max)

Any solution i in set D dominates all the other solutions in the set D' (check the min /max values in indicated in the tables.

Using the procedure explained above, 16 GVs were obtained for each the car and the robot tasks. Eight dominating solutions (V_G) and eight dominated solutions (V_B) for the car and robot tasks (Table B.1 and Table B.2).

Table B.1. V_G and V_B for the VMR case

i	Gn	GV	Z(i,1)	Z(i,2)	Z(i,3)	w_1	w_2
1	1 2 3 4 5 10 20 26	5 1 2 3 4 10 26 20	405	3490	88.125	10	0
2	1 2 3 4 5 10 20 23	5 1 2 23 4 10 3 20	418	3499	84.6875	10	0
3	1 2 3 4 5 10 22 26	5 1 2 22 4 10 26 3	402	3488	86.875	9	1
4	1 2 3 4 5 10 17 22	5 1 2 17 4 10 3 22	411	3497	86.5625	10	0
5	1 2 3 4 5 10 13 17	5 1 2 17 4 10 3 13	422	3496	87.8125	10	0
6	1 2 3 4 5 10 13 23	5 1 2 23 4 10 3 13	424	3496	84.6875	9	1
7	1 2 3 4 5 10 13 22	5 1 2 13 4 10 3 22	420	3493	86.5625	9	1
8	1 2 3 4 5 10 18 22	5 1 2 22 4 10 18 3	409	3492	86.875	10	0
9	6 7 8 10 12 13 17 21	21 7 6 17 12 10 8 13	3389	3546	99.375	9	1
10	6 7 8 10 12 17 20 21	21 7 6 17 12 10 8 20	3383	3549	99.375	7	3
11	6 7 8 10 12 17 21 23	21 7 6 17 12 10 8 23	3380	3548	96.25	10	0
12	6 7 8 10 12 17 21 24	21 7 6 17 12 10 8 24	3376	3552	99.0625	9	1
13	6 7 8 10 12 17 18 21	21 8 6 17 12 10 18 7	3157	3541	99.6875	10	0
14	6 7 8 10 12 17 22 24	22 8 6 17 12 10 24 7	3151	3556	97.5	8	2
15	6 7 8 10 12 17 18 20	8 20 6 17 12 10 18 7	3142	3539	99.375	9	1
16	6 7 8 10 17 21 26 27	21 7 6 17 26 27 8 10	3020	3801	99.6875	10	0

Table B.2. V_G and V_B for the robotic arm case

i	Gn	GV	Z(i,1)	Z(i,2)	Z(i,3)	w_1	w_2
1	4 5 6 7 8 10 11 13 14 16 17 19 24 26 27	5 7 17 6 4 16 14 24 10 11 26 27 19 13 8	6979	5287	98.33	10	0
2	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	14 7 17 6 4 16 24 20 10 11 26 27 19 13 8	6671	5458	98.5	9	1
3	4 5 6 7 8 10 11 13 14 16 17 19 24 26 27	5 7 17 6 4 16 24 8 10 11 26 27 19 13 14	6658	5398	98.33	8	2
4	4 5 6 7 8 10 11 14 16 17 19 20 24 26 27	5 7 17 6 4 16 24 20 10 11 26 27 19 8 14	6421	5396	98.5	8	2
5	4 5 6 7 8 10 11 13 14 16 19 20 24 26 27	5 7 14 6 4 16 24 20 10 11 26 27 19 13 8	6224	5259	98.33	9	1
6	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	10 8 17 6 4 16 24 20 26 27 7 11 19 13 14	6335	5633	98.5	8	2
7	4 5 6 7 8 10 11 13 14 16 19 20 24 26 27	5 7 8 6 4 16 24 20 10 11 26 27 19 13 14	6203	5439	98.33	8	2
8	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	10 14 17 6 4 16 24 20 26 27 7 11 19 13 8	6331	5645	98.5	7	3
9	1 2 3 4 5 9 12 13 15 16 17 19 20 23 27	2 1 17 12 4 16 5 20 15 3 9 27 19 13 23	4766	3535	90.67	10	0
10	1 2 3 4 5 8 9 12 13 15 16 17 19 20 23	8 1 17 5 4 16 2 20 15 3 9 12 19 13 23	4739	4359	90.67	10	0
11	1 2 3 4 5 9 10 12 13 15 16 17 19 20 23	9 15 17 1 4 3 2 20 16 12 23 10 19 13 5	1390	5128	90.67	4	6
12	1 2 3 4 5 9 10 12 13 15 16 17 19 20 23	10 9 20 1 3 4 23 19 16 12 17 15 2 5 13	262	5196	90.67	0	10
13	1 2 3 4 5 9 12 13 15 16 17 19 20 23 27	9 15 17 1 4 3 2 20 16 12 23 27 19 13 5	1394	5128	90.67	4	6
14	1 2 3 4 5 9 10 12 13 15 16 17 19 20 23	9 15 20 1 4 19 17 3 16 12 23 10 2 13 5	331	5192	90.67	3	7
15	1 2 3 4 5 9 10 12 13 15 16 17 19 20 23	5 15 17 1 4 16 2 20 10 12 3 9 19 13 23	4675	4939	90.67	8	2
16	1 2 3 4 5 9 10 12 13 15 16 17 19 20 23	3 15 17 1 4 16 2 20 10 12 9 5 19 13 23	5124	4735	90.67	10	0

Table B.3. Pareto set for the VMR study

Sol	i	Gn	Gn*	Z(i,1)	Z(i,2)	Z(i,3)	w1	w2
1	7	6 7 8 10 12 16 18 21	21 12 6 16 7 8 18 10	74	4152	100	6	4
1	8	6 7 8 10 12 16 18 21	21 8 6 12 7 16 18 10	140	4018	100	7	3
1	11	6 7 8 10 12 16 18 21	21 6 8 16 12 10 18 7	2797	3278	100	10	0
2	13	6 7 8 10 12 16 18 25	12 10 6 18 7 8 25 16	47	4167	100	1	9
2	15	6 7 8 10 12 16 18 25	12 25 6 16 7 8 18 10	57	4163	100	3	7
2	18	6 7 8 10 12 16 18 25	12 18 6 16 7 8 25 10	61	4161	100	6	4
2	19	6 7 8 10 12 16 18 25	12 8 6 18 7 16 25 10	123	4047	100	7	3
5	50	6 7 8 10 12 16 21 25	21 12 6 16 7 8 25 10	78	4148	100	5	5
5	51	6 7 8 10 12 16 21 25	21 12 6 16 7 8 25 10	78	4148	100	6	4
5	52	6 7 8 10 12 16 21 25	21 8 6 12 7 16 25 10	144	4014	100	7	3
5	55	6 7 8 10 12 16 21 25	21 7 6 16 12 10 8 25	2791	3531	100	10	0

Table B.4. Pareto set for the robotic arm study

sol	i	Gn	Gn*	Z(i,1)	Z(i,2)	Z(i,3)	w1	w2
1	1	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	27 14 8 6 4 7 20 24 13 10 19 11 17 26 16	67	5930	98.5	0	10
1	2	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	27 14 24 6 4 7 20 8 26 10 19 11 17 13 16	424	5929	98.5	1	9
1	3	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	10 14 24 6 4 7 20 8 26 27 19 11 17 13 16	651	5927	98.5	2	8
1	7	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	10 14 17 6 4 7 24 20 26 27 8 11 19 13 16	3602	5862	98.5	6	4
1	8	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	10 14 17 6 4 16 24 20 26 27 7 11 19 13 8	6331	5645	98.5	7	3
1	9	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	10 8 17 6 4 16 24 20 26 27 7 11 19 13 14	6335	5633	98.5	8	2
1	10	4 6 7 8 10 11 13 14 16 17 19 20 24 26 27	14 7 17 6 4 16 24 20 10 11 26 27 19 13 8	6671	5458	98.5	9	1
2	15	4 5 6 7 8 10 11 14 16 17 19 20 24 26 27	10 14 8 6 4 7 24 20 26 27 19 11 17 5 16	1092	5883	98.5	3	7
3	30	5 6 7 8 10 11 13 14 16 17 19 20 24 26 27	10 14 17 6 8 7 24 20 16 11 26 27 19 13 5	3890	5710	98.33	7	3
4	37	4 5 6 7 8 10 11 13 14 16 19 20 24 26 27	10 14 8 6 4 7 24 20 26 27 19 11 5 13 16	1117	5866	98.33	3	7
4	41	4 5 6 7 8 10 11 13 14 16 19 20 24 26 27	10 14 7 6 4 16 24 20 26 27 8 11 19 13 5	5086	5687	98.33	7	3
5	50	4 5 6 7 8 10 11 13 14 16 17 19 24 26 27	10 14 17 6 4 16 24 7 26 27 8 11 19 13 5	5541	5647	98.33	5	5
5	55	4 5 6 7 8 10 11 13 14 16 17 19 24 26 27	5 7 17 6 4 16 14 24 10 11 26 27 19 13 8	6979	5287	98.33	10	0

Table B.5. Pareto set for the VMR study using the multiobjective decision approach

num	i	Z(i,1)	Z(i,2)	Z(i,3)
1	140	3389	3546	99.4
2	726	2797	3278	100
3	913	2791	3531	100
4	1196	3383	3549	99.4
5	1230	3037	3553	98.8
6	1273	3450	3547	98.1
7	1296	3376	3552	99.1
8	1307	3374	3543	99.7
9	1351	3151	3556	97.5
10	1614	2757	3544	100
11	1625	2753	3547	100
12	**1804**	**2907**	**2992**	**100**
13	2197	2383	3798	100
14	2209	3456	3361	94.4
15	2936	1153	3905	99.4
16	2961	657	4224	98.8
17	3037	40	4231	98.4
18	3038	660	4227	98.4
19	3040	666	4226	98.4
20	3052	717	4186	99.1
21	3111	1159	3949	98.8
22	3308	2596	3810	99.4
23	3345	59	4228	98.8
24	3847	2386	3823	99.4
25	4023	924	4116	94.1
26	4221	2391	3830	99.1
27	4232	2393	3810	99.7
28	4254	729	4056	99.7
29	4256	1138	3971	99.7
30	4331	918	4065	99.4
31	4333	1144	3949	99.4
32	4430	1144	4015	99.1
33	4441	733	4052	99.7
34	4443	1142	3967	99.7
35	4453	1189	3845	99.7
36	4665	40	4225	99.1
37	4666	60	4224	99.1
38	4676	54	4185	99.7
39	4733	68	4209	99.4
40	4734	84	4204	99.4
41	4735	96	4198	99.4
42	**4744**	**72**	**4167**	**100**
43	4745	78	4164	100
44	4746	82	4162	100
45	4749	144	4048	100
46	4751	553	3963	100
47	4761	568	3849	100
48	4874	47	4215	99.4
49	4876	53	4214	99.4

num	i	Z(i,1)	Z(i,2)	Z(i,3)
50	4878	69	4206	99.4
51	4888	59	4210	99.4
52	4907	40	4173	100
53	4908	47	4172	100
54	4910	53	4170	100
55	4911	57	4168	100
56	4936	2378	3806	100
57	5167	2375	3814	99.7
58	5274	100	4194	99.4
59	5311	572	3845	100
60	5321	148	4034	100
61	5323	557	3949	100
62	5331	2437	3789	100
63	5343	913	4117	94.4
64	5441	2434	3820	98.4
65	5552	2382	3801	100
66	5558	862	4181	99.4
67	5565	1147	3908	99.4
68	5589	44	4230	98.8
69	5590	651	4227	98.8
70	5593	663	4221	98.8
71	5601	633	4192	99.4
72	5603	721	4185	99.4
73	5612	633	4192	99.4
74	5615	641	4187	99.4
75	5657	667	4216	99.1
76	5659	679	4210	99.1
77	5668	661	4176	99.7
78	5671	665	4174	99.7
79	5711	718	4226	97.5
80	5728	938	4097	98.1
81	5799	642	4222	99.1
82	5831	636	4182	99.7
83	5835	640	4180	99.7
84	5860	2778	3805	99.7
85	5887	868	4219	98.8
86	5894	1153	3952	98.8
87	5895	1221	3923	98.8
88	5899	866	4177	99.4
89	5906	1223	3903	99.4
90	5941	50	4232	97.2
91	5942	721	4229	97.2
92	5987	730	4227	95.3
93	6114	1220	3950	97.5
94	6198	683	4206	99.1
95	6255	3020	3801	99.7
96	6366	2657	3812	98.1
97	6557	48	4230	97.5
98	6558	136	4229	97.5

Appendix C. Good and bad vocabularies – graphical representation

In this appendix sixteen GVs are presented for each the VMR and the robotic arm tasks. The first eight are V_G and the last eight are V_B.

1


2


3


4


5


6


7


8


9


10


11


12


13


14

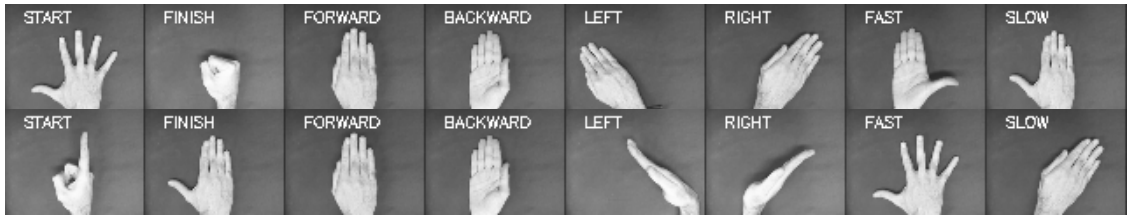
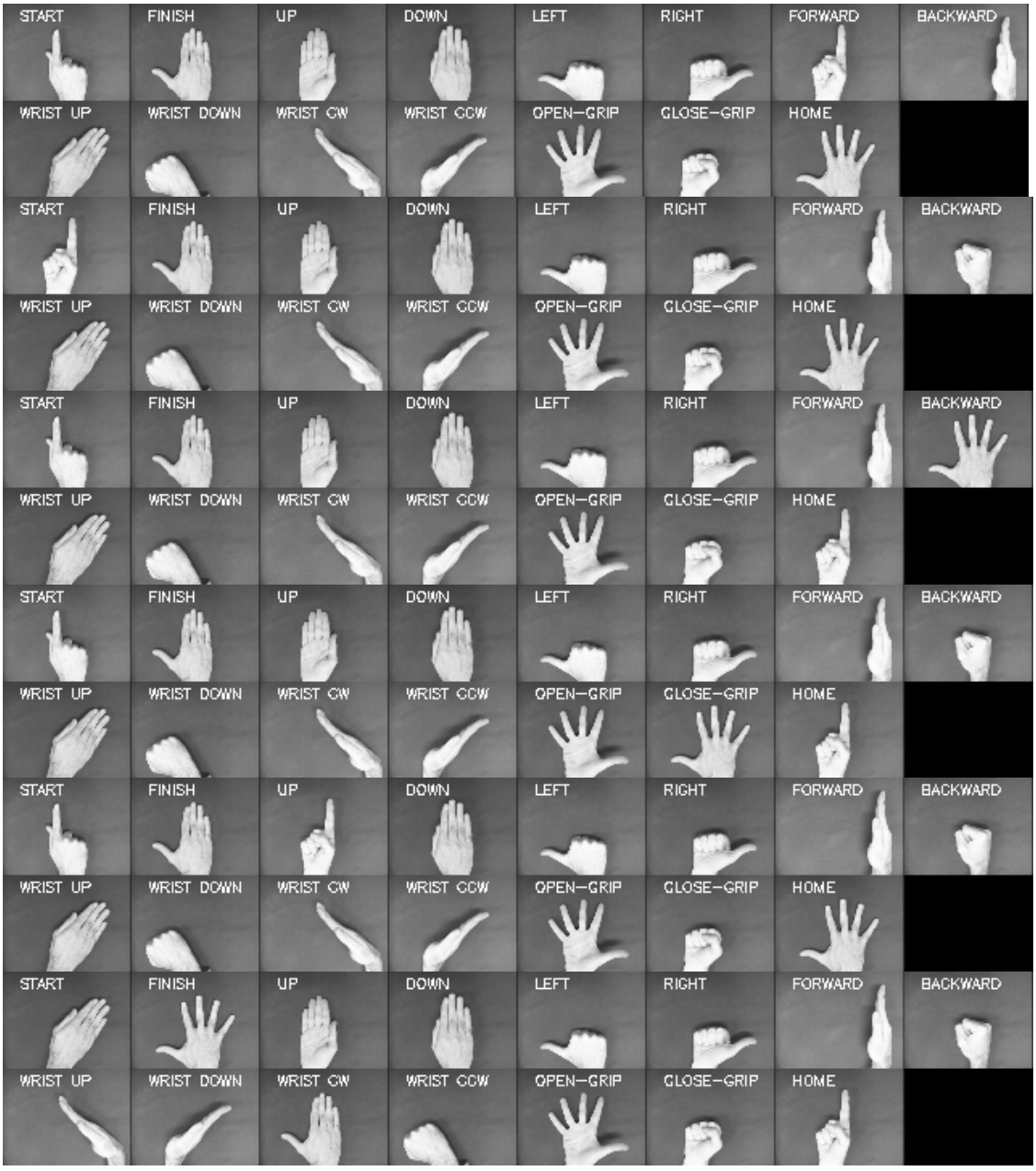
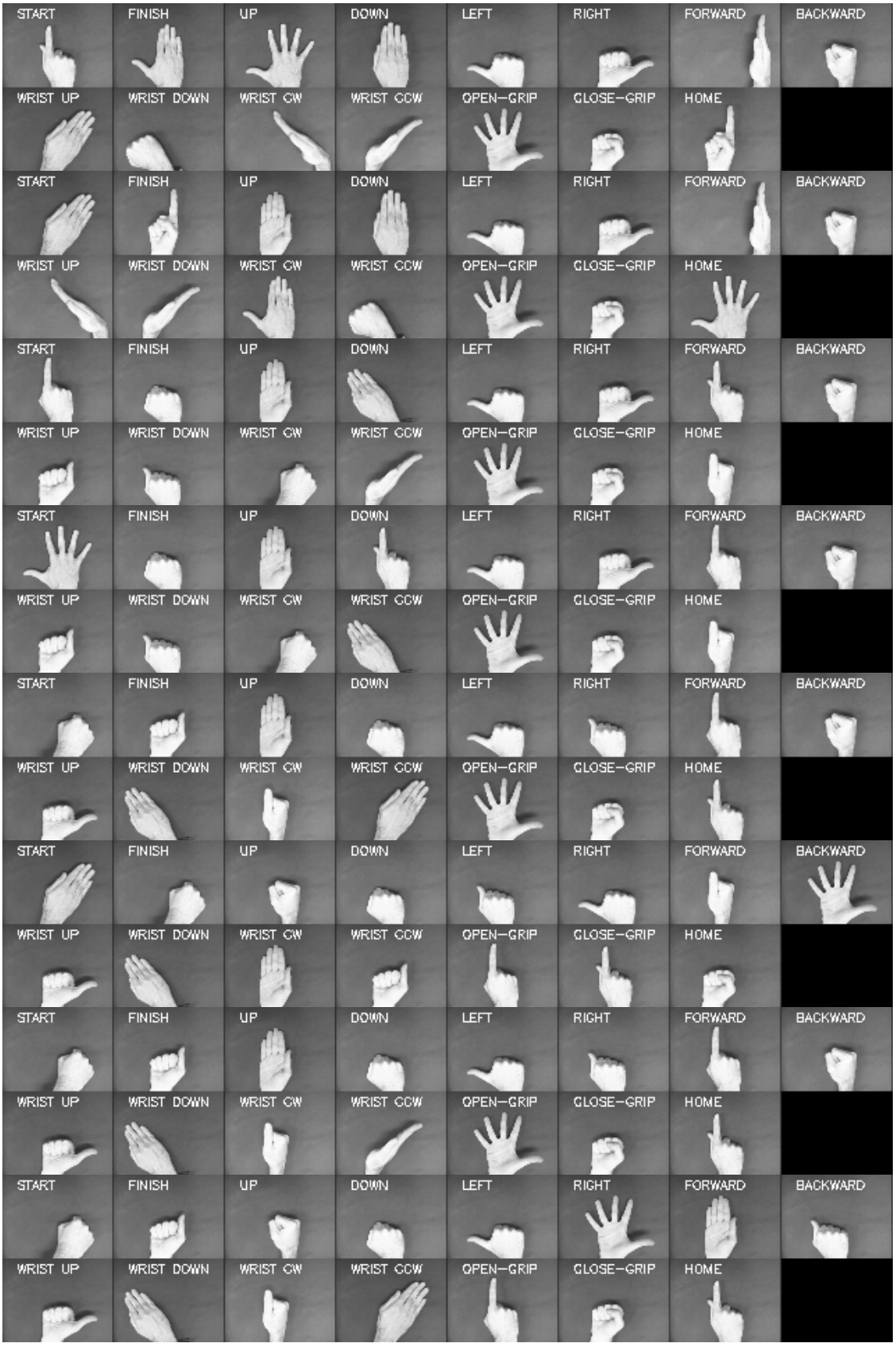



Figure C.1. GVs for the VMR study. 1-8 Bad GV. 9-16 Good GV



7



8

9

10

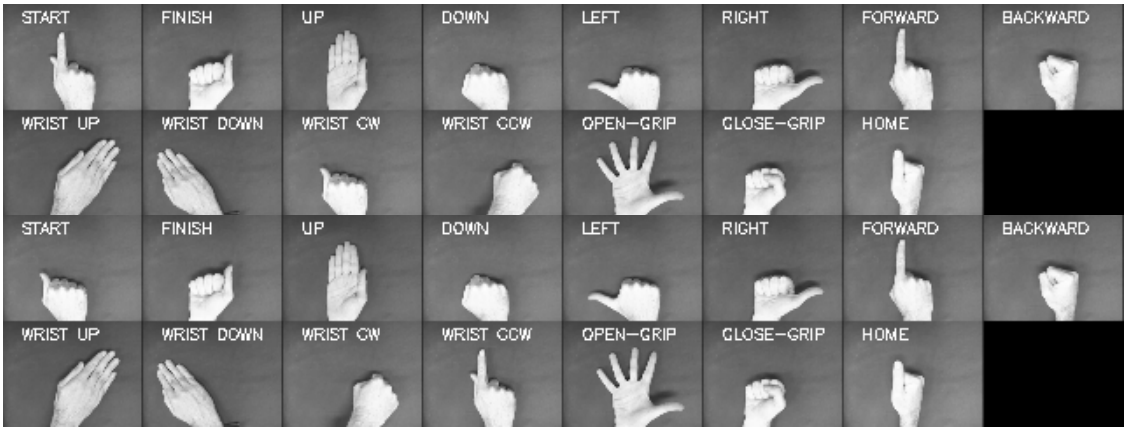
11

12

13

14

15



16

Figure C.2. GVs for the robotic arm study. 1-8 Good GV. 9-16 Bad GV

Appendix D. Human factors matrices

This appendix includes all the matrices obtained from the psycho-physiological experiments. The weighted intuitiveness, the extended intuitiveness using the extended master set of gestures, the complementary weighted intuitiveness, the stress, the duration and the frequency matrices are presented below.

Table D.1 Robot task intuitiveness matrix

Index	Gesture	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	sum
1	00000000000	1	3	1	6				1			1			6	7	26
2	00010000000	4		2				3	1				1			2	13
3	00100000000	2				2			2		1				2		9
4	00100001000	1			2	8			2		1		1			1	16
5	00110000000	3		1	1			1			1						7
6	00111110000	3	11		4			3							1	1	23
7	00111111000	1	5		2					2		1		2		2	15
8	00111111111	5	4											6	1	3	19
9	01000000000						1					3	1		2		7
10	01111110000						10	1		2			1				14
11	02000000000					1						1	4	1			7
12	02111110000					10			1		2	1					14
13	10000000000		1	1	4				2						9		17
14	10010000000			4				2		1							7
15	10100000000		3		1		1			2							7
16	10100001000	1		2			8			1		1			2		15
17	10111110000	2	1	6					2		1			1	1	4	18
19	10111111111		1											10		2	13
20	20000000000		2	2	3			1	4			1			1	1	15
23	20100000000	1		1					3	1					2	3	11
24	20111110000	1		2		1		6	1			1		1	1	5	19
26	21111110000	1				7		1	1	2	1	2					15
27	22111110000		1				8	1	1	1	1		2				15
	sum	26	32	22	23	29	28	19	21	12	8	12	10	21	28	31	322

Table D.2. VMR task intuitiveness matrix

Index	Gesture	1	2	3	4	5	6	7	8	sum
1	00000000000	1	5		3			3	2	14
2	00010000000	3		3				1	1	8
3	00100000000		2		1			1	1	5
4	00100001000	2				9				11
5	00110000000	2		2				2	2	8
6	00111110000	2	9	6		1		4	2	24
7	00111111000	4	4	1	2	1			3	15
8	00111111111	5	4					5		14
10	01111110000						10		1	11
12	02111110000					10				10
13	10000000000				6				4	10
16	10100001000				2		9			11
17	10111110000	2	1	1	3		1	1		9
18	10111111000		1				1	2		4
20	20000000000		2		2			1	3	8
21	20010000000	4								4
22	20011110000	1		1	2				1	5
23	20100000000	1	1		3			1	1	7
24	20111110000	3	2	11	2			2	1	21
25	20111112000		1					4		5
26	21111110000			1		8		1		10
27	22111110000				1		8			9
	sum	30	32	26	27	29	29	28	22	223

Table D.3. Robot task intuitiveness weighted matrix

Index	Gesture	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	sum
1	00000000000	3	9	2	11				1			1			15	15	57
2	00010000000	10		6				8	1				2			2	29
3	00100000000	4				6			4		3				5		22
4	00100001000	1			4	24			5		2		2			3	41
5	00110000000	6		3	1			2			2						14
6	00111110000	7	30		11			6							2	3	59
7	00111111000	2	13		4					4		1		4		4	32
8	00111111111	12	9											18	2	6	47
9	01000000000						2					6	3		3		14
10	01111110000						26	3		5			2				36
11	02000000000					2						3	7	2			14
12	02111110000					26			2		5	2					35
13	10000000000		3	2	8				4						22		39
14	10010000000			10				6		2							18
15	10100000000		6		2		3			6							17
16	10100001000	2		4			24			2		1			3		36
17	10111110000	4	2	15					3		3			3	1	8	39
19	10111111111		3											25		5	33
20	20000000000		5	4	5			2	10			2			3	2	33
23	20100000000	3		3					4	1					5	6	22
24	20111110000	2		5		3		14	3			2		3	2	8	42
26	21111110000	1				18		2	3	4	2	5					35
27	22111110000		1				19	3	2	3	2		5				35
	sum	57	81	54	46	79	74	46	42	27	19	23	21	55	63	62	749

Table D.4. VMR task intuitiveness weighted matrix

Index	Gesture	1	2	3	4	5	6	7	8	sum
1	00000000000	2	14		7			4	4	31
2	00010000000	7		8				1	3	19
3	00100000000		3		2			2	2	9
4	00100001000	3				27				30
5	00110000000	6		6				4	4	20
6	00111110000	5	23	17		2		11	2	60
7	00111111000	8	10	3	4	1			6	32
8	00111111111	11	12					8		31
10	01111110000						29		2	31
12	02111110000					28				28
13	10000000000				12				8	20
16	10100001000				3		27			30
17	10111110000	4	3	2	7		2	2		20
18	10111111000		2				1	5		8
20	20000000000		4		4			3	5	16
21	20010000000	11								11
22	20011110000	2		2	3				2	9
23	20100000000	2	3		8			2	3	18
24	20111110000	6	4	29	5			6	1	51
25	20111112000		2					7		9
26	21111110000			2		24		1		27
27	22111110000				2		24			26
	sum	67	80	69	57	82	83	56	42	536

Table D.5. Intuitiveness normalized weighted matrix for the robotic arm task

g	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	4	12	3	15	0	0	0	1	0	0	1	0	0	20	20
2	13	0	8	0	0	0	11	1	0	0	0	3	0	0	3
3	5	0	0	0	8	0	0	5	0	4	0	0	0	7	0
4	1	0	0	5	32	0	0	7	0	3	0	3	0	0	4
5	8	0	4	1	0	0	3	0	0	3	0	0	0	0	0
6	9	40	0	15	0	0	8	0	0	0	0	0	0	3	4
7	3	17	0	5	0	0	0	0	5	0	1	0	5	0	5
8	16	12	0	0	0	0	0	0	0	0	0	0	24	3	8
9	0	0	0	0	0	3	0	0	0	0	8	4	0	4	0
10	0	0	0	0	0	35	4	0	7	0	0	3	0	0	0
11	0	0	0	0	3	0	0	0	0	0	4	9	3	0	0
12	0	0	0	0	35	0	0	3	0	7	3	0	0	0	0
13	0	4	3	11	0	0	0	5	0	0	0	0	0	29	0
14	0	0	13	0	0	0	8	0	3	0	0	0	0	0	0
15	0	8	0	3	0	4	0	0	8	0	0	0	0	0	0
16	3	0	5	0	0	32	0	0	3	0	1	0	0	4	0
17	5	3	20	0	0	0	0	4	0	4	0	0	4	1	11
19	0	4	0	0	0	0	0	0	0	0	0	0	33	0	7
20	0	7	5	7	0	0	3	13	0	0	3	0	0	4	3
23	4	0	4	0	0	0	0	5	1	0	0	0	0	7	8
24	3	0	7	0	4	0	19	4	0	0	3	0	4	3	11
26	1	0	0	0	24	0	3	4	5	3	7	0	0	0	0
27	0	1	0	0	0	25	4	3	4	3	0	7	0	0	0

Table D.6. .Intuitiveness normalized weighted matrix for the VMR task

g	1	2	3	4	5	6	7	8
1	4	26	0	13	0	0	7	7
2	13	0	15	0	0	0	2	6
3	0	6	0	4	0	0	4	4
4	6	0	0	0	50	0	0	0
5	11	0	11	0	0	0	7	7
6	9	43	32	0	4	0	21	4
7	15	19	6	7	2	0	0	11
8	21	22	0	0	0	0	15	0
10	0	0	0	0	0	54	0	4
12	0	0	0	0	52	0	0	0
13	0	0	0	22	0	0	0	15
16	0	0	0	6	0	50	0	0
17	7	6	4	13	0	4	4	0
18	0	4	0	0	0	2	9	0
20	0	7	0	7	0	0	6	9
21	21	0	0	0	0	0	0	0
22	4	0	4	6	0	0	0	4
23	4	6	0	15	0	0	4	6
24	11	7	54	9	0	0	11	2
25	0	4	0	0	0	0	13	0
26	0	0	4	0	45	0	2	0
27	0	0	0	4	0	45	0	0

Table D.7. Agreement measures. (a) VMR task. (b) Robot task

Car Commands																			
id	gi	1	2	3	4	5	6	7	8	sum	p_i	α_i	$\alpha^{poss}{}_i$	Si	Φ	Σid(%)	Σpi (%)	Σid(%)+Σpi (%)
1	1	1	5		3			3	2	**14**	0.05	34	182	0.187	0.009	1.6949	5	6.695
2	2	3		3				1	1	8	0.0286	12	56	0.214	0.006	3.3898	7.8571	11.247
3	3		2		1			1	1	5	0.0179	2	20	0.1	0.002	5.0847	9.6429	14.728
4	4	2				9				11	0.0393	74	110	0.673	0.026	6.7797	13.571	20.351
5	5	2		2				2	2	8	0.0286	8	56	0.143	0.004	8.4746	16.429	24.903
6	6	2	9	6		1		4	2	**24**	0.0857	118	552	0.214	0.018	10.169	25	35.169
7	7	4	4	1	2	1			3	**15**	0.0536	32	210	0.152	0.008	11.864	30.357	42.222
8	8	5	4					5		**14**	0.05	52	182	0.286	0.014	13.559	35.357	48.916
9	10						**10**		1	11	0.0393	90	110	**0.818**	0.032	15.254	39.286	54.540
10	12					10				10	0.0357	90	90	1	0.036	16.949	42.857	59.806
11	13				6				4	10	0.0357	42	90	0.467	0.017	18.644	46.429	65.073
12	16				2		**9**			11	0.0393	74	110	0.673	0.026	20.339	50.357	70.696
10	12					**10**				10	0.0357	90	90	**1**	0.036	16.949	53.929	70.878
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
18	23	1	1		3			1	1	7	0.025	6	42	0.143	0.004	30.508	63.571	94.080
19	24	3	2	11	2			2	1	**21**	0.075	122	420	0.29	0.022	**32.203**	**71.071**	**103.275**
20	25		1					4		5	0.0179	12	20	0.6	0.011	33.898	72.857	106.755
21	26			1		8		1		10	0.0357	56	90	0.622	0.022	35.593	76.429	112.022
22	27				1		**8**			9	0.0321	56	72	0.778	0.025	37.288	79.643	116.931
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
59	64						1			1	0.0036	0	0	0	0	100	100	200.000
		35	35	35	35	35	35	35	35	280	1	924	2634	1	**0.338**			

(a)

Robot Commands																									
id	gi	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	sum	p_i	α_i	$\alpha^{poss}{}_i$	Si	Φ	Σid(%)	Σpi (%)	Σid(%)+Σpi (%)
1	1	1	3	1	6				1			1			6	7	**26**	0.05	108	650	0.17	0.01	0.8772	4.9524	5.830
2	2	4		2				3	1				1			2	13	0.02	22	156	0.14	0	1.7544	7.4286	9.183
3	3	2				2			2		1				2		9	0.02	8	72	0.11	0	2.6316	9.1429	11.774
4	4	1			2	8			2		1		1			1	16	0.03	60	240	0.25	0.01	3.5088	12.19	15.699
5	5	3		1	1			1			1						7	0.01	6	42	0.14	0	4.386	13.524	17.910
6	6	3	11		4			3							1	1	**23**	0.04	134	506	0.26	0.01	5.2632	17.905	23.168
7	7	1	5		2					2		1		2		2	15	0.03	28	210	0.13	0	6.1404	20.762	26.902
8	8	5	4											6	1	3	**19**	0.04	68	342	0.2	0.01	7.0175	24.381	31.398
9	9						1					3	1		2		7	0.01	8	42	0.19	0	7.8947	25.714	33.609
10	10						10	1		2			1				14	0.03	92	182	**0.51**	0.01	8.7719	28.381	37.153
11	11					1						1	4	1			7	0.01	12	42	0.29	0	9.6491	29.714	39.363
12	12					**10**			1		2	1					14	0.03	92	182	**0.51**	0.01	10.526	32.381	42.907
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
17	17	2	1	6					2		1			1	1	4	**18**	0.03	46	306	0.15	0.01	14.912	44.571	59.484
18	19		1											**10**		2	13	0.02	92	156	**0.59**	0.01	15.789	47.048	62.837
19	20		2	2	3			1	4			1			1	1	15	0.03	22	210	0.1	0	16.667	49.905	66.571
20	23	1		1					3	1					2	3	11	0.02	14	110	0.13	0	17.544	52	69.544
21	24	1		2		1		6	1			1		1	1	5	**19**	0.04	52	342	0.15	0.01	18.421	55.619	74.040
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
33	37				1					1	1	1	1				5	0.01	0	20	0	0	**28.947**	**71.238**	**100.185**
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
114	118				1												1	0	0	0	0	0	100	100	200.000
		35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	525	1	1174	5202	1	**0.18**			

(b)

Table D.8. Robot task intuitiveness complete matrix

re-index	Gesture	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	sum
1	00000000000	1	3	1	6				1			1			6	7	26
2	00111110000	3	11		4			3							1	1	23
3	00111111111	5	4											6	1	3	19
4	20111110000	1		2		1		6	1			1		1	1	5	19
5	10111110000	2	1	6					2		1			1	1	4	18
6	10000000000		1	1	4				2						9		17
7	00100001000	1			2	8			2		1		1			1	16
8	00111111000	1	5		2					2		1		2		2	15
9	10100001000	1		2			8			1		1			2		15
10	20000000000		2	2	3			1	4			1			1	1	15
11	21111110000	1				7		1	1	2	1	2					15
12	22111110000		1				8	1	1	1	1		2				15
13	01111110000						10	1		2			1				14
14	02111110000					10			1		2	1					14
15	00010000000	4		2				3	1				1			2	13
16	10111111111		1											10		2	13
17	20100000000	1		1					3	1					2	3	11
18	00100000000	2				2			2		1				2		9
19	00110000000	3		1	1			1			1						7
20	01000000000						1					3	1		2		7
21	02000000000					1						1	4	1			7
22	10010000000			4				2		1							7
23	10100000000		3		1		1			2							7
24	10111111000			2							1		1	2			6
25	20111112000				2			1	1	1			1				6
26	00011000000							1	1		1	1			1		5
27	10011000000								1		1		1	1	1		5
28	10110001000	1		2						1						1	5
29	12111110000						1				1		2	1			5
30	20010000000	2						2					1				5
31	20011110000	2							1				2				5
32	21000000000			1						1	1	1	1				5
33	22000000000				1					1	1	1	1				5
34	00000010000		1				1		1			1					4
35	00001110000	1	1		1					1							4
36	11111110000					1				1		2					4
37	20100002000			1					1		1					1	4
38	20110000000	1						2				1					4
39	21100002000				1					1	1		1				4
40	00011000100					1				1				1			3
41	00011110000							1			1				1		3
42	00101110000				2			1									3
43	00110001000				1			1			1						3
44	00111110111			1							1			1			3
45	01111111111											2	1				3
46	02111111111											1	2				3
47	10011110000								1	1				1			3
48	10101111000			1					1	1							3
49	11100001000									1		1	1				3
50	12000000000							1				1			1		3
51	12100001000							1			1	1					3
52	21111112000									2		1					3
53	22100002000										1	1	1				3
54	22111112000										2			1			3
55	00110011000											1				1	2
56	00111000000				1								1				2
57	00111111001									1	1						2

58	00111111100									1	1							2
59	01010000000							1					1					2
60	01100000000											2						2
61	01111111000									1				1				2
62	02010000000								1		1							2
63	02100000000												2					2
64	10110000000			2														2
65	10111000000							1				1						2
66	11000000000								1				1					2
67	11111111000					1					1							2
68	12111111000						1					1						2
69	21010000000					1								1				2
70	21011110000										1	1						2
71	21100000000					1					1							2
72	21110000000							1	1									2
73	22010000000						1									1		2
74	22011110000									1			1					2
75	22100000000						1			1								2
76	22110000000							1	1									2
77	00001000000			1														1
78	00001010000													1				1
79	00001100000															1		1
80	00001100010						1											1
81	00100010000										1							1
82	00101111000										1							1
83	00111001000			1														1
84	00111100000															1		1
85	00111101110													1				1
86	00111110001										1							1
87	00111110011																1	1
88	00111110100													1				1
89	01011000100										1							1
90	01011110011									1								1
91	01100001000	1																1
92	01110001000							1										1
93	01111111001											1						1
94	02011110011										1							1
95	02100001000								1									1
96	02111000100									1								1
97	02111001000				1													1
98	02111111000												1					1
99	02111111001												1					1
100	10001000000				1													1
101	10011000100													1				1
102	10100010000									1								1
103	10101110000			1														1
104	10110011000												1					1
105	10111110111									1								1
106	11010000000									1								1
107	11110000000					1												1
108	11110001000								1									1
109	11111111111	1																1
110	12010000000											1						1
111	12110000000						1											1
112	12111111111		1															1
113	21001110000										1							1
114	21101110000				1													1
		35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	525

Table D.9. VMR task intuitiveness complete matrix

re-index	Gesture	1	2	3	4	5	6	7	8	sum
1	00111110000	2	9	6		1		4	2	24
2	20111110000	3	2	11	2			2	1	21
3	00111111000	4	4	1	2	1			3	15
4	00000000000	1	5		3			3	2	14
5	00111111111	5	4					5		14
6	00100001000	2				9				11
7	01111110000						10		1	11
8	10100001000				2		9			11
9	02111110000					10				10
10	10000000000				6				4	10
11	21111110000			1		8		1		10
12	10111110000	2	1	1	3		1	1		9
13	22111110000				1		8			9
14	00010000000	3		3				1	1	8
15	00110000000	2		2				2	2	8
16	20000000000		2		2			1	3	8
17	20100000000	1	1		3			1	1	7
18	00100000000		2		1			1	1	5
19	20011110000	1		1	2				1	5
20	20111112000		1					4		5
21	10111111000		1				1	2		4
22	20010000000	4								4
23	00111000000			1				2		3
24	10010000000	1			1			1		3
25	10011110000			1	1				1	3
26	20100002000				1				2	3
27	20110000000	1		1					1	3
28	00011000000		1					1		2
29	00011110000			1	1					2
30	00110001000			2						2
31	00110010000								2	2
32	02100001000				2					2
33	10100000000				1				1	2
34	10111000000				1				1	2
35	20101112000	1							1	2
36	21110000000					1		1		2
37	22110000000						1		1	2
38	00001110000	1								1
39	00010010000								1	1
40	00101110000			1						1
41	01000000000								1	1
42	01010000000						1			1
43	01100001000	1								1
44	01111111000							1		1
45	02010000000					1				1
46	02111111000		1							1
47	10010010000		1							1
48	10011000100			1						1
49	10110001000							1		1
50	10111111111								1	1
51	11110000000					1				1
52	11111110000					1				1
53	12100001000			1						1
54	12110000000						1			1
55	12111110000						1			1
56	21010000000					1				1
57	21011110000					1				1
58	22010000000						1			1
59	22011110000						1			1
		35	35	35	35	35	35	35	35	280

Table D.10. Complementary intuitiveness matrix (robotic arm)

g1	g2	start/stop	up/down	left/right	fwrd/bwrd	w-up/w-down	wrist cw/ccw	open/close	**sum**
1	8	1							1
1	11						1		1
1	13		1						1
2	1	1	1						2
2	6	3							3
2	13				1				1
3	15	2		1					3
4	16			8					8
5	3				1				1
5	7	3							3
6	1	1							1
6	17	1			2				3
7	3							1	1
7	15	1							1
7	23							1	1
8	1	1						3	4
8	6	3							3
8	13							3	3
8	19	1							1
9	11						3		3
10	12				1	2			3
11	9			1			1	1	3
12	10			10			1		11
13	1		1						1
14	1		1						1
14	2				1				1
14	13		2						2
14	24				1				1
15	3					1			1
15	17					1			1
16	4		2			1	1		4
16	6	1							1
17	6	1	4						5
17	13	1	1					1	3
17	15		1						1
19	1							2	2
19	8							1	1
19	13							4	4
19	16							1	1
19	17							1	1
19	20							1	1
20	1		1						1
20	23				1				1
23	7	1							1
23	20		1						1
24	6	1							1
24	20				3				3
24	23				1			1	2
24	27			1					1
26	27	1		7	1	1	2		12
27	26				1	1			2
	sum	24	16	28	14	7	9	21	119

Table D.11. Complementary intuitiveness matrix (VMR)

g1	g2	start/stop	frwd/bkwd	left/right	fast/slow	**sum**
1	6				1	1
1	8	1				1
1	13				1	1
2	1		1		1	2
2	6	1				1
2	7	1				1
4	3	1				1
4	7	1				1
4	16			9		9
5	1	1				1
5	3		1			1
5	8	1				1
6	1	1	1			2
6	2				1	1
6	7		2			2
6	8	1				1
6	10				1	1
6	17		3	1		4
7	6	2				2
7	17	1				1
7	18	1		1		2
7	24		1			1
8	1	3				3
8	5				1	1
8	6	1			1	2
8	7				1	1
8	22				1	1
8	24	1				1
12	10			10		10
17	6	2				2
17	13				1	1
18	7				2	2
20	1				1	1
21	6	1				1
21	7	2				2
21	20	1				1
22	8	1				1
22	20		1			1
23	3	1				1
23	20				1	1
24	6	1				1
24	13		2			2
24	16		2			2
24	20		1		1	2
24	22		2			2
24	23	1	3		1	5
24	25	1				1
25	20				1	1
26	24				1	1
26	27		1	8		9
sum		29	21	29	18	97

Table D.12. Complementary gesture-commands weighted intuitiveness matrix: (a) Robot task, (b) VMR task

g1	g2	start/stop	up/down	left/right	fwrd/back	w. up/down	w. cw/ccw	open/close	sum
1	8	6							6
1	11						2		2
1	13		4						4
2	1	6	5						11
2	6	15							15
2	13				4				4
3	15	8		6					14
4	16			48					48
5	3				4				4
5	7	14							14
6	1	6							6
6	17	4			7				11
7	3							4	4
7	15	4							4
7	23							4	4
8	1	5						18	23
8	6	16							16
8	13							14	14
8	19	6							6
9	11						12		12
10	12				5	10			15
11	9			4			6	4	14
12	10			52			4		56
13	1		4						4
14	1		3						3
14	2				4				4
14	13		9						9
14	24				6				6
15	3					6			6
15	17					6			6
16	4		8			4	3		15
16	6	5							5
17	6	4	22						26
17	13	5	4					6	15
17	15		4						4
19	1							8	8
19	8							4	4
19	13							22	22
19	16							4	4
19	17							4	4
19	20							6	6
20	1		4						4
20	23				4				4
23	7	5							5
23	20		6						6
24	6	4							4
24	20				14				14
24	23				2			6	8
24	27			5					5
26	27	2		35	4	4	10		55
27	26				6	5			11
	sum	115	73	150	60	35	37	104	574

(a)

g1	g2	start/stop	frwrd/back	left/right	fast/slow	**sum**
1	6				2	2
1	8	5				5
1	13				2	2
2	1		5		3	8
2	6	5				5
2	7	5				5
4	3	3				3
4	7	4				4
4	16			54		54
5	1	6				6
5	3		5			5
5	8	6				6
6	1	4	6			10
6	2				6	6
6	7		10			10
6	8	6				6
6	10				4	4
6	17		15	4		19
7	6	9				9
7	17	5				5
7	18	4		2		6
7	24		6			6
8	1	17				17
8	5				3	3
8	6	5			2	7
8	7				2	2
8	22				4	4
8	24	3				3
12	10			57		57
17	6	8				8
17	13				4	4
18	7				10	10
20	1				5	5
21	6	6				6
21	7	11				11
21	20	5				5
22	8	5				5
22	20		4			4
23	3	4				4
23	20				4	4
24	6	4				4
24	13		10			10
24	16		8			8
24	20		5		4	9
24	22		7			7
24	23	5	16		6	27
24	25	4				4
25	20				4	4
26	24				2	2
26	27		4	48		52
	sum	139	101	165	67	472

(b)

Table D.13. Complementary intuitiveness normalized weighted matrix for the robotic arm task

g1	g2	1	2	3	4	5	6	7
1	8	10	0	0	0	0	0	0
1	11	0	0	0	0	0	3	0
1	13	0	7	0	0	0	0	0
2	1	10	9	0	0	0	0	0
2	6	26	0	0	0	0	0	0
2	13	0	0	0	7	0	0	0
3	15	14	0	10	0	0	0	0
4	16	0	0	84	0	0	0	0
5	3	0	0	0	7	0	0	0
5	7	24	0	0	0	0	0	0
6	1	10	0	0	0	0	0	0
6	17	7	0	0	12	0	0	0
7	3	0	0	0	0	0	0	7
7	15	7	0	0	0	0	0	0
7	23	0	0	0	0	0	0	7
8	1	9	0	0	0	0	0	31
8	6	28	0	0	0	0	0	0
8	13	0	0	0	0	0	0	24
8	19	10	0	0	0	0	0	0
9	11	0	0	0	0	0	21	0
10	12	0	0	0	9	17	0	0
11	9	0	0	7	0	0	10	7
12	10	0	0	91	0	0	7	0
13	1	0	7	0	0	0	0	0
14	1	0	5	0	0	0	0	0
14	2	0	0	0	7	0	0	0
14	13	0	16	0	0	0	0	0
14	24	0	0	0	10	0	0	0
15	3	0	0	0	0	10	0	0
15	17	0	0	0	0	10	0	0
16	4	0	14	0	0	7	5	0
16	6	9	0	0	0	0	0	0
17	6	7	38	0	0	0	0	0
17	13	9	7	0	0	0	0	10
17	15	0	7	0	0	0	0	0
19	1	0	0	0	0	0	0	14
19	8	0	0	0	0	0	0	7
19	13	0	0	0	0	0	0	38
19	16	0	0	0	0	0	0	7
19	17	0	0	0	0	0	0	7
19	20	0	0	0	0	0	0	10
20	1	0	7	0	0	0	0	0
20	23	0	0	0	7	0	0	0
23	7	9	0	0	0	0	0	0
23	20	0	10	0	0	0	0	0
24	6	7	0	0	0	0	0	0
24	20	0	0	0	24	0	0	0
24	23	0	0	0	3	0	0	10
24	27	0	0	9	0	0	0	0
26	27	3	0	61	7	7	17	0
27	26	0	0	0	10	9	0	0

Table D.14. Complementary intuitiveness normalized weighted matrix for the VMR task

g1	g2	1	2	3	4
1	6	0	0	0	4
1	8	11	0	0	0
1	13	0	0	0	4
2	1	0	11	0	6
2	6	11	0	0	0
2	7	11	0	0	0
4	3	6	0	0	0
4	7	8	0	0	0
4	16	0	0	114	0
5	1	13	0	0	0
5	3	0	11	0	0
5	8	13	0	0	0
6	1	8	13	0	0
6	2	0	0	0	13
6	7	0	21	0	0
6	8	13	0	0	0
6	10	0	0	0	8
6	17	0	32	8	0
7	6	19	0	0	0
7	17	11	0	0	0
7	18	8	0	4	0
7	24	0	13	0	0
8	1	36	0	0	0
8	5	0	0	0	6
8	6	11	0	0	4
8	7	0	0	0	4
8	22	0	0	0	8
8	24	6	0	0	0
12	10	0	0	121	0
17	6	17	0	0	0
17	13	0	0	0	8
18	7	0	0	0	21
20	1	0	0	0	11
21	6	13	0	0	0
21	7	23	0	0	0
21	20	11	0	0	0
22	8	11	0	0	0
22	20	0	8	0	0
23	3	8	0	0	0
23	20	0	0	0	8
24	6	8	0	0	0
24	13	0	21	0	0
24	16	0	17	0	0
24	20	0	11	0	8
24	22	0	15	0	0
24	23	11	34	0	13
24	25	8	0	0	0
25	20	0	0	0	8
26	24	0	0	0	4
26	27	0	8	102	0

Table D.15. Stress normalized matrix for the robot and VMR tasks

Gest	00000000000	00010000000	00100000000	00100001000	00110000000	00111110000	00111111000	00111111111	01000000000	01111110000	02000000000	02111110000	10000000000	10010000000	10100000000	10100001000	10111110000	10111111000	10111111111	20000000000	20010000000	20011110000	20100000000	20111110000	20111112000	21111110000	22111110000
00000000000	111	132	123	118	142	88	96	116	154	162	163	178	134	154	155	146	128	136	119	123	153	128	127	110	145	142	155
00010000000	113	135	125	120	144	90	98	118	156	164	165	180	136	156	157	148	131	139	121	125	155	131	129	112	147	144	157
00100000000	112	134	125	119	143	89	97	118	155	163	165	179	135	155	156	147	130	138	120	124	154	130	128	111	146	143	156
00100001000	112	133	124	119	142	89	97	117	155	163	164	179	134	155	156	147	129	137	120	124	153	129	128	110	145	142	156
00110000000	114	136	126	121	146	91	99	119	157	165	166	181	137	157	158	149	131	140	122	126	156	131	130	113	148	145	158
00111110000	109	130	121	115	140	86	94	114	152	160	161	176	131	152	153	144	126	134	117	121	150	126	125	107	142	140	153
00111111000	109	131	122	116	140	87	95	115	152	160	162	177	132	152	154	144	127	135	117	122	151	127	126	108	143	140	154
00111111111	111	133	124	118	142	89	97	117	154	163	164	179	134	154	156	146	129	137	120	124	153	129	128	110	145	142	156
01000000000	115	137	127	122	146	92	100	121	159	166	168	182	138	158	160	150	133	141	123	127	157	133	131	114	149	146	160
01111110000	116	138	128	123	147	93	101	121	159	168	168	183	139	159	160	151	134	142	124	128	158	134	132	115	150	147	160
02000000000	116	138	128	123	147	93	101	122	159	167	169	183	139	159	161	151	134	142	124	128	158	134	132	115	150	147	161
02111110000	118	139	130	124	149	95	103	123	161	169	170	186	141	161	162	153	135	143	126	130	159	135	134	116	151	149	162
10000000000	113	135	125	120	144	90	98	119	156	164	166	180	137	156	158	148	131	139	121	125	155	131	129	112	147	144	158
10010000000	115	137	127	122	146	92	100	121	158	166	168	182	138	159	160	150	133	141	123	127	157	133	131	114	149	146	160
10100000000	115	137	127	122	146	93	101	121	158	166	168	183	138	158	160	150	133	141	123	127	157	133	132	114	149	146	160
10100001000	114	136	127	121	145	92	100	120	157	165	167	182	137	157	159	150	132	140	122	127	156	132	131	113	148	145	159
10111110000	113	134	125	119	144	90	98	118	156	164	165	180	136	156	157	148	131	138	121	125	154	130	129	111	146	144	157
10111111000	113	135	126	120	144	91	99	119	156	165	166	181	136	156	158	148	131	140	122	126	155	131	130	112	147	144	158
10111111111	112	133	124	118	143	89	97	117	155	163	164	179	135	155	156	147	129	137	120	124	153	129	128	110	145	143	156
20000000000	112	134	124	119	143	89	97	118	155	163	165	179	135	155	156	147	130	138	120	125	154	130	128	111	146	143	156
20010000000	115	137	127	122	146	92	100	120	158	166	168	182	138	158	159	150	133	141	123	127	157	133	131	114	149	146	159
20011110000	113	134	125	119	144	90	98	118	156	164	165	180	136	156	157	148	130	138	121	125	154	131	129	111	146	144	157
20100000000	113	134	125	119	143	90	98	118	156	164	165	180	135	156	157	147	130	138	121	125	154	130	129	111	146	143	157
20111110000	111	132	123	118	142	88	96	116	154	162	163	178	134	154	155	146	128	136	119	123	152	128	127	110	144	142	155
20111112000	114	136	126	121	145	91	100	120	157	165	167	181	137	157	159	149	132	140	122	126	156	132	130	113	148	145	159
21111110000	114	136	126	121	145	91	99	119	157	165	166	181	137	157	158	149	131	140	122	126	156	131	130	113	148	146	158
22111110000	115	137	127	122	146	93	101	121	158	166	168	183	138	158	160	150	133	141	123	127	157	133	132	114	149	146	160

Table D.16. Average and std dev static stress values for 19 subjects

gi	Code	AVG	STD
1	00000000000	2.586207	1.592779
2	00010000000	3.137931	1.457104
3	00100000000	2.896552	1.113066
4	00100001000	2.758621	1.353703
5	00110000000	3.37931	1.236752
6	00111110000	2	1.28174
7	00111111000	2.206897	1.145778
8	00111111111	2.724138	1.532891
9	01000000000	3.689655	1.853634
10	01111110000	3.896552	1.472239
11	02000000000	3.931034	1.486391
12	02111110000	4.310345	1.794902
13	10000000000	3.172414	1.465532
14	10010000000	3.689655	1.441811
15	10100000000	3.724138	1.306483
16	10100001000	3.482759	1.66091
17	10111110000	3.034483	1.451176
18	10111111000	3.241379	1.479748
19	10111111111	2.793103	1.544097
20	20000000000	2.896552	1.739146
21	20010000000	3.655172	1.758162
22	20011110000	3.034483	1.475581
23	20100000000	3	1.558387
24	20111110000	2.551724	1.297971
25	20111112000	3.448276	1.616571
26	21111110000	3.37931	1.760961
27	22111110000	3.724138	1.386067
28	02110101000	8.137931	2.26
29	12001010000	7.034483	2.16

Table D.17. Subset 1 for the transition stress experiment

		1	7	25	27	28	29
	Gi \ Gj	00000000000	00111111000	20111112000	22111110000	02110101000	12001010000
1	00000000000	X	2.17	2.83	2.33	8.5	6.58
7	00111111000	1.917	X	2.33	2.67	8.17	5.92
25	20111112000	2.167	2.5	X	2.67	7.08	7
27	22111110000	2.25	2.17	2.83	X	8.17	6.58
28	02110101000	2.833	3.08	3.67	4.17	X	7.92
29	12001010000	2.417	3.17	3.5	3.67	8.25	X

Table D.18. Subset 2 for the transition stress experiment

	Gi \ Gj	00100001000	00111110000	00111111111	01111110000	10100001000	22111110000
		4	6	8	10	16	27
4	00100001000	X	3.14	2.57	4	3	3.86
6	00111110000	2.286	X	2	3.71	3.57	3.71
8	00111111111	1.571	2.29	X	4.57	3.86	2.43
10	01111110000	3.857	1.57	3	X	4.57	3
16	10100001000	1.857	3.57	3.71	5	X	3.86
27	22111110000	3.143	2	2.57	4	3.71	X

Table D.19. Subset 3 for the validation of the transition stress experiment

	Gi \ Gj	00011100110	01111111000	02111111000	10111001100	12111001000	20101112000
		30	31	32	33	34	35
30	00011100110	X	3	3.43	3.43	5.43	4
31	01111111000	2.143	X	3.14	3.86	4	3
32	02111111000	3.143	2	X	3.57	4.86	4
33	10111001100	2.857	3.43	3.86	X	4.86	4.43
34	12111001000	2.714	3.29	4.29	3.29	X	4
35	20101112000	2.714	2.86	3.43	4.71	5.43	X

Table D.20. Duration normalized matrix for the robot and VMR tasks

Gest	00000000000	00010000000	00100000000	00100001000	00110000000	00111110000	00111111000	00111111111	01000000000	01111110000	02000000000	02111110000	10000000000	10010000000	10100000000	10100001000	10111110000	10111111000	10111111111	20000000000	20010000000	20011110000	20100000000	20111110000	20111112000	21111110000	22111110000
00000000000	1	137	128	122	147	92	100	121	160	168	169	185	139	160	161	151	133	142	124	128	158	133	132	114	150	147	161
00010000000	118	1	130	124	149	94	102	123	162	170	172	187	141	162	163	154	136	144	126	130	161	136	134	116	152	149	163
00100000000	117	139	1	123	148	93	101	122	161	169	171	186	140	161	162	153	135	143	125	129	160	135	133	115	151	148	162
00100001000	116	138	128	1	148	92	101	121	160	169	170	185	140	160	162	152	134	142	124	128	159	134	133	115	151	148	162
00110000000	119	141	131	126	1	95	103	124	163	171	173	188	142	163	164	155	137	145	127	131	162	137	135	117	153	151	164
00111110000	113	135	125	120	145	1	97	118	157	165	167	182	136	157	158	149	131	139	121	125	156	131	129	111	147	145	158
00111111000	114	136	126	120	145	90	1	119	158	166	168	183	137	158	159	150	132	140	122	126	157	132	130	112	148	145	159
00111111111	116	138	128	123	148	92	101	1	160	168	170	185	139	160	162	152	134	142	124	128	159	134	132	114	150	148	162
01000000000	120	142	132	127	152	96	105	125	1	173	174	189	144	164	166	156	138	146	128	132	163	138	137	119	155	152	166
01111110000	121	143	133	128	153	97	106	126	165	1	175	190	144	165	167	157	139	147	129	133	164	139	137	119	156	153	167
02000000000	121	143	133	128	153	97	106	127	165	174	1	190	145	165	167	157	139	147	129	133	164	139	138	120	156	153	167
02111110000	123	145	135	130	155	99	107	128	167	175	177	1	146	167	168	159	141	149	131	135	166	141	139	121	157	155	168
10000000000	118	140	130	125	150	94	102	123	162	170	172	187	1	162	163	154	136	144	126	130	161	136	134	116	152	150	163
10010000000	120	142	132	127	152	96	105	125	164	173	174	189	144	1	166	156	138	146	128	132	163	138	137	119	155	152	166
10100000000	120	142	133	127	152	96	105	126	164	173	174	189	144	164	1	156	138	146	128	133	163	138	137	119	155	152	166
10100001000	119	141	132	126	151	95	104	125	163	172	173	188	143	163	165	1	137	145	127	132	162	137	136	118	154	151	165
10111110000	117	139	130	124	149	94	102	123	162	170	171	186	141	162	163	153	1	143	125	130	160	135	134	116	152	149	163
10111111000	118	140	130	125	150	94	103	124	162	171	172	187	142	162	164	154	136	1	126	130	161	136	135	117	153	150	164
10111111111	116	138	129	123	148	92	101	122	160	169	170	185	140	160	162	152	134	142	1	129	159	134	133	115	151	148	162
20000000000	117	139	129	123	148	93	101	122	161	169	171	186	140	161	162	153	135	143	125	1	160	135	133	115	151	148	162
20010000000	120	142	132	127	152	96	105	125	164	173	174	189	143	164	166	156	138	146	128	132	1	138	136	118	154	152	166
20011110000	117	139	130	124	149	94	102	123	162	170	171	186	141	162	163	153	135	143	125	130	160	1	134	116	152	149	163
20100000000	117	139	129	124	149	93	102	123	161	170	171	186	141	161	163	153	135	143	125	129	160	135	1	116	152	149	163
20111110000	115	137	128	122	147	91	100	121	159	168	169	184	139	159	161	151	133	141	123	128	158	133	132	1	150	147	161
20111112000	119	141	131	126	151	95	104	124	163	172	173	188	142	163	165	155	137	145	127	131	162	137	136	118	1	151	165
21111110000	119	141	131	126	151	95	103	124	163	171	173	188	142	163	164	155	137	145	127	131	162	137	135	117	153	1	164
22111110000	120	142	133	127	152	96	105	126	164	173	174	189	144	164	166	156	138	146	128	133	163	138	137	119	155	152	1

Table D.21 The frequency matrix for the robotic arm task with the 'rest' command

	0 (rest)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 (r)	42761	0	30	125	231	86	80	96	98	89	22	97	3	72	35	26
1	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	96	0	0	3560	0	0	0	0	28	0	0	0	0	0	0	4
4	212	0	0	1	5150	3	0	1	0	7	1	2	0	4	8	0
5	88	0	0	0	0	3110	0	1	1	0	0	0	0	0	0	0
6	91	0	0	0	1	0	6075	0	20	0	0	0	0	0	0	0
7	98	0	0	0	0	0	0	371	0	0	0	0	0	0	0	0
8	109	0	0	2	3	1	32	0	3182	0	0	0	0	0	0	0
9	96	0	0	0	0	0	0	0	0	208	0	0	0	0	0	0
10	22	0	0	0	0	0	0	0	0	0	7	0	0	0	1	0
11	95	0	0	0	4	0	0	0	0	0	0	1696	0	0	0	0
12	3	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0
13	76	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0
14	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table D.22. The frequency matrix for the VMR task with the 'rest' command

	0 (rest)	1	2	3	4	5	6	7	8
0 (r)	101462	0	23	986	61	336	367	24	20
1	30	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	1058	0	7	11772	13	294	177	25	0
4	64	0	0	0	1469	0	24	1	0
5	249	0	0	370	0	5290	0	1	10
6	329	0	0	215	15	0	5401	9	0
7	57	0	0	3	0	0	0	0	0
8	30	0	0	0	0	0	0	0	0

Table D.23. The frequency matrix for the robotic arm task

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	3624	1	1	3	0	29	0	0	0	0	1	0	29
4	0	0	2	5270	13	1	1	3	14	3	13	1	34	34	0
5	0	0	0	9	3159	4	14	12	0	1	0	0	1	0	0
6	0	0	0	20	3	6134	0	29	1	0	0	0	0	0	0
7	0	0	0	13	18	1	424	1	7	2	1	0	2	0	0
8	0	0	3	19	4	44	0	3255	0	1	1	0	2	0	0
9	0	0	0	15	1	0	0	0	272	4	5	0	7	0	0
10	0	0	0	4	0	0	0	0	9	15	0	0	1	1	0
11	0	0	0	29	1	0	0	0	0	1	1752	0	11	1	0
12	0	0	0	0	0	0	0	0	1	1	0	17	0	0	0
13	0	0	27	9	0	0	0	0	0	2	22	1	24	8	1
14	0	0	32	0	0	0	0	0	0	0	1	0	11	0	0
15	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0

Table D.24. The frequency matrix for the VMR task

	1	2	3	4	5	6	7	8
1	0	0	30	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	29	12460	30	475	313	32	7
4	0	0	7	1505	1	44	1	0
5	0	1	456	1	5436	4	1	21
6	0	0	314	19	1	5607	26	2
7	0	0	58	1	0	1	0	0
8	0	0	21	2	7	0	0	0

Table D.25. Normalized frequency matrix for the robotic arm task

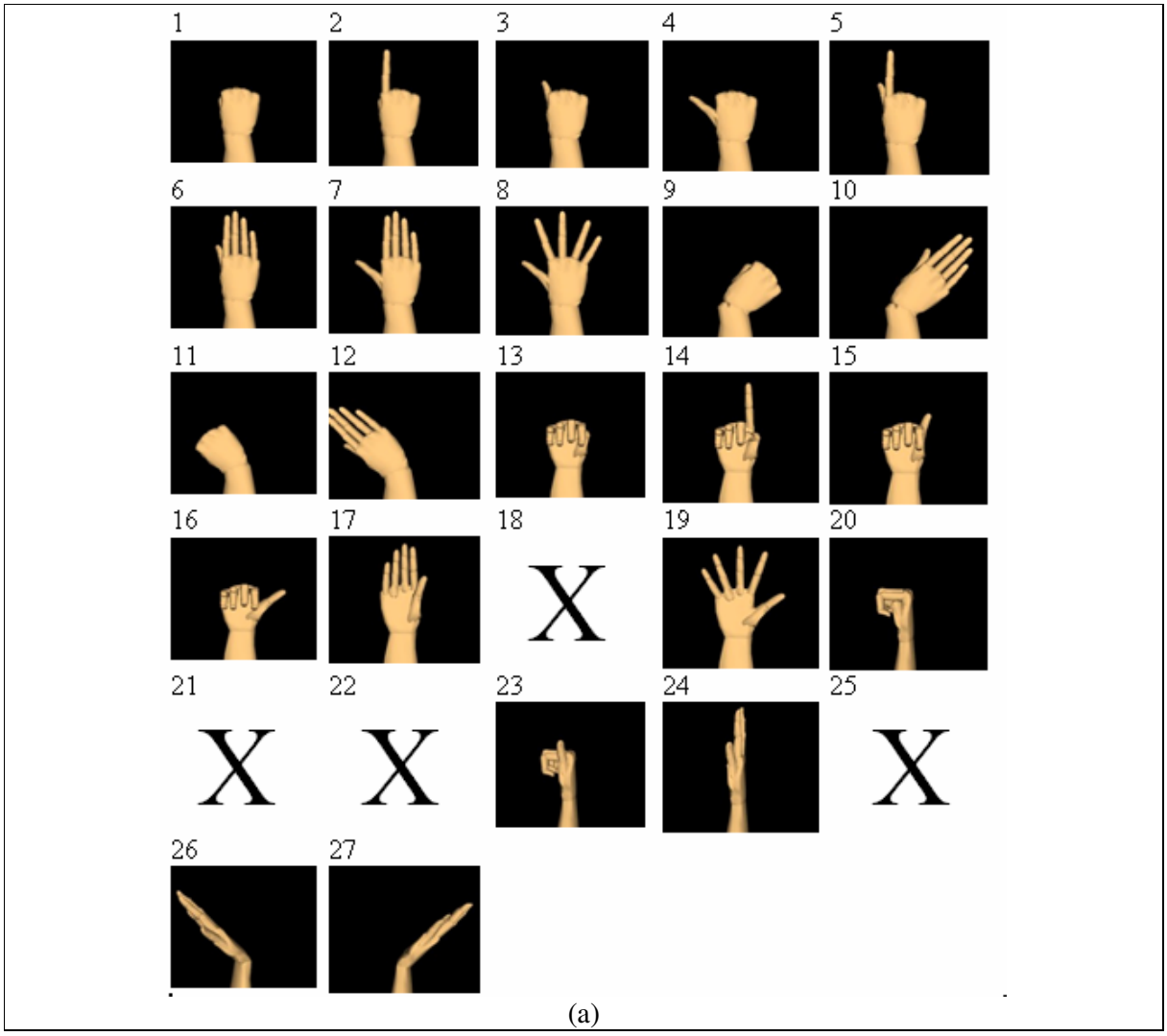
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	147	0	0	0	0	1	0	0	0	0	0	0	1
4	0	0	0	214	1	0	0	0	1	0	1	0	1	1	0
5	0	0	0	0	128	0	1	0	0	0	0	0	0	0	0
6	0	0	0	1	0	249	0	1	0	0	0	0	0	0	0
7	0	0	0	1	1	0	17	0	0	0	0	0	0	0	0
8	0	0	0	1	0	2	0	132	0	0	0	0	0	0	0
9	0	0	0	1	0	0	0	0	11	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	0	71	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
13	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0
14	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table D.26. Normalized frequency matrix for the VMR task

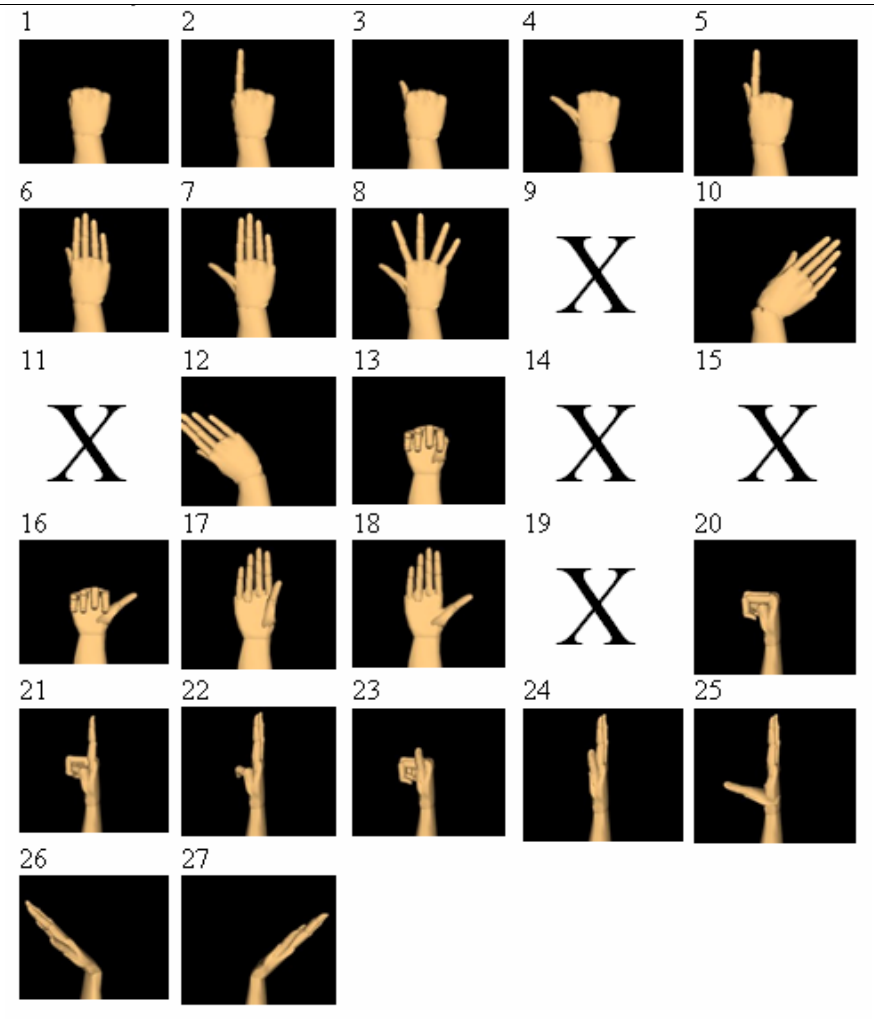
	1	2	3	4	5	6	7	8
1	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	1	463	1	18	12	1	0
4	0	0	0	56	0	2	0	0
5	0	0	17	0	202	0	0	1
6	0	0	12	1	0	208	1	0
7	0	0	2	0	0	0	0	0
8	0	0	1	0	0	0	0	0

Appendix E. Gesture master sets

This appendix presents the master set gestures images used for the robotic arm and VMR tasks. The combined set of gestures is presented in the last image.



(a)



(b)

Figure E.1 Gestures master set. (a) Robot task vocabulary. (b) VMR task vocabulary

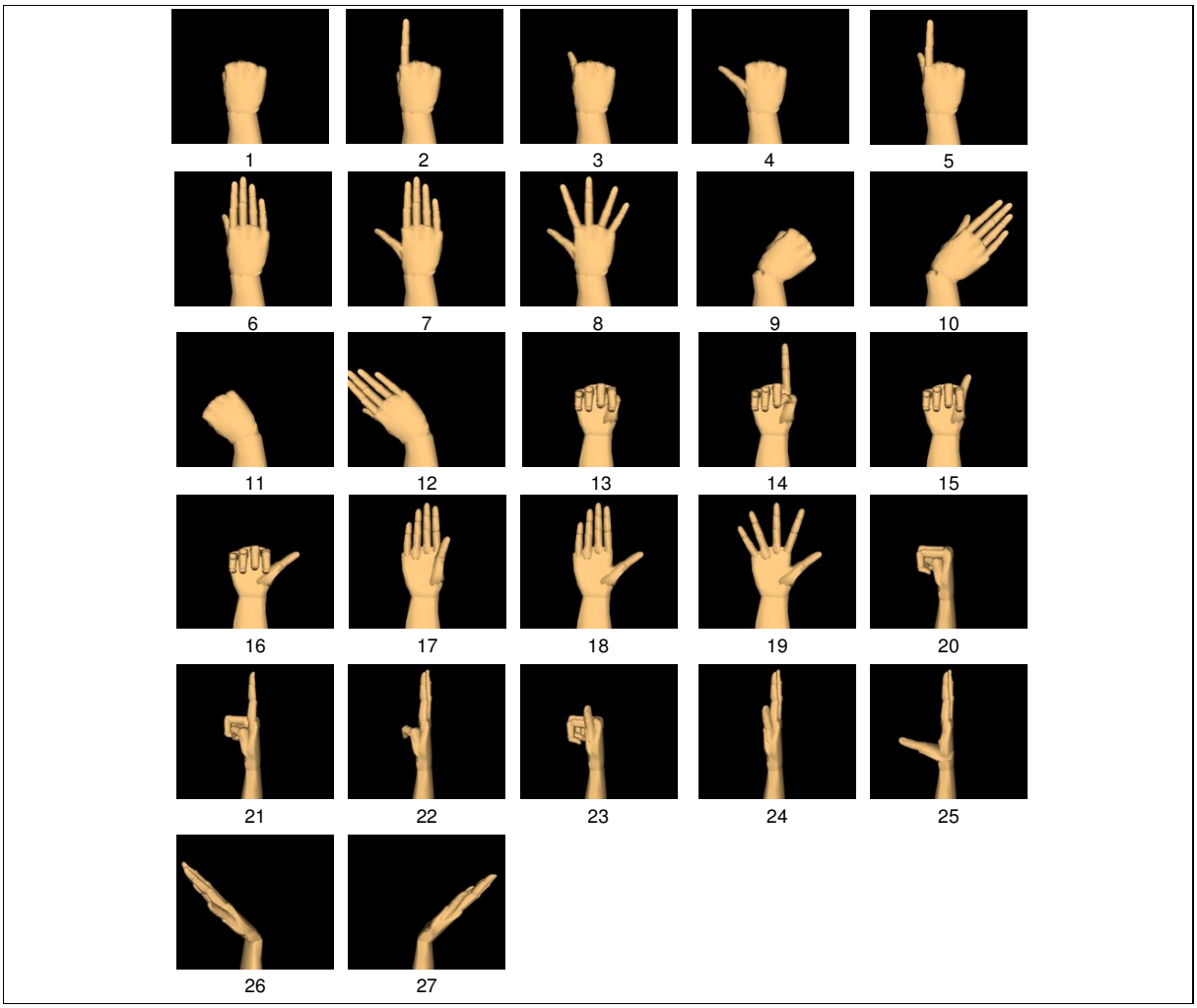
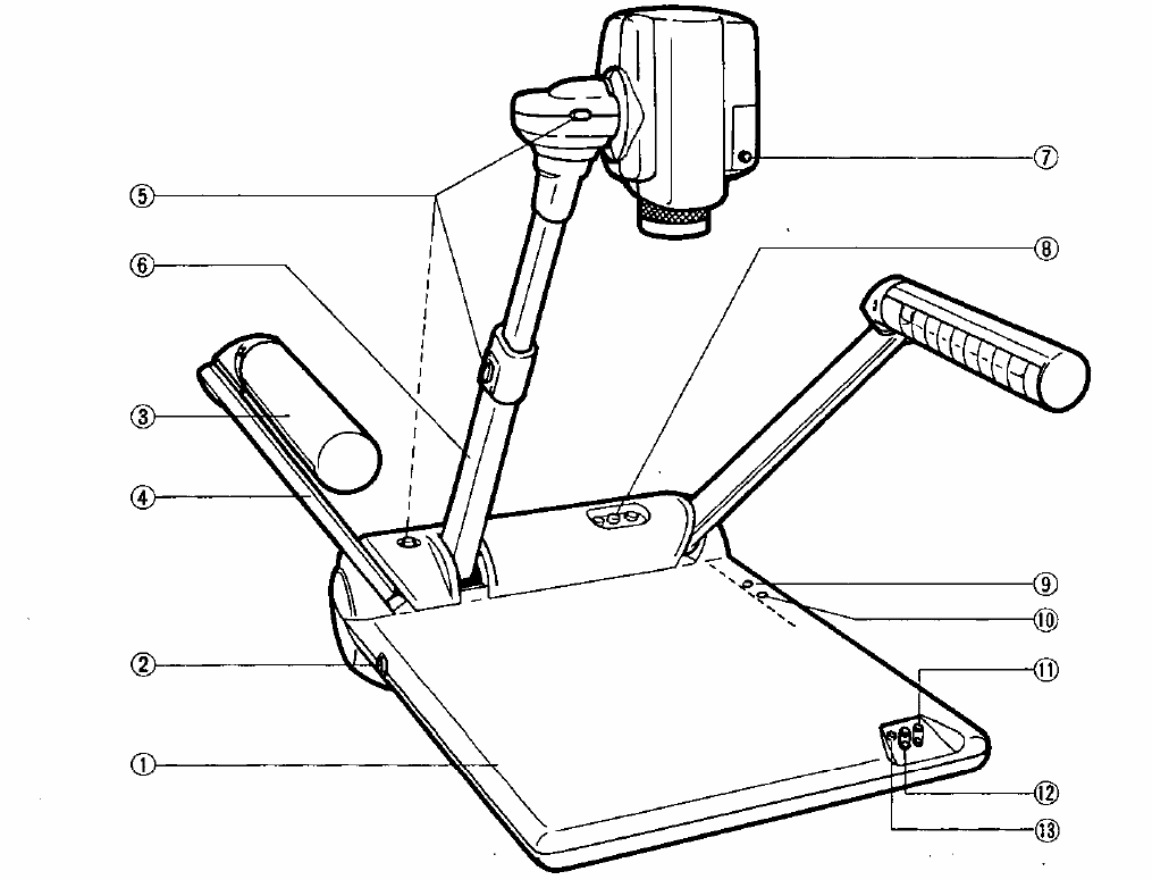
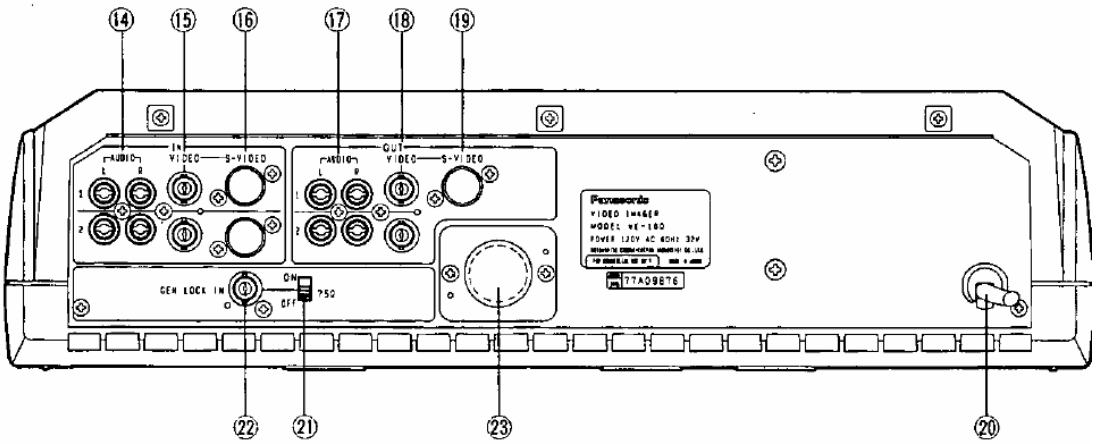


Figure E.2. Combined gestures master set

Appendix F. Panasonic WE-160 image viewer

This appendix shows the Panasonic WE-160 Image Viewer major operating controls and their function, from the original operating instructions manual.





1. **Stage**
   Place the document or the object on here.

2. **Power ON/OFF Switch (POWER, ON/OFF)**

3. **Lighting case**
   Two fluorescent lamps on both sides light up when the Light Selection Switch (10) is turned on.

4. **Lighting Arm**

5. **Lock Release button**

6. **Camera Arm**

7. **Auto White Balance Control Button (AWC)**
   This button is used to set the White Balance of the camera. Press the button shooting the white object by the camera.

8. **Input Selection Switch (INPUT SELECT, VIDEO IMAGER, A/V 1, A/V 2)**
   This switch is used to select the audio/Video signal from three source signals.
   When VIDEO IMAGER button is selected, the camera signal of the WE-160 is provided to the video output connector (18, 19).
   When either A/V 1 or A/V 2 button is selected, the external audio/video signal is provided to the audio output jack (17) and the video output connectors (18, 19).

9. **Backlight Connector (BACKLIGHT)**
   The power for the optional Backlight Unit WE-163 is supplied form this connector.

10. **Light Selection Switch (LIGHT SELECT, ARM, BACKLIGHT)**
    This switch is used to select Arm Light or Backlight.

11. **Focus Adjusting Switch (FOCUS, FAR, NEAR)**
    This switch is used to adjust the focus of the camera. Press FAR or NEAR to adjust the focus of video.

12. **Zoom Adjusting Switch (ZOOM, IN, OUT)**
    This switch is used to adjust the angle of view of the camera.
    Press IN or OUT to adjust the angle of view of video.

13. **Power Indicator (POWER)**
    The power is turned on, this indicator lights up.

14. **Audio Input Jack (AUDIO IN 1, 2)**
    The audio signal supplied to this jack is provided both Audio Output Jack (17) via Input Selection Switch (8) A/V 1 or A/V 2.

15. **Composite Video Input Connector (VIDEO IN 1, 2)**
    This connector accepts the 1.0 Vp-p / 75 ohms composite video signal. The video signal supplied to this connector is provided both Composite Video Output Connector (18) via Input Selection Switch (8) A/V 1 or A/V 2.

16. **S-Video Input Connector (S-VIDEO IN 1, 2)**
    This connector accepts the S-Video signal. The S-Video signal supplied to this connector is provided the S-Video Output Connector (19) via Input Selection Switch (8) A/V 1 or A/V 2.

17. **Audio Output Jack (AUDIO OUT 1, 2)**
    The audio signal selected by the Input Selection Switch (8) is provided to both jacks.

18. **Composite Video Output Connector (VIDEO OUT 1, 2)**
    The Video Selected by the Input Selection Switch (8) is provided to both connectors.

19. **S-Video Output Connector (S-VIDEO)**
    The S-Video signal selected by the Input Selection Switch (8) is provided at this Connector.

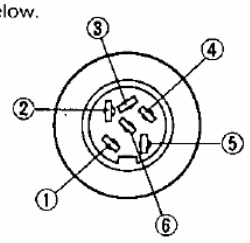
20. **Power Cord.**

21. **Gen-Lock Termination Switch (75 ohms, ON/OFF)**
    When looping through the gen-lock video input signal, set this switch to the OFF position and other cases, set this to the ON position.

22. **Gen-Lock Input Connector (GEN LOCK IN)**
    The color video signal of the camera is automatically synchronized to the gen-lock signal (composite or black burst) which is supplied to this connector. The gen-lock signal is used for system reference.

23. **Lens Remote Control Connector**
    Remove two screws and take the cover away. The lens remote control connector (DIN type) appears.
    Pin allocation is shown below.

    1. GROUND
    2. FOCUS/FAR
    3. FOCUS/NEAR
    4. ZOOM/IN
    5. ZOOM/OUT
    6. POWER



Note:

1. This connector can be used to remotely control the WE-160 in a custom designed system application.
   A connection between pin 6 and pin 2 or pin 3 will control the camera focus.
   A connection between pin 6 and pin 4 or pin 5 will control the zoom.
2. Prepare the plug purchased locally and fix it to compare with above pin allocation.
3. Connect plug and receptacle.

Appendix G. Learning curves

The following appendix shows the learning curves for the task performance for the robotic arm and tasks applications. The first learning curve is based in 8 runs using 8 different V_G for the robotic arm task. The following learning curve corresponds to the 8 runs using 8 V_B for the robotic arm task. The last two learning curves are similar to the first two but resulting from the VMR task.

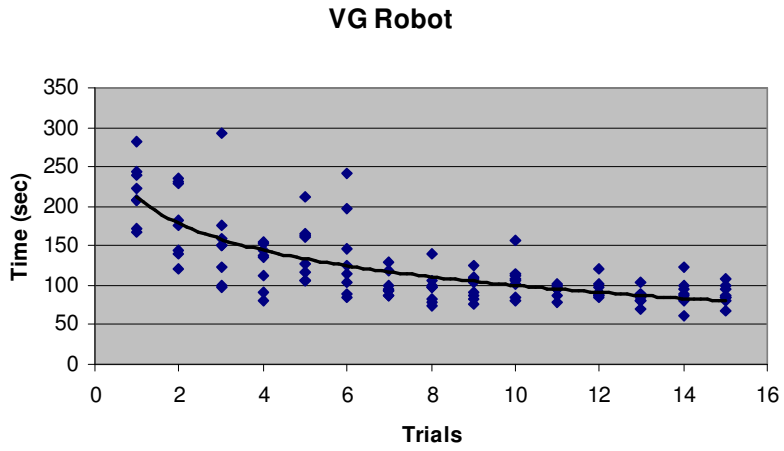


Figure G.1. Learning curve for the V_G vocabulary used in the robotic arm task

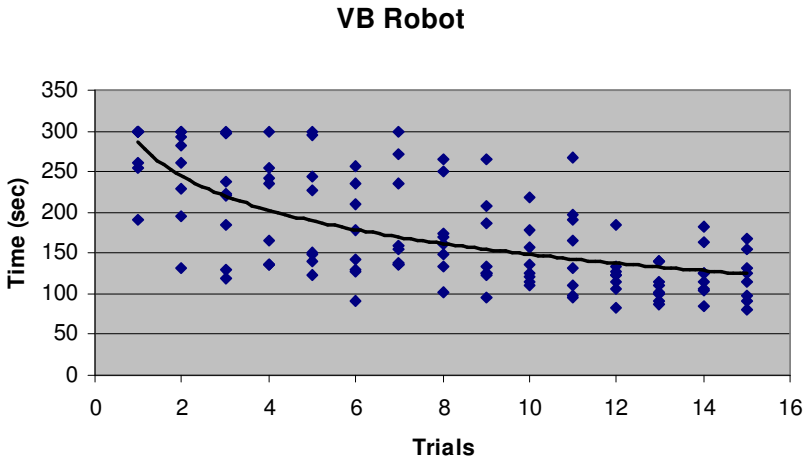


Figure G.2 Learning curve for the V_B vocabulary used in the robotic arm task

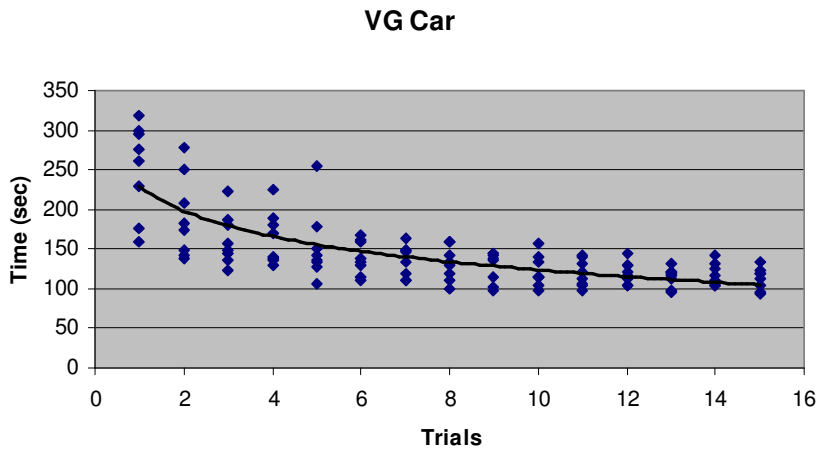


Figure G.3 Learning curve for the V_G vocabulary used in the VMR task

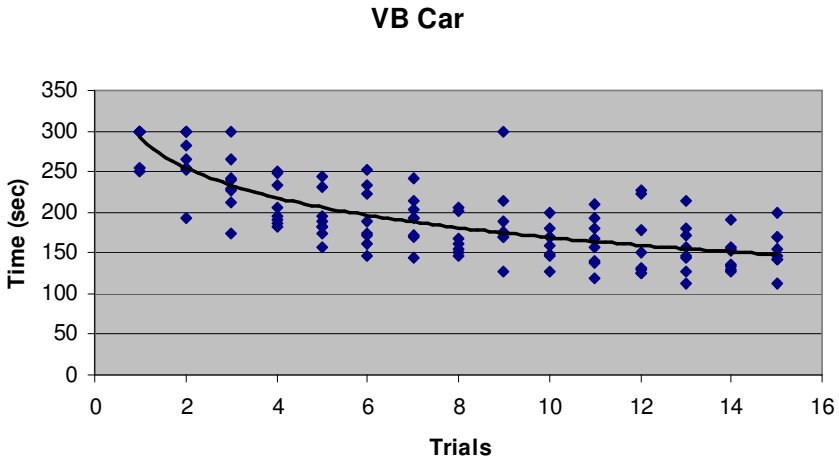


Figure G.4. Learning curve for the V_B vocabulary used in the VMR task

Appendix H. Statistical analysis

This appendix presents statistical results regarding different analysis done on the learning curve experiments, stress and duration prediction model, task completion time and memorability tests.

The table Table H.1 was obtained using the SPSS statistics package, and shows the regression results for the transition stress versus the static stress of the beginning and ending poses. The graph in Figure H.1 shows the result of the regression in a scatter plot. The same analysis was done regarding the duration time of the transition between poses versus the duration time of holding each pose see Table H.2 and Figure H.2.

Regression results for the linearization of the learning curves are presented in tables Table H.3- Table H.6. The tables present the results for the V_G and V_B for each the robotic arm (Table H.3 and

Table H.4) and for the VMR (Table H.5 and Table H.6) tasks.

Table H.7 and Table H.8 shows the results of the t-test comparing the task completion time between the V_G and V_B for the robotic arm and VMR tasks. The last tables, Table H.9 and Table H.10, show the results of the t-test comparing the memorability index between the V_G and V_B vocabularies.

Table H.1. Regression results for the transition stress model

Model Summary

Model	R	R Square(a)	Adjusted R Square	Std. Error of the Estimate
1	.988(b)	.977	.976	.6476997

ANOVA

Model		Sum of Squares	Df	Mean Square	F	Sig.
1	Regression	1024.757	2	512.379	1221.360	.000(a)
	Residual	24.332	58	.420		
	Total	1049.089(b)	60			

Coefficients

Model		Unstandardized Coefficients		Standardized Coefficients	T	Sig.
		B	Std. Error	Beta		
1	Stress_A	.091	.034	.092	2.660	.010
	Stress_B	.905	.034	.912	26.449	.000

Residuals Statistics

	Minimum	Maximum	Mean	Std. Deviation	N
Predicted Value	2.057517	8.003403	3.794374	1.6515060	60
Std. Predicted Value	-1.052	2.549	.000	1.000	60
Standard Error of Predicted Value	.051	.221	.106	.054	60
Adjusted Predicted Value	2.052688	7.986944	3.792189	1.6441112	60
Residual	-1.2716382	1.4448731	-.0360409	.6411580	60
Std. Residual	-1.963	2.231	-.056	.990	60
Stud. Residual	-1.977	2.246	-.054	1.003	60

Deleted Residual	-1.2891288	1.4643981	-.0338558	.6585088	60
Stud. Deleted Residual	-2.029	2.330	-.053	1.016	60
Mahal. Distance	.378	7.002	2.000	2.034	60
Cook's Distance	.000	.131	.014	.021	60
Centered Leverage Value	.006	.117	.033	.034	60

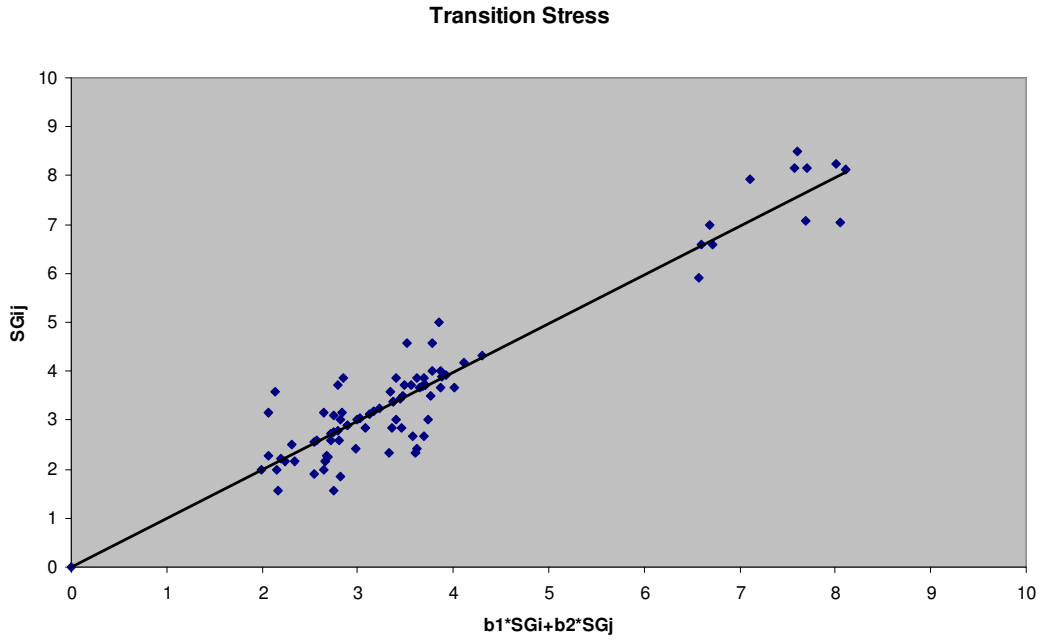


Figure H.1. Plot between real and predicted transition stress

Table H.2. Regression results for the transition duration time model

Model Summary

Model	R	R Square(a)	Adjusted R Square	Std. Error of the Estimate
1	.975(b)	.950	.949	1.03757

ANOVA

Model		Sum of Squares	Df	Mean Square	F	Sig.
1	Regression	1195.560	2	597.780	555.275	.000(a)
	Residual	62.440	58	1.077		
	Total	1258.000(b)	60			

Coefficients

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	Stress_A	.104	.055	.096	1.894	.063
	Stress_B	.973	.055	.895	17.748	.000

Residuals Statistics

	Minimum	Maximum	Mean	Std. Deviation	N
Predicted Value	2.2281	8.6455	4.1016	1.77630	60
Std. Predicted Value	-1.055	2.558	.000	1.000	60
Standard Error of Predicted Value	.082	.354	.169	.086	60
Adjusted Predicted Value	2.2260	8.5551	4.0928	1.75303	60
Residual	-2.02673	3.31321	-.13496	1.01969	60
Std. Residual	-1.953	3.193	-.130	.983	60
Stud. Residual	-1.968	3.298	-.126	1.009	60
Deleted Residual	-2.05777	3.53435	-.12617	1.07549	60
Stud. Deleted Residual	-2.020	3.627	-.118	1.043	60
Mahal. Distance	.378	7.002	2.000	2.034	60
Cook's Distance	.000	.499	.027	.080	60
Centered Leverage Value	.006	.117	.033	.034	60

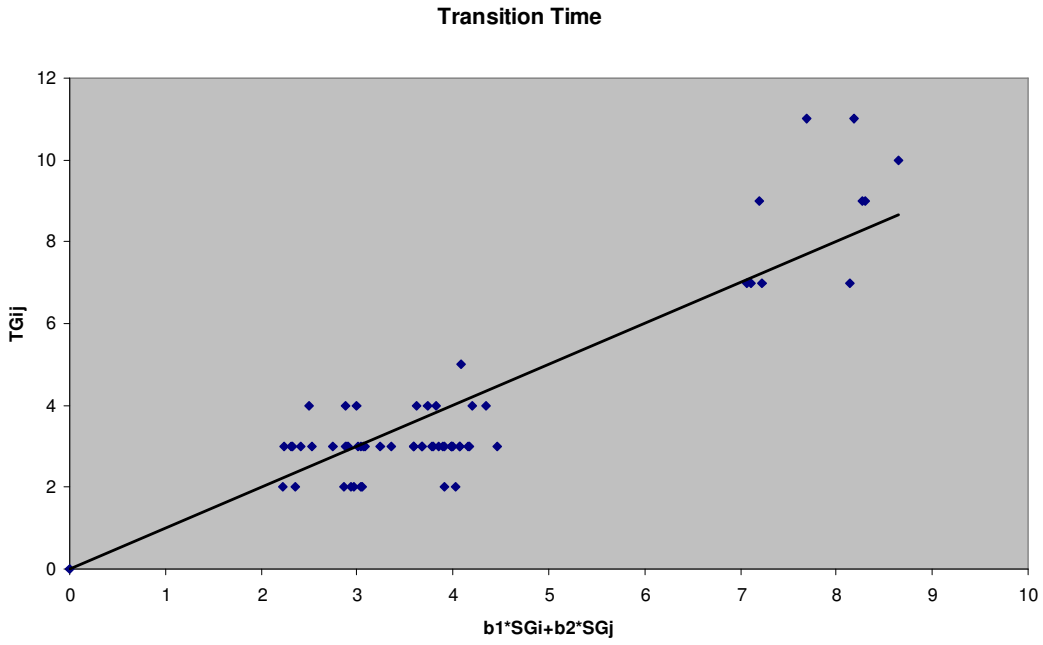


Figure H.2. Plot between the actual and predicted duration time

Table H.3. Results for the linear regression for the robotic arm task, V_G vocabulary (learning curve)

Model	Variables Entered	Variables Removed	Method
1	ln_n(a)	.	Enter

a All requested variables entered.
b Dependent Variable: ln_Yn

Model Summary(b)

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.775(a)	.601	.597	.2159597

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

ANOVA(b)

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	8.276	1	8.276	177.443	.000(a)
	Residual	5.503	118	.047		
	Total	13.779	119			

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

Coefficients(a)

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	5.384	.052		102.772	.000
	ln_n	-.348	.026	-.775	-13.321	.000

a Dependent Variable: ln_Yn

Casewise Diagnostics(a)

Case Number	Std. Residual	ln_Yn
81	3.370	5.4889
93	3.124	5.6768

a Dependent Variable: ln_Yn

Residuals Statistics(a)

	Minimum	Maximum	Mean	Std. Deviation	N
Predicted Value	4.442648	5.384013	4.737462	.2637112	120
Std. Predicted Value	-1.118	2.452	.000	1.000	120
Standard Error of Predicted Value	.020	.052	.027	.008	120
Adjusted Predicted Value	4.437874	5.400646	4.737458	.2638318	120
Residual	-.5076640	.7277712	.0000000	.2150504	120
Std. Residual	-2.351	3.370	.000	.996	120
Stud. Residual	-2.364	3.384	.000	1.003	120

Deleted Residual	-.5136268	.7339372	.0000034	.2184021	120
Stud. Deleted Residual	-2.412	3.546	.003	1.018	120
Mahal. Distance	.008	6.011	.992	1.477	120
Cook's Distance	.000	.085	.008	.014	120
Centered Leverage Value	.000	.051	.008	.012	120

a Dependent Variable: ln_Yn

Table H.4. Results for the linear regression for the robotic arm task, V_B vocabulary (learning curve)

Variables Entered/Removed(b)

Model	Variables Entered	Variables Removed	Method
1	Ln_n(a)	.	Enter

a All requested variables entered.
b Dependent Variable: ln_Yn

Model Summary(b)

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.650(a)	.422	.417	.2910501

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

ANOVA(b)

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	7.301	1	7.301	86.188	.000(a)
	Residual	9.996	118	.085		
	Total	17.297	119			

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

Coefficients(a)

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	5.698	.071		80.702	.000
	ln_n	-.327	.035	-.650	-9.284	.000

a Dependent Variable: ln_Yn

Residuals Statistics(a)

	Minimum	Maximum	Mean	Std. Deviation	N
Predicted Value	4.813634	5.697828	5.090544	.2476953	120
Std. Predicted Value	-1.118	2.452	.000	1.000	120
Standard Error of Predicted Value	.027	.071	.036	.011	120
Adjusted Predicted Value	4.807677	5.726015	5.090992	.2483787	120
Residual	-.6019490	.6723468	.00000	.2898246	120

			00		
Std. Residual	-2.068	2.310	.000	.996	120
Stud. Residual	-2.078	2.325	-.001	1.003	120
Deleted Residual	-.6136261	.6808980	-.0004487	.2939772	120
Stud. Deleted Residual	-2.108	2.370	.000	1.009	120
Mahal. Distance	.008	6.011	.992	1.477	120
Cook's Distance	.000	.080	.007	.011	120
Centered Leverage Value	.000	.051	.008	.012	120

a Dependent Variable: ln_Yn

Table H.5. Results for the linear regression for the VMR task, V_G vocabulary (learning curve)

Variables Entered/Removed(b)

Model	Variables Entered	Variables Removed	Method
1	Ln_n(a)	.	Enter

a All requested variables entered.
b Dependent Variable: ln_Yn

Model Summary(b)

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.770(a)	.593	.590	.1721787

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

ANOVA(b)

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	5.103	1	5.103	172.126	.000(a)
	Residual	3.498	118	.030		
	Total	8.601	119			

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

Coefficients(a)

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	5.438	.042		130.185	.000
	ln_n	-.273	.021	-.770	-13.120	.000

a Dependent Variable: ln_Yn

Casewise Diagnostics(a)

Case Number	Std. Residual	ln_Yn
65	3.154	5.5413

a Dependent Variable: ln_Yn

Residuals Statistics(a)

	Minimum	Maximum	Mean	Std. Deviation	N
Predicted Value	4.698318	5.437511	4.929816	.2070754	120
Std. Predicted Value	-1.118	2.452	.000	1.000	120
Standard Error of Predicted Value	.016	.042	.021	.007	120
Adjusted Predicted Value	4.694488	5.460166	4.929624	.2066966	120
Residual	-.3623374	.5430669	.0000000	.1714537	120
Std. Residual	-2.104	3.154	.000	.996	120
Stud. Residual	-2.169	3.169	.001	1.006	120
Deleted Residual	-.3849927	.5481370	.0001921	.1751525	120
Stud. Deleted Residual	-2.204	3.299	.001	1.015	120
Mahal. Distance	.008	6.011	.992	1.477	120
Cook's Distance	.000	.147	.011	.022	120
Centered Leverage Value	.000	.051	.008	.012	120

a Dependent Variable: ln_Yn

Table H.6. Results for the linear regression for the VMR task, V_B vocabulary (learning curve)

Variables Entered/Removed(b)

Model	Variables Entered	Variables Removed	Method
1	ln_n(a)	.	Enter

a All requested variables entered.
b Dependent Variable: ln_Yn

Model Summary(b)

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.768(a)	.590	.586	.1656439

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

ANOVA(b)

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	4.618	1	4.618	168.300	.000(a)
	Residual	3.210	117	.027		
	Total	7.828	118			

a Predictors: (Constant), ln_n
b Dependent Variable: ln_Yn

Coefficients(a)

Model		Unstandardized Coefficients	Standardized Coefficients	t	Sig.

		B	Std. Error	Beta		
1	(Constant)	5.711	.040		142.119	.000
	ln_n	-.260	.020	-.768	-12.973	.000

a Dependent Variable: ln_Yn

Casewise Diagnostics(a)

Case Number	Std. Residual	ln_Yn
113	3.402	5.7038

a Dependent Variable: ln_Yn

Residuals Statistics(a)

	Minimum	Maximum	Mean	Std. Deviation	N
Predicted Value	5.007515	5.710957	5.228296	.1978232	119
Std. Predicted Value	-1.116	2.440	.000	1.000	119
Standard Error of Predicted Value	.015	.040	.021	.006	119
Adjusted Predicted Value	5.001895	5.722557	5.228407	.1981446	119
Residual	-.3172990	.5635756	.0000000	.1649406	119
Std. Residual	-1.916	3.402	.000	.996	119
Stud. Residual	-1.931	3.420	.000	1.003	119
Deleted Residual	-.3223614	.5693164	-.0001113	.1673656	119
Stud. Deleted Residual	-1.954	3.589	.001	1.013	119
Mahal. Distance	.008	5.953	.992	1.466	119
Cook's Distance	.000	.060	.007	.011	119
Centered Leverage Value	.000	.050	.008	.012	119

a Dependent Variable: ln_Yn

Table H.7 t-test for the time completion time between V_G and V_B (robotic arm task)

	Time VG	*Time VB*
Mean	87.95833	118.958
Variance	91.22024	642.681
Observations	8	8
Pooled Variance	366.9504	
Hypothesized Mean Difference	0	
df	14	
t Stat	-3.236592	
P(T<=t) one-tail	0.002985	
t Critical one-tail	1.761309	
P(T<=t) two-tail	0.00597	
t Critical two-tail	2.144789	

Table H.8. t-test for the time completion time between V_G and V_B (VMR task)

	Time VG	*Time VB*
Mean	114.667	153.04167
Variance	144.063	379.18849
Observations	8	8
Pooled Variance	261.626	
Hypothesized Mean Difference	0	
df	14	
t Stat	-4.74502	
P(T<=t) one-tail	0.00016	
t Critical one-tail	1.76131	
P(T<=t) two-tail	0.00031	
t Critical two-tail	2.14479	

Table H.9. t-test for the memorability score for the robotic arm task

	Memo VG	*Memo VB*
Mean	87.5	70.83333
Variance	94.444444	405.5556
Observations	8	8
Pooled Variance	250	
Hypothesized Mean Difference	0	
df	14	
t Stat	2.1081851	
P(T<=t) one-tail	0.0267581	
t Critical one-tail	1.7613092	
P(T<=t) two-tail	0.0535161	
t Critical two-tail	2.1447886	

Table H.10. t-test for the memorability score for the VMR task

	Memo VG	*Memo VB*
Mean	96.666667	95
Variance	25.396825	47.619048
Observations	8	8
Pooled Variance	36.507937	
Hypothesized Mean Difference	0	
df	14	
t Stat	0.5516773	
P(T<=t) one-tail	0.2949337	
t Critical one-tail	1.7613092	
P(T<=t) two-tail	0.5898673	
t Critical two-tail	2.1447886	

Appendix I. Proof of convergence of the CNS method

Let p be the vector of parameters, and A the recognition accuracy. For any feasible solution $p=[p_1, \ldots, p_n]$ for the recognition system, define a set N(p) of neighboring solutions of vector p. The number of neighbors of p is 2n as each parameter is incremented up and down. This neighborhood search method starts with an arbitrary initial solution. A pseudo code of the algorithm is shown below:

```
Algorithm neighborhood search
   Begin
 Create an initial feasible solution p=[p1, …, pn]
  While there is a neighbor p' N(p) with A(p') > A(p) do
       Begin
          Replace p by p'
       End
Output p, which is the locally optimal solution
   End
```

Algorithm I-1 Neighborhood search

Define an iteration as one cycle starting from an initial solution p until the next neighbor solution p' is selected. An example sequence of the parameter vectors p, appears in Table 2. Recognition accuracy in each iteration is shown in Fig. 4.

Table I.1 Optimal parameter search

	Parameters			
Iterations	*p1*	*p2*	*p3*	*p4*
1	2	2	3.5	17
2	2	3	3.5	17
3	2	4	3.5	17
4	2	4	3.5	18
5	2	4	3.5	18

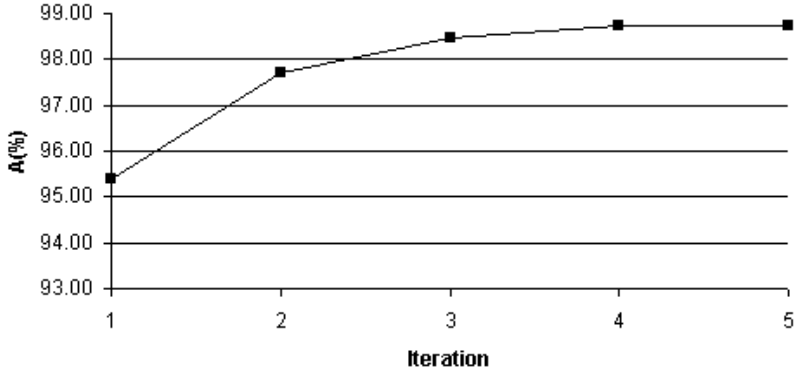


Figure Apx I.1Recognition accuracy vs. Iterations

Average complexity of the neighborhood search algorithm is O(n) (where n is the size of the parameter vector) times the number of iterations. In the previous example, for the given p, the number of neighborhood solutions examined is 2 x 4 x Ave. no. of iterations = 8 x 5 = 40 (convergence was fast in the order of 3 to 8 iterations). Complete evaluation requires an evaluation of 2940 (the size of the search space). It should be noted that the evaluation of each solution requires the determination of a new set of image features, executing the FCM algorithm, cluster label assignments, gesture recognition, and analysis of the confusion matrix.

Appendix J. Supervised FCM optimization procedure

The supervised FCM optimization procedure was applied first on the independent system for the VMR master set. To find a good initial solution of the parameter vector for the optimization of the supervised FCM, nine solutions were generated using the five heuristic rules explained in [Wachs *et al.,* 2005]. In the following tables the nine starting solutions are presented, and used to search for optimal parameter vector. Then the confusion matrices are presented, obtained using the optimal parameter vector set. The confusion matrices correspond to the car and robotic master set of gestures.

Table J.1. Parameter search results for VMR gesture set initial solutions

num	Rb	Cb	C	m	t	w											
1	2	2	15	2	142	0.2058	0.2000	0.1924	0.2111	0.1907							
2	2	2	19	2	142	0.2058	0.2000	0.1924	0.2111	0.1907							
3	2	2	23	2	142	0.2058	0.2000	0.1924	0.2111	0.1907							
4	5	5	15	2	142	0.0373	0.0940	0.0398	0.0304	0.0363	0.0434	0.0561	0.0389	0.0314	0.0297	0.0307	0.0366
5	5	5	19	2	142	0.0467	0.0438	0.0341	0.0355	0.0353	0.0427	0.0363	0.0306	0.0330	0.0305	0.0300	0.0374
6	5	5	23	2	142	0.0373	0.0940	0.0398	0.0304	0.0363	0.0434	0.0561	0.0389	0.0314	0.0297	0.0307	0.0366
7	8	8	15	2	142	0.0201	0.0200	0.0179	0.0148	0.0149	0.0155	0.0154	0.0160	0.0199	0.0157	0.0126	0.0118
8	8	8	19	2	142	0.0201	0.0200	0.0179	0.0148	0.0149	0.0155	0.0154	0.0160	0.0199	0.0157	0.0126	0.0118
9	8	8	23	2	142	0.0128	0.0128	0.0201	0.0200	0.0179	0.0148	0.0149	0.0155	0.0154	0.0160	0.0199	0.0157

Table J.2. Parameter search result for initial solution 5 – VMR gesture set

num	Rb	Cb	w	c	m	t	A(%)	n
1	5	5	0.047 0.044 0.034 0.036 0.035 0.043 0.036 0.031 0.033 0.031 0.03	23	2	142	77.25	690
2	5	5	0.512 0.023 0.017 0.018 0.018 0.022 0.018 0.016 0.017 0.016 0.015	23	2	142	90.43	690
3	4	5	0.557 0.024 0.019 0.02 0.019 0.024 0.02 0.017 0.018 0.017 0.017	23	2	142	90.72	690
4	4	5	0.557 0.024 0.019 0.02 0.019 0.024 0.02 0.017 0.018 0.017 0.017	25	1.5	142	94.35	690
5	4	5	0.557 0.024 0.019 0.02 0.019 0.024 0.02 0.017 0.018 0.017 0.017	25	1.5	142	94.35	690
6	4	5	0.557 0.024 0.019 0.02 0.019 0.024 0.02 0.017 0.018 0.017 0.017	24	1.5	142	95.07	690
7	4	4	0.615 0.027 0.021 0.022 0.022 0.026 0.022 0.019 0.02 0.019 0.018	24	1.5	142	96.23	690
8	4	4	0.615 0.027 0.021 0.022 0.022 0.026 0.022 0.019 0.02 0.019 0.018	24	1.5	142	96.23	690
9	4	4	0.615 0.027 0.021 0.022 0.022 0.026 0.022 0.019 0.02 0.019 0.018	24	1.5	142	96.23	690
10	4	4	0.615 0.027 0.021 0.022 0.022 0.026 0.022 0.019 0.02 0.019 0.018	24	1.5	142	96.23	690
11	4	4	0.629 0.028 0.021 0.023 0.022 0.027 0.023 0.019 0.021 0.019 0.019	24	1.5	142	96.52	690
12	4	4	.693 .023 .017 .019 .018 .022 .019 .016 .017 .016 .016 .019 .019	23	1.5	152	91.59	880
13	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	23	1.5	152	91.7	880
14	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	23	1.5	162	92.27	880
15	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	23	1.5	162	92.27	880
16	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	37	1.5	162	92.61	880
17	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	37	1.5	152	92.84	880
18	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	36	1.5	152	92.84	880
19	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	36	1.5	152	92.84	880
20	4	4	.744 .019 .014 .016 .015 .018 .016 .013 .014 .013 .013 .016 .016	36	1.5	152	92.84	880
21	4	4	.754 .019 .014 .016 .015 .019 .016 .014 .014 .014 .014 .016 .016	36	1.5	152	93.41	880
22	4	4	.754 .019 .014 .016 .015 .019 .016 .014 .014 .014 .014 .016 .016	35	1.5	152	93.41	880
23	4	4	.754 .019 .014 .016 .015 .019 .016 .014 .014 .014 .014 .016 .016	37	1.5	152	93.41	880
24	4	4	.754 .019 .014 .016 .015 .019 .016 .014 .014 .014 .014 .016 .016	37	1.5	152	93.41	880

Table J.3. Parameter search result using VMR optimal solution –robotic arm gesture set

num	Rb	Cb	w	c	m	t	A(%)	n
1	4	4	.693 .023 .017 .019 .018 .022 .019 .016 .017 .016	37	1.5	152	91.63	920
2	3	4	.758 .025 .019 .021 .02 .024 .021 .018 .019 .018	37	1.5	152	93.696	920
3	3	4	.758 .025 .019 .021 .02 .024 .021 .018 .019 .018	36	1.5	152	93.913	920
4	3	4	.758 .025 .019 .021 .02 .024 .021 .018 .019 .018	36	1.5	152	93.913	920

Table J.4. Confusion matrix for optimal solution - VMR case

gt \ gj	1	2	3	4	5	6	7	8	10	12	13	16	17	18	20	21	22	23	24	25	26	27
1	17	0	19	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
2	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	37	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	4	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	7	0	0	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0	0	39	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	39	0	0	0	0	1	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0
20	0	0	1	0	0	0	0	0	0	0	0	0	0	0	39	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	5	0	0	0
23	7	0	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0	29	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	38	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40

Table J.5. Confusion matrix for optimal solution -robotic arm case

gi \ gj	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20	23	24	26	27
1	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0
2	0	35	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	11	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	2	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	3	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	34	0	0	0	5	0	1	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	5	35	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	37	0	3	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	5	0	35	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	2	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0
20	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	1	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	3	35	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40

Appendix K. Software code

The robotic arm software is written in Microsoft MFC (using Visual Studio version 6.0), using the OpenCV machine vision library routines, under Windows2000).

The dissertation includes two main systems: a) the system that controls a telerobot/virtual robotic arm according to hand gestures evoked by the user, called "GestureRec" and, b) the system that finds GV solutions (matching commands to gestures), called "QAPI".

Additional smaller systems uses small variation of the codes presented for the "GestureRec" system, such the one to control the virtual VMR.

GestureRec

This system is used to train, test and run the recognition module for the hand gesture system. This system uses an optimized supervised FCM to cluster feature vectors that represent gesture instances. Each centroid represents a gesture class. Once the FCM classifier is trained and calibrated using a training set of gesture images, it is used to assign classes to gesture samples provided in real-time by the user. These gestures have assigned commands that are sent to a virtual robotic arm to carry out an action. A flowchart describing the operation of the system is presented in Figure K.1.

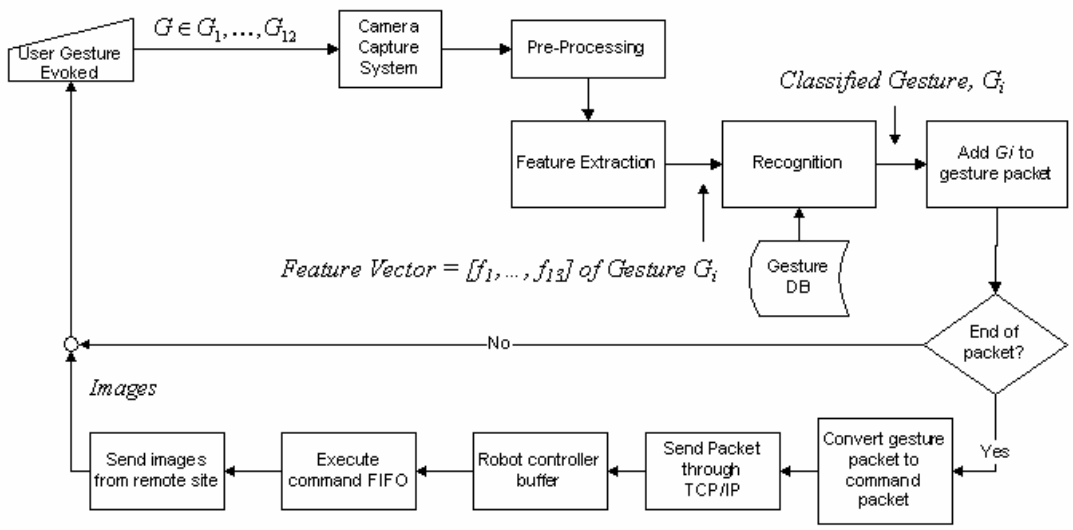


Figure K.1. Flowchart of the GestureRec system

GestureRec object and methods description

The following tables present the objects used in each system, all their members and methods, and the description of each of them.

GestureRec Class	
Members	**Description**
long SizeX;	Buffer Size X of the frame grabber
long SizeY;	Buffer Size Y of the frame grabber
long DigSizeX;	Digitizer input width of the frame grabber
long DigSizeY;	Digitizer input height of the frame grabber

long nBands;	Number of input color bands of the digitizer of the frame grabber
BOOL GrabIsStarted;	State of the grab of the frame grabber
CView *GrabInViewPtr;	Pointer to the view that has the grab of the frame grabber
long NumberOfDigitizer;	Number of digitizers available on the system
MIL_ID MilApplication;	The MIL application ID
MIL_ID MilSystem;	
MIL_ID MilDisplay;	The MIL system ID
MIL_ID MilDigitizer;	The MIL display ID
MIL_ID MilImage;	The MIL digitizer ID
	The MIL image ID
void MV1_Open();	Initialize the frame grabber
void MV1_Close();	Closes the frame grabber
void MV1_StartGrabIt();	Start grabbing images to the frame grabber
void MV1_StopGrabIt();	Stop grabbing images from the frame grabber
Methods	**Description**
int RoiNorm(int NumRows,int NumCols,IplImage *src,int minX,int maxX,int minY,int maxY);	Given a ROI rectangular, blocks are created as a result of rows and columns. Each block has a value which is the average of the grayscale values in the block
void Bouncing_Box(IplImage *src,int &minX,int &maxX,int &minY,int &maxY);	A bounding box is created around the biggest blob in the image. The biggest blob is obtained by calculating the largest perimeter of the contour of each blob
void CopyVector2Buffer(int counter,int minX,int maxX,int minY,int maxY,short int flag,int the_contador);	A vector including the values of the block partition is stored in the buffer memory
void CopyBuffer2DB(const int Nframes);	The buffer memory content including the feature vectors is copied to a database
int CopyDB2Buffer();	Copy the content of the feature vector table of the database to the buffer memory
void StringVector2ValueVector(int counter);	The string representing the feature vector is converted to a vector of numerical values
void Weight_String2Weight_Vector(); // Convert a string of weights to a vector of weights	The weighted string is converted to a weighted numerical vector
void CreateFeaturesMatrix();	The string vectors matrix is converted to a matrix of numerical values
void DisplayFeatures(int counter,int Rows,int Cols);	Displays an image composed of blocks with different grayscales values for a given feature vector
void AddValue2Vector(int value,int index);	Adds a value of grayscale for the current block to the feature vector
void RandomClusters(int Nclusters,int NumFrames);	Randomize the clusters centroids to init the FCM algorithm
float D(int i,int j);	Find the Euclidian distance between two vectors
float Find_MiuIJ(int i,int j);	Find the membership value for a given feature vector
void CreateMembership();	Creates the membership matrix (FCM algorithm)
void CreateCentroids();	Creates the centroids matrix (FCM algorithm)
void Find_Ci(int i);	Updates the current centroid using the information of the membership values (FCM algorithm)
float CostFunction();	Find the cost function (FCM algorithm)

void Membership2DB();	Copy the membership matrix to the membership table in the database
void Centroid2DB();	Copy the centroid matrix to the centroid table in the database
void Cost2DB(float cost);	Copy the cost values of every iteration to the cost table in the database
int DB2Centroid();	Copy from the centroids table of the database to the memory
int DB2Membership();	Copy from the membership table of the database to the memory
void CopyVector2Mat(int counter,short int flag);	Copy a vector to a matrix data type
void CreateNewMembership(int j,short int flag);	For a new feature vector from a real-time image, find the membership value
void NewMembership2DB(short int flag);	Copy the new membership value to the membership table in the database
void DrawGraphico(int j,short int flag);	Shows a bar graph representing the membership values of the current feature vector
int Pictures_inDB();	Count the number of feature vectors in the database
void RandomOneCluster(int Nclusters,int Nframes);	Randomize the initial position of only one new cluster
int Clusters_inDB();	Counts the number of clusters stored in the database
int Input_Parameters(char file_name[250]);	Copy the tuning parameters for the FCM, from the database to the memory
void AutomaticBatchMode(char file_name[250]);	Runs automatically a training session of the FCM without user interaction
void AutomaticTestMode(char file_name[250]);	Runs automatically a testing session of the FCM without user interaction
void SendTcpMessage(int j,short int flag,char sTotal[50],int &cont); //uses TCP/IP, send the command	Uses TCP protocol to send a message to the robotic arm server
void CloseTcpMessage(); //uses TCP/IP, send the command	Close the communication port between the client to the robotic arm server
int OpenTcpMessage(); //uses TCP/IP, send the command	Open the communication port between the client and the robotic arm server
void ShowLab(int &conter); //show an image of the lab, jpg pic	Shows an image of the distant scenario and sends it through FTP
bool Listen(int PortNum);	Listen to the server for specific command

```
// GestureRec.h : main header file for the GESTUREREC application
//

#if !defined(AFX_GESTUREREC_H__D111738D_241B_49C7_A936_F33231E3B2D6__INCLUDED_)
#define AFX_GESTUREREC_H__D111738D_241B_49C7_A936_F33231E3B2D6__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CGestureRecApp:
// See GestureRec.cpp for the implementation of this class
//

class CGestureRecApp : public CWinApp
{
public:
	CGestureRecApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CGestureRecApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CGestureRecApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_GESTUREREC_H__D111738D_241B_49C7_A936_F33231E3B2D6__INCLUDED_)
```

```
// GestureRecDlg.h : header file
//

#if !defined(AFX_GESTURERECDLG_H__CBD8D3DF_8219_4531_AE49_9855E11BC1BD__INCLUDED_)
#define AFX_GESTURERECDLG_H__CBD8D3DF_8219_4531_AE49_9855E11BC1BD__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

CV_TURN_ON_IPL_COMPATIBILITY();

//#include <ipl.h>
#include <cv.h>
#include "HighGUI.h"
#include "mil.h"
#include "mwinmil.h"
#include "milsetup.h"
#include "milerr.h"
#include <stdio.h>

#ifndef _WIN32          // If not compiling on a Windows system
        #include <sys/types.h>
        #include <sys/socket.h>
        #include <netinet/in.h>
        #include <unistd.h>
        #include <netdb.h>
        #include <arpa/inet.h>
        #define SOCKET                  int
        #define INVALID_SOCKET          -1
        #define closesocket             close
        #include <pthread.h>
#else// Yes this is a Windows system
        #include <winsock.h>
        #define socklen_t int
        // Programmatically setup the necessary library files
        #if defined(_MSC_VER)
                #pragma comment(lib, "wsock32.lib")
        #elif defined(__BORLANDC__)
                #pragma(lib, "mswsock.lib")
        #endif
#endif

//***** All this files were for the TCP/IP client**************//


/////////////////////////////////////////////////////////////////////////////
// CGestureRecDlg dialog

class CGestureRecDlg : public CDialog
{
// Construction
public:
        CGestureRecDlg(CWnd* pParent = NULL);// standard constructor

// Dialog Data
        //{{AFX_DATA(CGestureRecDlg)
        CButton m_ok;

        enum { IDD = IDD_GESTUREREC_DIALOG };
```

```
	long		SizeX;					// Buffer Size X
	long		SizeY;					// Buffer Size Y
	long		DigSizeX;				// Digitizer input width
	long		DigSizeY;				// Digitizer input heigh
	long		nBands;					// Number of input color bands of the digitizer
	BOOL		GrabIsStarted;		// State of the grab
	CView	*GrabInViewPtr;		// Pointer to the view that has the grab
	long		NumberOfDigitizer;	// Number of digitizers available on the system

	MIL_ID MilApplication;		// The MIL application ID
	MIL_ID MilSystem;			// The MIL system ID
	MIL_ID MilDisplay;			// The MIL display ID
	MIL_ID MilDigitizer;		// The MIL digitizer ID
	MIL_ID MilImage;			// The MIL image ID

	MAPPHOOKFCTPTR		HandlerPtr;
	void*				HandlerUserPtr;
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CGestureRecDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);		// DDX/DDV support


	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

		void ChangeSize();
	void MV1_Open();
	void MV1_Close();
	void MV1_StartGrabIt();
	void MV1_StopGrabIt();

	 int RoiNorm( int NumRows,int NumCols,IplImage *src,int minX,int maxX,int minY,int maxY);
	void Bouncing_Box(IplImage *src,int &minX,int &maxX,int &minY,int &maxY );
	void CopyVector2Buffer(int counter,int minX,int maxX,int minY,int maxY,short int flag,int the_contador);
	void CopyBuffer2DB(const int Nframes);
	int CopyDB2Buffer();
	void StringVector2ValueVector(int counter);
	void Weight_String2Weight_Vector(); // Convert a string of weights to a vector of weights
	void CreateFeaturesMatrix();
	void DisplayFeatures(int counter,int Rows,int Cols);
	void AddValue2Vector(int value,int index);
	void RandomClusters(int Nclusters,int NumFrames);
	float D(int i,int j);
	float Find_MiuIJ(int i,int j);
	void  CreateMembership();
	void CreateCentroids();
	void Find_Ci(int i);
	float CostFunction();
	void Membership2DB();
	void Centroid2DB();
	void Cost2DB(float cost);
	int DB2Centroid();
```

```
	int DB2Membership();
	void CopyVector2Mat(int counter,short int flag);
	void CreateNewMembership(int j,short int flag);
	void NewMembership2DB(short int flag);
	void DrawGraphico(int j,short int flag);
	int Pictures_inDB();
	void RandomOneCluster(int Nclusters,int Nframes);
	int Clusters_inDB();
	int Input_Parameters(char file_name[250]);
	void AutomaticBatchMode(char file_name[250]);
	void AutomaticTestMode(char file_name[250]);

	void SendTcpMessage(int j,short int flag,char sTotal[50],int &cont); //uses TCP/IP, send the command
	// the flag says if we use database, so continue add to array of frames
	// or stay always at the end of the array of frames (same place)
	// sTotal is the bunch of strings (each string is a number-gesture
	// cont is a counter of the bunch, for example 5.
	void CloseTcpMessage(); //uses TCP/IP, send the command
	int OpenTcpMessage(); //uses TCP/IP, send the command
	void ShowLab(int &conter); //show an image of the lab, jpg pic


		//************ TCP/IP Functions ********///
	bool SendMsg(char *Msg, int Len, char *host, short port);
	bool Listen(int PortNum);

	static void *ListenThread(void *data);
	SOCKET ListenSocket;	// the socket that we're listening for connections on
	sockaddr_in srv;			// the address that the server is listening on
	sockaddr_in client;				// the address that the last message was received from
	//************ Were TCP/IP Functions ********///


	// Generated message map functions
	//{{AFX_MSG(CGestureRecDlg)
		virtual BOOL OnInitDialog();
		virtual void OnOK();
		virtual void OnCancel();
		afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
		afx_msg void OnPaint();
		afx_msg HCURSOR OnQueryDragIcon();
		afx_msg void OnStart();
		afx_msg void OnProcess();
		afx_msg void OnFindClusters();
		afx_msg void OnLoad_Clusters();
		afx_msg void OnCapture_Gesture();
		afx_msg void OnAdd_Gesture();
		afx_msg void OnBatchMode();
		afx_msg void OnRunBatchMode();

		void EndDialog(int nResult); // Destructor TCP/IP
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif //
!defined(AFX_GESTURERECDLG_H__CBD8D3DF_8219_4531_AE49_9855E11BC1BD__INCLUDED_)
```

```
// GestureRec.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "GestureRec.h"
#include "GestureRecDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGestureRecApp

BEGIN_MESSAGE_MAP(CGestureRecApp, CWinApp)
	//{{AFX_MSG_MAP(CGestureRecApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGestureRecApp construction

CGestureRecApp::CGestureRecApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CGestureRecApp object

CGestureRecApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CGestureRecApp initialization

BOOL CGestureRecApp::InitInstance()
{
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

	CGestureRecDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
```

```
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

```
// GestureRecDlg.cpp : implementation file
//

// For the independent system found to be the best: Clusters=18, Rows=4, Cols=5,weight=3 Accuracy= 98.2%
// // For the dependent system found to be the best: Clusters=15, Rows=3, Cols=4,weight=2.5
// For the old dependent system found to be the best: Clusters=13, Rows=3, Cols=4,weight=2.5

#include "stdafx.h"
#include "GestureRec.h"
#include "GestureRecDlg.h"
#include "assert.h"
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <iostream>
#include <ole2.h>
#include "conio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

long nCols, nRows;                    // MIL image dimensions

#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
   no_namespace rename("EOF", "EndOfFile")

inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};

CImage imagen,resized,blackwh,featu,grafico,lab,i,im;
CStatic          mimg,mimg2,mimg3,mimg4,mimg5;

char buffer[640*480];

//const int Rows=4;   // originally=3
//const int Cols=5;       // originally=4
int Rows=3;
int Cols=4;
int Nframes=0; //((((((((((0))))))))))samples to learn, suggested 390;
                                        // We'll find the number of pictures in DB.
const int grab_cycle=60;   // how many pictures grab per cycle
//int Nclusters=18,CLUSTERS=Nclusters; // number of clusters (partitions)   // before this was 13
int Nclusters=18,CLUSTERS=Nclusters; // number of clusters (partitions)   // before this was 13
int FeatureLen=Rows*Cols+1;
int Old_FeatureLen=0; //number of features (incl. aspect ratio from the prev. run)
float m=2; // fixed value for fuzzy clustering
//int seed=1;   // best produced by 100000
const int NewFrames=10000; // images to grab in testing stage
double weights_val[100]; //weights in float format
int bw_threshold=0; // threshold for image processing grayscale to black and white
const double threshold=0.8; // Every gesture bigger than this value (membership) is recognized
int number_of_files=260;
char weights[1000];
int IMG_WIDTH=320;   //Width of all the working images beside the frame-grabber
int IMG_HEIGHT=240;  //Height of all the working images beside the frame-grabber

char vector[1000];
```

```
float IntVector[200];

struct BufferMem
{
    char gest[256];
    char file[256];
        char width[256];
        char height[256];
    char data[4000];
};

struct BufferValue
{
    //float data[FeatureLen];
        float data[101];
    unsigned int width;
        unsigned int height;

};

struct BufferCentroid
{
    //float data[FeatureLen];
       float data[101];

};


BufferMem Buffer[20000];
BufferMem DestBuffer[20000];
BufferValue MatFeatures[20000];

CvMat Uij = cvMat(100,Nframes+NewFrames,CV_MAT32F,NULL);
//CvMat Ci = {Nclusters,FeatureLen,CV_MAT32F,0,NULL};
BufferCentroid Ci[100];


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
```

```
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGestureRecDlg dialog

CGestureRecDlg::CGestureRecDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CGestureRecDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CGestureRecDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CGestureRecDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CGestureRecDlg)
		DDX_Control(pDX, IDOK, m_ok);
		DDX_Control(pDX, IDC_IMG, mimg);
		DDX_Control(pDX, IDC_IMG2, mimg2);
		DDX_Control(pDX, IDC_IMG3, mimg3);
		DDX_Control(pDX, IDC_IMG4, mimg4);
		DDX_Control(pDX, IDC_IMG5, mimg5);
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGestureRecDlg, CDialog)
	//{{AFX_MSG_MAP(CGestureRecDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, OnStart)
	ON_BN_CLICKED(IDC_BUTTON2, OnProcess)
	ON_BN_CLICKED(IDC_BUTTON3, OnFindClusters)
	ON_BN_CLICKED(IDC_BUTTON4, OnLoad_Clusters)
	ON_BN_CLICKED(IDC_BUTTON5, OnCapture_Gesture)
	ON_BN_CLICKED(IDC_BUTTON6, OnAdd_Gesture)
```

```
	ON_BN_CLICKED(IDC_BUTTON7, OnBatchMode)
	ON_BN_CLICKED(IDC_BUTTON8, OnRunBatchMode)

	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGestureRecDlg message handlers

BOOL CGestureRecDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	char file_name[250];
	//**** TCP/IP Files ******
	ListenSocket = INVALID_SOCKET;		// Set to INVALID to begin with
	#ifdef _WIN32	// don't need to do anything if not a Windows machine
		WORD VersionRequested = MAKEWORD(1,1);
		WSADATA wsaData;
		WSAStartup(VersionRequested, &wsaData);		// starts the Winsock service
		if ( wsaData.wVersion != VersionRequested )
		{
			//printf("Wrong version or WinSock not loaded\n");
			AfxMessageBox("Wrong version or WinSock not loaded\n");
			fflush(0);
		}
	#endif
		//**** Were TCP/IP Files ******
	Nframes=Pictures_inDB(); // Find how many pictures are already in DB
    int status=Input_Parameters(file_name); // Find ROWS, COLS, WEIGHTS,M,BW_THRESHOLD
number_of_files (samples per gesture)
	FeatureLen=Rows*Cols+1;
	CLUSTERS=Nclusters;
	Weight_String2Weight_Vector(); // Convert the weights string to a weight vector

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
```

```
	// All the automatic part will be here *****************

	if (status==1) // status show if is train or test run (status=1 => train)
	{
		AutomaticBatchMode(file_name);// //++++++++++++++++++++++++++UNCOMMENT IT FOR
EXTERNAL RUN
		OnProcess();// //++++++++++++++++++++++++++UNCOMMENT IT FOR EXTERNAL RUN
		OnFindClusters();//  //++++++++++++++++++++++++++UNCOMMENT  IT  FOR  EXTERNAL
RUN
	}
	else
	{
		OnLoad_Clusters();//  //++++++++++++++++++++++++++UNCOMMENT IT FOR EXTERNAL
RUN
		AutomaticTestMode(file_name);//  //++++++++++++++++++++++++++UNCOMMENT  IT  FOR
EXTERNAL RUN
	}
	OnOK();  //++++++++++++++++++++++++++UNCOMMENT IT FOR EXTERNAL RUN
	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CGestureRecDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CGestureRecDlg::OnPaint()
{
if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
```

```
                CDC *dc1 = mimg.GetDC();
                CDC *dc2 = mimg2.GetDC();
                CDC *dc3 = mimg3.GetDC();
                CDC *dc4 = mimg4.GetDC();
                CDC *dc5 = mimg5.GetDC();


                i.Show (dc1->m_hDC, 0, 0, i.Width(), i.Height(), 0, 0);
                blackwh.Show (dc2->m_hDC, 0, 0, blackwh.Width(), blackwh.Height(), 0, 0);
                featu.Show (dc3->m_hDC, 0, 0, featu.Width(), featu.Height(), 0, 0);
                grafico.Show(dc4->m_hDC,0,0, grafico.Width(),grafico.Height(),0,0);
                lab.Show(dc5->m_hDC,0,0, lab.Width(),lab.Height(),0,0);

                featu.Destroy();
                grafico.Destroy();

                mimg.ReleaseDC( dc1 );
                mimg2.ReleaseDC( dc2 );
                mimg3.ReleaseDC( dc3 );
                mimg4.ReleaseDC( dc4 );
                mimg5.ReleaseDC( dc5 );
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CGestureRecDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}


void CGestureRecDlg::ChangeSize()
{
                // resize window
                RECT r;
                GetWindowRect (&r);
                r.bottom=r.top+max(resized.Height()+40, 400);
                r.right=r.left+resized.Width()+165;
                MoveWindow (&r);
                mimg.MoveWindow (10, 10, resized.Width(), resized.Height());

}

void CGestureRecDlg::MV1_Open()
{
   MappAllocDefault(M_SETUP,&MilApplication,&MilSystem,&MilDisplay,&MilDigitizer,NULL);
      MdigInquire(MilDigitizer, M_SIZE_BAND, &nBands);
      MdigInquire(MilDigitizer, M_SIZE_X,    &nCols);
      MdigInquire(MilDigitizer, M_SIZE_Y,    &nRows);
      MbufAllocColor(MilSystem,                    nBands,                    nCols,
nRows,8L+M_UNSIGNED,M_IMAGE+M_DISP+M_GRAB+M_PROC+M_OFF_BOARD+M_BGR24+M_PACK
ED,&MilImage);

}

void CGestureRecDlg::MV1_Close()
{
```

```
  MappFreeDefault(MilApplication,MilSystem,MilDisplay,MilDigitizer,MilImage);
}

void CGestureRecDlg::MV1_StartGrabIt()
{
    MdigGrabContinuous(MilDigitizer, MilImage);
}


void CGestureRecDlg::MV1_StopGrabIt()
{
      MdigHalt(MilDigitizer);
}

// Find the number of pictures already in the DB
int CGestureRecDlg::Pictures_inDB()
{

      int number=0;
      HRESULT  hr = S_OK;
      _bstr_t gest_num;

  if(FAILED(::CoInitialize(NULL)))
  {
        AfxMessageBox("Problems opening Gesture DB.");
       exit(1);
    return 1;
  }
  if (SUCCEEDED(hr))
  {
      _RecordsetPtr pRstGestures("ADODB.Recordset");
      // Connection String
   _bstr_t strCnn("DSN=gesture;");
       // Open table
        try
   {
                pRstGestures->Open("SELECT COUNT(*) AS result FROM GESTURE;", strCnn, adOpenStatic,
adLockReadOnly, adCmdText);
      gest_num =((_bstr_t) pRstGestures->GetFields()->GetItem("result")->GetValue());
      number=atoi(gest_num);
                pRstGestures->Close();
       }
    catch (_com_error &e)
    {
      // Notify the user of errors if any.
      // Pass a connection pointer accessed from the Recordset.
      _variant_t vtConnect = pRstGestures->GetActiveConnection();


                        AfxMessageBox((char*) e.Description());

                        exit(1);

      // GetActiveConnection returns connect string if connection
      // is not open, else returns Connection object.

    }
       // Clean up objects before exit.
    if (pRstGestures)
      if (pRstGestures->State == adStateOpen)
```

```
            pRstGestures->Close();


      }
      return number;

    }

    // This function finds the number of clusters (centroids) that already exist in the DB
    int CGestureRecDlg::Clusters_inDB()
    {

          int number=0;
          HRESULT  hr = S_OK;
          _bstr_t gest_num;

      if(FAILED(::CoInitialize(NULL)))
        return 1;
      if (SUCCEEDED(hr))
      {
          _RecordsetPtr pRstGestures("ADODB.Recordset");
         // Connection String
         _bstr_t strCnn("DSN=gesture;");
          // Open table
          try
          {
                    pRstGestures->Open("SELECT  COUNT(*)  AS  result  FROM  CENTROID;",  strCnn,
adOpenStatic, adLockReadOnly, adCmdText);
          gest_num =((_bstr_t) pRstGestures->GetFields()->GetItem("result")->GetValue());
              number=atoi(gest_num);
          pRstGestures->Close();
          }
          catch (_com_error &e)
       {
         // Notify the user of errors if any.
         // Pass a connection pointer accessed from the Recordset.
          _variant_t vtConnect = pRstGestures->GetActiveConnection();


                          AfxMessageBox((char*) e.Description());

                          exit(1);
          // GetActiveConnection returns connect string if connection
          // is not open, else returns Connection object.

       }
      }
      return number;

    }


    void CGestureRecDlg::AddValue2Vector(int value,int index)
    {
      char val[10]="";
      char zeros[10]="";

      IntVector[index]=value; // add the integer value to a vector
      sprintf(val, "%d",value); // add the string...
          if (value < 10) //padd with 0 zero
```

```
        {
                                        strcpy(zeros,"00");
                                        strcat(zeros,val);
                                        strcpy(val,zeros);
        }
        else if (value < 100) // padd with 00 zeros
        {
                                        strcpy(zeros,"0");
                                        strcat(zeros,val);
                                        strcpy(val,zeros);
        }
        // put inside inside an string (string of vectors)
        strcat(vector,val);
        strcat(vector," ");

    }


    int CGestureRecDlg::RoiNorm( int NumRows,int NumCols,IplImage *src,int minX,int maxX,int minY,int
maxY)
    {

      strcpy(vector,""); //init vector with features
      const rawSizex = IMG_WIDTH;/// raw size of image
      const rawSizey = IMG_HEIGHT;/// raw size of image
      int tsizeX = (maxX-minX) / NumCols;
      int tsizeY = (maxY-minY) / NumRows;
      int tarea = tsizeX*tsizeY;
      int index=0;

      if (tsizeX<=0) tsizeX=1;
      if (tsizeY<=0) tsizeY=1;

      if (minX>=maxX) minX=1;
      if (minY>=maxY) minY=1;


      featu.Create(IMG_WIDTH,IMG_HEIGHT,8);
      IplImage *dst=featu.GetImage();
      cvSet( dst, CV_RGB(0,0,0) );

      //insert the size ratio (height/width) in the beginning of the feature vector
      AddValue2Vector((int) (weights_val[0]*97.32*(maxY-minY)/(maxX-minX)),index);
          //AddValue2Vector(maxY-minY);
      __try {
        /// the images will use ROI for operations,
        /// the source image uses the ROI to calculate L1 norm,
        /// the destination image uses the ROI to set value
        IplROI roiSource = { 0, 0,0, tsizeX,tsizeY };
            IplROI roiDest = { 0, 0,0, tsizeX,tsizeY };
        src->roi = &roiSource;
            dst->roi = &roiDest;


        /// for each ROI
        for( int y=0; y<NumRows; ++y ) {
          roiSource.yOffset = tsizeY * y + minY;
                    roiDest.yOffset = tsizeY * y;  // fixed in place
          for( int x=0; x<NumCols; ++x ) {
            roiSource.xOffset = tsizeX * x + minX;
```

```
                        roiDest.xOffset = tsizeX * x; // fixed in place
            /// get mean value from source
            int value = (int)( cvNorm( src, NULL, CV_L1 ) / tarea );
            /// put this value to destination

                        cvSet(dst,cvScalar(value));

                        // transforms the value to a string.
                        index++;
                        AddValue2Vector((int)floor(10*weights_val[index]*value),index);    //add    the    other
features
              }
            }
          }
          __finally {
            /// preserve automatic variable and free memory
            src->roi = dst->roi = NULL;
            //iplDeallocate( src, IPL_IMAGE_ALL );
            //iplDeallocate( dst, IPL_IMAGE_ALL );
          }
          //memcpy(vec,vector, sizeof(buffer));

          return(0);
        }

        void CGestureRecDlg::Bouncing_Box(IplImage *src,int &minX,int &maxX,int &minY,int &maxY )
        {

                CImage cont;
                cont.Create(IMG_WIDTH,IMG_HEIGHT,8);
                IplImage* contur = cont.GetImage();
                cvCopy(src,contur);
                int surface=0;
                CvSeq *contour = NULL;
                CvSeqReader reader;
                CvMemStorage *storage = cvCreateMemStorage(0);
                surface=cvCountNonZero (contur);
                CvPoint corner1;
                CvPoint corner2;

                minX=IMG_WIDTH;
                maxX=0;
                minY=IMG_HEIGHT;
                maxY=0;
                //cvRect windo;

                if (surface>80)  // if this surface is bigger than 80 pixels
                {
                        cvFindContours(contur,          storage,&contour,sizeof          (CvContour),
CV_RETR_LIST,CV_CHAIN_APPROX_SIMPLE);
                        if( contour )
                        {
                                for(  CvSeq*  copycontour  =  contour;  copycontour  !=  0;  copycontour  =
copycontour->h_next )
                                {
                                        cvStartReadSeq( copycontour, &reader, 0 );
                                        if (copycontour->total>40)
                                        {
                                                for( int i = 0; i < copycontour->total; i++ )
                                                {
```

```
                                                    CvPoint pt;
                                                    CV_READ_SEQ_ELEM( pt, reader );
                                                    if (pt.x<minX) minX=abs(pt.x);
                                                    if (pt.x>maxX) maxX=abs(pt.x);
                                                    if (pt.y<minY) minY=abs(pt.y);
                                                    if (pt.y>maxY) maxY=abs(pt.y);

                                                }
                                            }
                                        }
                                        corner1.x=minX;
                                        corner1.y=minY;
                                        corner2.x=maxX;
                                        corner2.y=maxY;
                                    }
                                }
                                cvRectangle(src,corner1,corner2,CV_RGB(128,128,128),1);

                                cont.Destroy();
                                cvReleaseMemStorage(&storage);

    }

    void CGestureRecDlg::CopyVector2Buffer(int counter,int minX,int maxX,int minY,int maxY,short int flag,int
the_contador)
    {
        char filen[255]="";
        char fpath[255]="";
        char width[255]="";
        char height[255]="";
    //  time_t long_time;
    //  long int i;

    //  i=time(&long_time);
        //i=1066962813;
    //  i=i+counter;

        sprintf(filen, "%d", the_contador);
        strcat(filen,".bmp");
        if (flag==1)
                strcat(fpath,"C:\\OpenCV_projects\\GestureRec\\pics\\");
        else
           strcat(fpath,"C:\\OpenCV_projects\\GestureRec\\picsNew\\");

        strcat(fpath,filen); // create a filename based in path
        //and a big number (time in seconds).
        sprintf(width, "%d", maxX-minX);
        sprintf(height, "%d", maxY-minY);

        if (flag!=3)
                resized.Save(fpath); //save the picture with this unique name

        if (flag!=3)
        {
                strcpy(Buffer[counter].gest,fpath);
                strcpy(Buffer[counter].file,fpath); //save in buffer the filename
                strcpy(Buffer[counter].data,vector); //save in buffer the vector of picture
                strcpy(Buffer[counter].width,width);//save in buffer the width of the bounc box
                strcpy(Buffer[counter].height,height);//save in buffer the hight of the bounc box
        }
```

```
        else
        {
                strcpy(Buffer[Nframes].gest,fpath);
                strcpy(Buffer[Nframes].file,fpath); //save in buffer the filename
                strcpy(Buffer[Nframes].data,vector); //save in buffer the vector of picture
                strcpy(Buffer[Nframes].width,width);//save in buffer the width of the bounc box
                strcpy(Buffer[Nframes].height,height);//s
        }
}

void CGestureRecDlg::CopyBuffer2DB(const int num_frames)
{
        char vec[600]="";
        char number_pics[50]="";
        HRESULT  hr = S_OK;
        _bstr_t feat;
        _bstr_t filen;
        _bstr_t width;
        _bstr_t height;
        _bstr_t n_pics;

  if(FAILED(::CoInitialize(NULL)))
    return;

  if (SUCCEEDED(hr))
  {
          // Define ADO object pointers.
   // Initialize pointers on define.
  _RecordsetPtr pRstGestures = NULL;
  _ConnectionPtr pConnection  = NULL;

  HRESULT hr = S_OK;

 //Replace Data Source value with your server name.
  _bstr_t strCnn("DSN=gesture;");
   _bstr_t strMessage;


                try
                {
                        //Open a connection
                        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                        pConnection->Open(strCnn,"","",adConnectUnspecified);


                        //Open gestures table
                        TESTHR(pRstGestures.CreateInstance(__uuidof(Recordset)));

                        //You have to explicitly pass the Cursor type and LockType to the Recordset here
                        pRstGestures->Open("gesture",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);

                        for (int num_pics=Nframes; num_pics<num_frames; num_pics++)
                        {
                                //strncpy(vec,(Buffer+num_pics)->data,576);
//                              vec[576]='\0';
                                feat=(Buffer+num_pics)->data;
                                filen=(Buffer+num_pics)->file;
                                width=(Buffer+num_pics)->width;
                                height=(Buffer+num_pics)->height;
```

```
                                        sprintf(number_pics,"%d",num_pics);
                                        n_pics=number_pics;

                                        pConnection->Execute("INSERT INTO GESTURE
(filename,gest_name,features,width,height,numb,membership) VALUES
('"+filen+"',1,'"+feat+"','"+width+"','"+height+"','"+n_pics+"',0);",NULL,adCmdText);
                                        //pConnection->Execute("INSERT INTO GESTURE
(filename,gest_name,features,width,height,number,membership) VALUES
('"+filen+"',1,'"+feat+"','"+width+"','"+height+"','"+n_pics+"','0');",NULL,adCmdText);
                                }
                                pRstGestures->Close();
                                pConnection->Close();

                        }
                        catch (_com_error &e)
                {
                (char*) e.Description();
                        }

        ::CoUninitialize();
    }
}

int CGestureRecDlg::CopyDB2Buffer()
{
        char vec[600]="";
        int num_pics=0;
        HRESULT  hr = S_OK;
        _bstr_t feat;
        _bstr_t filen;
        _bstr_t width;
        _bstr_t height;


    if(FAILED(::CoInitialize(NULL)))
      return 1;
    if (SUCCEEDED(hr))
    {
        _RecordsetPtr pRstGestures("ADODB.Recordset");
       // Connection String
       _bstr_t strCnn("DSN=gesture;");
        // Open table

        try
        {
                pRstGestures->Open("SELECT * FROM GESTURE ORDER BY ordered;", strCnn,
adOpenStatic, adLockReadOnly, adCmdText);

         pRstGestures->MoveFirst();


                while (!pRstGestures->EndOfFile) {
                                        feat    =((_bstr_t)     pRstGestures->GetFields()->GetItem("features")-
>GetValue());
                                        width   =((_bstr_t)     pRstGestures->GetFields()->GetItem("width")-
>GetValue());
                                        height  =((_bstr_t)     pRstGestures->GetFields()->GetItem("height")-
>GetValue());
                        strcpy(DestBuffer[num_pics].data,feat);
                                        strcpy(DestBuffer[num_pics].width,width);
```

```
                                        strcpy(DestBuffer[num_pics].height,height);
                                        pRstGestures->MoveNext();
                                         num_pics++;
                }
                pRstGestures->Close();
        }
         catch (_com_error &e)
                {
       // Notify the user of errors if any.
       // Pass a connection pointer accessed from the Recordset.
        _variant_t vtConnect = pRstGestures->GetActiveConnection();

        // GetActiveConnection returns connect string if connection
        // is not open, else returns Connection object.

                AfxMessageBox((char*) e.Description());
            //  printf("Errors occured.");
                        //              (char*) e.Description();
       exit(1);
                }

   }
  return 0;
}

void CGestureRecDlg::StringVector2ValueVector(int counter)
{
        int digit,total,width,height,index=0;
        char *tokenPtr;
        total = Rows*Cols;
        width=atoi(DestBuffer[counter].width);
        height=atoi(DestBuffer[counter].height);

        strcpy(vector,DestBuffer[counter].data);
        tokenPtr=strtok(vector, " ");

        while (tokenPtr !=NULL )
        {
                digit=atof(tokenPtr);
                tokenPtr = strtok(NULL," ");
                MatFeatures[counter].data[index]=digit;
                index++;
        }
        MatFeatures[counter].width=width;
        MatFeatures[counter].height=height;

}


void CGestureRecDlg::Weight_String2Weight_Vector()
{
        float weight;
        char *tokenPtr;
        int index=0;

        tokenPtr=strtok(weights, " ");

        while (tokenPtr !=NULL )
        {
                weight=atof(tokenPtr);
```

```
                tokenPtr = strtok(NULL," ");
                weights_val[index]=weight;
                index++;
        }

}



void CGestureRecDlg::CreateFeaturesMatrix()
{
        for (int counter=0;counter<Nframes;counter++)
    {
                StringVector2ValueVector(counter);
        }

}

void CGestureRecDlg::DisplayFeatures(int counter,int Rows,int Cols)
{
  int tsizeX = MatFeatures[counter].width / Cols;
  int tsizeY = MatFeatures[counter].height / Rows;
  int tarea = tsizeX*tsizeY;

  featu.Create(IMG_WIDTH,IMG_HEIGHT,8);
  IplImage *dst=featu.GetImage();
  __try {
    /// the images will use ROI for operations,
    /// the source image uses the ROI to calculate L1 norm,
    /// the destination image uses the ROI to set value
        IplROI roiDest = { 0, 0,0, tsizeX,tsizeY };
//      src->roi = &roiSource;
        dst->roi = &roiDest;

        //from the second place exist features of greyscale
        // before this there is the SizeX and the SizeY of the image
        int index=1; //because the place 0 is for the size ratio
    /// for each ROI
    for( int y=0; y<Rows; ++y ) {
//        roiSource.yOffset = tsizeY * y + minY;
                roiDest.yOffset = tsizeY * y;  // fixed in place
      for( int x=0; x<Cols; ++x ) {
//          roiSource.xOffset = tsizeX * x + minX;
                        roiDest.xOffset = tsizeX * x; // fixed in place
         /// get mean value from source
         int value = MatFeatures[counter].data[index];
         /// put this value to destination
         cvSet(dst,cvScalar(value));
                        index++;
                }
        }
  }
  __finally {
    /// preserve automatic variable and free memory
    dst->roi = NULL;
        OnPaint();
        featu.Destroy();
        }
}
```

```
//Creates a matrix of N clusters containing features vectors of N
// random pictures
void CGestureRecDlg::RandomClusters(int Nclusters,int NumFrames)
{

	bool sign[20000];
	int Upper=NumFrames;
	float r;
	int ran=0;
	for (int index=0;index<NumFrames;index++)
		sign[index]=false;   //sets all to false, in order to know
                                                        // which number already been choosen
	for (index=0;index<Nclusters;index++)
	{
		r=((double) rand() / (double) (RAND_MAX+1));
		ran = (int) (r*Upper);
		//ran=rand() % NumFrames;
		while (sign[ran]==true)
		{
		  r=((double) rand() / (double) (RAND_MAX+1));
		  ran = (int) (r*Upper);
		  //ran=rand() % NumFrames;
		}
		sign[ran]=true;
		memcpy(Ci[index].data,MatFeatures[ran].data,FeatureLen*4);
	}
}

//Creates a matrix of N clusters containing features vectors of the last cycle
// random pictures
void CGestureRecDlg::RandomOneCluster(int Nclusters,int NumFrames)
{

	int Upper=grab_cycle;
	float r;
	int ran=0,rando;

	r=((double) rand() / (double) (RAND_MAX+1));
	r=((double) rand() / (double) (RAND_MAX+1));

	ran = (int) (r*Upper);

	rando = NumFrames-ran;
	int index = Nclusters;
	memcpy(Ci[index].data,MatFeatures[rando].data,FeatureLen*4);

}


// Euclidian Distance between the cluster i(vector) and the cluster
// j (vector).

float CGestureRecDlg::D(int i,int j)
{
	float u=0;
	CvMat PointI = cvMat(1,FeatureLen,CV_MAT32F,NULL);
	CvMat PointJ = cvMat(1,FeatureLen,CV_MAT32F,NULL);
	CvMat PointDiff = cvMat(1,FeatureLen,CV_MAT32F,NULL);
	//CvMat Result = {1,1,CV_MAT32F,0,NULL};
```

```
	CvMat Result = cvMat(1,1,CV_MAT32F,NULL);

	cvmAlloc(&PointI);
	cvmAlloc(&PointJ);
	cvmAlloc(&PointDiff);
	cvmAlloc(&Result);

	float *pI = PointI.data.fl;
	float *pJ = PointJ.data.fl;

	memcpy(pI,Ci[i].data,FeatureLen*4);
	//memcpy(pI,Ci[i].data,sizeof(Ci[i].data));

	//for (int index=0;index<FeatureLen;index++)
	//		cvmSet(&PointI,0,index ,Ci[i].data[index]);

	memcpy(pJ,MatFeatures[j].data,FeatureLen*4);
	//memcpy(pJ,MatFeatures[j].data,sizeof(MatFeatures[j].data));

	//for (index=0;index<FeatureLen;index++)
	//		cvmSet(&PointJ,0,index, MatFeatures[j].data[index]);

	cvmSub(&PointI,&PointJ,&PointDiff);
	cvmMulTransposed(&PointDiff,&Result,0);

	u=cvmGet(&Result,0,0);

	cvmFree(&PointJ);
	cvmFree(&Result);
	cvmFree(&PointDiff);
	cvmFree(&PointI);
	return u;
}

float CGestureRecDlg::Find_MiuIJ(int i,int j)
{
	float numerador,denominador;
	float result,acc=0;
	int num=1;

	numerador=D(i,j);
	for (int index=0;index<Nclusters;index++)
	{
		denominador=D(index,j);
		if ((denominador==0) && (numerador!=0)) //centroid very far!
		{
			acc=1;
			num=0;
			break;
		}

		if ((denominador==0) && (numerador==0)) //centroid overlap!!
		{
			acc=1;
			num=1;
			break;
		}

		acc=acc+pow((numerador/denominador),2/(m-1));
	}
```

```
	result=num/acc;
	return result;
}

void CGestureRecDlg::CreateMembership()
{
	float acc=0,u=0;
	float sums[20000];

	for (int j=0; j<Nframes; j++)
	{
		acc=0;
		for (int i=0; i<Nclusters; i++)
		{
			u=Find_MiuIJ(i,j);
			cvmSet(&Uij,i,j,u);
			acc=acc+u;
		}
		sums[j]=acc;
	}
}


void CGestureRecDlg::Find_Ci(int i)
{
	float u_ij=0,Acc_Denominador=0,val=0;

	CvMat uij = cvMat(1,1,CV_MAT32F,NULL);
	CvMat multi = cvMat(FeatureLen,1,CV_MAT32F,NULL);
	CvMat Xj= cvMat(FeatureLen,1,CV_MAT32F,NULL);
	CvMat Acc_Numerador= cvMat(FeatureLen,1,CV_MAT32F,NULL);
	CvMat ci=cvMat(FeatureLen,1,CV_MAT32F,NULL);

	cvmAlloc(&uij);
	cvmAlloc(&multi);
	cvmAlloc(&Xj);
	cvmAlloc(&Acc_Numerador);
	cvmAlloc(&ci);

	cvmSetZero(&Acc_Numerador);
	Acc_Denominador=0;

	for (int j=0;j<Nframes;j++)
	{

	// remeber to improve this loop to an array of vectors using cvMatArray
	// ontherways it'll remains very slowly
		for (int index=0;index<FeatureLen;index++)
			cvmSet(&Xj,index,0, MatFeatures[j].data[index]);

		u_ij=cvmGet(&Uij,i,j);
		u_ij=pow(u_ij,m);
		cvmScale(&Xj,&multi,u_ij);
		cvmAdd(&Acc_Numerador,&multi,&Acc_Numerador);
	}

	for (j=0;j<Nframes;j++)
	{

		u_ij=cvmGet(&Uij,i,j);
```

```
                u_ij=pow(u_ij,m);
                Acc_Denominador=Acc_Denominador+u_ij;
        }

        Acc_Denominador=1/Acc_Denominador;
        cvmScale(&Acc_Numerador,&ci,Acc_Denominador);

        for (int index=0;index<FeatureLen;index++)
        {
                val=cvmGet(&ci,index,0);
                Ci[i].data[index]=val;
        }

// clean pointers!!!
        cvmFree(&uij);
        cvmFree(&multi);
        cvmFree(&Xj);
        cvmFree(&Acc_Numerador);
        cvmFree(&ci);
}

void CGestureRecDlg::CreateCentroids()
{
        for (int i=0;i<Nclusters;i++)
                Find_Ci(i);
}

float CGestureRecDlg::CostFunction()
{
        float first,second,acc,total=0;

        for (int i=0;i<Nclusters;i++)
        {
                acc=0;
                for(int j=0;j<Nframes;j++)
                {
                        first=cvmGet(&Uij,i,j);
                        first=pow(first,m);
                        second=D(i,j);
                        second=pow(second,1);
                        // Sholud be second=pow(second,1);
                        // but I didn't use the root in the calculation of the ditance
                        // so now, I can eliminate the power 2
                        acc=acc+first*second;
                }
                total=total+acc;
        }
        return total;
}

void CGestureRecDlg::Membership2DB()
{
        float u,max=0;
        int max_i;
        char vec[600]="";
        char number_pics[50]="";
        char member[500]="";
        char Su[100]="";
        char Suu[15]="";
        char gestu[100]="";
```

```
        char s_max[100]="";

        HRESULT  hr = S_OK;
        _bstr_t n_pics;
        _bstr_t memberF;
        _bstr_t centro;
        _bstr_t bs_max;

      if(FAILED(::CoInitialize(NULL)))
        return;

      if (SUCCEEDED(hr))
      {
            // Define ADO object pointers.
       // Initialize pointers on define.
      _RecordsetPtr pRstGestures = NULL;
      _ConnectionPtr pConnection  = NULL;

      HRESULT hr = S_OK;

     //Replace Data Source value with your server name.
      _bstr_t strCnn("DSN=gesture;");
       _bstr_t strMessage;


                  try
                  {
                        //Open a connection
                        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                        pConnection->Open(strCnn,"","",adConnectUnspecified);


                        //Open gestures table
                        TESTHR(pRstGestures.CreateInstance(__uuidof(Recordset)));

                        //You have to explicitly pass the Cursor type and LockType to the Recordset here
                        pRstGestures->Open("gesture",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);
                        for(int j=0;j<Nframes;j++)
                        {
                          strcpy(Su,"");
                          strcpy(member,"");
                          max=0;

                          for (int i=0;i<Nclusters;i++)
                          {
                                u=cvmGet(&Uij,i,j);
                        //      if ((u*1000-floor(u*1000)) > 0.5)
                        //      {
                        //            u=ceil(u*1000);
                                      //u=u/1000;    //added
                        //      }
                        //      else
                        //            u=floor(u*1000);
                                 //u=u/1000;   //added

                                if (u>max)
                                {
                                      max=u;
                                      max_i=i;
```

```
                                        }


                                        sprintf(Su,"%f",u);
                                        strncpy(Suu,Su,5);
                                        strcat(member,Suu);
                                        strcat(member," ");


                                        sprintf(s_max,"%d",max_i);
                                        bs_max=s_max;
                                        //sprintf(Su,"%d",(int)u);
                                        //strcat(member,Su);
                                        //strcat(member," ");

                                    }
                                    sprintf(number_pics,"%d",j);
                                    n_pics=number_pics;
                                    memberF=member;
                                    pConnection->Execute("UPDATE GESTURE SET
membership='"+memberF+"',gest_name='"+bs_max+"' WHERE numb='"+n_pics+"';",NULL,adCmdText);

                                }

                                pRstGestures->Close();
                                pConnection->Close();

                        }
                        catch (_com_error &e)
                {
                (char*) e.Description();
                        }

            ::CoUninitialize();
        }
    }

    void CGestureRecDlg::Centroid2DB()
    {
            char ci[500]="",c[15]="",s_i[15]="";



            HRESULT  hr = S_OK;
            _bstr_t centro;
            _bstr_t bs_i;

      if(FAILED(::CoInitialize(NULL)))
        return;

      if (SUCCEEDED(hr))
      {
              // Define ADO object pointers.
       // Initialize pointers on define.
      _RecordsetPtr pRstGestures = NULL;
      _ConnectionPtr pConnection  = NULL;

      HRESULT hr = S_OK;

     //Replace Data Source value with your server name.
```

```
    _bstr_t strCnn("DSN=gesture;");
    _bstr_t strMessage;


                try
                {
                        //Open a connection
                        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                        pConnection->Open(strCnn,"","",adConnectUnspecified);


                        //Open gestures table
                        TESTHR(pRstGestures.CreateInstance(__uuidof(Recordset)));

                        //You have to explicitly pass the Cursor type and LockType to the Recordset here
                        pRstGestures->Open("centroid",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);

                        pConnection->Execute("DELETE * FROM CENTROID;",NULL,adCmdText);

                        for (int i=0;i<Nclusters;i++)
                        {
                                strcpy(ci,"");
                                for (int index=0;index<FeatureLen;index++)
                                {
                                        sprintf(c,"%d",(int)Ci[i].data[index]);
                                        strcat(ci,c);
                                        strcat(ci," ");
                                }

                                sprintf(s_i,"%d",i);
                                bs_i=s_i;
                                centro=ci;
                                pConnection->Execute("INSERT INTO CENTROID (gest_num,center)
VALUES ('"+bs_i+"','"+centro+"');",NULL,adCmdText);
                        }
                        pRstGestures->Close();
                        pConnection->Close();

                }
                catch (_com_error &e)
        {
           (char*) e.Description();
                }

        ::CoUninitialize();
    }

}

void CGestureRecDlg::Cost2DB(float cost)
{
        char s_cost[30]="";

        HRESULT  hr = S_OK;
        _bstr_t bs_cost;

    if(FAILED(::CoInitialize(NULL)))
      return;
```

```
  if (SUCCEEDED(hr))
  {
        // Define ADO object pointers.
   // Initialize pointers on define.
  _RecordsetPtr pRstGestures = NULL;
  _ConnectionPtr pConnection  = NULL;

  HRESULT hr = S_OK;

 //Replace Data Source value with your server name.
  _bstr_t strCnn("DSN=gesture;");
  _bstr_t strMessage;


                try
                {
                        //Open a connection
                        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                        pConnection->Open(strCnn,"","",adConnectUnspecified);


                        //Open gestures table
                        TESTHR(pRstGestures.CreateInstance(__uuidof(Recordset)));

                        //You have to explicitly pass the Cursor type and LockType to the Recordset here
                        pRstGestures->Open("cost",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);
                        sprintf(s_cost,"%f",cost);
                        bs_cost=s_cost;
                        pConnection->Execute("INSERT INTO COST (cost) VALUES
('"+bs_cost+"');",NULL,adCmdText);
                        pRstGestures->Close();
                        pConnection->Close();

                }
                catch (_com_error &e)
        {
           (char*) e.Description();
                }
        ::CoUninitialize();
  }
}

int CGestureRecDlg::DB2Centroid()
{

       char vec[600]="";
       int num_pics=0,number=0;
       HRESULT  hr = S_OK;
       _bstr_t gest_num;
       _bstr_t center;

       int digit,index=0;
       char *tokenPtr;


  if(FAILED(::CoInitialize(NULL)))
    return 1;
  if (SUCCEEDED(hr))
  {
```

```
        _RecordsetPtr pRstGestures("ADODB.Recordset");
        // Connection String
        _bstr_t strCnn("DSN=gesture;");
         // Open table
                try
                {
                pRstGestures->Open("SELECT * FROM CENTROID ORDER BY gest_num;", strCnn,
adOpenStatic, adLockReadOnly, adCmdText);

         pRstGestures->MoveFirst();


                 while (!pRstGestures->EndOfFile)
                                {
                                        gest_num =((_bstr_t) pRstGestures->GetFields()-
>GetItem("gest_num")->GetValue());
                                        center =((_bstr_t) pRstGestures->GetFields()->GetItem("center")-
>GetValue());

                                        number=atoi(gest_num);

                                        strcpy(vec,center);
                                        tokenPtr=strtok(vec, " ");
                                        index=0;

                                        while (tokenPtr !=NULL )
                                        {
                                                digit=atof(tokenPtr);
                                                tokenPtr = strtok(NULL," ");
                                                Ci[number].data[index]=digit;
                                                index++;
                                        }


                                pRstGestures->MoveNext();
                                        num_pics++;
                 }
                 pRstGestures->Close();
                }
                 catch (_com_error &e)
                {
       // Notify the user of errors if any.
       // Pass a connection pointer accessed from the Recordset.
        _variant_t vtConnect = pRstGestures->GetActiveConnection();

        // GetActiveConnection returns connect string if connection
        // is not open, else returns Connection object.

                AfxMessageBox((char*) e.Description());
            //  printf("Errors occured.");
                        //          (char*) e.Description();
                exit(1);
        }
    }
    return index;
}

int CGestureRecDlg::DB2Membership()
{
                char vec[600]="";
```

```
        int num_pics=0,number=0;
        HRESULT  hr = S_OK;
        _bstr_t numb;
        _bstr_t membership;

        int index=0;
        float acc=0,digit;
        char *tokenPtr;


      if(FAILED(::CoInitialize(NULL)))
        return 1;
      if (SUCCEEDED(hr))
      {
          _RecordsetPtr pRstGestures("ADODB.Recordset");
         // Connection String
         _bstr_t strCnn("DSN=gesture;");
          // Open table

                    try
                    {
                    pRstGestures->Open("SELECT * FROM GESTURE ORDER BY ordered;", strCnn,
adOpenStatic, adLockReadOnly, adCmdText);

           pRstGestures->MoveFirst();
                              int u=0;

                     while (!pRstGestures->EndOfFile)
                                        {
                                                  numb =((_bstr_t) pRstGestures->GetFields()->GetItem("numb")-
>GetValue());
                                                  membership =((_bstr_t) pRstGestures->GetFields()-
>GetItem("membership")->GetValue());

                                                  number=atoi(numb);

                                                  strcpy(vec,membership);
                                                  tokenPtr=strtok(vec, " ");
                                                  index=0;
                                                  acc=0;

                                                  while (tokenPtr !=NULL )
                                                  {
                                                            digit=atof(tokenPtr);
                                                            tokenPtr = strtok(NULL," ");
                                                            cvmSet(&Uij,index,number,digit);  // delete 1000
                                                            u=cvmGet(&Uij,index,number);
                                                            index++;
                                                            acc=acc+digit;  // delete 1000
                                                  }

                                        pRstGestures->MoveNext();
                                                   num_pics++;
                     }
                     pRstGestures->Close();
                    }
           catch (_com_error &e)
                    {
          // Notify the user of errors if any.
          // Pass a connection pointer accessed from the Recordset.
```

```
	_variant_t vtConnect = pRstGestures->GetActiveConnection();

	// GetActiveConnection returns connect string if connection
	// is not open, else returns Connection object.

			AfxMessageBox((char*) e.Description());
		//  printf("Errors occured.");
				//		(char*) e.Description();
			exit(1);

			}
  }
  return 0;
}

void CGestureRecDlg::CopyVector2Mat(int counter,short int flag)
{
	if (flag!=3)
			memcpy(MatFeatures[counter].data,IntVector,FeatureLen*4);
	else
			memcpy(MatFeatures[Nframes].data,IntVector,FeatureLen*4);
}

void CGestureRecDlg::CreateNewMembership(int j,short int flag)
{
	float acc=0,u=0;
	float sums;
	acc=0;
	if (flag==3) j=Nframes;

	for (int i=0; i<Nclusters; i++)
	{
			u=Find_MiuIJ(i,j);
			cvmSet(&Uij,i,j,u);
			u=cvmGet(&Uij,i,j);
			acc=acc+u;
	}
	sums=acc;
}

void CGestureRecDlg::NewMembership2DB(short int flag)
{
	float u,max=0;
	int max_i,num_pics=0;
	char vec[600]="";
	char number_pics[50]="";
	char member[500]="";
	char Su[15]="";
	char Suu[15]="";
	char gestu[15]="";
	char s_max[15]="";

	HRESULT  hr = S_OK;
	_bstr_t n_pics;
	_bstr_t memberF;
	_bstr_t centro;
	_bstr_t bs_max;
	_bstr_t feat;
	_bstr_t filen;
	_bstr_t width;
```

```
        _bstr_t height;


      if((FAILED(::CoInitialize(NULL)))|| (flag==3))
        return;

      if (SUCCEEDED(hr))
      {
            // Define ADO object pointers.
       // Initialize pointers on define.
      _RecordsetPtr pRstGestures = NULL;
      _ConnectionPtr pConnection  = NULL;

      HRESULT hr = S_OK;

     //Replace Data Source value with your server name.
      _bstr_t strCnn("DSN=gesture;");
       _bstr_t strMessage;


                    try
                    {
                              //Open a connection
                              TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                              pConnection->Open(strCnn,"","",adConnectUnspecified);


                              //Open gestures table
                              TESTHR(pRstGestures.CreateInstance(__uuidof(Recordset)));

                              //You have to explicitly pass the Cursor type and LockType to the Recordset here
                              pRstGestures->Open("actual",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);
                              for(int j=Nframes;j<(Nframes+NewFrames);j++)
                              {
                               strcpy(Su,"");
                               strcpy(member,"");
                               max=0;

                               for (int i=0;i<Nclusters;i++)
                               {
                                        u=cvmGet(&Uij,i,j);
                                        if (u>max)
                                        {
                                                  max=u;
                                                  max_i=i;
                                        }

                                 sprintf(Su,"%f",u);
                                        strncpy(Suu,Su,5);
                                        strcat(member,Suu);
                                        strcat(member," ");

                               }

                               feat=Buffer[j].data;
                               filen=Buffer[j].file;
                               width=Buffer[j].width;
                               height=Buffer[j].height;
```

```
                                sprintf(number_pics,"%d",num_pics);
                                n_pics=number_pics;
                                sprintf(s_max,"%d",max_i);
                                bs_max=s_max;
                                memberF=member;

                                pConnection->Execute("INSERT INTO ACTUAL
(filename,gest_name,features,width,height,numb,membership) VALUES
('"+filen+"','"+bs_max+"','"+feat+"','"+width+"','"+height+"','"+n_pics+"','"+memberF+"');",NULL,adCmdText);
                                num_pics++;
                               }

                               pRstGestures->Close();
                               pConnection->Close();

                        }
                        catch (_com_error &e)
                {
                (char*) e.Description();
                        }

            ::CoUninitialize();
          }
        }

        void CGestureRecDlg::DrawGraphico(int j,short int flag)
        {
         int space1=5;
         int space2=8;
         int color=20;
         float u;

         CvPoint pt1[100];
         CvPoint pt2[100];

         CvPoint p1,p2;

         p1.x=0;
         p1.y=25;
         p2.x=IMG_WIDTH;
         p2.y=24;

         if (flag==3) j=Nframes;

         for (int index=0;index<Nclusters;index++)
         {
              pt1[index].x=space1;
              pt2[index].x=space2;
              space1=space1 + (int)(330/Nclusters);
              space2=space2 + (int)(330/Nclusters);
              pt1[index].y=100;
              u=cvmGet(&Uij,index,j);
              pt2[index].y=100-u*100;
         }

         grafico.Create(IMG_WIDTH,100,24);
         grafico.Fill(RGB(255,255,255));
         IplImage* graphi = grafico.GetImage();

         cvRectangle(graphi,p1,p2,CV_RGB(0,0,255),1);
```

```
 for (index=0;index<Nclusters;index++)
 {
      cvRectangle(graphi,pt1[index],pt2[index],CV_RGB(150,color,color),8);
      color=color+15;

 }

// grafico.Destroy();
}

int CGestureRecDlg::OpenTcpMessage()
{

      short listeningport;          // port to listen on
      short destport;                     // port to send to
      char *desthost;                     // address of destination machine

      listeningport=10001;
      destport=10000;
      desthost="132.72.135.14";

            if (! Listen(listeningport))  // Try to listen to requested port
            {
                  AfxMessageBox("Error listening to port");   // Made a booboo, exit the app
                  return 1;
            }
      return 0;
}

void CGestureRecDlg::CloseTcpMessage()
{

      if ( ListenSocket != INVALID_SOCKET )
            closesocket( ListenSocket );          // close if socket was created
}


void CGestureRecDlg::SendTcpMessage(int j,short int flag,char sTotal[1500],int &cont)
{

      char buffer[100];           // buffer we'll use to store msg read in from stdin
      short listeningport;        // port to listen on
      short destport;                     // port to send to
      char *desthost;                     // address of destination machine
      float u=0,maxU=0;
      int maxIndex;
      char filen[255]="";
      char fpath[255]="";
      time_t long_time;
      long int ii=0;

      listeningport=10001;
      destport=10000;
      desthost="132.72.135.14";

      if (flag==3) j=Nframes; //Stocks the Array always in Nframes posi

      for (int index=0;index<Nclusters;index++)
      {
```

```
                u=cvmGet(&Uij,index,j);
                if (u>maxU)
                {
                        maxU=u;
                        maxIndex=index;
                }
        }

        if (maxU>threshold)
                sprintf(buffer,"%d",maxIndex);
        else
                sprintf(buffer,"%d",-1);

        strcat(buffer," ");
        strcat(sTotal,buffer);
        cont++;
        if (cont==140) // A BUNCH OF n GESTURES IS SENT TO THE SERVER
        {
                SendMsg( sTotal, strlen(sTotal), desthost, destport );  // Forward the msg to destination machine
                cont=0;
                strcpy(sTotal,"");
                ii=time(&long_time);
                ii=ii+j;
                sprintf(filen, "%d", ii);
                strcat(filen,".bmp");
           strcat(fpath,"C:\\OpenCV_projects\\GestureRec\\sequence\\");
                strcat(fpath,filen); // create a filename based in path
//          i.Save(fpath);
        }

}


void CGestureRecDlg::ShowLab(int &conter)
{
        if (conter==7)  // EACH n SNAPS SHOWS THE LAB PICTURE
        {
                lab.Load("C:\\OpenCV_projects\\GestureRec\\dest_pics\\webcam32.jpg",8);
                conter=0;
        }
        conter++;
}

void CGestureRecDlg::OnOK()
{
        EndDialog(0);

        CDialog::OnOK();
}

void CGestureRecDlg::OnCancel()
{
        // TODO: Add extra cleanup here

        CDialog::OnCancel();
}


//* TCP/IP Function
void CGestureRecDlg::EndDialog(int nResult)
```

```
{

	if ( ListenSocket != INVALID_SOCKET )
		closesocket( ListenSocket );		// close if socket was created

	#ifdef _WIN32	// Windows only
		WSACleanup();
	#endif
		cvmFree(&Uij);
}

//* TCP/IP Function
bool CGestureRecDlg::SendMsg( char *Msg, int Len, char *host, short port )
{
	signed int Sent;
	hostent *hostdata;
	if ( atoi(host) )	// Is the host passed in IP format?
	{
		u_long ip = inet_addr( host );
		hostdata = gethostbyaddr( (char *)&ip, sizeof(ip), PF_INET );
	}
	else	// otherwise, assume it's a name
	{
		hostdata = gethostbyname( host );
	}

	if ( !hostdata )
	{
		printf("Error getting host address\n");
		fflush(0);
		return false;
	}

	sockaddr_in dest; // the address of the destination computer
	dest.sin_family = PF_INET;
	dest.sin_addr = *(in_addr *)(hostdata->h_addr_list[0]);
	dest.sin_port = htons( port );
	printf("Message being sent to host %s port %i\n", inet_ntoa(dest.sin_addr), ntohs(dest.sin_port));
	Sent = sendto(ListenSocket, Msg, Len, 0, (sockaddr *)&dest, sizeof(sockaddr_in));

	if ( Sent != Len )
	{
		printf("Error sending UDP packet from listen socket\n");
		fflush(0);
		return false;
	}

	return true;
}

//****** TCP/IP
void *CGestureRecDlg::ListenThread( void *data )
{
	char buf[4096];
	CGestureRecDlg *Comm = (CGestureRecDlg *)data;
	int len = sizeof(Comm->client);
	while(1) // loop forever
	{
		int result = recvfrom( Comm->ListenSocket, buf, sizeof(buf)-1, 0, (sockaddr *)&Comm->client,
(socklen_t *)&len);
```

```
                if ( result > 0 )
                {
                        buf[result] = 0;
                        printf("Message received from host %s port %i\n", inet_ntoa(Comm->client.sin_addr),
ntohs(Comm->client.sin_port));
                        printf(">> %s", buf);
                        fflush(0);
                }       // end check to see if socket read was ok
        }       // end infinite loop
    }

    //****** TCP/IP
    bool CGestureRecDlg::Listen( int PortNum )
    {
        ListenSocket = socket(PF_INET, SOCK_DGRAM, 0);
        if ( ListenSocket == INVALID_SOCKET )
        {
                printf("Error: listen socket creation failed\n");
                fflush(0);
                return false;
        }

        srv.sin_family = PF_INET;
        srv.sin_addr.s_addr = htonl( INADDR_ANY );      // any address
        srv.sin_port = htons( PortNum );

        if ( bind( ListenSocket, (struct sockaddr *)&srv, sizeof(srv)) != 0 )
        {
                printf("Error: bind on listen socket failed\n");
                fflush(0);
                closesocket( ListenSocket );
                return false;
        }

        int ThreadID;   // the listening thread's handle

        #ifdef _WIN32
                DWORD thread;
                ThreadID                =               (int)CreateThread(NULL,                 0,
(LPTHREAD_START_ROUTINE)(CGestureRecDlg::ListenThread), (void *)this, 0, &thread);
                ThreadID = ThreadID ? 0 : 1;        // reverse the value for Windows
        #else   // not windows machine
                pthread_t thread;
                ThreadID = pthread_create(&thread, 0, CComm::ListenThread, (void *)this);
        #endif

        if(ThreadID)    // if failed creating thread
        {
                printf("Error creating listen thread\n");
                return false;
        }
        else
                return true;
    }

    void CGestureRecDlg::OnProcess()
    {
        CopyDB2Buffer();
```

```
//	if (CLUSTERS==Clusters_inDB()) //if the new number of clusters now is the same from the prev. run (in the DB there is the same number of centroids)
//		Old_FeatureLen=DB2Centroid(); // you can use the centroids that you found instead of randmozing them again

	CreateFeaturesMatrix();
//	for (int index=0;index<Nframes;index++)
//	{
//		DisplayFeatures(index,Rows,Cols);
//		Sleep(300);
//	}
}



void CGestureRecDlg::OnStart()
{

	short int flag=1;  // the flag tells if we do setup=1, run time with
	// saving & DB =2, or just run time=3. // here always is 1
	int counter=0;
	int minX,maxX,minY,maxY;
	imagen.Create(640,480,8);
	blackwh.Create(IMG_WIDTH,IMG_HEIGHT,8);
	i.Create(IMG_WIDTH,IMG_HEIGHT,24);
	im.Create(IMG_WIDTH,IMG_HEIGHT,8);

	IplImage* img =imagen.GetImage();
	IplImage* blac =blackwh.GetImage();
	IplImage *imgIPL=i.GetImage();
	IplImage *tmpIPL=im.GetImage();
    IplImage *sndIPL=cvCreateImage( cvSize( 768, 576 ), IPL_DEPTH_8U, 3 );
	resized.Create(IMG_WIDTH,IMG_HEIGHT,8);
	IplImage *temp = resized.GetImage();
	OnPaint();
	MV1_Open();
	MV1_StartGrabIt();

	while (counter<grab_cycle)
   {
		sndIPL->imageData = (char *)MbufInquire(MilImage,M_HOST_ADDRESS,M_NULL);
		cvResize(sndIPL,imgIPL);
		cvCvtColor(imgIPL,temp,CV_BGR2GRAY);
		//bw_threshold=cvOtsuThreshold(imgIPL);
		cvThreshold( temp, blac, bw_threshold, 255, CV_THRESH_BINARY );
		Bouncing_Box(blac,minX,maxX,minY,maxY);
		RoiNorm(Rows,Cols,blac,minX,maxX,minY,maxY);
		CopyVector2Buffer(counter+Nframes,minX,maxX,minY,maxY,flag,counter);
		OnPaint();
		counter++;
      Sleep(300);
		if (GetAsyncKeyState(VK_ESCAPE) & 0x0001)
		break; // ESCAPE key is currently pressed
	}
	MV1_StopGrabIt();
	MV1_Close();
	CopyBuffer2DB(Nframes+counter);
	Nframes=Pictures_inDB();
	imagen.Destroy();
	blackwh.Destroy();
```

```
	resized.Destroy();
}

void CGestureRecDlg::OnFindClusters()
{
	float result=0,min_cost=9000000000;
	float old_cost=9000000000;
	float cost=8000000000;
	int min_seed;
	int contad=0;
	float epsilon=2;

	cvmAlloc(&Uij);
	cvmSetZero(&Uij);

//	if (CLUSTERS!=Clusters_inDB()|| Old_FeatureLen!=FeatureLen) //if the new number of clusters now is the same from the prev. run (in the DB there is the same number of centroids)
	// you can use the centroids that you found instead of randmozing them again
 //  {
		//for (int seed=1;seed<=20;seed++)
	for (int seed=1;seed<=10;seed++)
		{
			srand(seed);
			RandomClusters(Nclusters, Nframes);
			//for (int index=0;index<10;index++)
			for (int index=0;index<4;index++) //just to make this go faster, but is less accurate than the line above
			{
				CreateMembership();
				CreateCentroids();
				cost=CostFunction();
			}
			if (cost<min_cost)
			{
				min_cost=cost;
				min_seed=seed;
			}
		}

		srand(min_seed);
		RandomClusters(Nclusters, Nframes);
	//}

	while ((abs(cost-old_cost)>=epsilon) && (contad<40))
	{
		old_cost=cost;
		CreateMembership();
		CreateCentroids();
		cost=CostFunction();
		contad=contad+1;
	}

	Centroid2DB();
	Membership2DB();
	Cost2DB(cost);
	cvmFree(&Uij);

}

void CGestureRecDlg::OnLoad_Clusters()
```

```
{
	cvmAlloc(&Uij);
	cvmSetZero(&Uij);
	DB2Centroid();
	DB2Membership();
	OnProcess();

}

// Add a new gesture by re-evaluating the membership value and the centroids. Doesn't Rand all
// from a scratch, just rand the new cluster (between the last cycle of images)

void CGestureRecDlg::OnAdd_Gesture()
{
	float result=0,cost;
	int seed=5;

	Nclusters=Clusters_inDB(); // actual number of clusters in DB
	Nclusters=Nclusters++;
	cvmAlloc(&Uij);
	cvmSetZero(&Uij);

	OnStart();

	DB2Centroid();
	DB2Membership();
	OnProcess();

	srand(seed);
	RandomOneCluster(Nclusters, Nframes);
	for (int index=0;index<10;index++)
	{
		CreateMembership();
		CreateCentroids();
		cost=CostFunction();
	}

	Centroid2DB();
	Membership2DB();
	Cost2DB(cost);
	cvmFree(&Uij);
}

// Automatic Load of images from 1-XXX
// This function call the batch mode without user choice of files. This means that you can prepare a set of BMP files in
// some directory, and instead of grabbing live images to future cluster creation, you just
// load the BMP files for future cluster creation in order from 1 to XXX.

void CGestureRecDlg::AutomaticBatchMode(char file_name[250])
{

	short int flag=1;  // the flag tells if we do setup (learn) =1, run time with
	// saving & DB =2, or just run time=3. // here always is 1
	int counter=0;
	int indice=0;
	int the_contador=100000; //for old images (or smaller set of gestures, this should be 10,000)
	int minX,maxX,minY,maxY;
	char filen[255]="";
	char fpath[255]="";
```

```
	CFileFind finder;
	BOOL bWorking;

	imagen.Create(640,480,8);
	blackwh.Create(IMG_WIDTH,IMG_HEIGHT,8);
	i.Create(IMG_WIDTH,IMG_HEIGHT,24);
	im.Create(IMG_WIDTH,IMG_HEIGHT,8);

	IplImage* img =imagen.GetImage();
	IplImage* blac =blackwh.GetImage();
	IplImage *imgIPL=i.GetImage();
	IplImage *tmpIPL=im.GetImage();
	IplImage *sndIPL=cvCreateImage( cvSize( 768, 576 ), IPL_DEPTH_8U, 3 );


	resized.Create(IMG_WIDTH,IMG_HEIGHT,8);
	IplImage *temp = resized.GetImage();
	//OnPaint();
	//int indice=0;
	bWorking=0;

	while (indice<number_of_files)
//	for (int indice=0;indice<number_of_files;indice++)
	{
		bWorking=0;

		while (bWorking==0)
		{
			strcpy(filen,"");
			strcpy(fpath,"");
			sprintf(filen, "%d", the_contador);
			strcat(filen,".bmp");
			strcat(fpath,file_name);
			strcat(fpath,filen);

			bWorking=finder.FindFile(fpath);

			if (the_contador>1110000000)
			{
				AfxMessageBox("Gestures images files not found. TIME OUT!");
				bWorking=1;
				exit(1);
			}

			the_contador=the_contador++;

		}
			i.Load(fpath,8);
			cvCvtColor(imgIPL,temp,CV_BGR2GRAY);
			//bw_threshold=cvOtsuThreshold(temp);
			cvThreshold(temp,blac,bw_threshold,255,CV_THRESH_BINARY);
			Bouncing_Box(blac,minX,maxX,minY,maxY);
			RoiNorm(Rows,Cols,blac,minX,maxX,minY,maxY);
			CopyVector2Buffer(counter+Nframes,minX,maxX,minY,maxY,flag,the_contador-1);
			counter++;
			indice++;
  }

	CopyBuffer2DB(Nframes+counter);
	Nframes=Pictures_inDB();
```

```
	imagen.Destroy();
	blackwh.Destroy();
	resized.Destroy();

}

// Automatic Load of images from 1-XXX
// This function call the batch mode without user choice of files. This means that you can prepare a set of BMP files in
// some directory, and instead of grabbing live images to future cluster creation, you just
// load the BMP files for future cluster creation in order from 1 to XXX.

void CGestureRecDlg::AutomaticTestMode(char file_name[250])
{
	short int flag=2;  // the flag tells if we do setup (learn) =1, run time with
	// saving & DB =2, or just run time=3. // here always is 1
	int counter=Nframes;
	int indice=0;
	int the_contador=100000; //for old images (or smaller set of gestures, this should be 10,000);
	int cont=0,conter=0;
	char sTotal[500];
	char filen[255]="";
	char fpath[255]="";
	CFileFind finder;
	BOOL bWorking=0;

	strcpy(sTotal,"");

	int minX,maxX,minY,maxY;
	imagen.Create(640,480,8);
	blackwh.Create(IMG_WIDTH,IMG_HEIGHT,8);

	i.Create(IMG_WIDTH,IMG_HEIGHT,24);
	im.Create(IMG_WIDTH,IMG_HEIGHT,8);

	IplImage* img =imagen.GetImage();
	IplImage* blac =blackwh.GetImage();
	IplImage *imgIPL=i.GetImage();
	IplImage *tmpIPL=im.GetImage();
	IplImage *sndIPL=cvCreateImage( cvSize( 768, 576 ), IPL_DEPTH_8U, 3 );


	resized.Create(IMG_WIDTH,IMG_HEIGHT,8);
	IplImage *temp = resized.GetImage();

	// Now the part of the choosing files for loading with GUI
		while (indice<number_of_files)
		{
		//for (int indice=0;indice<number_of_files;indice++)
			bWorking=0;

			while (bWorking==0)
			{
				strcpy(filen,"");
				strcpy(fpath,"");
				sprintf(filen, "%d", the_contador);
				strcat(filen,".bmp");
				strcat(fpath,file_name);
				strcat(fpath,filen);
```

```
                        bWorking=finder.FindFile(fpath);
                        the_contador++;

                }

                        i.Load(fpath,8);
                        cvCvtColor(imgIPL,temp,CV_BGR2GRAY);
                        //bw_threshold=cvOtsuThreshold(temp);
                        cvThreshold(temp,blac,bw_threshold,255,CV_THRESH_BINARY);
                        Bouncing_Box(blac,minX,maxX,minY,maxY);
                        RoiNorm(Rows,Cols,blac,minX,maxX,minY,maxY);
                        CopyVector2Buffer(counter,minX,maxX,minY,maxY,flag,the_contador-1);
                        CopyVector2Mat(counter,flag);
                        CreateNewMembership(counter,flag);
                        counter++;
                        indice++;

        }
        NewMembership2DB(flag);
        imagen.Destroy();
        blackwh.Destroy();
        resized.Destroy();

}


// This function call the batch mode. This means that you can prepare a set of BMP files in
// some directory, and instead of grabbing live images to future cluster creation, you just
// load the BMP files for future cluster creation.

void CGestureRecDlg::OnBatchMode()
{

        short int flag=1;  // the flag tells if we do setup (learn) =1, run time with
        // saving & DB =2, or just run time=3. // here always is 1
        int counter=0;
        int minX,maxX,minY,maxY;
        imagen.Create(640,480,8);
        blackwh.Create(IMG_WIDTH,IMG_HEIGHT,8);
        i.Create(IMG_WIDTH,IMG_HEIGHT,24);
        im.Create(IMG_WIDTH,IMG_HEIGHT,8);

        IplImage* img =imagen.GetImage();
        IplImage* blac =blackwh.GetImage();
        IplImage *imgIPL=i.GetImage();
        IplImage *tmpIPL=im.GetImage();
        IplImage *sndIPL=cvCreateImage( cvSize( 768, 576 ), IPL_DEPTH_8U, 3 );


        resized.Create(IMG_WIDTH,IMG_HEIGHT,8);
        IplImage *temp = resized.GetImage();
        OnPaint();
        //MV1_Open();
        //MV1_StartGrabIt();

        int iBufferSize = 300000;
        CFileDialog dlg(TRUE, NULL, NULL,
OFN_FILEMUSTEXIST|OFN_HIDEREADONLY|OFN_ALLOWMULTISELECT,
                "Images (*.jpg, *.bmp)|*.jpg; *.bmp|Windows Bitmap (*.bmp)|*.bmp|JPEG-File (*.jpg)|*.jpg|All
Files (*.*)|*.*||");
```

```
        dlg.m_ofn.lpstrTitle = "My File Dialog";

        dlg.m_ofn.nMaxFile = iBufferSize;
      char* cNewBuffer = new char[iBufferSize];
      dlg.m_ofn.lpstrFile = cNewBuffer;
      dlg.m_ofn.lpstrFile[0] = NULL;


        int result = dlg.DoModal();


        if (result==IDOK)
        {
                POSITION ps=dlg.GetStartPosition();          //
                while (ps)
                {
                        CString name=dlg.GetNextPathName(ps);
                        i.Load(name,8);

                        //sndIPL->imageData                                     =                                       (char
*)MbufInquire(MilImage,M_HOST_ADDRESS,M_NULL);
                        //cvResize(sndIPL,imgIPL);
                        cvCvtColor(imgIPL,temp,CV_BGR2GRAY);
                        //bw_threshold=cvOtsuThreshold(temp);
                        cvThreshold(temp,blac,bw_threshold,255,CV_THRESH_BINARY);
                        Bouncing_Box(blac,minX,maxX,minY,maxY);
                        RoiNorm(Rows,Cols,blac,minX,maxX,minY,maxY);
                        CopyVector2Buffer(counter+Nframes,minX,maxX,minY,maxY,flag,counter);
                        OnPaint();
                        counter++;
                        Sleep(20);

                //          AfxMessageBox(dlg.GetNextPathName(ps));//
        }
        }
     delete []cNewBuffer;//

        //MV1_StopGrabIt();
        //MV1_Close();
        CopyBuffer2DB(Nframes+counter);
        Nframes=Pictures_inDB();
        imagen.Destroy();
        blackwh.Destroy();
        resized.Destroy();

    }

    void CGestureRecDlg::OnRunBatchMode()
    {

        short int flag=2;  // the flag tells if we do setup (learn) =1, run time with
        // saving & DB =2, or just run time=3. // here always is 1
        int counter=Nframes;
        int cont=0,conter=0;
        char sTotal[500];
        strcpy(sTotal,"");

        int minX,maxX,minY,maxY;
        imagen.Create(640,480,8);
```

```
	blackwh.Create(IMG_WIDTH,IMG_HEIGHT,8);

	i.Create(IMG_WIDTH,IMG_HEIGHT,24);
	im.Create(IMG_WIDTH,IMG_HEIGHT,8);

	IplImage* img =imagen.GetImage();
	IplImage* blac =blackwh.GetImage();
	IplImage *imgIPL=i.GetImage();
	IplImage *tmpIPL=im.GetImage();
	IplImage *sndIPL=cvCreateImage( cvSize( 768, 576 ), IPL_DEPTH_8U, 3 );


	resized.Create(IMG_WIDTH,IMG_HEIGHT,8);
	IplImage *temp = resized.GetImage();
	OnPaint();
	//MV1_Open();
	//MV1_StartGrabIt();
	OpenTcpMessage();

	// Now the part of the choosing files for loading with GUI

	int iBufferSize = 300000;
	CFileDialog dlg(TRUE, NULL, NULL,
OFN_FILEMUSTEXIST|OFN_HIDEREADONLY|OFN_ALLOWMULTISELECT,
		"Images (*.jpg, *.bmp)|*.jpg; *.bmp|Windows Bitmap (*.bmp)|*.bmp|JPEG-File (*.jpg)|*.jpg|All
Files (*.*)|*.*||");

	dlg.m_ofn.lpstrTitle = "My File Dialog";

	dlg.m_ofn.nMaxFile = iBufferSize;
      char* cNewBuffer = new char[iBufferSize];
      dlg.m_ofn.lpstrFile = cNewBuffer;
      dlg.m_ofn.lpstrFile[0] = NULL;


	int result = dlg.DoModal();


	if (result==IDOK)
	{
		POSITION ps=dlg.GetStartPosition();	//
		while (ps)
		{
			CString name=dlg.GetNextPathName(ps);
			i.Load(name,8);

			//sndIPL->imageData = (char
*)MbufInquire(MilImage,M_HOST_ADDRESS,M_NULL);
			//cvResize(sndIPL,imgIPL);
			cvCvtColor(imgIPL,temp,CV_BGR2GRAY);
			//bw_threshold=cvOtsuThreshold(temp);
			cvThreshold(temp,blac,bw_threshold,255,CV_THRESH_BINARY);
			Bouncing_Box(blac,minX,maxX,minY,maxY);
			RoiNorm(Rows,Cols,blac,minX,maxX,minY,maxY);
			CopyVector2Buffer(counter,minX,maxX,minY,maxY,flag,counter);
			CopyVector2Mat(counter,flag);
			CreateNewMembership(counter,flag);
			DrawGraphico(counter,flag);
			// SendTcpMessage(counter,flag,sTotal,cont);
			ShowLab(conter);
```

```
                        OnPaint();
                        counter++;
                        Sleep(20);
        }
        }
    delete []cNewBuffer;//

        //MV1_StopGrabIt();
        //MV1_Close();
    //CloseTcpMessage();
        NewMembership2DB(flag);
        imagen.Destroy();
        blackwh.Destroy();
        resized.Destroy();

}

int CGestureRecDlg::Input_Parameters(char file_name[250])
{

        int status;
        HRESULT  hr = S_OK;
        _bstr_t rows_s;
        _bstr_t cols_s;
        _bstr_t weights_s;
        _bstr_t clusters_s;
        _bstr_t m_s;
        _bstr_t bw_threshold_s;
        _bstr_t number_of_files_s;
        _bstr_t status_s;
        _bstr_t file_name_s;

    if(FAILED(::CoInitialize(NULL)))
    {
          AfxMessageBox("Problems opening Gesture DB.");
          exit(1);

      return 1;
    }
    if (SUCCEEDED(hr))
    {
        _RecordsetPtr pRstGestures("ADODB.Recordset");
       // Connection String
       _bstr_t strCnn("DSN=gesture;");
        // Open table

                try
                {
                pRstGestures->Open("SELECT * FROM PARAMETER ORDER BY ID;", strCnn, adOpenStatic,
adLockReadOnly, adCmdText);

        //pRstGestures->MoveFirst();
                        pRstGestures->MoveLast();


                while (!pRstGestures->EndOfFile)
                        {
                                rows_s      =((_bstr_t)      pRstGestures->GetFields()->GetItem("rows")-
>GetValue());
```

```
                                        cols_s=((_bstr_t)        pRstGestures->GetFields()->GetItem("cols")-
>GetValue());
                                        weights_s=((_bstr_t)  pRstGestures->GetFields()->GetItem("weights")-
>GetValue());
                                        clusters_s=((_bstr_t)  pRstGestures->GetFields()->GetItem("clusters")-
>GetValue());
                                        m_s=((_bstr_t)           pRstGestures->GetFields()->GetItem("m")-
>GetValue());
                                        bw_threshold_s=((_bstr_t)                pRstGestures->GetFields()-
>GetItem("bw_threshold")->GetValue());
                                        number_of_files_s=((_bstr_t)             pRstGestures->GetFields()-
>GetItem("samples")->GetValue());
                                        status_s=((_bstr_t)      pRstGestures->GetFields()->GetItem("train")-
>GetValue());
                                        file_name_s=((_bstr_t)   pRstGestures->GetFields()->GetItem("path")-
>GetValue());

                                        Rows=atoi(rows_s);
                                        Cols=atoi(cols_s);
                                        //weight=atof(weight_s);
                                        Nclusters=atoi(clusters_s);
                                        m=atof(m_s);
                                        bw_threshold=atoi(bw_threshold_s);
                                        number_of_files=atoi(number_of_files_s);
                                        status = atoi(status_s);
                                    strcpy(weights,weights_s);
                                        strcpy(file_name,file_name_s);

                                pRstGestures->MoveNext();
                            }
                            pRstGestures->Close();
                            }
                             catch (_com_error &e)
                            {
        // Notify the user of errors if any.
        // Pass a connection pointer accessed from the Recordset.
         _variant_t vtConnect = pRstGestures->GetActiveConnection();

         // GetActiveConnection returns connect string if connection
         // is not open, else returns Connection object.

                    AfxMessageBox((char*) e.Description());

                    exit(1);
              //  printf("Errors occured.");
                                //            (char*) e.Description();

                    }
    }
    return status;

}

void CGestureRecDlg::OnCapture_Gesture()
{

    short int flag=3;  // the flag tells if we do setup=1, run time with
        // saving in DB =2, or just run time=3
        int counter=Nframes;
        int cont=0,conter=0; //group of discret gestures numbers
```

```
        char sTotal[500];  // string containig the bunch message to TCP/IP
        strcpy(sTotal,"");  // Initialize
        int minX,maxX,minY,maxY;

        imagen.Create(640,480,8);
        blackwh.Create(IMG_WIDTH,IMG_HEIGHT,8);

        i.Create(IMG_WIDTH,IMG_HEIGHT,24);
        im.Create(IMG_WIDTH,IMG_HEIGHT,8);
        resized.Create(IMG_WIDTH,IMG_HEIGHT,8);

        IplImage* img =imagen.GetImage();
        IplImage* blac =blackwh.GetImage();
        IplImage *imgIPL=i.GetImage();
        IplImage *tmpIPL=im.GetImage();
        IplImage *sndIPL=cvCreateImage( cvSize( 768, 576 ), IPL_DEPTH_8U, 3 );
        IplImage *temp = resized.GetImage();

        OnPaint();
        MV1_Open();
        MV1_StartGrabIt();
        OpenTcpMessage();

        while (counter<Nframes+NewFrames)
    {
                sndIPL->imageData = (char *)MbufInquire(MilImage,M_HOST_ADDRESS,M_NULL);
                cvResize(sndIPL,imgIPL);
                cvCvtColor(imgIPL,temp,CV_BGR2GRAY);
                //bw_threshold=cvOtsuThreshold(temp);
                cvThreshold(temp,blac,bw_threshold,255,CV_THRESH_BINARY);
                Bouncing_Box(blac,minX,maxX,minY,maxY);
                RoiNorm(Rows,Cols,blac,minX,maxX,minY,maxY);
                CopyVector2Buffer(counter,minX,maxX,minY,maxY,flag,counter);
                CopyVector2Mat(counter,flag);
                CreateNewMembership(counter,flag);
                DrawGraphico(counter,flag);
                SendTcpMessage(counter,flag,sTotal,cont);   //++++++++++++++++++++++++++COMMENT IT
FOR EXTERNAL RUN
                ShowLab(conter);
                OnPaint();
                counter++;
                if (GetAsyncKeyState(VK_ESCAPE) & 0x0001)                break; // ESCAPE key is currently
pressed

        }
        MV1_StopGrabIt();
        MV1_Close();
        CloseTcpMessage();
        NewMembership2DB(flag);
        imagen.Destroy();
        blackwh.Destroy();
        resized.Destroy();

    }
```

QAPI

This system uses the matrices obtained from ergonomic studies and the results obtained from the GestureRec system to find the best GV. The matrices used as inputs are the intuitiveness, stress, duration and frequency. For each subset of gestures, the recognition accuracy is calculated using the CMD or the DCM methods. For this subset, the associations to the commands are found, in such a way so the intuitiveness and comfort are maximized. For this, the quadratic assignment problem (QAP) is used to model this problem. It implementation code is based on an enhanced simulated annealing scheme proposed by Mr. Eric Taillard. A flowchart of the system is presented in Figure K.2.

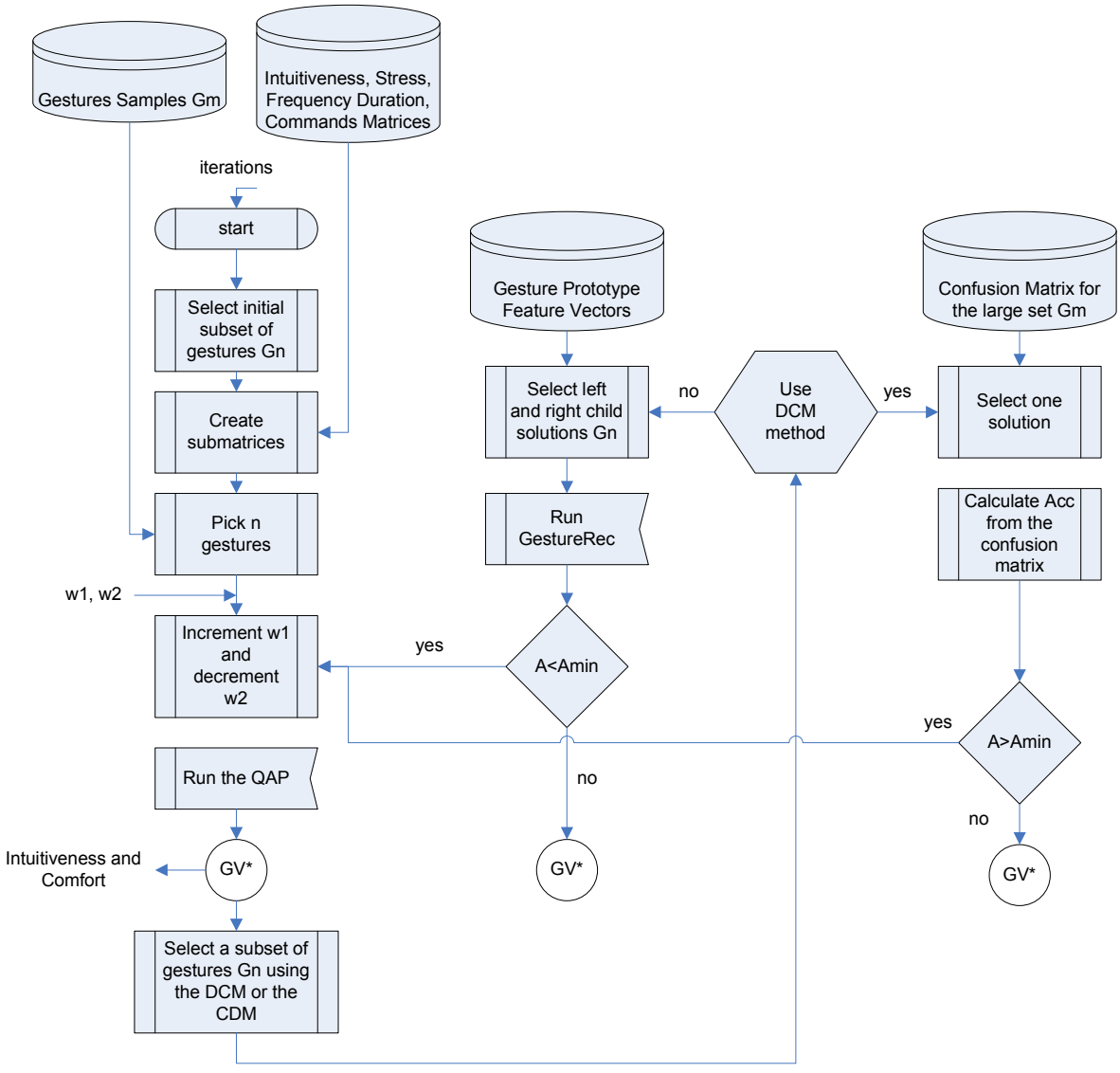


Figure K.2. Flowchart of the QAPI system

class Gmanager	Name	Description
Members		
public:		
	int *confus_vect;	Keeps here the confusion matrix (in vector form) of the last run of the name_gesture_VMR application
	int commando;	Number of commands
	char *vec;	The centroids vector, string
	int *vector;	The centroids vector, numeral
	float acc;	The Accuracy of the last run of the name_gesture_VMR application
	float fcm_time;	time in seconds that takes to run the FCM
	int confused_A,confused_B;	The two most confused gestures
	long *gestures_subset;	Vector of gestures ordered from low to high
	long *gestures_matched;	Vector of gestures not ordered, matched with their index (command)
	int gestures;	Total number of gestures in the reduced master set
Methods		
	Gmanager(int commands,int total_gestures);	Object constructor, receives the subset of gestures indices, and also the number of total commands
	virtual ~Gmanager();	Destructor
	void FindAccuracy();	Call the GestureRec system to find the Accuracy
	int RenameLabelsDB(long *gestures_matched);	After the tree process, the centroids of the gesture DB are re-named according to the QAP result match.
	void RunGL_map();	Draws an image with all the gestures and the commands written on it
private:		
	bool RunProcessAndWait(char *sCmdLine, char *sRunningDir, int *nRetValue);	This calls a process is ran as a console window to run the Gesture VMR application
	int DB2Accuracy();	Extract from the DB Gestures, the accuracy and confusion matrix data
	void FindMostConf();	Find the two most confused gestures
	void Acc2DB();	Copy the Accuracy and 2 most confused gestures to the DB

class OrganizeImages	Name	Description
public:		
Members		
	long *gestures_subset;	Pointer to subset of gestures indices
Methods		
	OrganizeImages(int commands);	Constructor receives the number of commands
	virtual ~OrganizeImages();	The main procedure, call all the others
	void MovePics();	Moves the subset of gesture images to the working folder
private:		
Members		
	int actions;	Number of commands
	char *pics_path;	Path of the gesture images
Methods		
	int DB2Path(int index);	Get a index of a gesture, and set the path for all the pictures samples of that gesture
	void Move2Temp(int m_from);	Rename images in temp directory

class qap	Name	Description
public:		
Members		
	long n,nb_iterations, nb_res;	nb - Number of iterations , nb - Number of iterations
	long Z1, Z2, Zt;	Zt-total cost, Z1 comfort measure, Z2 intuitve measure
	long *p;	Permutation, Result of the QAP
	long ** a, ** b,** w, ** d, ** ic;	Pointers to: Matrix a is F, matrix b is S', matrix w is I, ic is the complementary intuitveness matrix IC Pointers to: Matrix d is D (duration), Important remark: IC is a matrix where the rows are the gestures, and the columns pairs of complementary commands. the first two columns are gestures g1 and g2, the following ones are pair of complementary commands. To convert this to a fast access matrix, we create a matrix where the column index is obtianed by: g1*commands+g2. The rows are the values for Complementary pairs of commands.
	int *oG,*oC;	Pointers to vector of opposed of gestures, and to vector to opposed commands
	float k1,k2,k3;	weights for the intuitiveness, for the stress, and for the complementary intuitiveness respectively
	double h2;	coefficient to reduce the size of the stress to match the range of intuitveness
	double tperiod;	period of time to solve the QAP (all the iterations included)
	void solve();	Solve the QAP problem. Maximization of Total comfort and intutiveness
Methods		
	qap(long N);	Contructor of the qap object, receives numfer of commands

	virtual ~qap();	Destructor of the qap object
private:		
Members		
	long n_max, infini, nb_iter_initialisation, no_res;	Internal parameters of the simulated annealing
	long Cout;	Value of the goal function
Methods		
	double max(double a, double b);	Finds maximum between two values
	double min(double a, double b);	Finds minimum between two values
	void swap(long &a, long &b);	Swap content of the cells, between the places a and b
	double temps();	CPU time in milliseconds
	double mon_rand();	Returns a random value within limits
	long unif(long low, long high);	Returns a random value from uniform distribution
	long calc_delta_complet2(long n, long ** a, long ** b, long ** w, long **d, long * p, long r, long s);	Finds the delta increment (step) according the simmulated annealing formula
	long calcule_cout(long n, long ** a, long ** b, long ** w, long **d, long * p);	Calculates the goal function Zt=Z1+Z2
	void calcule_cout_bout(long & co, long & bo,long n, long ** a, long ** b, long ** w, long **d,long * p);	Calculates the goal function for Z1 and Z2 individually
	void tire_solution_aleatoire(long n, long * p);	Swaps in random order the solution vector
	void recuit(long n, long ** a, long ** b, long ** w, long **d,	Main body of the simmulated annealing procedure
	long * meilleure_sol, long & meilleur_cout,	
	long nb_iterations);	

class QAP_DB	**Names**	**Description**
public:		
Members		
	long *gestures_subset;	Data from the QAP object (the combinative solution)
	long *pai;	Data from QAP (weight for direct intuitiveness, for stress, and for comp. intuitiveness)
	float W1,W2,W3;	
	double H2;	coefficient to reduce the size of the stress to be in the same range of the intuitiveness
Methods		
	QAP_DB(long n);	Constructor, includes vector with the indices of the gestures
		from the big matrix and num of nodes
	virtual ~QAP_DB();	Destructor of the main object
	long commands,gestures;	Number of Commands and Gestures in DB
	void Initial();	The indexes of the n gestures in the big matrix
private:		
Members		
	long ** F, ** S,** UI,**D, ** IC;	Big matrices containing all the data in DB

	long ** f, ** s, **ui,**d, **ic;	Small matrices containing just the data to be passed to QAP
	int *oC;	Small vectors of Opposed Commands, and Opposed Gestures, to be passed to QAP
	int *equiv_table;	vector with equivalences between the gestures names, and their order in the task master set, for example: gesture 27 is the 23 in the robotic arm task
	long Z1,Z2,Zt;	Data accessible from the QAP object.
	qap *qap_obj;	Pointer to the QAP object
	int number_comp_gestures;	number of records in the comp intuitive table, this is the number of comp. gestures in the database
	double tperiod;	period of time to solve the QAP
Methods		
	int CandG_inDB();	Find number of gestures, and commands.
	int DB2Matrices();	Copy the matrices data from DB to memory
	void Allocate_Mem();	Allocate memory of all the kinds of matrices
	void ExtractSubMatrix();	Extract the small matrix of size nxn
	void RunQAP();	Run the QAP object using the sub matrices data and some parameters
	void Insert_Results2DB();	Insert the results on the DB
	int renumbered_index(int i);	returns the new index of the gesture of the subset, according to renumbering it from 0 to num. of commands
	int extract_equiv_index(int i);	finds the equiv order number of the gesture number as presented, for ex: the gesture 27, is may be the 22 in the order

class SimilarityMat	**Name**	**Description**
public:		
Members		
	long *gestures_indices;	Pointer to subset of gestures indices
	SimilarityMat(int total_gestures,int n);	Constructor of the object. Receives the number of gestures and commands in the GV
	virtual ~SimilarityMat();	
	float Dist(int i,int j);	Euclidian Distance between the cluster i (vector) and the cluster j (vector).
	int GetDistinct(int j);	You change the gesture j, by a new gesture not included in the GV, but yes in DB.
	int GetIndexOfGesture(int g);	for a given gesture g, we can get it index in the vector gesture_indices
Methods		
	void OrderGestureVector();	Order the gesture vector from low to high, to get always the same Accuracy for the same vector
private:		

Members		
	float Ci[100][200];	Memory to hold the centroids matrix
	void DB2Centroid();	Copy the centroids from the DB to the memory buffer
	int FeatureLen,commands;	Length of features and number of commands
Methods		
	void CreateCentroid2DB(int total_gestures);	Creates the feature averages of each gesture types
	void RunGestureCentroids();	Run execute for Centroids creation
	bool RunProcessAndWait(char *sCmdLine,	Calls a external shell execution process, the GestureRec
	char *sRunningDir,int *nRetValue);	waits until execute die
	int all_gestures;	Number of gestures
	long *gestures_indices_out	gestures not selected in GV but in DB, are signed with 1/ the others are signed with 0

```
// Gmanager.h: interface for the Gmanager class.
//
//////////////////////////////////////////////////////////////////////
#include "windows.h"
#include <process.h>
#include <shellapi.h>
#include "string.h"
//#include "afx.h"

#if !defined(AFX_GMANAGER_H__27D218ED_0630_41DC_BFDC_51F37BB8A63B__INCLUDED_)
#define AFX_GMANAGER_H__27D218ED_0630_41DC_BFDC_51F37BB8A63B__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Gmanager
{
public:
	Gmanager(int commands,int total_gestures); //Constructor, recieves the subset of gestures
	// indices, and also the number of total commands
	virtual ~Gmanager();
	void FindAccuracy(); // Run all the others functions
	int *confus_vect;//Keeps here the confusion matrix (in vector form) of the last run
                                                // of the name_gesture_VMR application
	char *vec;
	int *vector;
	float acc;                          // The Accuracy of the last run of the name_gesture_VMR appl.
	float fcm_time; //time in seconds that takes to run the FCM
	int confused_A,confused_B; //The two most confused gestures
	long *gestures_subset;// vector of gestures ordered from low to high
	long *gestures_matched; //vector of gestures not ordered, matched with their index (command)
	int gestures; //total number of gestures in the reduced master set

	int RenameLabelsDB(long *gestures_matched); //After the tree process, the centroids of the gesture DB//
                                                                //          are re-named according to the QAP result match.
	int commando; //Number of commands

	void RunGL_map(); //draws an image with all the gestures and the commands written on it

private:

//-------------------------------------------------------------
//
//      Run a synchronized other command line EXE. Returns only
// after this exits. The process is runned as a console window.
// Returns Values : TRUE if the process was created
//                  FALSE if not.
// see *nRetValue for the LastError number

	bool RunProcessAndWait(char *sCmdLine,
                char *sRunningDir,
                int *nRetValue);
	void RunGestureName(); // Run the name_gesture_VMR
	int DB2Accuracy();  //Extract from the DB Gestures, the accuracy and confusion data
	void FindMostConf(); //Find the two most confused gestures
	void Acc2DB(); //Pass the Accuracy and 2 most confused gestures to DB
};

#endif // !defined(AFX_GMANAGER_H__27D218ED_0630_41DC_BFDC_51F37BB8A63B__INCLUDED_)
```

```
// OrganizeImages.h: interface for the OrganizeImages class.
//
//////////////////////////////////////////////////////////////////////

//#include <iostream>
//#include <fstream>
#include <windows.h>
#include <stdio.h>
#include <shellapi.h>
#include "string.h"

#if
!defined(AFX_ORGANIZEIMAGES_H__B3F1CA14_7BCA_4A9F_B861_A3645D69077D__INCLUDED_)
#define AFX_ORGANIZEIMAGES_H__B3F1CA14_7BCA_4A9F_B861_A3645D69077D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class OrganizeImages
{
public:
	long *gestures_subset; //Pointer to subset of gestures indices
	OrganizeImages(int commands);
	virtual ~OrganizeImages();
	void MovePics(); //The main procedure, call all the others


private:
	int actions; //Number of commands
	char *pics_path;
	int DB2Path(int index); //Get a index of a gesture, and set the path for all
	// the pictures samples of that gesture
	void Move2Temp(int m_from); //Rename images in temp directory

};

#endif                                                                                    //
!defined(AFX_ORGANIZEIMAGES_H__B3F1CA14_7BCA_4A9F_B861_A3645D69077D__INCLUDED_)
```

```
// qap.h: interface for the qap class.
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <time.h>
#include <memory>
#if !defined(AFX_QAP_H__9F486B0B_1D89_46FB_B988_8DF5ECD8B4EF__INCLUDED_)
#define AFX_QAP_H__9F486B0B_1D89_46FB_B988_8DF5ECD8B4EF__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
enum booleen {faux, vrai};
class qap
{
public:
        qap(long N);
        virtual ~qap();
        long  n,nb_iterations, nb_res; // n - Number of nodes ,
        // nb - Number of iterations , nb - Number of iteration a
        long Z1, Z2, Zt; // Zt-total cost, Z1 comfort measure, Z2  intuitve measure
        long *p;  // Permutation, Result of the QAP
        long ** a, ** b,** w, ** d, ** ic; // Pointers to: Matrix a is F, matrix b is S', matrix w is I, ic is the complementary intuitveness matrix IC
                                // Pointers to: Matrix d is D (duration),
        //important remark: IC is a matrix where the rows are the gestures, and the columns pairs of complementary commands.
        //the first two columns are gestures g1 and g2, the following ones are pair of complementary commands.
        // to convert this to a fast access matrix, we create a matrix where the column index is obtianed by: g1*commands+g2.  The rows are the values for
        // complementary pairs of commands.
        int *oG,*oC; //Pointers to vector of opposed of gestures, and to vector to opposed commands
        float k1,k2,k3; //weights for the intuitiveness, for the stress, and for the complementary intuitiveness respectively
        double h2; //coefficient to reduce the size of the stress to match the range of intuitveness
        double tperiod; //period of time to solve the QAP (all the iterations included)
        void solve();  // Solve the QAP problem. Maximization of Total comfort and intutiveness
private:
        long n_max, infini, nb_iter_initialisation, no_res;
        long Cout;
        //long maxi(long a, long b);
        long max(long a, long b);
        double max(double a, double b);
        long min(long a, long b);
        double min(double a, double b);
                void swap(long &a, long &b);
        double temps();
//      void a_la_ligne(ifstream & fichier_donnees);
        double mon_rand();
        long unif(long low, long high);
//      void lire(long &n, long ** a,long ** b,long ** w);
        long calc_delta_complet2(long n, long ** a, long ** b,
                                                long ** w, long **d, long * p, long r, long s);
        long calcule_cout(long n, long ** a, long ** b, long ** w, long **d, long * p);
    void calcule_cout_bout(long & co, long & bo,long n, long ** a, long ** b, long ** w, long **d,long * p);
        void tire_solution_aleatoire(long n, long * p);
        void recuit(long n, long ** a, long ** b, long ** w, long **d,
          long * meilleure_sol, long & meilleur_cout,
          long nb_iterations);
};
#endif // !defined(AFX_QAP_H__9F486B0B_1D89_46FB_B988_8DF5ECD8B4EF__INCLUDED_)
```

```
// QAP_DB.h: interface for the QAP_DB class.
//
//////////////////////////////////////////////////////////////////////
#include "qap.h"

#if !defined(AFX_QAP_DB_H__1B9E5B52_1EA9_4A26_A9EB_AC1233E8BE46__INCLUDED_)
#define AFX_QAP_DB_H__1B9E5B52_1EA9_4A26_A9EB_AC1233E8BE46__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

// Object that extract from db data of 3 matrices (F,S',I) Freq, Stres and Intuitivness
        // and run the QAP, and after that insert Z1,Z2,Zt to the gl databse

class QAP_DB
{
public:
        QAP_DB(long n); // Constructor, includes vector wiht the indices of the gestures
                                                                        // from the big matrix and num of nodes
        virtual ~QAP_DB();
        long commands,gestures; // Number of Commands and Gestures in DB
        void Initial();
        long *gestures_subset; //The indexes of the n gestures in the big matrix
        long *pai; //Data from the QAP object (the combinaty solution)
        float W1,W2,W3; // Data from QAP (weight for direct intuitiveness, for stress, and for compl. intuitveness)
        double H2; //coefficient to reduce the size of the stress to be in the same range of the intuitiveness


private:
        long ** F, ** S,** UI,**D, ** IC;  //Big matrices containing all the data in DB
        long ** f, ** s, **ui,**d, **ic;        //Small matrices containing just the data to be passed to QAP
        int *oC; //Small vectors of Opposed Commands, and Opposed Gestures, to be passed to QAP
        int *equiv_table; // vector with equivalneces between the gestures names, and their order in the task master
set, for exampl: ges 27 is the 23 in the robotic arm tassk
        long  Z1,Z2,Zt; //Data accesible from the QAP object.
        qap *qap_obj; //Pointer to the QAP object
        int number_comp_gestures; //number of records in the comp intutive table, this is the number of compl
gestures in the database
        double tperiod;// period of time to solve the QAP
        int CandG_inDB(); //Find number of gestures, and commands.
        int DB2Matrices();  // Copy the matrices data from DB to memory
        void Allocate_Mem(); // Allocate memory of all the kinds of matrices
        void ExtractSubMatrix();  // Extract the small matrix of size nxn
        void RunQAP(); //Run the QAP object using the submatrices data and some parameters
        void Insert_Results2DB(); //Insert the results on the DB
        int renumbered_index(int i); //returns the new index of the gesture of the subset, according to renumbering it
from 0 to num. of commands
        int extract_equiv_index(int i); //finds the equiv order number of the gesture number as presented, for ex: the
gest 27, is may be the 22 in the order

};
#endif // !defined(AFX_QAP_DB_H__1B9E5B52_1EA9_4A26_A9EB_AC1233E8BE46__INCLUDED_)
```

```
// SimilarityMat.h: interface for the SimilarityMat class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_SIMILARITYMAT_H__68C27103_CB12_4815_9B8C_AAADB1354947__INCLUDED_)
#define AFX_SIMILARITYMAT_H__68C27103_CB12_4815_9B8C_AAADB1354947__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//This object creates a similarity matrix between all the gestures in the DB. So, if there
// are a total voacbulary of 12 gestures, so the matrix will be 12x12. Each entry Simi(i,j)
// represent the distance (simliariyty) between gesture i and j.
class SimilarityMat
{
public:
        SimilarityMat(int total_gestures,int n);
        virtual ~SimilarityMat();
        float Dist(int i,int j); // Euclidian Distance between the cluster i(vector) and the cluster
                                                                // j (vector).
        long *gestures_indices; //Pointer to subset of gestures indices
        int  GetDistinct(int j); //You change the gesture j, by a new gesture not included
                                                                        /// in the GV, but yes in DB.
        int GetIndexOfGesture(int g); // for a given gesture g, we can get it index in the vector
                                                                        // gesture_indices
    void OrderGestureVector(); //Order thr gesture vector from low to high, to get always
                                                                //the same Accuracy for the same vector

private:
        float Ci[100][200];
        void DB2Centroid();
        int FeatureLen,commands;
        void CreateCentroid2DB(int total_gestures);  //Creates the feature averages of each gesture types
        void RunGestureCentroids();  //Run execute for Centroids creation
        bool RunProcessAndWait(char *sCmdLine,
                char *sRunningDir,int *nRetValue); //waits until execute die
        int all_gestures;
        long *gestures_indices_out; //gestures not selected in GV but in DB, are signed with 1
                                                                // the others are signed with 0
};


    #endif                                                                                                      //
!defined(AFX_SIMILARITYMAT_H__68C27103_CB12_4815_9B8C_AAADB1354947__INCLUDED_)
```

```
// Gmanager.cpp: implementation of the Gmanager class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Gmanager.h"
#include "math.h"

#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Gmanager::Gmanager(int commands, int total_gestures)
{
  gestures=total_gestures;
  vec=new char[gestures*gestures*4]; //="0";
  vector=new int[gestures*gestures];
  confus_vect=new int[600];
  commando=commands;

}


Gmanager::~Gmanager()
{
        delete [] vec;
        delete [] vector;
        delete [] confus_vect;


}

void Gmanager::FindAccuracy()
{
        RunGestureName();//********************* REMEMBER TO UNCOMMENT THIS COMMANDS
        DB2Accuracy();
        FindMostConf();
        Acc2DB();
}


int Gmanager::DB2Accuracy()
{

        int row=0,number=0;
        HRESULT  hr = S_OK;
        //_bstr_t gest_num;
        _bstr_t acc_data;
        _bstr_t fcm_time_data;
        _bstr_t confus_data;

        int digit,col=0;
        char *tokenPtr;
```

```
    if(FAILED(::CoInitialize(NULL)))
      return 1;
    if (SUCCEEDED(hr))
    {
        _RecordsetPtr pRstGestures("ADODB.Recordset");
       // Connection String
       _bstr_t strCnn("DSN=gesture;");
        // Open table

                try
                {
                pRstGestures->Open("SELECT * FROM parameter ORDER BY id;", strCnn, adOpenStatic,
adLockReadOnly, adCmdText);

         pRstGestures->MoveLast();

                                        acc_data        =((_bstr_t)     pRstGestures->GetFields()-
>GetItem("recognized")->GetValue());
                                        acc=(float)atof(acc_data);
                                        fcm_time_data   =((_bstr_t)     pRstGestures->GetFields()-
>GetItem("fcm_time")->GetValue());
                                        fcm_time=(float)atof(fcm_time_data);
                                        confus_data     =((_bstr_t)     pRstGestures->GetFields()-
>GetItem("confusion")->GetValue());
                                        strcpy(vec,confus_data);
                                        tokenPtr=strtok(vec, " ");
                                        col=0;

                                        while (tokenPtr !=NULL )
                                        {
                                                digit=atoi(tokenPtr);
                                                tokenPtr = strtok(NULL," ");
                                                confus_vect[col]=digit;
                                                col++;
                                        }

                 pRstGestures->Close();
                }
                 catch (_com_error &e)
                {
       // Notify the user of errors if any.
       // Pass a connection pointer accessed from the Recordset.
        _variant_t vtConnect = pRstGestures->GetActiveConnection();

        // GetActiveConnection returns connect string if connection
        // is not open, else returns Connection object.

        //          AfxMessageBox((char*) e.Description());
            printf("Errors occured.");
                        (char*) e.Description();

                }

    }


        return 1;
  }
```

```
void Gmanager::RunGestureName()
{
	int memor[5];
	int *nRetValue=memor;
	char sCmdLine[200]="D:\\PHD_PROJECTS\\name_gest_robotic arm_batch2\\name_gestures_batch";
	char sRunningDir[200]="D:\\";
	RunProcessAndWait(sCmdLine,sRunningDir ,nRetValue);
}



void Gmanager::RunGL_map()
{
	 char runa[200]="";
	 char digit[8]="";

	 for (int i=0;i<commando;i++)
	  {
	     sprintf(digit,"%d",gestures_matched[i]);
		strcat(runa,digit);
		strcat(runa," ");
	  }
	 //ShellExecute(NULL,	"open","C:\\WINDOWS\\SYSTEM32\\cmd.exe",	runa,
NULL,SW_SHOWNORMAL );
	 ShellExecute(NULL,	"open","D:\\PHD_Projects\\QAPI\\GL_map.exe",runa,
NULL,SW_SHOWNORMAL );
}


bool Gmanager::RunProcessAndWait(char *sCmdLine,
		char *sRunningDir,int *nRetValue)
{

 int nRetWait;
 int nError;

 // That means wait 300 s before returning an error
 // You can change it to the value you need.
 // If you want to wait for ever just use 'dwTimeout = INFINITE'>
 DWORD dwTimeout = 1000 *300;


 STARTUPINFO stInfo;
 PROCESS_INFORMATION prInfo;
 BOOL bResult;
 ZeroMemory( &stInfo, sizeof(stInfo) );
 stInfo.cb = sizeof(stInfo);
 stInfo.dwFlags=STARTF_USESHOWWINDOW;
 stInfo.wShowWindow=SW_MINIMIZE;

 bResult = CreateProcess(NULL,
		(LPSTR)(LPCSTR)sCmdLine,
		NULL,
		NULL,
		TRUE,
		CREATE_NEW_CONSOLE
		| NORMAL_PRIORITY_CLASS,
		NULL,
		(LPCSTR)sRunningDir,
		&stInfo,
```

```
                    &prInfo);
 *nRetValue = nError = GetLastError();

 if (!bResult) return FALSE;
  nRetWait =  WaitForSingleObject(prInfo.hProcess,dwTimeout);

 CloseHandle(prInfo.hThread);
 CloseHandle(prInfo.hProcess);

 if (nRetWait == WAIT_TIMEOUT) return FALSE;
  return TRUE;
}

void Gmanager::FindMostConf()
{
        int ind,indice;
        int max=0;
        int min=100000;
        int row;
        int col;

        // Create a copy of the confus_vector, and damage it!, in order to avoid the diagonals

        //for (int i=0; i<pow(commando,2); i++)
        //          vector[i]=confus_vect[i];

        for (int i=0; i<pow(commando,2); i++)
                  vector[i]=0;

        for (i=0;i<commando;i++)   // Take all the diagonals
                  vector[i]=confus_vect[i+i*commando];

        for (i=0;i<commando;i++)   // Take min over all the diagonals
                  if (vector[i]<=min)
                  {
                            min=vector[i];
                            ind=i;
                  }
        for (i=0; i<pow(commando,2); i++)  //Copy the original matrix
                  vector[i]=confus_vect[i];

        vector[ind+commando*ind]=0; //destroy it a little

        for (i=0;i<commando;i++)   // Take max over the row with the min diag.
                  if (vector[i+commando*ind]>=max)
                  {
                            max=vector[i+commando*ind];
                            indice=i;
                  }
    indice=ind*commando+indice;

        //for (i=0;i<commando;i++)
        //          vector[i+i*commando]=0;

//
//      for (i=(int)pow(commando,2);i>0;i--)
//      {
//                if (vector[i]>=max)
//                {
//                          max=vector[i];
```

```
//                              indice=i;
//                      }
//      }

        //Now find the rows and cols of the conf. matrix form the vector
        row=(indice/commando);
        col=(indice%commando);

        confused_A=gestures_subset[row];
        confused_B=gestures_subset[col];

}

void Gmanager::Acc2DB()
{

        char str_confus1[15]="";
        char str_confus2[15]="";
        char sAcc[15]="";
        char sfcm_time[15]="";

        HRESULT  hr = S_OK;

        _bstr_t sstr_confus1,sstr_confus2,sstr_Acc,ssfcm_time;

  if (FAILED(::CoInitialize(NULL)))
    return;

  if (SUCCEEDED(hr))
  {
          // Define ADO object pointers.
   // Initialize pointers on define.
  _RecordsetPtr pRstGL = NULL;
  _ConnectionPtr pConnection  = NULL;

  HRESULT hr = S_OK;


 //Replace Data Source value with your server name.
  _bstr_t strCnn("DSN=gl;");
   _bstr_t strMessage;


                try
                {
                        //Open a connection
                        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                        pConnection->Open(strCnn,"","",adConnectUnspecified);


                        //Open results table
                        TESTHR(pRstGL.CreateInstance(__uuidof(Recordset)));

                        //You have to explicitly pass the Cursor type and LockType to the Recordset here
                        pRstGL->Open("results",_variant_t((IDispatch                *)                pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);



                         sprintf(sAcc,"%f",acc);
```

```
                                sprintf(sfcm_time,"%f",fcm_time);
                                sprintf(str_confus1,"%d",confused_A);
                                sprintf(str_confus2,"%d",confused_B);

                                sstr_Acc=sAcc;
                                ssfcm_time=sfcm_time;
                                sstr_confus1=str_confus1;
                                sstr_confus2=str_confus2;

                                pConnection->Execute("UPDATE results SET Acc='"+ sstr_Acc+"',fcm_time='"+
ssfcm_time+"',confused1='"+sstr_confus1+"', confused2='"+sstr_confus2+"' WHERE id=(SELECT max(id) FROM
results);",NULL,adCmdText);

                                pRstGL->Close();
                                pConnection->Close();
                        }
                        catch (_com_error &e)
                        {
                // Notify the user of errors if any.
                // Pass a connection pointer accessed from the Recordset.
                 _variant_t vtConnect = pRstGL
                                ->GetActiveConnection();

                 // GetActiveConnection returns connect string if connection
                 // is not open, else returns Connection object.

                        //AfxMessageBox((char*) e.Description());
                    //printf("Errors occured.");
                                        fprintf(stderr, "Database gl Problems: %s\n",(char*) e.Description());
                                        exit(1);



                        }

                ::CoUninitialize();
            }
        }

        int Gmanager::RenameLabelsDB(long *gestures_matched)
        {

                char sold_lbl[15]="";
                char snew_lbl[15]="";
            char sold_ind[10]="";
                char snew_ind[10]="";
                char tmp[1]="";

                int old_ind,new_ind;

                        HRESULT  hr = S_OK;

                  if(FAILED(::CoInitialize(NULL)))
                            return 1;

                  if (SUCCEEDED(hr))
                  {
                                // Define ADO object pointers.
                        // Initialize pointers on define.
                          _RecordsetPtr pRstGesture = NULL;
                          _ConnectionPtr pConnection  = NULL;
```

```
                HRESULT hr = S_OK;


               //Replace Data Source value with your server name.
                _bstr_t strCnn("DSN=gesture;");
                        _bstr_t strMessage;


                                try
                                {
                                        //Open a connection
                                        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                                        pConnection->Open(strCnn,"","",adConnectUnspecified);


                                        //Open results table
                                        TESTHR(pRstGesture.CreateInstance(__uuidof(Recordset)));

                                        //You have to explicitly pass the Cursor type and LockType to the
Recordset here
                                        pRstGesture->Open("centroid",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);

                                          pConnection->Execute("UPDATE centroid SET
command=";",NULL,adCmdText);

                                        pRstGesture->Close();
                                        pConnection->Close();
                                }
                                 catch (_com_error &e)
                                {
                              // Notify the user of errors if any.
                              // Pass a connection pointer accessed from the Recordset.
                                        _variant_t vtConnect = pRstGesture
                                                ->GetActiveConnection();

                                        // GetActiveConnection returns connect string if connection
                                        // is not open, else returns Connection object.

                                        //AfxMessageBox((char*) e.Description());
                                                        //printf("Errors occured.");
                                                         fprintf(stderr, "Database gl Problems: %s\n",(char*)
e.Description());
                                                         exit(1);


                                        }
          }

        for (int j=0;j<commando;j++)
        {
                old_ind=j;
                new_ind=gestures_matched[j];


                HRESULT  hr = S_OK;

        _bstr_t scmdo_ind;
                //_bstr_t gest_num;
```

```
                _bstr_t ssold_lbl;
                _bstr_t ssnew_lbl;
                _bstr_t ssold_ind;
                _bstr_t ssnew_ind;


        if(FAILED(::CoInitialize(NULL)))
                return 1;
        if (SUCCEEDED(hr))
        {
                _RecordsetPtr pRstGL("ADODB.Recordset");
                        // Connection String
                        _bstr_t strCnn("DSN=gl;");
                 // Open table

                        sprintf(sold_ind,"%d",old_ind);
                        sprintf(snew_ind,"%d",new_ind);

                        ssold_ind=sold_ind;
                        ssnew_ind=snew_ind;

                        try
                        {
                        pRstGL->Open("SELECT * FROM commands WHERE id="+ ssold_ind +";", strCnn,
adOpenStatic, adLockReadOnly, adCmdText);
                        ssold_lbl =((_bstr_t) pRstGL->GetFields()->GetItem("command")->GetValue());
                        strcpy(sold_lbl, ssold_lbl);
                        pRstGL->Close();
                        }
                         catch (_com_error &e)
                        {
                        _variant_t vtConnect = pRstGL->GetActiveConnection();
                          printf("Errors occured.");
                                (char*) e.Description();
                        }

        }

        if (SUCCEEDED(hr))
        {
                        // Define ADO object pointers.
                // Initialize pointers on define.
                  _RecordsetPtr pRstGesture = NULL;
                  _ConnectionPtr pConnection  = NULL;

                  HRESULT hr = S_OK;


                 //Replace Data Source value with your server name.
                  _bstr_t strCnn("DSN=gesture;");
                        _bstr_t strMessage;


                                try
                                {
                                        //Open a connection
                                        TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                                        pConnection->Open(strCnn,"","",adConnectUnspecified);
```

```
                                        //Open results table
                                        TESTHR(pRstGesture.CreateInstance(__uuidof(Recordset)));

                                        //You have to explicitly pass the Cursor type and LockType to the
Recordset here
                                        pRstGesture->Open("centroid",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);

                                         pConnection->Execute("UPDATE centroid SET command='"+
ssold_lbl +"' WHERE name='"+ ssnew_ind +"';",NULL,adCmdText);

                                        pRstGesture->Close();
                                        pConnection->Close();
                                }
                                 catch (_com_error &e)
                                {
                        // Notify the user of errors if any.
                        // Pass a connection pointer accessed from the Recordset.
                                _variant_t vtConnect = pRstGesture
                                        ->GetActiveConnection();

                                // GetActiveConnection returns connect string if connection
                                // is not open, else returns Connection object.

                                //AfxMessageBox((char*) e.Description());
                                                //printf("Errors occured.");
                                                 fprintf(stderr, "Database gl Problems: %s\n",(char*)
e.Description());
                                                 exit(1);


                                }
        }

    }
    return (0);

    }
```

```
// qap.cpp: implementation of the qap class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "qap.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

qap::qap(long N)
{
	p = new long [N];

	n_max=851;
	infini=1399999999;
	nb_iter_initialisation = 1000;
	n=N;

//	k1=1000; //for intu
//    k2=1; //for stress;
//    k3=k1; //for compl. intutivenss

}

qap::~qap()
{
	delete [] p;

}

long qap::max(long a, long b){if (a > b) return(a);else return(b);};
double qap::max(double a, double b) {if (a > b) return(a); else return(b);}
long qap::min(long a, long b) {if (a < b) return(a); else return(b);}
double qap::min(double a, double b) {if (a < b) return(a); else return(b);}

void qap::swap(long &a, long &b) {long temp = a; a = b; b = temp;}
double qap::temps() {return(double(clock())/double(1000000));}

/*-------------------------------------------------*/

/************* random number generators ****************/

const long m = 2147483647; const long m2 = 2145483479;
const long a12 = 63308; const long q12 = 33921; const long r12 = 12979;
const long a13 = -183326; const long q13 = 11714; const long r13 = 2883;
const long a21 = 86098; const long q21 = 24919; const long r21 = 7417;
const long a23 = -539608; const long q23 = 3976; const long r23 = 2071;
const double invm = 4.656612873077393e-10;
long x10 = 12345, x11 = 67890, x12 = 13579,
    x20 = 24680, x21 = 98765, x22 = 43210;

double qap::mon_rand()
 {long h, p12, p13, p21, p23;
  h = x10/q13; p13 = -a13*(x10-h*q13)-h*r13;
  h = x11/q12; p12 = a12*(x11-h*q12)-h*r12;
  if (p13 < 0) p13 = p13 + m; if (p12 < 0) p12 = p12 + m;
  x10 = x11; x11 = x12; x12 = p12-p13; if (x12 < 0) x12 = x12 + m;
  h = x20/q23; p23 = -a23*(x20-h*q23)-h*r23;
```

```
 h = x22/q21; p21 = a21*(x22-h*q21)-h*r21;
 if (p23 < 0) p23 = p23 + m2; if (p21 < 0) p21 = p21 + m2;
 x20 = x21; x21 = x22; x22 = p21-p23; if(x22 < 0) x22 = x22 + m2;
 if (x12 < x22) h = x12 - x22 + m; else h = x12 - x22;
 if (h == 0) return(1.0); else return(h*invm);
 }

long qap::unif(long low, long high)
 {return(low + long(double(high - low + 1) *  mon_rand() - 0.5));
 }

/************************* sa for qap *******************************/

long qap::calc_delta_complet2(long n, long ** a, long ** b,
                                                    long ** w, long ** d, long * p, long r, long s)
 {

 long dd;

// if ((p[0]==2) && (p[1]==7) && (p[2]==0) && (p[3]==5) && (p[4]==4) && (p[5]==3) &&
  //   (p[6]==6) && (p[7]==1))
  // int toti=0;


 //effect of (conitrbution or not) this couple of commands and their assignment, when sweeped.
 dd = k1*(w[r][p[s]]+w[s][p[r]]-w[r][p[r]]-w[s][p[s]]); // new added by Juan  - intuitve term

 dd = dd  - h2*k2*((a[r][r]-a[s][s])*(b[p[s]][p[s]]*d[p[s]][p[s]]-b[p[r]][p[r]]*d[p[r]][p[r]])+
       (a[r][s]-a[s][r])*(b[p[s]][p[r]]*d[p[s]][p[r]]-b[p[r]][p[s]]*d[p[r]][p[s]])); //stress term
 //see above that the stress has a minus sign before, since we want that a high delta means low stress (high comfort)


 //effect on the other nodes (except the couple). Minus sign before stress is becasue we want to minimize stress
 for (long k = 0; k < n; k = k + 1) if (k!=r && k!=s)
   dd = dd - h2*k2*((a[k][r]-a[k][s])*(b[p[k]][p[s]]*d[p[k]][p[s]]-b[p[k]][p[r]]*d[p[k]][p[r]]) +
        (a[r][k]-a[s][k])*(b[p[s]][p[k]]*d[p[s]][p[k]]-b[p[r]][p[k]]*d[p[r]][p[k]]));

 if (oC[r]==s)// if the n are complementary
       dd=dd+k3*(ic[int(r/2)][p[s]*n+p[r]]-ic[int(r/2)][p[r]*n+p[s]]);
 else
       if (oC[s]==r)// if the n are complementary
          dd=dd+k3*(ic[int(s/2)][p[r]*n+p[s]]-ic[int(s/2)][p[s]*n+p[r]]);

 for (k = 0; k < n; k = k + 1) if (k!=r && k!=s) //check how the swap will affect the other relations. Add reward for new couples, punish demolition of couples
 {
       if (oC[k]==s)  //if there is a command that is complementary of one of the pair candidates for swaping, check the contribution for the swap of the pair
              dd= dd + k3*ic[int(k/2)][p[k]*n+p[r]];
       else
              if (oC[s]==k) // same as above, but check the n in reverse, first command2 and them command1
                     dd= dd + k3*ic[int(s/2)][p[r]*n+p[k]];

              if (oC[k]==s) //if is a comnd compli of one of the pair, check the lost for the swap of the pair
                dd= dd - k3*ic[int(k/2)][p[k]*n+p[s]];
              else
                     if (oC[s]==k) // same as above, but check the n in reverse, first command2 and them command1
                       dd= dd - k3*ic[int(s/2)][p[s]*n+p[k]];
```

```
                if (oC[k]==r)  //if there is a command that is complementary of one of the pair candidates (check the second candidate) for swaping, check the contribution for the swap of the pair
                  dd= dd + k3*ic[int(k/2)][p[k]*n+p[s]];
          else
                  if (oC[r]==k) // same as above, but check the n in reverse, first command2 and them command1
                        dd= dd + k3*ic[int(r/2)][p[s]*n+p[k]];

                if (oC[k]==r) //if is a comnd compli of one of the pair, check the lost for the swap of the pair
                  dd= dd - k3*ic[int(k/2)][p[k]*n+p[r]];
                else
                        if (oC[r]==k) // same as above, but check the n in reverse, first command2 and them command1
                          dd= dd - k3*ic[int(r/2)][p[r]*n+p[k]];
       }
      return(dd);
     }

    long qap::calcule_cout(long n, long ** a, long ** b, long ** w,long ** d,long * p)
     {long i, j;
      long c = 0;
      int comp_intu;
     // long sk = 400000000;


      for (i = 0; i < n; i = i + 1) for (j = 0; j < n; j = j + 1)
        c = c - h2*k2*a[i][j] * b[p[i]][p[j]]*d[p[i]][p[j]]; //total stress
     // c=c+sk;

      for (i = 0; i < n; i = i + 1)
            c = c + k1*w[i][p[i]]; //total intutiveness (added to total comfort)


      for (i = 0; i < n; i = i + 1)  //complementary intuitveness
            for (j = 0; j < n; j = j + 1)
            {
                  comp_intu=0;

                  if (oC[i]==j)
                        comp_intu=ic[int(i/2)][p[i]*n+p[j]];
                  else
                        if (oC[j]==i)
                            comp_intu=ic[int(j/2)][p[j]*n+p[i]];

                  c = c + k3*comp_intu;
            }

      return(c);
     }

    void qap::calcule_cout_bout(long & co, long & bo,long n, long ** a, long ** b, long ** w,long ** d, long * p)
     {long i, j;

      long c = 0;
      int comp_intu=0;
      co=0;
      bo=0;
      //long sk = 400000000;

      for (i = 0; i < n; i = i + 1)    //This is the total comfort
```

```
            for (j = 0; j < n; j = j + 1)
                                    c = c   - h2*a[i][j] * b[p[i]][p[j]] * d[p[i]][p[j]] ; // a-freq, b-comfort, w-intutivenss, d-
duration ---
             //c = c  - k2*a[i][j] * b[p[i]][p[j]] * d[p[i]][p[j]] ; // a-freq, b-comfort, w-intutivenss, d- duration ---
            //c = c + sk;

      co=c;

      for (i = 0; i < n; i = i + 1)
      {
            c = c + w[i][p[i]];
            bo = bo + w[i][p[i]];  //  This is the total intuitveness
            //c = c + k1*w[i][p[i]];
            //bo = bo + k1*w[i][p[i]];  //  This is the total intuitveness
      }

      for (i = 0; i < n; i = i + 1)  //this is complementary intuitveness
            for (j = 0; j < n; j = j + 1)
            {
                  if (oC[i]==j)
                        comp_intu=ic[int(i/2)][p[i]*n+p[j]];
                  else
                        if (oC[j]==i)
                            comp_intu=ic[int(j/2)][p[j]*n+p[i]];

                  bo = bo + comp_intu ; //total
                  //bo = bo + k3*comp_intu ; //total
            }

      c=c+bo;

      }


     void qap::tire_solution_aleatoire(long n, long * p)
      {long i;
       for (i = 0; i < n; i = i+1) p[i] = i;
       for (i = 1; i < n; i = i+1) swap(p[i], p[unif(i, n-1)]);
      }

     void qap::recuit(long n, long ** a, long ** b, long ** w,long ** d,
               long * meilleure_sol, long & meilleur_cout,
               long nb_iterations)

      {long * pp;
       long i, r, s;
       long delta;
       double cpu = temps();
       long k = n*(n-1)/2, mxfail = k, nb_fail, no_iteration;
       long dmin = infini, dmax = 0;
       double t0, tf, beta, tfound, temperature;
       long co=0;
       long bo=0;
     //  long Cout1;

       pp = new long[n]; //added by me!

       for (i = 0; i < n; i = i + 1) pp[i] = meilleure_sol[i];
       long Cout = calcule_cout(n, a, b,w,d, pp);
       meilleur_cout = Cout;
```

```
for (no_iteration = 0; no_iteration < nb_iter_initialisation;
    no_iteration = no_iteration+1)
 {
                r = unif(0, n-1);
                s = unif(0, n-2);
                if (s >= r) s = s+1;

                delta = calc_delta_complet2(n,a,b,w,d,pp,r,s);
                if (delta > 0)
                 {dmin = min(dmin, delta); dmax = max(dmax, delta);};
                Cout = Cout + delta;
                swap(pp[r], pp[s]);
                //Cout1 = calcule_cout(n, a, b,w,d, pp);
 };

t0 = dmin + (dmax - dmin)/10.0;
tf = dmin;
beta = (t0 - tf)/(nb_iterations*t0*tf);

nb_fail = 0;
tfound = t0;
temperature = t0;
r = 0; s = 1;
for (no_iteration = 0;
    no_iteration < nb_iterations - nb_iter_initialisation;
    no_iteration = no_iteration + 1)
 { temperature = temperature / (1.0 + beta*temperature);

   s = s + 1;
   if (s > n-1)
    {r = r + 1;
     if (r > n - 2) r = 0;
     s = r + 1;
    };

   delta = calc_delta_complet2(n,a,b,w,d,pp,r,s);

   if ((delta > 0) || (mon_rand() <= exp(double(delta)/temperature)) || //Modified to Maximiz
       mxfail == nb_fail)
       {
                Cout = Cout + delta; swap(pp[r], pp[s]);
                //Cout1=calcule_cout(n, a, b, w,d,pp); //just added
          nb_fail = 0;
       }
   else nb_fail = nb_fail + 1;

   if (mxfail == nb_fail)
       {beta = 0; temperature = tfound;};
   if (Cout > meilleur_cout) //Modified to Maximiza
    {
                        meilleur_cout = Cout;
                        for (i = 0; i < n; i = i + 1)
                                meilleure_sol[i] = pp[i];
                        tfound = temperature;
                        //Cout=calcule_cout(n, a, b, w,d,meilleure_sol); //just added
                        // cout << "Iteration = " << no_iteration
                        //           << "  Cost = " << meilleur_cout
                        //           << "  Cout = " << Cout <<  '\n';
                                  //<< "  Computational time = " << temps() - cpu <<  '\n';
```

```
      };

  };
 // cout << "Best solution found : \n";

  Cout=calcule_cout(n, a, b, w,d,meilleure_sol);
  calcule_cout_bout(co,bo, n, a, b, w,d,meilleure_sol);
  Z1=co; //Stress
  Z2=bo; //Total Intutiveness (Normal + Complementary )
  Zt=k2*co+k1*bo;
  // for (i = 0; i < n; i = i + 1) meilleure_sol[i] = meilleure_sol[i]+1;
  /// End

 //  cout << "Best solution for distance : ";
//  cout << co << ' ';
//  cout << bo << ' ';
//  cout << '\n';

  delete [] pp;
 }


void qap::solve()
{

    long best_Z1,best_Z2,best_Zt,*best_p;
        best_p = new long [n];
        double cpu = temps();

        tire_solution_aleatoire(n, p);
    recuit(n,a,b,w,d,p,Cout, nb_iterations);

        best_Z1=Z1;
        best_Z2=Z2;
        best_Zt=Zt;

    memcpy(best_p, p, n * sizeof(long));

        for (no_res = 0; no_res < nb_res-1; no_res = no_res + 1)
   {
                tire_solution_aleatoire(n, p);
                recuit(n,a,b,w,d,p,Cout, nb_iterations);
                if (Zt>best_Zt)
                {
                        best_Z1=Z1;
                        best_Z2=Z2;
                        best_Zt=Zt;
                        memcpy(best_p, p, n * sizeof(long));
                }

        }
   Z1=best_Z1;
   Z2=best_Z2;
   Zt=best_Zt;

   memcpy(p, best_p, n * sizeof(long));
   tperiod=temps()-cpu;
```

```
/*
  long co=0;
  long bo=0;

  p[0]=4;
  p[1]=0;
  p[2]=3;
  p[3]=5;
  p[4]=1;
  p[5]=2;
  p[6]=7;
  p[7]=6;

  calcule_cout_bout(co,bo, n, a, b, w,d,p);
  Z1=co; //Stress
  Z2=bo; //Total Intutiveness (Normal + Complementary )
  Zt=k2*co+k1*bo;
*/

  delete [] best_p;

}
```

```
// QAP_DB.cpp: implementation of the QAP_DB class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "QAP_DB.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
   no_namespace rename("EOF", "EndOfFile")

inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};


QAP_DB::QAP_DB(long n)  //Constructor recieves the indeces of n gestures in the
                                                                    // big matrix
{
//      gestures_subset=subset;
        qap_obj=new qap(n);
        qap_obj->nb_iterations=600000;   // can be more than this (inner iterations)  600000
        qap_obj->nb_res=4;   // can be more iterations (outter iterations)              4
        int ans=0;
        ans=CandG_inDB();  //extracts the data from DB called GL
        Allocate_Mem();


}

int QAP_DB::CandG_inDB()
{

        // Find the number of commands and gestures in DB
        int number=0;
        commands=0;
        gestures=0;
        HRESULT  hr = S_OK;
        _bstr_t num;

    if(FAILED(::CoInitialize(NULL)))
      return 0;
    if (SUCCEEDED(hr))
    {
          _RecordsetPtr pRstGL("ADODB.Recordset");
        // Connection String
        _bstr_t strCnn("DSN=GL;");
         // Open table
        try
        {
                pRstGL->Open("SELECT COUNT(*) AS result FROM COMMANDS;", strCnn, adOpenStatic,
adLockReadOnly, adCmdText);
        num =((_bstr_t) pRstGL->GetFields()->GetItem("result")->GetValue());
            number=atoi(num);
        pRstGL->Close();
                commands=number;

                pRstGL->Open("SELECT COUNT(*) AS result FROM stress_matrix;", strCnn, adOpenStatic,
adLockReadOnly, adCmdText);
```

```
        num =((_bstr_t) pRstGL->GetFields()->GetItem("result")->GetValue());
            number=atoi(num);
        pRstGL->Close();
                    gestures=number;
         }
          catch (_com_error &e)
         {
        // Notify the user of errors if any.
        // Pass a connection pointer accessed from the Recordset.
         _variant_t vtConnect = pRstGL->GetActiveConnection();

         // GetActiveConnection returns connect string if connection
         // is not open, else returns Connection object.

                     fprintf(stderr, "Database gl Problems: %s\n",(char*) e.Description());
                     exit(1);

         //          AfxMessageBox((char*) e.Description());
               // printf("Errors occured.");
                                //          (char*) e.Description();

         }
    }

   return 1;
}

int QAP_DB::DB2Matrices()
{

        char vec[1000]="0";
        int row=0,number=0;
        HRESULT  hr = S_OK;
        //_bstr_t gest_num;
        _bstr_t ui_data, stress_data, duration_data, frequency_data, oC_data, oG_data;

        int digit,digit2,col=0;
        char *tokenPtr;


      if(FAILED(::CoInitialize(NULL)))
        return 1;
      if (SUCCEEDED(hr))
      {
          _RecordsetPtr pRstGL("ADODB.Recordset");
         // Connection String
         _bstr_t strCnn("DSN=gl;");
          // Open table

         try
         {
                    pRstGL->Open("SELECT * FROM intutive_matrix ORDER BY id;", strCnn, adOpenStatic,
adLockReadOnly, adCmdText);

          pRstGL->MoveFirst();


                    while (!pRstGL->EndOfFile)
                                {
                                        col=0;
```

```
                                        ui_data =((_bstr_t) pRstGL->GetFields()->GetItem("id")->GetValue());
                                        strcpy(vec,ui_data);
                                        digit=atoi(vec);
                                        equiv_table[row]=digit;

                                        ui_data =((_bstr_t) pRstGL->GetFields()->GetItem("data")->GetValue());
                                        strcpy(vec,ui_data);
                                        tokenPtr=strtok(vec, " ");


                                        while (tokenPtr !=NULL )
                                        {
                                                digit=atoi(tokenPtr);
                                                tokenPtr = strtok(NULL," ");
                                                UI[row][col]=digit;
                                                col++;
                                        }


                        pRstGL->MoveNext();
                                        row++;
                }
                pRstGL->Close();


                        pRstGL->Open("SELECT * FROM stress_matrix ORDER BY id;", strCnn, adOpenStatic, adLockReadOnly, adCmdText);

        pRstGL->MoveFirst();

                                row=0;
                while (!pRstGL->EndOfFile)
                                {
                                        stress_data =((_bstr_t) pRstGL->GetFields()->GetItem("data")->GetValue());

                                        strcpy(vec,stress_data);
                                        tokenPtr=strtok(vec, " ");
                                        col=0;

                                        while (tokenPtr !=NULL )
                                        {
                                                digit=atoi(tokenPtr);
                                                tokenPtr = strtok(NULL," ");
                                                S[row][col]=digit;
                                                col++;
                                        }


                        pRstGL->MoveNext();
                                        row++;
                }
                pRstGL->Close();

        pRstGL->Open("SELECT * FROM duration_matrix ORDER BY id;", strCnn, adOpenStatic, adLockReadOnly, adCmdText);
```

```
pRstGL->MoveFirst();

                        row=0;
        while (!pRstGL->EndOfFile)
                        {
                                duration_data   =((_bstr_t)   pRstGL->GetFields()->GetItem("data")->GetValue());

                                strcpy(vec,duration_data);
                                tokenPtr=strtok(vec, " ");
                                col=0;

                                while (tokenPtr !=NULL )
                                {
                                        digit=atoi(tokenPtr);
                                        tokenPtr = strtok(NULL," ");
                                        D[row][col]=digit;
                                        col++;
                                }


                pRstGL->MoveNext();
                                row++;
        }
        pRstGL->Close();



        pRstGL->Open("SELECT * FROM frequency_matrix ORDER BY id;", strCnn, adOpenStatic, adLockReadOnly, adCmdText);

pRstGL->MoveFirst();

                        row=0;
        while (!pRstGL->EndOfFile)
                        {
                                frequency_data  =((_bstr_t)  pRstGL->GetFields()->GetItem("data")->GetValue());

                                strcpy(vec,frequency_data);
                                tokenPtr=strtok(vec, " ");
                                col=0;

                                while (tokenPtr !=NULL )
                                {
                                        digit=atoi(tokenPtr);
                                        tokenPtr = strtok(NULL," ");
                                        F[row][col]=digit;
                                        col++;
                                }


                pRstGL->MoveNext();
                                row++;
        }
        pRstGL->Close();


        pRstGL->Open("SELECT * FROM comp_commands ORDER BY id;", strCnn, adOpenStatic, adLockReadOnly, adCmdText);
```

```
pRstGL->MoveFirst();

                for (int index=0;index<commands;index++)
                        oC[index]=-1;

                        col=0;
        while (!pRstGL->EndOfFile)
                        {
                                oC_data =((_bstr_t) pRstGL->GetFields()->GetItem("data")->GetValue());
                                strcpy(vec,oC_data);
                                digit=atoi(vec);

                                oC_data =((_bstr_t) pRstGL->GetFields()->GetItem("id")->GetValue());
                                strcpy(vec,oC_data);
                                digit2=atoi(vec);
                                oC[digit2]=digit;
                                col++;
                        pRstGL->MoveNext();
        }
        pRstGL->Close();


        pRstGL->Open("SELECT * FROM comp_intuitive ORDER BY id,id2;", strCnn, adOpenStatic, adLockReadOnly, adCmdText);

pRstGL->MoveFirst();
                row=0;
                col=0;

        while (!pRstGL->EndOfFile)
                        {
                                col=0;
                                oG_data =((_bstr_t) pRstGL->GetFields()->GetItem("id")->GetValue());
                                strcpy(vec,oG_data);
                                digit=atoi(vec);
        IC[row][col]=digit;
                                col=col+1;

                                oG_data =((_bstr_t) pRstGL->GetFields()->GetItem("id2")->GetValue());
                                strcpy(vec,oG_data);
                                digit=atoi(vec);
        IC[row][col]=digit;
                                col=col+1;


                                oG_data =((_bstr_t) pRstGL->GetFields()->GetItem("data")->GetValue());
                                strcpy(vec,oG_data);
                                tokenPtr=strtok(vec, " ");

                                while (tokenPtr !=NULL )
                                {
                                        digit=atoi(tokenPtr);
                                        tokenPtr = strtok(NULL," ");
                                        IC[row][col]=digit;
                                        col++;
```

```
                                        }


                        pRstGL->MoveNext();
                                        row++;
                }
                pRstGL->Close();
                                number_comp_gestures=row;


        }
        catch (_com_error &e)
                {
      // Notify the user of errors if any.
      // Pass a connection pointer accessed from the Recordset.
       _variant_t vtConnect = pRstGL->GetActiveConnection();

       // GetActiveConnection returns connect string if connection
       // is not open, else returns Connection object.

//              AfxMessageBox((char*) e.Description());
            //printf("Errors occured.");
                                //(char*) e.Description();
                fprintf(stderr, "Database gl Problems: %s\n",(char*) e.Description());
                exit(1);

                }

  }


 //  delete vec;
  return 0;

}


void QAP_DB::Allocate_Mem()
{

  // Allocate memory for big Matrices according to this quantities
  long i,j;

  for (i = 0; i < gestures; i++)
      {
   F = new long *[gestures];
       S = new long *[gestures];
       D = new long *[gestures];
       UI = new long *[gestures];
      }

      for (i = 0; i < gestures*3; i++)
                IC = new long *[gestures*3];

      for (i = 0; i < gestures*3; i++)
        IC[i]=new long[gestures*3]; //Complementary  intuitivety

      for (i = 0; i < gestures; i++)
      {
                F[i] = new long[gestures]; //frequency
```

```
                S[i] = new long[gestures];  // Stress
                D[i] = new long[gestures]; //Duration
                UI[i] = new long[gestures]; //intutiveness
        }

        for (i = 0; i < commands; i++)
        {
    //ui = new long *[gestures]; // you must fix this!!!!!!!!!!!!!!!!!
         f = new long *[commands];
        }

        for (i = 0; i < gestures; i++)
        {
                ui = new long *[commands];
                s = new long *[gestures];
                d = new long *[gestures];

        }

        for (i = 0; i < commands; i++)
    {
                f[i] = new long[commands];
                ui[i] = new long[gestures];
        }


        for (i = 0; i < gestures; i++)
        {
        //          ui[i] = new long[commands];
                s[i] = new long[gestures];
                d[i] = new long[gestures];
        }

        for (i = 0; i < commands*commands; i++)
       ic=new long *[commands];

      for (i = 0; i < commands; i++)
                ic[i]=new long[commands*commands];

//      ic[1][150]=7;

        // cleaning before use
        for(i=0;i<3*gestures;i++)
                for(j=0;j<3*gestures;j++)
                 IC[i][j]=0;

        oC=new int[commands]; //opposed command. entry oC[i]=j means that command 'j' is the complementary of command 'i' (like 'fast' and 'slow')
        equiv_table=new int[gestures]; //table of equivalences between the gesture number, and it order in the subset. For example, gesture 27, will be 5 (23 gestures maximum)



    }

    void QAP_DB::ExtractSubMatrix()
    {
        long rowcol,row,col,rowcol_equiv;
        long index=0;
        int gest=0;
```

```
int g1,g2;
int composite_index;


//cleaning a little bit the old matrices
for(int i=0;i<commands;i++) //makes zero all the complementary matrix, so later on, the compl not filled, wiill have automatically zero
        for (int j=0;j<commands*commands;j++)
    ic[i][j]=0;

// Copy the matrix UI to a submatrix ui
for (i=0; i<commands;i++) // Assumption that the number of sub-gestures is
{                                                                       // equal to number of commands
        rowcol=gestures_subset[i];   //CHANGED 19/03/06 BECAUSE now we enumerate the gestures from 1 to 27, instead of from 0.

        rowcol_equiv=extract_equiv_index(rowcol);

        for (int j=0; j<commands;j++)
        {
        ui[j][index]=UI[rowcol_equiv][j];  //ui is transposed of UI. So now, rows are commands, and cols are gestures.
//      op[j][index]=OP[rowcol][j];
        }
        index++;
}

index=0;
//Copy the matrix S to a submatrix s, including only the rows/cols of the subset of gestures
//Copy the matrix D to a submatrix d  including only the rows/cols of the subset of gestures
for (i=0;i<commands;i++)
{

 row=gestures_subset[i]-1;  //CHANGED 19/03/06 BECAUSE now we enumerate the gestures from 1 to 27, instead of from 0.
  for (long j=0; j<commands;j++)
  {
        col=gestures_subset[j]-1;   //CHANGED 19/03/06 BECAUSE now we enumerate the gestures from 1 to 27, instead of from 0.
        s[index][j]=S[row][col];
        d[index][j]=D[row][col];


  }
  index++;
}

//This parts takes the IC matrix, with the first two columns are g1 and g2 respectively, and g1 and g2 are complementary gestures.
//The rest of the values in the row is the value of intuitivety for each column.
// The other columns represents the pairs of complementary commands, left-right, up-down, etc.
//We want to cpy this to a new matrix ic, that the columns are a composite index of both g1, and g2: g1*commands + g2.
// The rows of ic are the values of the intuitivety for each pair of complementary commands
for (index=0; index<number_comp_gestures;index++)
{
        g1=IC[index][0];
        g2=IC[index][1];
        row=0;
```

```
                int in1=renumbered_index(g1);
                int in2=renumbered_index(g2);

        //          ic[1][150]=7;

                if ((in1!=-1) && (in2!=-1))
        {
            composite_index=in1*commands+in2;
            for (int indice=2;indice<commands;indice++)
            {
                                        ic[row][composite_index]=IC[index][indice];
                row=row+1;

                            }
                }

                if (composite_index==206)
                        int tio=1;

        }

        //Copy the matrix F to the submatrix f  (nothing to do, they are equal)
        f=F;

}

int QAP_DB::renumbered_index(int i)
{
        int indi;
        indi=-1;

   for (int index=0; index<commands; index++)
          if (i==gestures_subset[index])
            indi=index;

   return (indi);
}


int QAP_DB::extract_equiv_index(int i)
{
        int indi;
        indi=-1;

   for (int index=0; index<gestures; index++)
          if (i==equiv_table[index])
            indi=index;

   return (indi);

}

void QAP_DB::RunQAP()
{
        long val;

        //copies all the matrices here to the qap object
        qap_obj->a=f;
        qap_obj->b=s;
        qap_obj->w=ui;
```

```
	qap_obj->d=d;
	qap_obj->ic=ic;
	qap_obj->oC=oC;

	qap_obj->k1=W1; //weight for direct intuitveness
	qap_obj->k2=W2; // for stress (the increments of stress are more signif than the intu)
	qap_obj->k3=W3; //weight for complementary intuitveness
	qap_obj->h2=H2; //coefficient to reduce the size of the stress, to make in same scale as intuitive



	qap_obj->solve();
   Z1=qap_obj->Z1;
	Z2=qap_obj->Z2;
	Zt=qap_obj->Zt;
	tperiod=qap_obj->tperiod*1000;

	pai=qap_obj->p;  // comb[1], comb[2], etc

	//Here we try to copy the gesture permutation, using their original indexes
	for (long i=0;i<commands;i++)
	{
		val=pai[i];
		pai[i]=gestures_subset[val];
	}

}

void QAP_DB::Insert_Results2DB()
{

	char str_pai[1000]="";
	char str_subset[1000]="";
	char Su[15]="";
	char sZ1[15]="";
	char sZ2[15]="";
	char sZt[15]="";
	char sW1[15]="";
	char sW2[15]="";
	char sW3[15]="";
	char sTime[15]="";

	HRESULT  hr = S_OK;

	_bstr_t sstr_pai,sstr_subset,ssZ1,ssZ2,ssZt,ssW1,ssW2,ssW3,ssTime;

  if (FAILED(::CoInitialize(NULL)))
    return;

  if (SUCCEEDED(hr))
  {
        // Define ADO object pointers.
   // Initialize pointers on define.
  _RecordsetPtr pRstGL = NULL;
  _ConnectionPtr pConnection  = NULL;

  HRESULT hr = S_OK;


 //Replace Data Source value with your server name.
```

```
_bstr_t strCnn("DSN=gl;");
 _bstr_t strMessage;


            try
            {
                  //Open a connection
                  TESTHR(pConnection.CreateInstance(__uuidof(Connection)));
                  pConnection->Open(strCnn,"","",adConnectUnspecified);


                  //Open results table
                  TESTHR(pRstGL.CreateInstance(__uuidof(Recordset)));

                  //You have to explicitly pass the Cursor type and LockType to the Recordset here
                  pRstGL->Open("results",_variant_t((IDispatch *) pConnection,
true),adOpenKeyset,adLockOptimistic,adCmdTable);

                  for(int j=0;j<commands;j++)
                  {
                     strcpy(Su,"");
                     sprintf(Su,"%d",pai[j]);
                           strcat(str_pai,Su);
                           strcat(str_pai," ");

                           strcpy(Su,"");
                     sprintf(Su,"%d",gestures_subset[j]);
                           strcat(str_subset,Su);
                           strcat(str_subset," ");

                  }

                   sprintf(sZ1,"%d",Z1);
                   sprintf(sZ2,"%d",Z2);
                   sprintf(sZt,"%d",Zt);
                   sprintf(sW1,"%f",W1);
                   sprintf(sW2,"%f",W2);
                   sprintf(sW3,"%f",W3);
                   sprintf(sTime,"%f",tperiod);

                   sstr_pai=str_pai;
                   sstr_subset=str_subset;
                   ssZ1=sZ1;
                   ssZ2=sZ2;
                   ssZt=sZt;
                   ssW1=sW1;
                   ssW2=sW2;
                   ssW3=sW3;
                   ssTime=sTime;

      pConnection->Execute("INSERT INTO results (solution,ordered,z_str,z_int,zt,w_int,w_str,Tann)
VALUES
('"+sstr_pai+"','"+sstr_subset+"','"+ssZ1+"','"+ssZ2+"','"+ssZt+"','"+ssW1+"','"+ssW2+"','"+ssTime+"');",NULL,adC
mdText);

                  pRstGL->Close();
                  pConnection->Close();
            }
             catch (_com_error &e)
            {
```

```
        // Notify the user of errors if any.
        // Pass a connection pointer accessed from the Recordset.
         _variant_t vtConnect = pRstGL
                                ->GetActiveConnection();

         // GetActiveConnection returns connect string if connection
         // is not open, else returns Connection object.

                    //AfxMessageBox((char*) e.Description());
               //printf("Errors occured.");
                                        fprintf(stderr, "Database gl Problems: %s\n",(char*) e.Description());
                                        exit(1);


                    }

        ::CoUninitialize();
      }
    }



    void QAP_DB::Initial()
    {
          int ans=0;
    //    ans=CandG_inDB();
    //    Allocate_Mem();
          ans=DB2Matrices();
          ExtractSubMatrix();
          RunQAP();
          Insert_Results2DB();

    }

    QAP_DB::~QAP_DB()
    {
          delete [] qap_obj;    //are you sure that you destroy the object this way??  I think that the object destroys
itself
          delete [] F;
          delete [] S;
          delete [] UI;
          delete [] IC;
          delete [] D;
    //    delete [] f; Commented since f=F and already deallocated (before 2 lines ago)
          delete [] ui;
          delete [] ic;
          delete [] s;
       delete [] d;
          delete [] oC;
          delete [] equiv_table;

    }
```

```
// QAPI.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "QAP_DB.h"
#include  "Gmanager.h"
#include "OrganizeImages.h"
#include "SimilarityMat.h"
#include "string.h"


void main()
{

	long a,b,c,d,e,f,g,h;
	long n=8;   // Number of n nodes of the problem
	long total_gestures=22; //Number of Gestures in Master Set Vocabulary
	long *gestures_indices;
	long *gestures_indices_matched; //the same indices, but in the order corresponding each command
	gestures_indices=new long[n];
	gestures_indices_matched=gestures_indices;
	float distan=0;

	// Object that extract from db data of 3 matrices (F,S,I)
	// and run the QAP, and after that insert Z1,Z2,Zt to the gl databse

	SimilarityMat Simat(total_gestures,n); //Constructor of Simlarity Matrix

	QAP_DB qap_db_obj(n); //Constructor
	OrganizeImages oi(n);  //Constructor
	Gmanager Gman(n,total_gestures); //Constructor

	int iterat=0;
	int W1,W2;

	W1=0;
	W2=10;

//	while (iterat<=10)
//	{
			iterat++;

/*
	for (a=1;a<20;a++)
	 for (b=a+1;b<21;b++)
	  for (c=b+1;c<22;c++)
	   for (d=c+1;d<23;d++)
			for (e=d+1;e<24;e++)
			 for (f=e+1;f<25;f++)
	  for (g=f+1;g<26;g++)
					for (h=g+1;h<27;h++)
					{

	iterat++;

   gestures_indices[0]=a;
	gestures_indices[1]=b;
	gestures_indices[2]=c;
	gestures_indices[3]=d;
	gestures_indices[4]=e;
```

```
        gestures_indices[5]=f;
        gestures_indices[6]=g;
        gestures_indices[7]=h;
    */
        W1=8;
        W2=2;

        gestures_indices[0]=6;
        gestures_indices[1]=7;
        gestures_indices[2]=8;
        gestures_indices[3]=10;
        gestures_indices[4]=12;
        gestures_indices[5]=18;
        gestures_indices[6]=23;
        gestures_indices[7]=24;

        qap_db_obj.W1=W1; //intu weight
        qap_db_obj.W2=W2;    //stress weight
        qap_db_obj.W3=qap_db_obj.W1;   //compl. intu
        qap_db_obj.H2=0.001;   //reduction factor for the stress.

        W1=W1+1;
        W2=W2-1;

        qap_db_obj.gestures_subset=gestures_indices; //Give the subset of gestures indices
                                                                                    //      to
object
        qap_db_obj.Initial(); //Run the object

        gestures_indices_matched=qap_db_obj.pai;
                                                                                    //      to
object
        oi.gestures_subset=gestures_indices;//Give the subset of gestures indices
                                                                                    //      to
object

    //      oi.MovePics();  //Move the gestures pics to train folder

        //Object that run the accuracy module with preselected gestures
        // and extract the accuracy, and put it in the gl db.
                                                                                    //      to
object
        Gman.gestures_subset=gestures_indices;//Give the subset of gestures indices      // to object

        Gman.gestures_matched=gestures_indices_matched;//Give the subset of gestures indices

        Gman.FindAccuracy();

        distan=Simat.Dist(Gman.confused_A,Gman.confused_B);

        distan=Simat.Dist(Gman.confused_A,0);

        Gman.RunGL_map();

    //      }
        Gman.RenameLabelsDB(gestures_indices_matched);

        //************ Remember to uncomment the name_gesture_VMR running applic
        delete [] gestures_indices;
    }
```

```
// SimilarityMat.cpp: implementation of the SimilarityMat class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "SimilarityMat.h"
#include "OrganizeImages.h"
#include <cv.h>

#import "C:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace rename("EOF", "EndOfFile")

inline void TESTHR(HRESULT x) {if FAILED(x) _com_issue_error(x);};

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

SimilarityMat::SimilarityMat(int total_gestures,int n)
{
  gestures_indices=new long[total_gestures];
  gestures_indices_out=new long[total_gestures];

  all_gestures=total_gestures;
  commands=n;

  for (int i=0;i<total_gestures;i++)
        gestures_indices[i]=i;     //Order the images by their oginially index order: 1,2,3,..12

 // CreateCentroid2DB(total_gestures); //Creates for the first time a prototype...
                                                                                    //..vector matrix of the
gestures. You can comment
                                                                                  // this line, after the first run
  // THIS DATA is saved in a DB called INITIAL.DBM (the centroids of each group of gestures type)

  DB2Centroid();

}

void SimilarityMat::CreateCentroid2DB(int total_gestures)
{

 OrganizeImages oi(total_gestures);  //Constructor of the pictures organizer object
 oi.gestures_subset=gestures_indices;//Give the subset of gestures indices
                                                                                          // to object
  oi.MovePics();  //Move the gestures pics to train folder


 // *** RUN THE GestureRecCentroids ***//
  RunGestureCentroids();

}

void SimilarityMat::RunGestureCentroids()
{
      int memor[5];
      int *nRetValue=memor;
      char sCmdLine[200]="D:\\PHD_PROJECTS\\GestureRecCentroids\\Debug\\GestureRec.exe";
      char sRunningDir[200]="D:\\";
      RunProcessAndWait(sCmdLine,sRunningDir ,nRetValue);
```

```
}
bool SimilarityMat::RunProcessAndWait(char *sCmdLine,
                    char *sRunningDir,int *nRetValue)
{

 int nRetWait;
 int nError;

 // That means wait 300 s before returning an error
 // You can change it to the value you need.
 // If you want to wait for ever just use 'dwTimeout = INFINITE'>
 DWORD dwTimeout = 1000 *300;


 STARTUPINFO stInfo;
 PROCESS_INFORMATION prInfo;
 BOOL bResult;
 ZeroMemory( &stInfo, sizeof(stInfo) );
 stInfo.cb = sizeof(stInfo);
 stInfo.dwFlags=STARTF_USESHOWWINDOW;
 stInfo.wShowWindow=SW_MINIMIZE;

 bResult = CreateProcess(NULL,
                    (LPSTR)(LPCSTR)sCmdLine,
                    NULL,
                    NULL,
                    TRUE,
                    CREATE_NEW_CONSOLE
                    | NORMAL_PRIORITY_CLASS,
                    NULL,
                    (LPCSTR)sRunningDir,
                    &stInfo,
                    &prInfo);
 *nRetValue = nError = GetLastError();

 if (!bResult) return FALSE;
  nRetWait =  WaitForSingleObject(prInfo.hProcess,dwTimeout);

 CloseHandle(prInfo.hThread);
 CloseHandle(prInfo.hProcess);

 if (nRetWait == WAIT_TIMEOUT) return FALSE;
  return TRUE;
}

void SimilarityMat::DB2Centroid()
{

        char vec[600]="";
        int num_pics=0,number=0;
        HRESULT  hr = S_OK;
        _bstr_t gest_num;
        _bstr_t center;

        int digit,index=0;
        char *tokenPtr;


   if(FAILED(::CoInitialize(NULL)))
      return;
```

```
    if (SUCCEEDED(hr))
    {
        _RecordsetPtr pRstGestures("ADODB.Recordset");
       // Connection String
       _bstr_t strCnn("DSN=initial;");
        // Open table
                try
                {
                pRstGestures->Open("SELECT * FROM CENTROID ORDER BY gest_num;", strCnn,
adOpenStatic, adLockReadOnly, adCmdText);

         pRstGestures->MoveFirst();


                 while (!pRstGestures->EndOfFile)
                                {
                                        gest_num =((_bstr_t) pRstGestures->GetFields()-
>GetItem("gest_num")->GetValue());
                                        center =((_bstr_t) pRstGestures->GetFields()->GetItem("center")-
>GetValue());

                                        number=atoi(gest_num);

                                        strcpy(vec,center);
                                        tokenPtr=strtok(vec, " ");
                                        index=0;

                                        while (tokenPtr !=NULL )
                                        {
                                                digit=atoi(tokenPtr);
                                                tokenPtr = strtok(NULL," ");
                                                Ci[number][index]=digit;
                                                index++;
                                        }


                        pRstGestures->MoveNext();
                                         num_pics++;
                 }
                                FeatureLen=index;
                 pRstGestures->Close();
                 }
                 catch (_com_error &e)
                 {
       // Notify the user of errors if any.
       // Pass a connection pointer accessed from the Recordset.
        _variant_t vtConnect = pRstGestures->GetActiveConnection();

        // GetActiveConnection returns connect string if connection
        // is not open, else returns Connection object.


         printf("Errors occured.");
                (char*) e.Description();
                exit(1);
       }
    }
    return;
  }
```

```
float SimilarityMat::Dist(int i,int j)
{
	float u=0;
	CvMat PointI = cvMat(1,FeatureLen,CV_MAT32F,NULL);
	CvMat PointJ = cvMat(1,FeatureLen,CV_MAT32F,NULL);
	CvMat PointDiff = cvMat(1,FeatureLen,CV_MAT32F,NULL);
	//CvMat Result = {1,1,CV_MAT32F,0,NULL};
	CvMat Result = cvMat(1,1,CV_MAT32F,NULL);

	cvmAlloc(&PointI);
	cvmAlloc(&PointJ);
	cvmAlloc(&PointDiff);
	cvmAlloc(&Result)

	float *pI = PointI.data.fl;
	float *pJ = PointJ.data.fl;

	memcpy(pI,Ci[i],FeatureLen*4);
	//memcpy(pI,Ci[i].data,sizeof(Ci[i].data));

	//for (int index=0;index<FeatureLen;index++)
	//		cvmSet(&PointI,0,index ,Ci[i].data[index]);

	memcpy(pJ,Ci[j],FeatureLen*4);
	//memcpy(pJ,MatFeatures[j].data,sizeof(MatFeatures[j].data));

	//for (index=0;index<FeatureLen;index++)
	//		cvmSet(&PointJ,0,index, MatFeatures[j].data[index]);

	cvmSub(&PointI,&PointJ,&PointDiff);
	cvmMulTransposed(&PointDiff,&Result,0);

	u=(float)cvmGet(&Result,0,0);

	cvmFree(&PointJ);
	cvmFree(&Result);
	cvmFree(&PointDiff);
	cvmFree(&PointI);
	return u;
}

int SimilarityMat::GetDistinct(int j)
{
  int max=0;
  int index=-1;
  int x,y,min,c;
  long *gestures_min;

  gestures_min=new long[all_gestures];
  for (c=0;c<all_gestures;c++)  //Initialization
        gestures_indices_out[c]=1;

  for (c=0;c<all_gestures;c++)  //Initialization
        gestures_min[c]=0;

  for (c=0;c<commands;c++) //gestures_indices_out[x]=1, mean x is not used in subset
        gestures_indices_out[gestures_indices[c]]=0;
//Helman way MinmMax
  // Now find the MIN distances between OUT and IN
   for (x=0;x<all_gestures;x++)
```

```
	{
		min=100000000; //Start from some value to compare
		for (y=0;y<all_gestures;y++)
			if ((Dist(x,y)<=min) && (gestures_indices_out[x]==1)
				&& (gestures_indices_out[y]==0) && (y!=j))
			{
				min=cvRound(Dist(x,y));
				gestures_min[x]=min;
			}
	}
			// Now find the MAX over the mins.
	for (x=0;x<all_gestures;x++)
	  if (gestures_min[x]>=max)
	  {
		max=gestures_min[x];
		index=x;
	  }

/// JUAN OLD WAY
//   for (c=0;c<all_gestures;c++)
//     if  ((Dist(j,c)>=max) && (gestures_indices_out[c]==1))
//       {
//        max=Dist(j,c);
//		index=c;
//       }

    delete [] gestures_min;
	return(index);
}
int SimilarityMat::GetIndexOfGesture(int g)
{
  int ind,c;
  for (c=0;c<commands;c++)
	if (g==gestures_indices[c])
		ind=c;
	return(ind);
}
void SimilarityMat::OrderGestureVector()
{
  int contador=0;
  long *gestvector_cpy;
  gestvector_cpy=new long[commands];
  for (int g=0;g<all_gestures;g++)
    for (int c=0;c<commands;c++)
	if (g==gestures_indices[c])
	{
		gestvector_cpy[contador]=g;
		contador++;
	}
  for (int c=0;c<commands;c++)
	gestures_indices[c]=gestvector_cpy[c];
	delete [] gestvector_cpy;
}
SimilarityMat::~SimilarityMat()
{
//   delete [] gestures_indices; //Don't delete this now, it is deleted
                                                  //later, at the end of the main program
  delete [] gestures_indices_out;
}
```

תקציר

ממשקים מבוססי מחוות ידיים מציעים חלופה לממשקים מסורתיים יותר כגון שלטים, תפריטים וממשקי מניפולציה ישירה. היכולת לציין עצמים, פעולות ,פקודות ניווט ופרמטרים נוספים באמצעות מחווה אינטואיטיבית אחת מושכת משתמשים מתחילים ומשתמשים מנוסים כאחד. ניתן לראות בממשקים מבוססי מחוות ידיים כחלופה מועדפת לטכניקות ממשק קיימות מכיוון שהן מציעות יתרונות כגון טבעיות, סטרילית ותגובה מהירה. אחד מהיתרונות הבולטים הוא סיוע לאנשים בעלי מוגבלויות פיזיות לגשת למחשבים ולהתקנים פיזיים נוספים. שלב מכריע בממשק מבוסס מחוות ידיים הנועד למשימות לא גנריות הוא הבחירה של מחוות הידיים הנכללות במשוב בקרה. לרוע המזל, פרוצדורות לתכנון שפות מחוות ידיים לאינטראקציה בין אדם למחשב לא נחקרו לעומק. יצירת שפת מחוות ידיים קשורה לבעיית אופטימיזציה בעלת מרחב חיפוש גדול במיוחד ואמורה להתבסס על גורמי שימושיות של המשתמשים וגם זיהוי המכונה. הגורמים הבאים הם המשפיעים ביותר על הביצועים של תכנון שפת מחוות ידיים מכוון אדם-מחשב:

1. **עייפות** (או נוחות): תקשורת מבוססת מחוות ידיים כוללת יותר אינטראקציה של השרירים מאשר עכבר או דיבור. פרק היד, האצבעות והזרוע כולם תורמים לביטוי הפקודות. לכן מחוות ידיים צריכות להיות תמציתיות ונוחות, ולמזער את המאמץ ביד ובזרוע. במיוחד, תכנון השפה חייב להימנע ממחוות ידיים שדורשות מתיחת שרירים גבוהה במשך זמן ממושך. מחוות חוזרות ומגושמות משפיעות באופן משמעותי על מתיחת הרקמות וגורם לחץ על עצם שורש כף היד. פרוצדורה מוצלחת תעודד תנוחות טבעיות ותזניח את אלו שגורמות מאמץ חוזר. שני סוגים של מאמץ נקבעו בתיזה הזו: מאמץ סטטי, שהוא המאמץ הדרוש על מנת להחזיק מחווה סטטית במשך זמן מוגדר, ומאמץ דינאמי, שהוא המאמץ הדרוש לביצוע מעבר בין מחוות סטטיות. מטריצת מאמץ S נוצרה על מנת לשמור את המידע אודות מדדי המאמץ של המחוות שנכללו במתודולוגיה נוכחית. מטריצה U היא פונקציה של S.
2. **אינטואיטיביות**: אינטואיטיביות היא הקוגניטיביות, הקשר הטבעי בין מחווה לפקודה או כוונה. דבר זה לא קשור לאילוצים הנובעים מאנטומית היד. מחוות מסובכות ולא טבעיות מוזכרות לעתים רחוקות על ידי המשתמשים בעת ביצוע משימה. המחווה צריכה להיות פשוטה, שניתן לזכור אותה אפילו כאשר אין פעולה קוגניטיבית ברורה הקשורה אליה. אינטואיטיביות קשורה ללמידה ויכולת זיכרון. גורמים נוספים שמשפיעים על המחוות המועדפות ע"י המשתמשים הם רקע כללי, רקע תרבותי ויכולות בלשניות של המשתמש. שני סוגים של אינטואיטיביות מוצגים בתיזה זו: אינטואיטיביות ישירה, שהיא קשורה לקשר הקוגניטיבי בין מחווה לפקודה, ואינטואיטיביות משלימה, שהיא קשורה לשימוש מחוות משלימות על מנת להציג פקודות משלימות. מטריצת האינטואיטיביות הישירה I מכילה מידע על אינטואיטיביות ישירה של סביבת העבודה. מידע לגבי האינטואיטיביות המשלימה נשמר במטריצת האינטואיטיביות המשלימה, IC. לכן קבוצת האינטואיטיביות היא {IC,I}.
3. **דיוק הזיהוי**: דיוק הזיהוי הוא אחוז המחוות שהתקבלו שזוהו נכון. זיהוי מחוות ידיים היא משימה קשה בתחום הראייה שהיא כוללת הנחות לגבי רקע אחיד/מורכב, מצבים סטטיים-דינאמיים, במודלים צבע של עור. מיקום, כיוון ותצורת אצבעות כף היד יכולות להוות דגש על הבדלים בין מחוות ידיים ולכן להשפיע על ההבחנה ביניהן. עיבוד תמונה ואלגוריתמים יעילים לזיהוי מהווים גורם מכריע לסיווג מחוות ידיים. אלגוריתם לזיהוי מחוות ידיים פותח על מנת למצוא את דיוק הזיהוי של שפת מחוות ידיים, A.

שני גורמים הראשונים, עייפות ואינטואיטיביות, הם ממוקדי אדם כאשר הגורם השלישי, דיוק, תלוי בתכונות המכונה (למשל: חומרה, תוכנה). תיזה זו עוסקת בתכנון אופטימאלי של שפות מחוות ידיים, תוך שיפור חווית השליטה של המשתמשים (אינטואיטיביות ונוחות) מבלי להשפיע על ההיבט הטכני (דיוק הזיהוי). שלושת גורמים אלו יקבעו את התכנון האופטימאלי של שפת מחוות הידיים.
מטרתה העיקרית של התיזה היא לנסח את בעיית התכנון האופטימאלי של שפת מחוות הידיים באופן קפדני, לפתח ולאמת מתודולוגיה לפתרון על-ידי שימוש בתכנות מתמטי, גישות יוריסטיות, אלגוריתמים לעיבוד תמונה והערכת מדדים פסיכו-פיזיולוגיים.

<u>מתודולוגיה</u>
שפת מחוות ידיים אופטימאלית, GV, מוגדרת כאוסף זוגות מחווה-פקודה אשר מביא למינימום את זמן ביצוע המשימה (או המשימות). אוסף הפקודות C מוגדר על-פי המשימה ואוסף המחוות נבחר מאוסף רחב של מחוות

ידיים, G_z. ביצוע המשימה תלוי בדיוק הזיהוי של תת קבוצת המחוות G_n, התלוי במדדי גורמי אנוש המייצגים את טבעיות האסוציאציות מחווה-פקודה, ונוחות המחוות.

<u>הגדרת הבעיה וגישות לפתרון</u>

הבעיה העיקרית היא לצמצם זמן ביצוע משימה עבור אוסף שפות מחוות הידיים האפשריות, GV. מאחר שזמן ביצוע משימה, כפונקציה של GV, אינו בעל צורה ידוע מראש, שלושה מדדי ביצוע מומלצים כקירובים: אינטואיטיביות $Z_1(GV)$, נוחות $Z_2(GV)$, ודיוק הזיהוי $Z_3(GV)$. מציאת המקסימום של כל המטרות בו זמנית מגדיר בעיית אופטימיזציה מרובת מטרות (MOP) הניתנת לפתרון על-ידי בחירת GV מתוך גבול הפרטו ע"פ עדיפויות "מקבל ההחלטות". ניתן למצוא את פתרון גבול הפרטו על-ידי מספור הפתרונות, אך גישה זו איננה מוצדקת אפילו עבור שפות בעלות ממדים סבירים בשל זמן חישוב גבוה.

<u>ארכיטקטורה</u>

ארכיטקטורת המתודולוגיה של שפות מחוות ידיים מורכבת משלושה מודולים טוריים. במודול 1 נקבעים גורמי אנוש פיזיו-פסיכולוגים. במודול 2 מתבצע חיפוש תת-אוסף מחוות ידיים אפשרי, המאולץ על-ידי דיוק זיהוי המחוות במכונה. מודול 3 מורכב מתהליך שידוך בין פקודות למחוות. אוסף המשימות T, אוסף מחוות הידיים הראשי הרחב G_z ואוסף הפקודות C, הם פרמטרי הקלט של מודול 1. אוסף כל הפקודות הדרושות על מנת לבצע את כל המשימות T מסומן ב C. מטרות מודול 1 הן יצירת אסוציאציות בין פקודות למחוות על סמך אינטואיטיביות המשתמש (ישירה ומשלימה), חיפוש מטריצת הנוחות המבוססת על מעברים בין פקודות, מדדי עייפות וצמצום האוסף הרחב של מחוות הידיים לאוסף הראשי G_m. עבור מודול 2, נתוני הקלט ההכרחיים הם האוסף הראשי של מחוות הידיים G_m, ואלגוריתם זיהוי למציאת A.

ודול זה משתמש בתהליך חיפוש איטרטיבי למציאת תת קבוצת מחוות הידיים האפשרי היחיד G_{n^*} (או לחלופין אוסף תתי קבוצות מחוות הידיים האפשריים) המספקת רמת דיוק זיהוי המוגדרת מראש על-ידי מקבל ההחלטות. שתי גישות מטה- יוריסטיות פותחו עבור תהליך החיפוש. הגישה הראשונה נקראת "מטריצת שגיאה מתפלגת" (DCM), והשנייה נקראת "פתרון נגזר ממטריצת שגיאה" (CMD). בנוסף, מוצגת דוגמה של מספור חלקי.

אלגוריתם מבוקר FCM הניתן לשינוי תצורה פותח לצורך קבלת דיוק זיהוי, A. הפרמטרים של אלגוריתם עיבוד התמונה ואישכול נמצאו בו זמנית על-ידי רוטינות "חיפוש פרמטרים שכנים". כמו כן פותחו שתי גרסאות לאלגוריתם ל"חיפוש שכנים מקומי". שתי גרסאות אלו הותאמו למערכת לכיול פרמטרי ביצוע עבור משימה, כאשר מספר הפרמטרים בווקטור הפתרון השתנה באופן דינאמי.

לצורך מציאת דיוק הזיהוי של תת קבוצת מחוות הידיים המועמדת, היה צורך לאמן את המסווג. שתי גישות שונות אומצו, אחת על-ידי אימון חוזר של ה-FCM פעמים רבות עבור כל מועמד G_n, והשנייה אימון ה- FCM וכיולו פעם אחת עבור האוסף הראשי G_m וממנו גזירת דיוק הזיהוי עבור מועמדי G_n. הגישה שנייה הינה מקורבת אך מהירה יותר. הקלט למודול השלישי הוא מטריצות האינטואיטיביות $V=\{I,IC\}$, מטריצת הנוחות U, פקודות C, ותת אוסף המחוות G_{n^*}. מטרת מודול זה היא לשייך את אוסף המחוות G_n עם אוסף הפקודות C, על מנת למקסם את מדדי גורמי אנוש. אלגוריתם ה-QAP לפתרון בשלמים פותר את בעיית שידוך המחוות לפקודות. תוצאת ההשמה מחוות-פקודות מהווה את שפת המחוות, GV.

<u>ניסויים, ניתוח ותוצאות</u>

המדדים הסובייקטים נתקבלו באמצעות סדרת ניסויים החוקרים תגובת בני אדם. הניסוי הראשון כלל מציאת מחוות אינטואיטיביות לבקרת מודל וירטואלי של זרוע רובוטית ו-VMR. על מנת לאסוף נתונים על אינטואיטיביות, הוצגה למשתמש סדרת פקודות (שנלקחו ממשימות מוגדרות מראש על זרוע רובוטית ועל VMR), והמשתמש שייך את המחוות לפקודות אלו באופן חופשי. מחוות הידיים של המשתמש נתקבלו כאשר הוא יצר פיזית את המחווה, והזין את המידע הקשור לתצורה של המחווה. בחירת המחוות שמר על הכלל 70/30, כאשר 70% של המשתמשים השתמשו רק ב-30% מהמחוות שבשפה. נתון זה סותר את הטענה שמשתמשים משתמשים באופן עקבי באותן מחוות על מנת להציג את אותן פקודות לביצוע משימות, כפי שהוצע על-ידי האופטמן [Hauptmann and McAvinney , 1993].

לגבי מדדי העייפות, ניסוי ארגונומי התנהל ובו המחוות דורגו על ידי המשתמש, מחלש לחזק בסקאלת בורג [Borg, 1982]. המודל המתאר את מאמץ (עייפות) המעבר פותח ואומת על סמך מדדי עייפות סטטית עבור כל המחוות באוסף המחוות הראשי, אוסף מצומצם של מדדי מאמץ המעבר. על פי המודל, 90% מהמאמץ הדינאמי ומשך פעולתו נקבעים על-ידי המחווה סופית במעבר בין שתי מחוות, ורק 10% על-ידי המחווה ההתחלתית. באמצעות יחס זה חיזוי המאמץ הדינאמי ומשך פעולתו מבוססים על סמך מדדי מאמץ סטטי. מודל חיזוי זה חסך 86 שעות מניסויים סובייקטיביים.

לצורך אימות המודל שתי קבוצות של GV נוצרו: V_G קבוצת השפות בעלות אינטואיטיביות גבוהה, נוחות וקלות לזיהוי, ו-V_B קבוצת השפות בעלות אינטואיטיביות נמוכה, קשות לביצוע ובעלות דיוק זיהוי נמוך. GV_G ו-GV_B מייצגות מדגם של שפות מ- V_G ו-V_B בהתאמה.

אימות התהליכים האנליטיים למציאת שפות מחוות ידיים אופטימאלי מורכב מבדיקת ההנחות הבאות: א) H_1: Min $(\max(Z_1), \max(Z_2)) \propto \tau(GV^*)$ – זמן ביצוע משימה τ יכול להיות מוצג על-ידי מדדי קירוב ו-$\max(Z_3)$ לפונקציה מרובת מטרות. יתרה מזאת, מקסום פונקציה מרובת מטרות גורם למזעור זמן ביצוע המשימה. ב) H_2: $\tau(GV_G) < \tau(GV_B)$- שימוש ב- GV_G גורם לזמני השלמת משימה קצרים יותר מאשר GV_B. ג) H_3: $m(GV_G) > m(GV_B)$ – שפות מסוג GV_G קלות יותר לזיכרון מאשר שפות מסוג GV_B.

על מנת לבדוק את שתי ההנחות הראשונות (H1, H2), בוצע מבחן סטטיסטי t בין זמנים סטנדרטים להשלמת המשימה עבור 8 שפות V_G ו-8 שפות V_B למשימות הזרוע הרובוטית וגם ה-VMR. הזמן הממוצע להשלמת המשימות כאשר משתמשים ב-V_G היה קצר יותר מהזמן כאשר משתמשים ב-V_B

($\tau(GV_G)$ =87.98 sec < $\tau(GV_B)$=118.95 sec עם p=0.0059)

ו-($\tau(GV_G)$=114.67 sec < $\tau(GV_B)$=153.04 sec עם p=0.00031), עבור משימות הזרוע הרובוטית וה-VMR, בהתאמה.

זמן הלמידה של משתמש עבור GV ומשימה ספציפיים חושב במונחים של קצב הלמידה הנובע מעקומת הלמידה שלו. נמצא שזמן הלמידה היה נמוך עבור V_G ביחס ל-V_B (עבור המשימה עם הזרוע הרובוטית 0.785>0.797 ועבור משימת ה-VMR היה 0.827>0.835) המעיד על למידה מהירה יותר. ההנחה האחרונה (H_3), הציעה שה-GV_G הינה קלה לזכירה מאשר GV_B. זכירה נקבעה על-ידי יכולת הזיכרון של אסוציאציות מחוות-פקודה של משתמש מנוסה. ציון הזכירה הממוצע עבור משימת הזרוע הרובוטית כאשר משתמשים ב-V_G היה גבוה מהציון הממוצע שהתקבל משימוש ב- V_B (87.5 ו-70.83% עם p=0.05), בכל אופן לא היה הבדל משמעותי בין הזכירה עבור המשימות עם ה-VMR. כל התוצאות האלו ניתנות לניסוח באופן הבא:שימוש ב- GV ים עם ערכים גבוהים עבור שלושת המטרות גורם להקטנת זמן ביצוע, למידה מהירה וזכירה גבוהה.

<u>מסקנות</u>

בתיזה זו הוצגה מתודולוגיה לתכנון שפות מחוות ידיים טבעיות, הכוללת היבטים פסיכו-פיזיולוגים (אינטואיטיביות ונוחות) וטכניים (דיוק זיהוי), ומאחדת את שני ההיבטים בגישה אחידה.

התרומות העיקריות במחקר זה הן:

ניסוח אנליטי של בעיית תכנון GV: פותחה מתודולוגיה למציאת שפת מחוות ידיים אופטימאלית באמצעות גישה אנליטית. המטרה העיקרית של המתודולוגיה היא למנוע בחירה שרירותית של מחוות ידיים כאשר מתכננים יישום אדם-רובוט למשימות בפקודות ידועות מראש. התרומה היא ניסוח מתמטי קפדני הכולל שיטות אופטימיזציה מיושמות, אילוצים, ומדידת איכות הפתרון. **אלגוריתם לזיהוי מחוות ידיים הניתן לשינוי תצורה**: בעיית כיול הפרמטרים בו זמנית של מערכת לזיהוי מחוות ידיים מבוסס על עיבוד תמונה- Fuzzy C Means (FCM). הוצע אלגוריתם למיכון תהליך כיול הפרמטרים. תכנון מערכת מונחית מחוות ידיים הוגדר כבעיית אופטימיזציה.

שתי שיטות לפתרון בעיית תכנון GV: פותחו שתי שיטות לפתרון בעיית התכנון האופטימאלי של השפה: גישה מרובת מטרות החלטה ותהליך לפירוק לשני שלבים. לבעיה ראשונה, בוצע מספור מקורב של הפתרונות, ותת קבוצה של פתרונות בלתי נשלטים נבחר לתצוגה למקבל ההחלטות. שיטת פירוק לשני שלבים הינה בעיית מטרות דואלית, כאשר מטרות דיוק זיהוי ומטרות ממוקדי אדם (אינטואיטיביות ונוחות) מקבלות עדיפות ראשונה ושנייה בהתאם. **פיתוח אינדקסים לאינטואיטיביות ונוחות, ושיטה אוטומטית לאספן**: מטרות הקשורות לגורמים פסיכו-פיזיולוגיים אנושיים, נוחות ו אינטואיטיביות, התווספו למחקר זה. עוצבו ניסויים למציאת רמת אסוציאטיביות קוגניטיבית של המשתמש (אינטואיטיביות) בין זוגות פקודות-מחוות על סמך הדמיית תרחישים שונים ולימוד אופן החלטת המשתמש על האסוציאציות הטבעיות ביותר בין פקודות למחוות. בנוגע לאינטואיטיביות, בחירת המחוות שמר על הכלל – 70/30 , כאשר 70% מן הפרטים משתמשים ב-30% מהמחוות בשפה. הוגדר מדד לאינטואיטיביות משלימה כאסוציאציה קוגניטיבית בין זוגות פקודות משלימות (כגון: למעלה-למטה) לזוג מחוות משלימות (כגון: אגודל למעלה-אגודל למטה). בנוסף על כך, שני סוגים של מאמץ (עייפות) זוהו: סטטי ודינאמי. מודל פותח על מנת לחזות את המאמץ הדינאמי ומשכו על סמך מדדי מאמץ סטטיים.

תוצאות אימות ושימושיות: GV בעל אינטואיטיביות ,נוחות ודיוק זיהוי גבוהים גורם לזמני ביצוע משימה קצרים יותר, לימוד מהיר יותר וזכירה גבוהה יותר.

מילות מפתח: תכנון שפת מחוות ידיים, ראיית מכונה, fuzzy c-means, בחירת מאפיינים, עיבוד תמונה, זיהוי מחוות ידיים, ממשקי אדם מחשב, שליטה רובוטית, גורמי אנוש, מחוות אינטואיטיביות, עייפות היד.

העבודה נעשתה בהדרכה של פרופ' הלמן שטרן ופרופ' יעל אידן

המחלקה להנדסת תעשיה וניהול

הפקולטה למדעי ההנדסה

שיטה מיטבית לפיתוח שפת מחוות ידיים לשליטת רובוט וירטואלי

מחקר לשם מילוי חלקי של הדרישות לקבלת

"דוקטורט לפילוסופיה"

מאת

חואן ווקס

הוגש לסינאט אוניברסיטת בן-גוריון בנגב

אישור מנחה פרופ' הלמן שטרן _____

פרופ' יעל אידן _____

אישור דיקן בית הספר ללימודי מחקר מתקדמים _____

תשס"ו 2006

באר-שבע

שיטה מיטבית לפיתוח שפת מחוות ידיים לשליטת רובוט וירטואלי

מחקר לשם מילוי חלקי של הדרישות לקבלת

"דוקטורט לפילוסופיה"

מאת

חואן ווקס

הוגש לסינאט אוניברסיטת בן-גוריון בנגב

תשס"ו 2006

באר-שבע