3D OBJECTS DESCRIPTION AND CLASSIFICATION BY IMPLICIT POLYNOMIALS

HILLA BEN-YAACOV

3D OBJECTS DESCRIPTION AND CLASSIFICATION BY IMPLICIT POLYNOMIALS

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

HILLA BEN-YAACOV

SUBMITTED TO THE SENATE OF THE TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY

ELUL, 5768 HAIFA SEPTEMBER, 2008

THE RESEARCH THESIS WAS DONE UNDER THE SUPERVISION OF PROF. DAVID MALAH AND DR. MEIR BARZOHAR IN THE FACULTY OF ELECTRICAL ENGINEERING

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY ACKNOWLEDGED

Contents

List of Figures

List of Tables

Abstract

Implicit polynomials (IP) are used for the representation of 2D curves and 3D surfaces specified by discrete data. We explore the description abilities of existing 3D implicit polynomials fitting algorithms, Gradient1, Min-Max and Min-Var, and suggest a modification for the Min-Max and Min-Var algorithms, so that they will be rotation invariant. We develop a set of 3D rotation invariants that are linear combinations of the IP coefficients, using a tensor representation of the IP, and two 3D quadratic rotation invariants using trigonometric identities. We explore the quaternion representation as an alternative method for rotation invariants derivation, in a similar way to the complex representation used in 2D. We describe the pre-processing stages required in order to improve the classification performance: locating the center of mass at the origin, scaling, mirroring and selecting 2D projections. We then present a 3D classification method which is based on the Multi Order and Fitting Errors Technique (MOFET) proposed earlier for 2D object classification. This classification approach is based on fitting several polynomials to the object surface, each having a different degree, and on their fitting errors. The advantage of this approach, is that it does not require the use of high computational registration (pose estimation) between the new object

representation we would like to classify and the representations of different objects in the dictionary. Instead, it uses a rotation invariant features vector for classification. The classification features we use are the 3D IP rotation invariants and fitting errors, as well as 2D IP rotation invariants and fitting errors, derived from the most informative 2D projections of the 3D objects, and 3D PCA eigenvalues. We demonstrate the classification results on both a rigid objects database and a faces database (acquired in a cooperative situation). Simulation results show that our method outperforms classification based on an IP fitting after pose estimation as well as the Shape Spectrum Descriptor (SSD) classification, which was adopted by the MPEG-7 standard.

List of Symbols

a_{klm}- implicit polynomial coefficient

$\underline{a}$ - implicit polynomial coefficients vector

$\underline{p}^n(x,y,z)$ - monomials vector (of implicit polynomial of degree n)

$P^n(x,y,z)$ - implicit polynomial of degree n

$P^n_{\underline{a}}(x,y,z)$- implicit polynomial of degree n and coefficients vector $\underline{a}$

$\nabla P^n_{\underline{a}}(x,y,z)$ - gradient vector of the polynomial $P^n_{\underline{a}}(x,y,z)$

$e_{\underline{a}}(i)$ - fitting error of the implicit polynomial $P^n_{\underline{a}}(x,y,z)$ at the i^{th} data-point

$\underline{p}^n_x(x,y,z)$ - the monomial vector derivative with respect to x

$\underline{p}^n_y(x,y,z)$- the monomial vector derivative with respect to y

$\underline{p}^n_z(x,y,z)$- the monomial vector derivative with respect to z

$\underline{p}^n_k(x,y,z)$ - the k^{th} element of the monomials vector

$S_{\underline{a}}$ - sensitivity function with respect to the polynomial coefficients

$\underline{\varepsilon}(x,y,z)$ - the change in the location of a zero-set point

$\underline{\delta}_{\underline{a}}$ - the change in the polynomial coefficients

$H_r(x,y,z)$ - a form of degree r (a homogeneous polynomial of degree r)

S_n - the tensor representation of a form of degree n (H_n)

$V(H_n)$ - the 3x3 matrix created by contractions of a tensor of order $n = 2p$

$L_{2D,k}$ - a 2D linear rotation invariant, derived from a k^{th} degree form

$Q_{2D,k}$ - a 2D quadratic rotation invariant, derived from a k^{th} degree form

$L_{3D,k}$ - a 3D linear rotation invariant, derived from a k^{th} degree form

$Q_{3D,k}$ - a 3D quadratic rotation invariant, derived from a k^{th} degree form

$S_{75\%}$ - scaling factor (the 75^{th} percentile of the distances from the origin)

$E_{75\%,3D,n}$ - 3D IP (degree n) fitting error (the 75^{th} percentile of $e_{\underline{a}}(i)$)

$E_{75\%,2D,n}$ - 2D IP (degree n) fitting error (the 75^{th} percentile of $e_{\underline{a}}(i)$)

List of Abbreviations

Gradient1 - Gradient-One Fitting Algorithm

IP - Implicit Polynomial

ICP - Iterative Closest Point

LS - Least Squares

Min-Max - Min-Max Fitting Algorithm

Min-Var - Min-Var Fitting Algorithm

MOFET - Multi Order and Fitting Errors Technique

MPEG - Moving Picture Experts Group

PDF - Probability Density Function

PCA - Principal Component Analysis

SSD - Shape Spectrum Descriptor

Chapter 1

Introduction

Implicit polynomials (IP) are used for the representation of 2D curves and 3D surfaces specified by discrete data [1, 2, 6, 7, 8]. Over time, different algorithms for fitting Implicit Polynomials to the data were developed. The state of the art fitting algorithms are Gradient1 [1], Min-Max and Min-Var [2], all of them apply a linear Least Squares (LS) solution to the fitting problem. Compared with older nonlinear iterative algorithms [5], the Least Squares algorithms have a much better performance in both representation and classification tasks, with lower complexity. These abilities were previously tested mainly for 2D implicit polynomials [3]. In this work we first explore the description abilities of existing 3D polynomial fitting algorithms for both low and high degree fitting. We then use a tensor representation of implicit polynomials in order to derive a set of rotation invariants which are linear combinations of the IP coefficients. Using the 3D coordinate system rotation matrix and trigonometric identities, we derive two rotation invariants which are quadratic combinations of the IP coefficients.

The IP based 3D rotation invariants are well suited for the Gradient1 fitting algorithm which is rotation invariant (i.e., the relation between polynomials fitted to different orientations of the same data is characterized by rotation only). In order to apply them for classification using other fitting algorithms, we suggest a modification of Min-Max and Min-Var fitting algorithms so that they will be rotation invariant as well. This modification is based on the 2D rotation invariant Min-Max and rotation invariant Min-Var [3].

Following the 2D IP classification method Multi Order and Fitting Error Technique (MOFET) [3], we propose a classifier based on 3D IP rotation invariants and fitting errors, 2D IP rotation invariants and fitting errors (from the most descriptive 2D projections) and the eigenvalues of a PCA decomposition. The suggested classifier uses various IP degrees (both for 2D and 3D) in order to utilize both the stability of low degree polynomials and the descriptiveness of high degree polynomials. We explore the results of our classifier using Gradient1, Rotation-Invariant Min-Max and Rotation-Invariant Min-Var fitting algorithms. We apply our method to both a rigid objects database and a faces database (in a cooperative situation) and compare our classification results with pose estimation methods followed by IP fitting and with the Shape Spectrum Descriptor (SSD) technique, which was adopted by the MPEG-7 standard for 3D descriptors.

The rest of the thesis is organized as follows:

Chapter 2 provides background information on Implicit Polynomials fitting algorithms, and presents a rotation invariant modification of Min-Max and Min-Var algorithms.

Chapter 3 reviews existing pose estimation approaches, one of them is based on implicit polynomials.

Chapter 4 describes the process of rotation invariants derivation using various approaches.

Chapter 5 deals with feature extraction for 3D object recognition. We describe the classification problem and the objects and faces databases. We discuss the pre-processing stages and the classification features selection.

Chapter 6 describes the classifier design (learning and testing) for both the objects and the faces databases.

Chapter 7 describes our classification results and compares them with other classification methods.

Chapter 8 summarizes the work, emphasizes its main contributions and suggests research directions for future study.

Chapter 2

3D Object Description by Implicit Polynomials

2.1 Background

Implicit polynomials (IP) are used for efficient representation of 2D curves and 3D surfaces specified by discrete data. The ability to efficiently describe complicated boundaries using the coefficients of implicit polynomials is attractive for applications in the fields of object recognition, and might also be used as a basis for pose estimation.

Over time, different algorithms for fitting implicit polynomials to the data were developed.

The technique assumes that the data points are part of a zero set of a polynomial of degree n (i.e., $P^n(x, y, z) = 0$, where $P^n(x, y, z)$ is a polynomial of degree n). Therefore, the first and most intuitive requirement that the polynomial should satisfy is that its value should be zero at the location

of the data points.

Note that a polynomial may contain several zero-sets, where a zero-set is defined as a continuous surface satisfying $P^n(x,y,z) = 0$. Actually, a polynomial of degree n may have up to n zero sets.

2.2 Problem Definition

The input is a 3D object, described by a set of discrete data (a cloud of data points $\{(x_i, y_i, z_i)\}_{i=1,...,N}$).

The output of the fitting process is a 3D implicit polynomial $P^n_{\underline{a}}(x,y,z)$:

$$\begin{aligned} P^n_{\underline{a}}(x,y,z) = \; & a_{000} + a_{100}x + a_{010}y + a_{001}z + a_{200}x^2 + a_{110}xy + \\ & + a_{020}y^2 + a_{101}xz + a_{011}yz + a_{002}z^2 + ... + \\ & + a_{n00}x^n + ... + a_{0n0}y^n + ... + a_{00n}z^n \end{aligned} \tag{2.1}$$

Where a_{klm} is the coefficient of the monomial with x to the power of k, y to the power of l and z to the power of m.

The IP is defined by its coefficients vector $\underline{a}$:

$$\underline{a} = [a_{000}\ a_{100}\ a_{010}\ a_{001}\ a_{200}\ a_{110}\ a_{020}\ a_{101}\ a_{011}\ a_{002}\ ...\ a_{n00}\ ...\ a_{0n0}\ ...\ a_{00n}]^T \tag{2.2}$$

and its monomials vector $\underline{p}^n(x,y,z)$:

$$\underline{p}^n(x,y,z) = \left[1\ x\ y\ z\ x^2\ xy\ y^2\ xz\ yz\ z^2\ ...\ x^n\ ...\ y^n\ ...\ z^n\right]^T \tag{2.3}$$

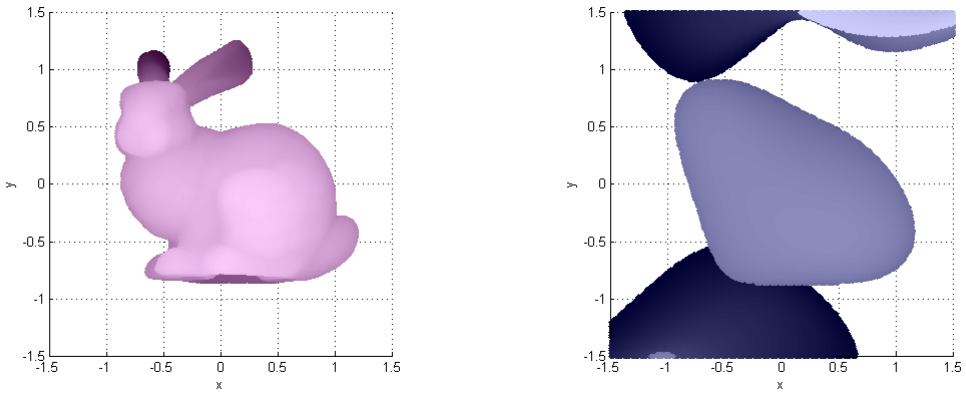
and can be also written as:

$$P^n(x,y,z) = \underline{a}^T \underline{p}^n(x,y,z) \tag{2.4}$$

The dimension of $\underline{a}$, as well as the dimension of $\underline{p}^n(x,y,z)$, is $\frac{(n+1)(n+2)(n+3)}{2 \cdot 3}$.

The fitting problem definition is therefore to find a coefficients vector $\underline{a}$ that describes the polynomial which best fits the data under certain criteria.

An example of a 3D object and an implicit polynomial describing it is shown in Fig. 2.1. In this figure (as well as in all the figures to follow), the lighter the shade of the point - the higher z value it has (i.e., it is closer to the reader).



(a) (b)

Figure 2.1: "The Stanford Bunny" object (taken from "The Stanford 3D Scanning Repository" [9]) : (a) the original data points (b) the zero-sets of a 4th degree polynomial attempting to fit the object. Note that several zero-sets appear in the image, but only the central one describes the original object.

2.3 Overview of Existing Fitting Algorithms

2.3.1 Least Squares Approach

The classical and simplest way to fit an algebraic surface to a data set, is to minimize the algebraic distance over a set of given data points.

Let us denote by $e_{\underline{a}}(i)$ the minimum Euclidean distance (i.e., the fitting error) between the i^{th} point of the data set and the zero sets of the polynomial $P^n_{\underline{a}}(x, y, z)$.

If we use the l_2 norm, we would like to minimize the sum of squares of the fitting errors:

$$\min_{\underline{a}} \sum_{i=1}^{N} \left(e_{\underline{a}}(i)\right)^2 \tag{2.5}$$

This problem formulation leads to an iterative approach with high computational cost [4], since in each iteration one has to go over all the data points, and for each point - to look for the closest zero-set point.

A less accurate but easier to solve minimization problem is:

$$\min_{\underline{a}} \sum_{i=1}^{N} \left(P^n_{\underline{a}}(x_i, y_i, z_i)\right)^2 \tag{2.6}$$

Here we don't minimize the distance between the data points and the zero sets of the polynomial. Instead, we wish to minimize the values of the IP at the locations of the data points, thus forcing the zero set to lie nearby.

This is actually the first requirement we mentioned earlier - the polynomial value should be zero at the location of the data points.

We define the matrix of monomials as:

$$M_0 = \left[\begin{array}{ccccc} \underline{p}^n(x_1,y_1,z_1) & \underline{p}^n(x_2,y_2,z_2) & \underline{p}^n(x_3,y_3,z_3) & \ldots & \underline{p}^n(x_N,y_N,z_N) \end{array} \right]^T \tag{2.7}$$

since for each data point $P^n_{\underline{a}}(x_i,y_i,z_i) = \underline{a}^T\underline{p}^n(x_i,y_i,z_i)$, the minimization problem can be written as:

$$\min_{\underline{a}} \|M_0\underline{a}\|^2 \tag{2.8}$$

Where $\|\cdot\|$ is l_2 norm.

The optimization problem became a linear least squares problem. The trivial solution to this minimization problem is $\underline{a} = \underline{0}$ (where $\underline{0}$ is a vector of zeros of size $\frac{(n+1)(n+2)(n+3)}{2\cdot 3}$). In order to avoid the trivial solution, we can add an additional constraint, for example $\|\underline{a}\| = c$ where c is a positive constant. Using this constraint the solution is known to be the eigenvector which corresponds to the minimal eigenvalue of $M_0^T M_0$.

This solution is highly unstable when the data points are affected by noise, and small perturbation of the data may result in very different solutions (i.e., different polynomials). This sensitivity to noise has led to more advanced, state of the art, least square algorithms - Gradient1, Min-Max and Min-Var. All three of them use additional requirements in order to improve the stability of the polynomial coefficients in the presence of noise.

An example to fitting artifacts is shown in Fig. 2.2. In this example, the unconstrained least squares solution is unbounded and intersects itself several times.

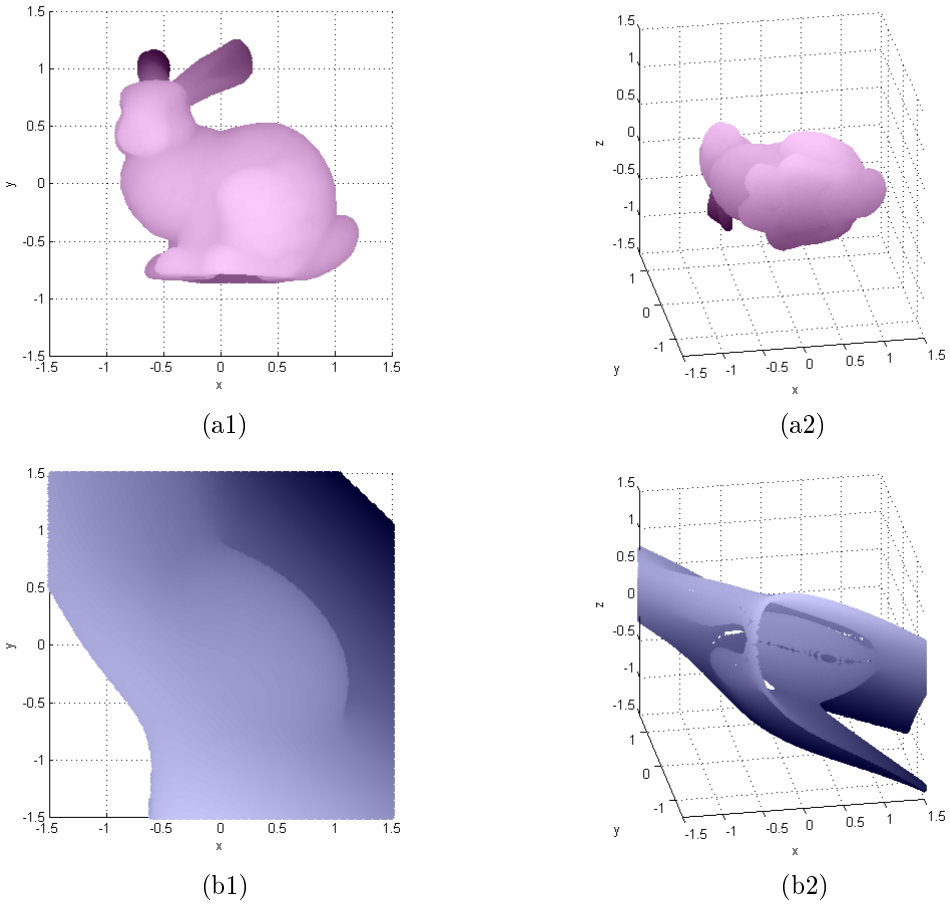


Figure 2.2: An example for fitting artifacts: (a1,a2) the original data points (b1,b2) the zero-set of a 4^{th} degree polynomial fit. The left column and the right column describe different points of view

2.3.2 Gradient1 Fitting Algorithm

The Gradient1 fitting algorithm is presented in [1].

The first fitting criterion is the demand that the zero set of the polynomial will fit the data-set.

The Gradient1 algorithm requires one more fitting criterion: the polynomial gradient at the location of each data point needs to be in the same

direction as the normal vector of the data set in the same location. In addition, the algorithm requires that the value of the gradient will be the same (and equal to 1) at all the data-set points. The unit norm value of the gradient can be replaced by any positive constant value, as long as it is required to be the same at all the data points. Setting it equal to a constant c will scale all the polynomial coefficients by c, but will not change the zero-sets of the polynomial. Therefore, it is chosen to be 1 without loss of generality.

It is a well-known fact that the gradient vector of a surface at a certain point is perpendicular to the surface tangent plane at this point. Therefore, in order to calculate the gradient vector of the data-set at each point, a local plane is fitted to the data point and its closest neighbors (see Fig. 2.3). The normal vector to the plane is then calculated and scaled to be a unit vector.

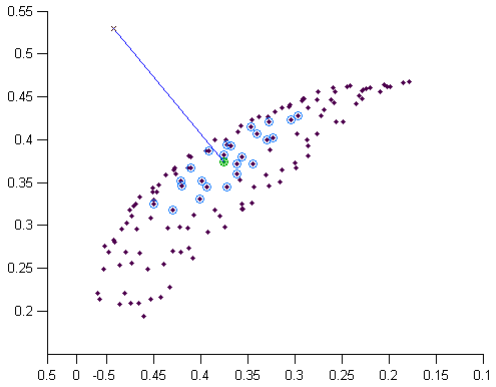


Figure 2.3: Estimation of the gradient direction at a certain data point, using a tangent plane fitted to the the point and its 24 closest neighboring data points

We define $\underline{p}_x^n(x,y,z)$, $\underline{p}_y^n(x,y,z)$ and $\underline{p}_z^n(x,y,z)$ as the derivatives of the monomials vector $\underline{p}^n(x,y,z)$, with respect to x, y and z, respectively.

We denote the local normal unit vector at each of the data set points

by (v_k, w_k, q_k) where $k = 1, 2, ..., N$ and N is the number of points in the data-set.

We construct the matrix of monomials and the matrices of monomials partial derivatives in the following way:

$$M_0 = \begin{bmatrix} \underline{p}^n(x_1, y_1, z_1) & \underline{p}^n(x_2, y_2, z_2) & ... & \underline{p}^n(x_N, y_N, z_N) \end{bmatrix}^T \tag{2.9}$$

$$M_x = \begin{bmatrix} \underline{p}^n_x(x_1, y_1, z_1) & \underline{p}^n_x(x_2, y_2, z_2) & ... & \underline{p}^n_x(x_N, y_N, z_N) \end{bmatrix}^T \tag{2.10}$$

$$M_y = \begin{bmatrix} \underline{p}^n_y(x_1, y_1, z_1) & \underline{p}^n_y(x_2, y_2, z_2) & ... & \underline{p}^n_y(x_N, y_N, z_N) \end{bmatrix}^T \tag{2.11}$$

$$M_z = \begin{bmatrix} \underline{p}^n_z(x_1, y_1, z_1) & \underline{p}^n_z(x_2, y_2, z_2) & ... & \underline{p}^n_z(x_N, y_N, z_N) \end{bmatrix}^T \tag{2.12}$$

and the following gradient coordinates vectors:

$$\underline{v} = \begin{bmatrix} v_1 & v_2 & v_3 & ... & v_N \end{bmatrix}^T \tag{2.13}$$

$$\underline{w} = \begin{bmatrix} w_1 & w_2 & w_3 & ... & w_N \end{bmatrix}^T \tag{2.14}$$

$$\underline{q} = \begin{bmatrix} q_1 & q_2 & q_3 & ... & q_N \end{bmatrix}^T \tag{2.15}$$

and by defining $M = \begin{bmatrix} M_0 \\ M_x \\ M_y \\ M_z \end{bmatrix}$ and $\underline{b} = \begin{bmatrix} \underline{0} \\ \underline{v} \\ \underline{w} \\ \underline{q} \end{bmatrix}$, we can formulate the fitting criteria as:

$$M\underline{a} = \underline{b} \tag{2.16}$$

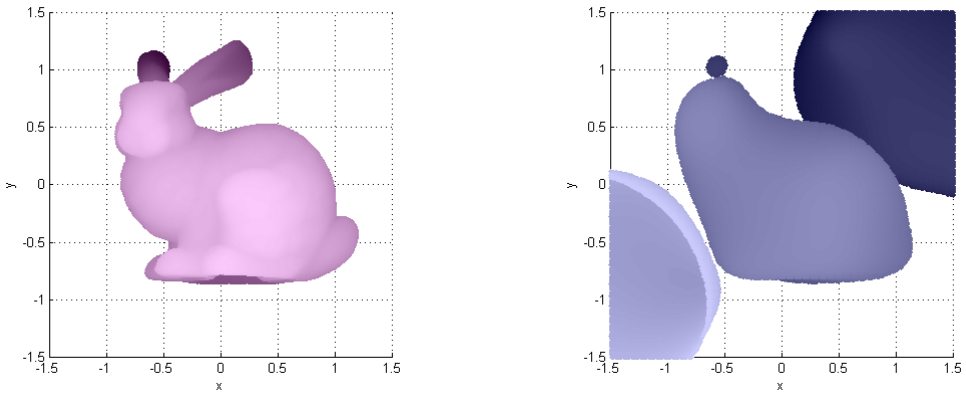
If we formulate it as a minimization problem of the l_2 norm of the fitting error vector we get:

$$\min_{\underline{a}} \|M\underline{a} - \underline{b}\|^2 \tag{2.17}$$

which is again a linear least squares problem with a known solution:

$$\underline{a}_{LS} = \left(M^T M\right)^{-1} M^T \underline{b} \tag{2.18}$$

Fig. 2.4 shows an example of the Gradient1 fitting.



(a) (b)

Figure 2.4: Gradient1 fitting using polynomial of 6^{th} degree applied to "The Stanford Bunny": (a) original data points (b) 6^{th} degree polynomial fit

2.3.3 Min-Max and Min-Var Fitting Algorithms

Although Gradient1 has good fitting properties, it still suffers from zero-set stability problems. An implicit polynomial may have several zero-sets, and only one of them is describing the data-set. The zero-sets which do not describe the data-set are called *spurious zero-sets*. When using Gradient1 fitting algorithm, the spurious zero-sets are sometimes very close to the desired

zero-set (which describes the data-set). Consequently, if the data points are perturbed, these external zero-sets might become too close, and in extreme cases - even intersect the desired zero-set.

The Min-Max and Min-Var fitting algorithms are presented in [2, 19].

These improved algorithms are based on analysis of the sensitivity of zero-set points to small changes in the polynomial coefficients. Small changes in the position of zero-set points along a tangent direction move zero-set points back into the zero-set. Therefore, for the purpose of evaluating zero-set changes, it is sufficient to examine changes in the direction that is perpendicular to the zero-set.

We denote by $d\underline{a}$ the change in the polynomial coefficients, by $dP^n_{\underline{a}}(x,y,z)$ the change in each zero-set point, and by $du(x,y,z)$ the component of $dP^n_{\underline{a}}(x,y,z)$ that is locally perpendicular to the zero-set.

The sensitivity function, which we would like to minimize, is therefore:

$$S_{\underline{a}} = \frac{du(x,y,z)}{d\underline{a}} \tag{2.19}$$

In [2] it is shown that the sensitivity function can be calculated at each point by:

$$S_{\underline{a}} = \frac{\underline{p}^n(x,y,z)}{\left\|\nabla P^n_{\underline{a}}(x,y,z)\right\|} \tag{2.20}$$

where $\nabla P^n_{\underline{a}}(x,y,z)$ is the gradient of the polynomial at each point.

The change in the location of a zero-set point $\underline{\varepsilon}(x,y,z) = [\varepsilon_x, \varepsilon_y, \varepsilon_z]^T$ resulting from a small change in the polynomial coefficients (denoted by

$\underline{\delta}_{\underline{a}} = [\delta_{a000}, \delta_{a100}, ..., \delta_{a00n}]^T$) can be approximated by:

$$\underline{\varepsilon}(x,y,z) \cong S_{\underline{a}}^T \underline{\delta}_{\underline{a}} = \frac{\left\langle \underline{p}^n(x,y,z), \underline{\delta}_{\underline{a}} \right\rangle}{\left\| \nabla P_{\underline{a}}^n(x,y,z) \right\|} \tag{2.21}$$

If we denote by δ_{max} the maximum of the absolute values of $\underline{\delta}_{\underline{a}}$, then for a given zero-set point (x,y,z), the maximum error is bounded by:

$$max\left|\underline{\varepsilon}(x,y,z)\right| \leq \delta_{max} \frac{\sum_{k=1}^{r} \left| \underline{p}_k^n(x,y,z) \right|}{\left\| \nabla P_{\underline{a}}^n(x,y,z) \right\|} \tag{2.22}$$

where $r = \frac{(n+1)(n+2)(n+3)}{2 \cdot 3}$.

And if the components of $\underline{\delta}_{\underline{a}}$ are independent random variables with zero mean and variance of σ_δ^2, the variance of of the error is:

$$var\left\{\underline{\varepsilon}(x,y,z)\right\} = \sigma_\delta^2 \frac{\sum_{k=1}^{r} \left| \underline{p}_k^n(x,y,z) \right|^2}{\left\| \nabla P_{\underline{a}}^n(x,y,z) \right\|^2} \tag{2.23}$$

The Min-Max fitting algorithm has three fitting requirements:

1. The zero set of the polynomial should fit the data-set.

2. The polynomial gradient at the location of each data point should be in the same direction as the normal vector of the data set in the same location.

3. Minimize the maximum error.

Since no data point has priority over any other point, we can limit the maximal fitting error to a constant value by requiring that the value of

$max\left|\underline{\varepsilon}\left(x_i, y_i, z_i\right)\right|$ would be the same for all the given data points. Since the value of this constant does not affect the optimization, we require that for $i = 1, 2, ..., N$:

$$\left\|\nabla P_{\underline{a}}^{n}\left(x_i, y_i, z_i\right)\right\| = \sum_{k=1}^{r}\left|\underline{p}_{k}^{n}\left(x_i, y_i, z_i\right)\right| \tag{2.24}$$

Prioritizing the error minimization at different data points can be done, using different weights for the error requirements at different points.

This additional requirement modifies the least squares matrices of (2.9)-(2.12) in the following way:

$$M_0 = \left[\begin{array}{cccc} \underline{p}^{n}\left(x_1, y_1, z_1\right) & \underline{p}^{n}\left(x_2, y_2, z_2\right) & ... & \underline{p}^{n}\left(x_N, y_N, z_N\right) \end{array}\right]^T \tag{2.25}$$

$$M_x = \left[\begin{array}{cccc} \frac{\underline{p}_{x}^{n}(x_1,y_1,z_1)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_1,y_1,z_1)\right|} & \frac{\underline{p}_{x}^{n}(x_2,y_2,z_2)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_2,y_2,z_2)\right|} & ... & \frac{\underline{p}_{x}^{n}(x_N,y_N,z_N)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_N,y_N,z_N)\right|} \end{array}\right]^T \tag{2.26}$$

$$M_y = \left[\begin{array}{cccc} \frac{\underline{p}_{y}^{n}(x_1,y_1,z_1)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_1,y_1,z_1)\right|} & \frac{\underline{p}_{y}^{n}(x_2,y_2,z_2)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_2,y_2,z_2)\right|} & ... & \frac{\underline{p}_{y}^{n}(x_N,y_N,z_N)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_N,y_N,z_N)\right|} \end{array}\right]^T \tag{2.27}$$

$$M_z = \left[\begin{array}{cccc} \frac{\underline{p}_{z}^{n}(x_1,y_1,z_1)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_1,y_1,z_1)\right|} & \frac{\underline{p}_{z}^{n}(x_2,y_2,z_2)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_2,y_2,z_2)\right|} & ... & \frac{\underline{p}_{z}^{n}(x_N,y_N,z_N)}{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}(x_N,y_N,z_N)\right|} \end{array}\right]^T \tag{2.28}$$

The gradient coordinates vectors remain the same as in (2.13), (2.14) and (2.15), and by defining $M = \left[\begin{array}{c} M_0 \\ M_x \\ M_y \\ M_z \end{array}\right]$ and $\underline{b} = \left[\begin{array}{c} \underline{0} \\ \underline{v} \\ \underline{w} \\ \underline{q} \end{array}\right]$, we can again formulate the fitting problem as in (2.17) and get the least squares solution of the same form as in (2.18).

Similarly, the Min-Var fitting algorithm has three fitting requirements: the first two are the same as in Min-Max, and the third is to minimize the variance of the error (instead of the maximum error as in Min-Max). Using the same formulation as in Min-Max, we get the following least squares problem:

$$M_0 = \begin{bmatrix} \underline{p}^n(x_1,y_1,z_1) & \underline{p}^n(x_2,y_2,z_2) & \dots & \underline{p}^n(x_N,y_N,z_N) \end{bmatrix}^T \tag{2.29}$$

$$M_x = \begin{bmatrix} \frac{\underline{p}_x^n(x_1,y_1,z_1)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_1,y_1,z_1)\right|^2}} & \frac{\underline{p}_x^n(x_2,y_2,z_2)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_2,y_2,z_2)\right|^2}} & \dots & \frac{\underline{p}_x^n(x_N,y_N,z_N)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_N,y_N,z_N)\right|^2}} \end{bmatrix}^T \tag{2.30}$$

$$M_y = \begin{bmatrix} \frac{\underline{p}_y^n(x_1,y_1,z_1)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_1,y_1,z_1)\right|^2}} & \frac{\underline{p}_y^n(x_2,y_2,z_2)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_2,y_2,z_2)\right|^2}} & \dots & \frac{\underline{p}_y^n(x_N,y_N,z_N)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_N,y_N,z_N)\right|^2}} \end{bmatrix}^T \tag{2.31}$$

$$M_z = \begin{bmatrix} \frac{\underline{p}_z^n(x_1,y_1,z_1)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_1,y_1,z_1)\right|^2}} & \frac{\underline{p}_z^n(x_2,y_2,z_2)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_2,y_2,z_2)\right|^2}} & \dots & \frac{\underline{p}_z^n(x_N,y_N,z_N)}{\sqrt{\sum_{k=1}^{r}\left|\underline{p}_k^n(x_N,y_N,z_N)\right|^2}} \end{bmatrix}^T \tag{2.32}$$

The gradient coordinates vectors remain the same as in (2.13), (2.14) and (2.15), and by defining $M = \begin{bmatrix} M_0 \\ M_x \\ M_y \\ M_z \end{bmatrix}$ and $\underline{b} = \begin{bmatrix} \underline{0} \\ \underline{v} \\ \underline{w} \\ \underline{q} \end{bmatrix}$, we can again formulate the fitting problem as in (2.17) and get the least squares solution of the same form as in (2.18).

A comparison between Gradient1, Min-Max and Min-Var fitting performance is shown in Fig. 2.5 and Fig. 2.6.

Based on these observations we can see that compared with Gradient1,

Min-Max and Min-Var spurious zero-sets appear farther away from the desired zero-set.

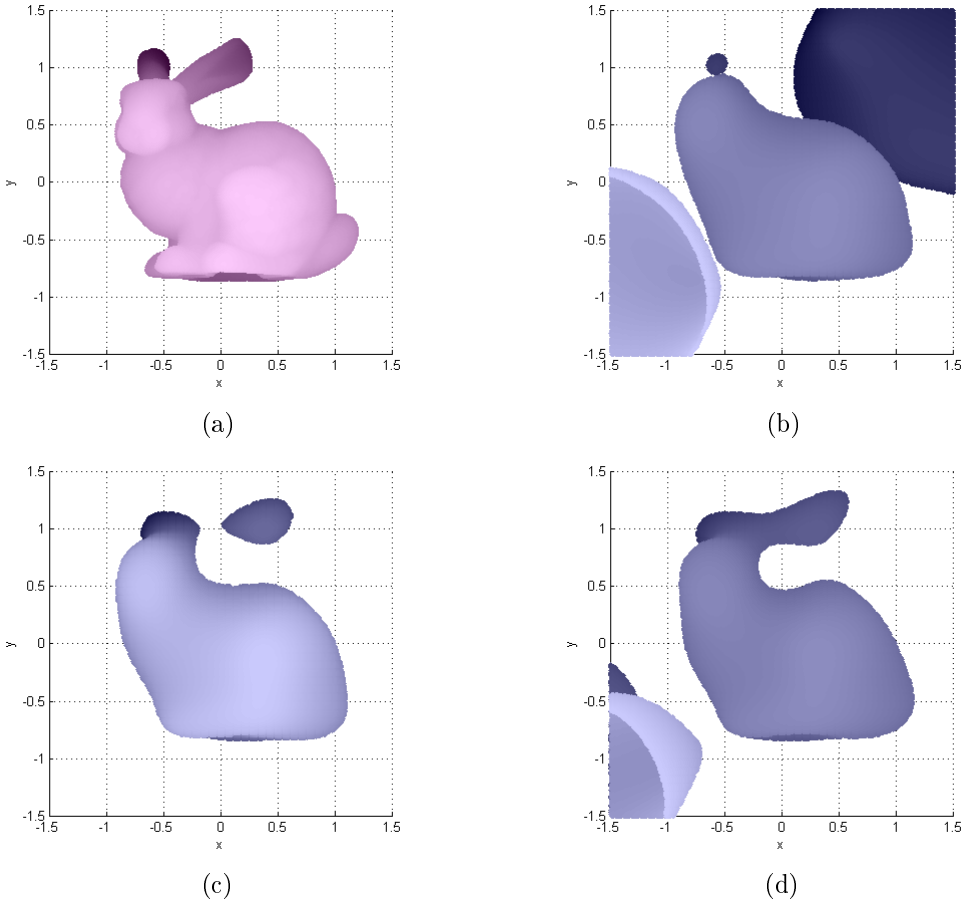


Figure 2.5: Fit of a 6^{th} degree polynomial to "The Stanford Bunny": (a) original data points (b) Gradient1 fitting (c) Min-Var fitting (d) Min-Max fitting

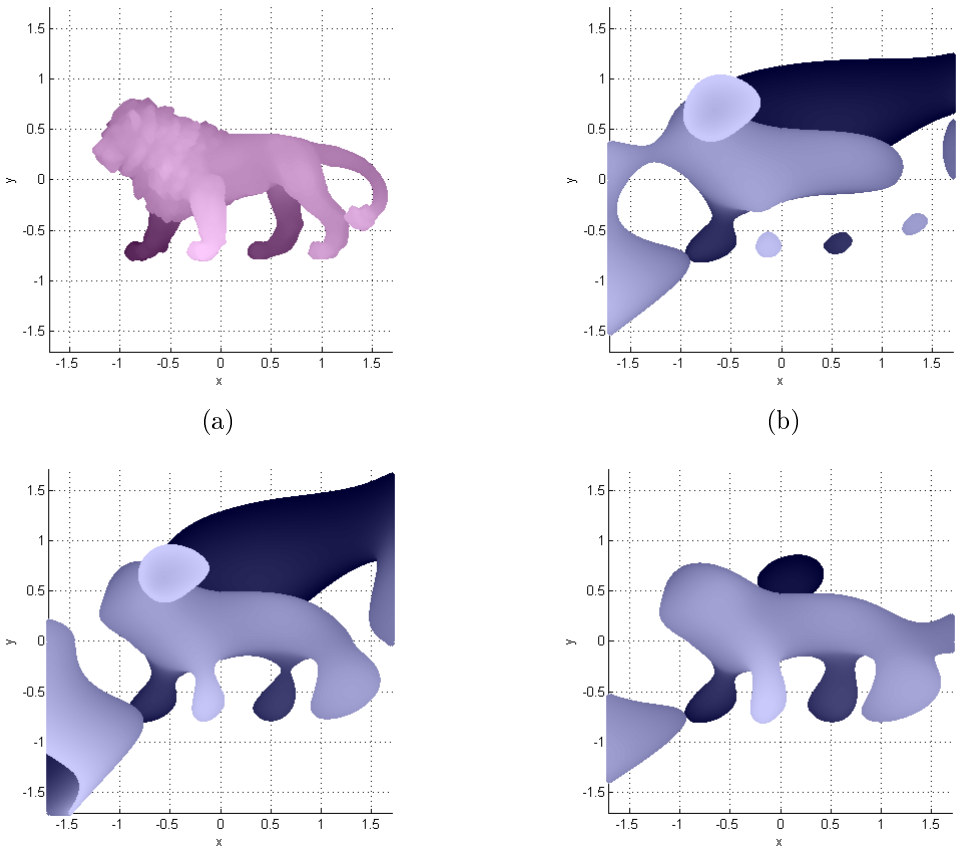


Figure 2.6: Fit of a 6^{th} degree polynomial to "lion" (taken from the "Suggestive Contour Gallery" [10]) (a) original data points (b) Gradient1 fitting (c) Min-Var fitting (d) Min-Max fitting.

2.4 Rotation Invariant Fitting Algorithms

For the application of object recognition that we will present later, we would like that the relation between polynomials fitted to the original and the rotated data-sets will be rotation only. This section is based on 2D rotation invariant IP fitting algorithms [3] and extended here for 3D rotation invariant IP fitting algorithms.

In all the least squares algorithms (Gradient1, Min-Max and Min-Var) we had the following requirements:

1. The polynomial value should be zero at each data point location.
2. The gradient direction should coincide with the normal to the data-set at each data point location.
3. The gradient size at each data point should be equal to a certain value (algorithm dependent).

We will examine each of these requirements and adjust them so that for each data point they will not be affected by rotation.

We denote by $\{(x_i, y_i, z_i)\}_{i=1,\ldots,N}$ the coordinates of the data points, and by $\{(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)\}_{i=1,\ldots,N}$ the coordinates of the data points after rotation by angles (α, β, γ) ((α, β, γ) are the rotation angles around x, y and z axes respectively).

If we fit an IP to the data-set and then rotate the data-set by angles (α, β, γ) and re-fit an IP, we would like the requirements on the IP at the given data-set points before and after rotation to fulfill the following:

$$P^n_{\underline{\tilde{a}}}(\tilde{x}, \tilde{y}, \tilde{z}) = P^n_{\underline{a}}(x, y, z) \tag{2.33}$$

$$\angle \nabla P^n_{\underline{\tilde{a}}}(\tilde{x}, \tilde{y}, \tilde{z}) = \angle \nabla P^n_{\underline{a}}(x, y, z) + (\alpha, \beta, \gamma) \tag{2.34}$$

$$\left\| \nabla P^n_{\underline{\tilde{a}}}(\tilde{x}, \tilde{y}, \tilde{z}) \right\| = \left\| \nabla P^n_{\underline{a}}(x, y, z) \right\| \tag{2.35}$$

Where $P^n_{\underline{\tilde{a}}}(\tilde{x}, \tilde{y}, \tilde{z})$ denotes the IP fitted after rotation.

The polynomial value at each data point is required to be zero, both before and after the rotation, this means that (2.33) is fulfilled.

The gradient direction should coincide with the normal to the data-set. If the data-set is rotated by (α, β, γ), the normals to it would be rotated by (α, β, γ) as well. This means that (2.34) is fulfilled.

Since the algorithms differ in the gradient size requirement, let us now examine each fitting algorithm separately.

2.4.1 Rotation Invariant Gradient1 Fitting Algorithm

The Gradient1 algorithm requires that the gradient size is equal to 1 for each data-set point, both before and after the rotation, and therefore (2.35) is fulfilled.

We conclude that Gradient1 requirement fulfills all the rotation invariance conditions and no adjustment is needed.

2.4.2 Rotation Invariant Min-Max Fitting Algorithm

The Min-Max algorithm requires that the value of $\left\|\nabla P^n_{\underline{a}}(x,y,z)\right\|$ is equal to $\sum_{k=1}^{r}\left|\underline{p}^n_k(x,y,z)\right|$. After rotation the value of $\left\|\nabla P^n_{\underline{\tilde{a}}}(\tilde{x},\tilde{y},\tilde{z})\right\|$ should be equal to $\sum_{k=1}^{r}\left|\underline{p}^n_k(\tilde{x},\tilde{y},\tilde{z})\right|$. The sum of the absolute values of the monomials for the same data point depends on the rotation angles (α, β, γ). Therefore, (2.35) is not fulfilled, and an adjustment is needed.

Following the same solution as in 2D [3], we transform into spherical coordinates: $(x, y, z) \rightarrow (rcos\theta sin\varphi, rsin\theta sin\varphi, rcos\varphi)$, and replace the gradient value requirement with the following requirement:

$$\left\|\nabla P_{\underline{a}}^{n}\left(rcos\theta sin\varphi, rsin\theta sin\varphi, rcos\varphi\right)\right\| =$$
$$= E_{\theta,\varphi}\left\{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}\left(rcos\theta sin\varphi, rsin\theta sin\varphi, rcos\varphi\right)\right|\right\}$$

where $0 \leq \theta < 2\pi$ and $0 \leq \varphi < \pi$, and $E_{\theta,\varphi}\{\cdot\}$ is the expectation with respect to θ and φ.

Now the sum of monomials is averaged over all the possible values of coordinates (θ, φ) and the gradient value requirement is invariant to rotation.

The use of the average value hardly affects the Min-Max properties. The comparison of the fitting performance appears in Fig. 2.7. As it can be seen, the results of Min-Max and the rotation invariant Min-Max (RI-Min-Max) are quite similar.

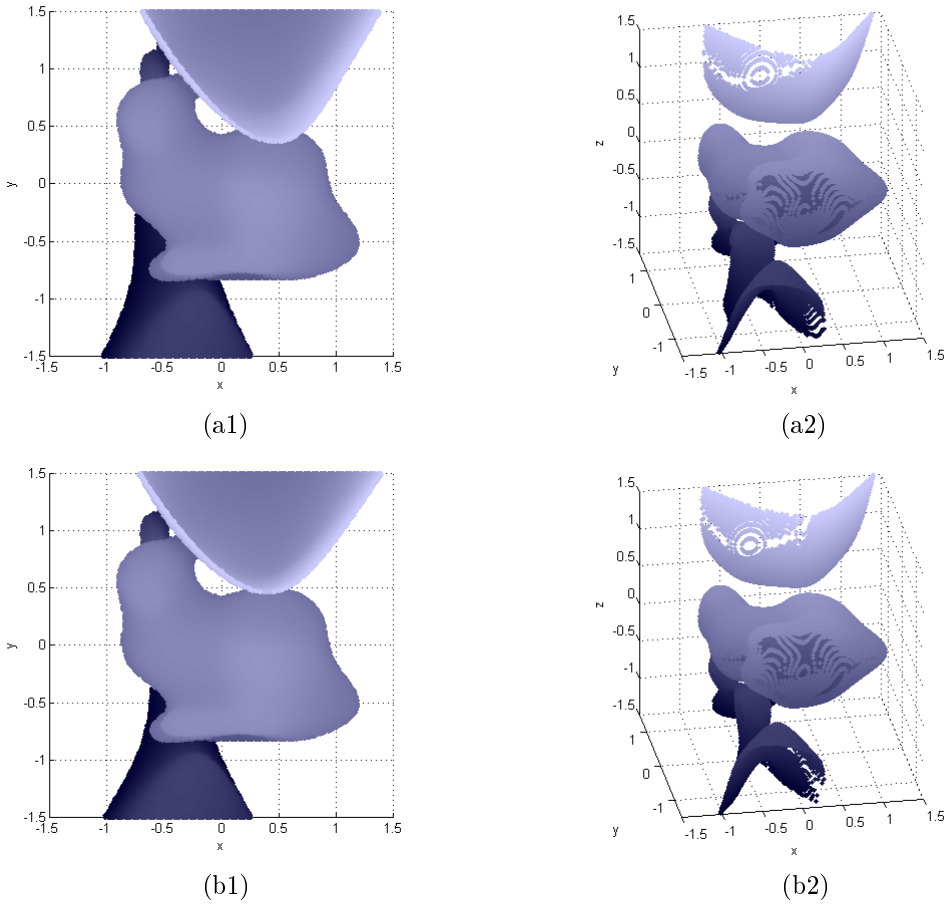


Figure 2.7: Fit of an 8^{th} degree polynomial to "The Stanford Bunny" object: (a1,a2) Min-Max fitting (b1,b2) RI-Min-Max fitting. The left column and the right column describe different points of view

2.4.3 Rotation Invariant Min-Var Fitting Algorithm

The Min-Var algorithm requires that the value of $\left\|\nabla P_{\underline{a}}^{n}\left(x,y,z\right)\right\|$ is equal to $\sqrt{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}\left(x,y,z\right)\right|^{2}}$. After rotation, the value of $\left\|\nabla P_{\underline{\tilde{a}}}^{n}\left(\tilde{x},\tilde{y},\tilde{z}\right)\right\|$ should be equal to $\sqrt{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}\left(\tilde{x},\tilde{y},\tilde{z}\right)\right|^{2}}$.

Let us examine the expression $\sqrt{\sum_{k=1}^{r}\left|\underline{p}_{k}^{n}\left(x,y,z\right)\right|^{2}}$. For a 2^{nd} degree IP,

this expression is equal to:

$$\sqrt{1+x^2+y^2+z^2+x^4+x^2y^2+y^4+x^2z^2+y^2z^2+z^4} =$$

$$= \sqrt{1+d^2+d^4-x^2y^2-x^2z^2-y^2z^2}$$

where d is the Euclidean distance from the origin ($d = \sqrt{x^2+y^2+z^2}$).

This expression is coordinate system dependent and not invariant to rotation. Only the terms of the form d^{2p}, where $p \in \mathbb{N}$, are invariant to rotation. However, if the 2^{nd} degree monomials xy, xz and yz are replaced by $\sqrt{2}xy$, $\sqrt{2}xz$ and $\sqrt{2}yz$ correspondingly, this expression becomes

$\sqrt{1+x^2+y^2+z^2+x^4+2x^2y^2+y^4+2x^2z^2+2y^2z^2+z^4} = \sqrt{1+d^2+d^4}$

and it is now invariant to rotation.

Following the same solution as in 2D [3], we use the binomial expansion for 3 variables in order to obtain only terms of the form d^{2p}:

$$\begin{aligned}\left(x^2+y^2+z^2\right)^n &= \sum_{0\le k,l,m,k+l+m\le n} \frac{n!}{k!l!m!}\left(x^2\right)^k\left(y^2\right)^l\left(z^2\right)^m = \\ &= \sum_{0\le k,l,m,k+l+m\le n}\left[\sqrt{\frac{n!}{k!l!m!}}x^k y^l z^m\right]^2\end{aligned}$$

Thus, in order to obtain only rotation invariant terms of the form d^{2p}, we need to replace each monomial $x^k y^l z^m$ with $\sqrt{\frac{n!}{k!l!m!}}x^k y^l z^m$.

In other words, the adjustment we do is the following:

$$
\begin{aligned}
P_{\underline{a}}^{n}(x,y,z) &= \sum_{0\leq k,l,m,k+l+m\leq n} a_{klm}x^{k}y^{l}z^{m} = \\
&= \sum_{0\leq k,l,m,k+l+m\leq n} \underbrace{\frac{a_{klm}}{\sqrt{\frac{n!}{k!l!m!}}}}_{a'_{klm}} \sqrt{\frac{n!}{k!l!m!}} x^{k}y^{l}z^{m} = \\
&= \sum_{0\leq k,l,m,k+l+m\leq n} a'_{klm}\sqrt{\frac{n!}{k!l!m!}} x^{k}y^{l}z^{m}
\end{aligned}
$$

Actually we create a new factorized monomials vector, and solve the least squares problem for $\underline{a}'$. After we have $\underline{a}'$, we can transform back to our original coefficients vector $\underline{a}$. This adjustment makes sure that (2.35) is fulfilled.

The use of the factorized monomials vector hardly affects the Min-Var properties. A comparison of the fitting performance appears in Fig. 2.8. As it can be seen, the results of Min-Var and the rotation invariant Min-Var (RI-Min-Var) are quite similar.

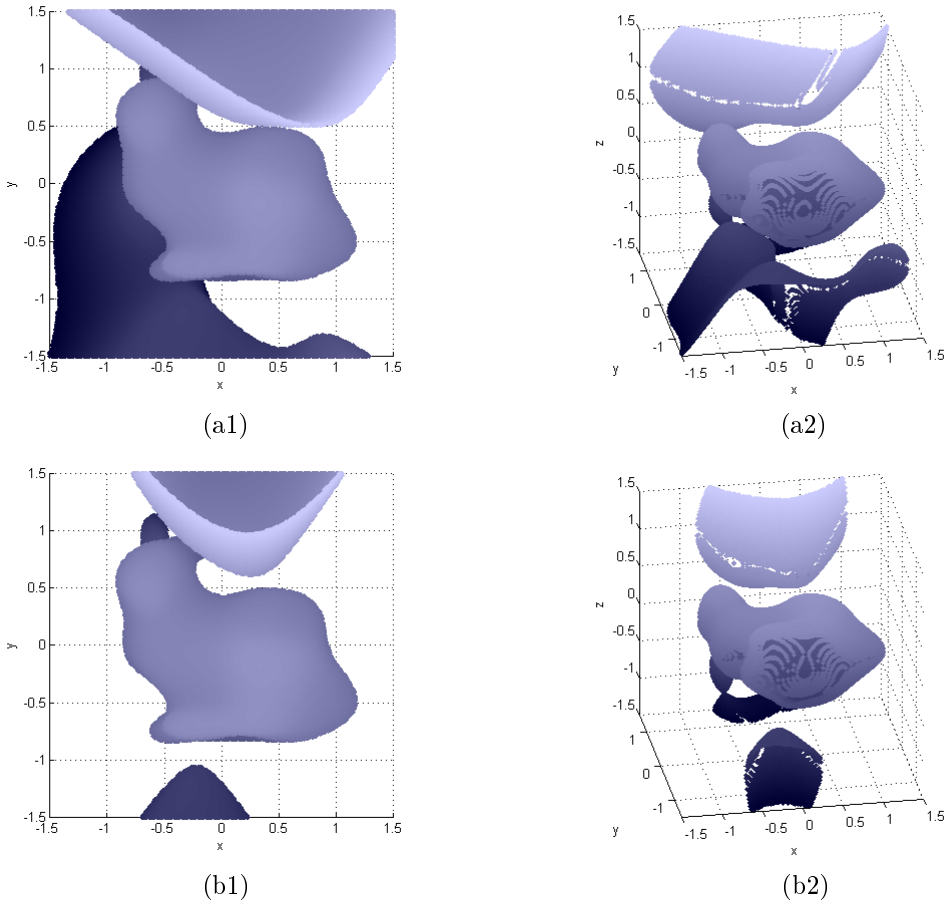


Figure 2.8: Fit of an 8^{th} degree polynomial to "The Stanford Bunny" object: (a1,a2) Min-Var fitting (b1,b2) RI-Min-Var fitting. The left column and the right column describe different points of view

Chapter 3

3D Pose Estimation Algorithms - An Overview

3.1 Problem Definition

3D Pose estimation involves determining the coordinate transformation between two occurrences of the same object. In 3D, the coordinate transformation is usually a rigid motion composed of a rotation and a translation. Pose estimation is often needed in computer vision applications, since determining an object's position and orientation relative to some coordinate system, is essential for classification and identification. In this chapter we will review 3D pose estimation algorithms which assume that the given 3D objects are represented as clouds of data points. We chose to review relatively low complexity algorithms, which have interesting mathematical background that can be utilized when looking for rotation invariant representation.

An example for 2 instances of the same object with relation of translation

and rotation is shown in Fig. 3.1.

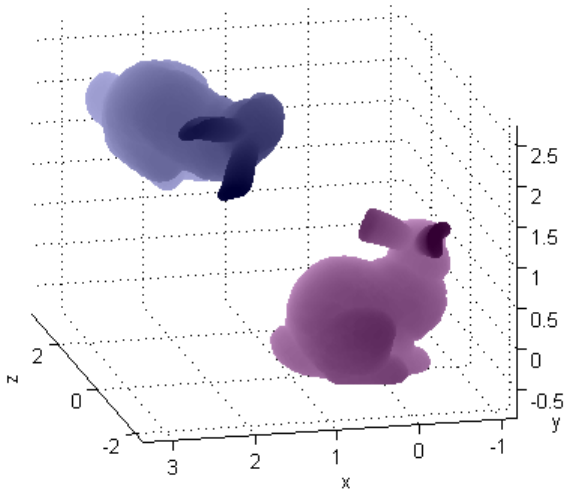


Figure 3.1: 2 instances of "The Stanford Bunny" with relation of translation and rotation

3.2 Translation

The center of mass of an object is invariant to rotation and scaling. Therefore, estimating the translation is relatively easy; we locate the center of mass of the object at the origin:

$$x_{i,translated} = x_i - \frac{1}{N}\sum_{k=1}^{N} x_k \tag{3.1}$$

$$y_{i,translated} = y_i - \frac{1}{N}\sum_{k=1}^{N} y_k \tag{3.2}$$

$$z_{i,translated} = z_i - \frac{1}{N}\sum_{k=1}^{N} z_k \tag{3.3}$$

An example of 2 instances of the same object with relation of rotation only (after locating the center of mass of each object at the origin) is shown in Fig. 3.2.

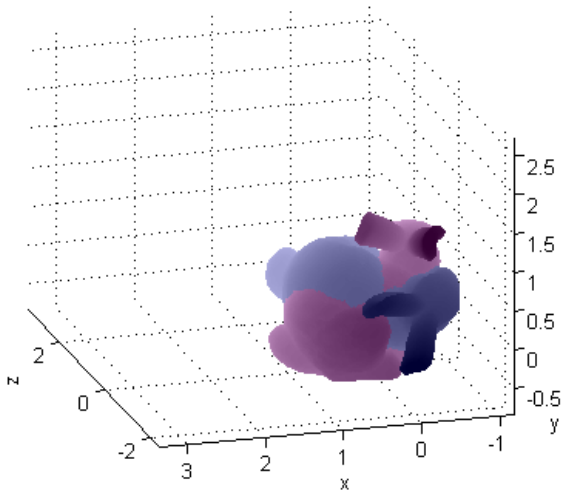


Figure 3.2: 2 instances of "The Stanford Bunny" after translation, with relation of rotation only

3.3 Rotation Estimation Approaches

The output of this stage is a rotation matrix, which should be applied to the data points.

The different approaches can be divided into two distinct groups:

1. Intrinsic Orientation Methods:

   (a) Only one occurrence of the object is required as input

   (b) The object is registered according to its intrinsic characteristics

2. Methods that use point matching:

   (a) Both occurrences of the object are required as input

   (b) The two occurrences are registered with respect to each other's data points

3.4 Intrinsic Orientation Methods

3.4.1 Principal Component Analysis (PCA)

This method is often used for lowering the dimension of the data [11], but it can also be used for pose estimation.

The input is a 3D object, which is represented by a cloud of N data points, $\{(x_i, y_i, z_i)\}|_{i=1,\ldots,N}$, and its center of mass is located at the origin.

3.4.1.1 Scatter Matrix

PCA is based on the 3D scatter matrix of the object, which is defined as follows:

$$\mathcal{S}\left(\underline{x},\underline{y},\underline{z}\right) = \frac{1}{N}\begin{bmatrix} \underline{x}^T \\ \underline{y}^T \\ \underline{z}^T \end{bmatrix}\begin{bmatrix} \underline{x} & \underline{y} & \underline{z} \end{bmatrix} = \frac{1}{N}\begin{bmatrix} \sum_i x_i^2 & \sum_i x_i y_i & \sum_i x_i z_i \\ \sum_i x_i y_i & \sum_i y_i^2 & \sum_i y_i z_i \\ \sum_i x_i z_i & \sum_i y_i z_i & \sum_i z_i^2 \end{bmatrix} \tag{3.4}$$

3.4.1.2 PCA Pose Estimation Algorithm

The PCA pose estimation algorithm is as follows:

1. Compute the scatter matrix of the data points, $\mathcal{S}\left(\underline{x}, \underline{y}, \underline{z}\right)$.

2. Find the eigenvectors and eigenvalues of the scatter matrix:

$$\mathcal{S}v = \lambda v \tag{3.5}$$

3. Sort the eigenvectors and eigenvalues according to decreasing value of the eigenvalues. We denote the eigenvectors after sorting by v_1, v_2, v_3.

4. Use the eigenvectors matrix as an orthogonal linear transformation that rotates the data to a new coordinate system:

$$R = \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix} \tag{3.6}$$

   This coordinate system is such that the largest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

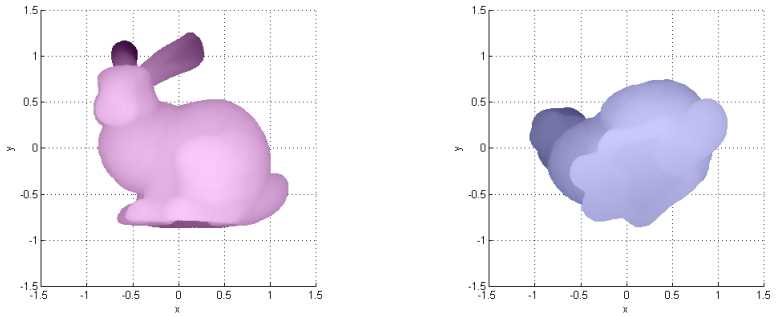
Thus, the output of this method is a rotation matrix consisting of the eigenvectors of the scatter matrix. Applying this rotation matrix to the input object, we get an intrinsic rotation of the object.

The rotation matrix actually defines 8 different solutions, since reflections of the principal axes were not taken into account (each eigenvector v can be taken as v or $-v$, so there are 2^n orthogonal systems for an nxn scatter matrix).

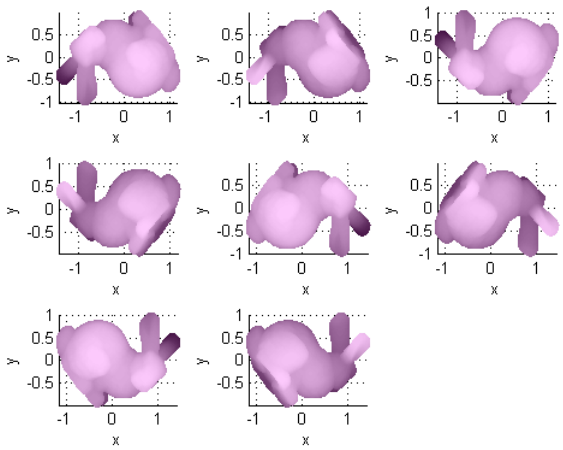
Examples are shown in Fig. 3.3 and Fig. 3.4.

In Fig. 3.3 the 2 instances were identical except for a synthetic rotation. It can be seen that we got identical 8 PCA intrinsic rotations (but in a different order) for the 2 instances in this case.

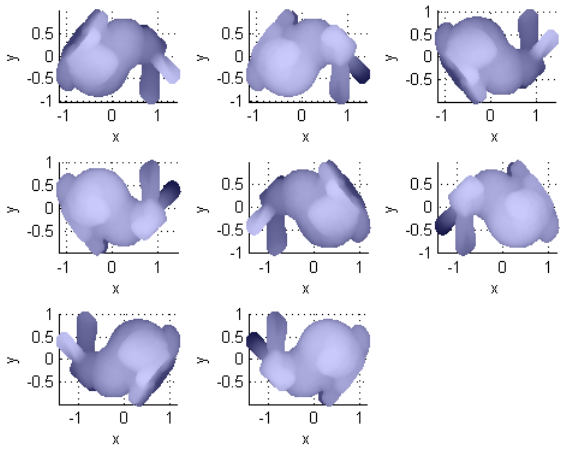
In Fig. 3.4 the 2 instances were 2 different acquisitions of the same face, but with a different expression and some missing/additional patches of points. It can be seen that this slight deformation together with the re-sampling resulted in a slightly different pose estimation for each instance.



(a) (b)

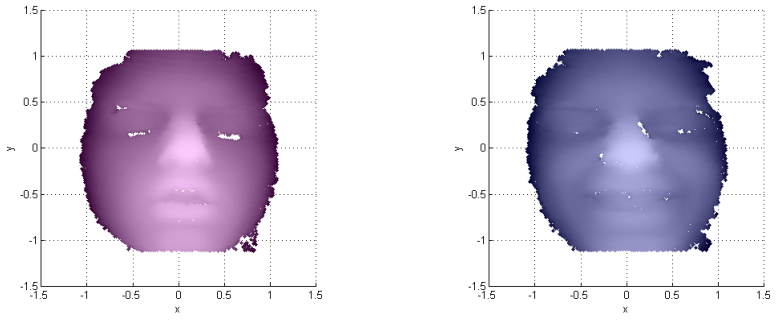


(c)

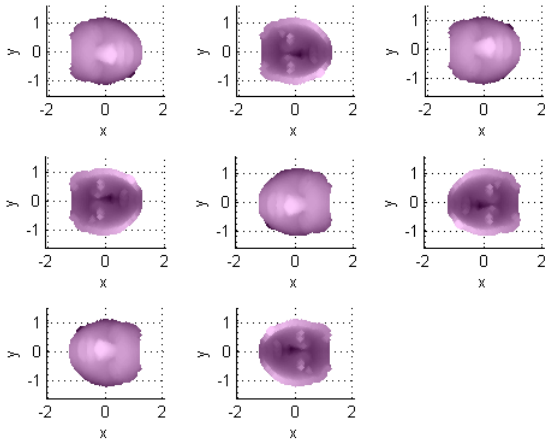


(d)

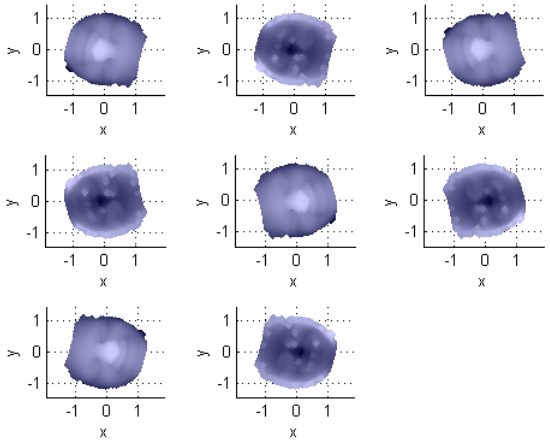
Figure 3.3: An example for the 8 PCA intrinsic rotations of 2 different poses of "The Stanford Bunny": (a) the original data points (b) the data points after rotation (c) the 8 PCA intrinsic rotations of the original data points (d) the 8 PCA intrinsic rotations of the data points after rotation



(a) (b)



(c)



(d)

Figure 3.4: An example for the 8 PCA intrinsic rotations of 2 different scans of the same face: (a) neutral expression (b) smiling expression (c) the 8 PCA intrinsic rotations of the neutral expression (d) the 8 PCA intrinsic rotations of the smiling expression

3.4.2 Implicit Polynomials Pose Estimation (Tensorial Approach)

The method in this section is described in [6]. Here we use a somewhat different coordinate notation ((x_1, x_2, x_3) instead of (x, y, z) for convenience).

The input is a 3D object, which is represented by a cloud of N data points, $\{(x_{1,i}, x_{2,i}, x_{3,i})\}|_{i=1,...,N}$, and its center of mass is located at the origin.

A 3D implicit polynomial is fitted to the data:

$$f_n(x_1, x_2, x_3) = \sum_{0 \le k,l,m,k+l+m \le n} a_{klm} x_1^k x_2^l x_3^m \tag{3.7}$$

Then, the implicit polynomial can be separated into homogeneous parts in the following way:

$$\begin{aligned} f_n(x_1, x_2, x_3) = & \underbrace{a_{000}}_{H_0} + \underbrace{a_{100}x_1 + a_{010}x_2 + a_{001}x_3}_{H_1(x_1,x_2,x_3)} + \\ & + \underbrace{a_{200}x_1^2 + +a_{110}x_1x_2 + a_{020}x_2^2 + a_{101}x_1x_3 + a_{011}x_2x_3 + a_{002}x_3^2}_{H_2(x_1,x_2,x_3)} + \\ & + ... + \underbrace{a_{n00}x_1^n + ... + a_{0n0}x_2^n + ... + a_{00n}x_3^n}_{H_n(x_1,x_2,x_3)} \end{aligned} \tag{3.8}$$

Or in short:

$$f_n(x_1, x_2, x_3) = \sum_{r=0}^{n} H_r(x_1, x_2, x_3) \tag{3.9}$$

Where $H_r(x_1, x_2, x_3)$ denotes a homogeneous ternary polynomial of degree

r, also called a *'form'* of degree r:

$$H_r(x_1, x_2, x_3) = \sum_{k+l+m=r} a_{klm} x_1^k x_2^l x_3^m \tag{3.10}$$

$H_n(x_1, x_2, x_3)$ is called *'the leading form'* of an implicit polynomial of degree n. Translations leave the leading form unaffected, but affect the rest of the forms. Therefore, the orientation of the implicit polynomial can be calculated directly from $H_n(x_1, x_2, x_3)$.

3.4.2.1 Tensor Representation

For pose estimation purposes, it is more convenient to use tensor representation for $H_n(x_1, x_2, x_3)$:

$$H_n(x_1, x_2, x_3) = \sum_{k+l+m=n} a_{klm} x_1^k x_2^l x_3^m \tag{3.11}$$

$$H_n(x_1, x_2, x_3) = \sum_{i_1=1}^{3} \sum_{i_2=1}^{3} ... \sum_{i_n=1}^{3} s^{i_1 i_2 ... i_n} x_{i_1} x_{i_2} ... x_{i_n} \tag{3.12}$$

Where $(s^{i_1 i_2 ... i_n})_{1 \leq i_1, i_2, ..., i_n \leq 3}$ is a symmetric tensor of order n, denoted S_n, which represents the leading form. It can also be considered as an n-dimensional array with 3^n entries. Each entry $s^{i_1 i_2 ... i_n}$ can be expressed by the leading form coefficients:

$$s^{i_1 i_2 ... i_n} = \frac{a_{klm}}{\frac{n!}{k!l!m!}} \tag{3.13}$$

Where k is the number of tensor indexes $(i_1, i_2, ..., i_n)$ which are equal to 1, l is the number of tensor indexes which are equal to 2, and m is number of tensor indexes which are equal to 3. See Appendix B for a more detailed

explanation about tensors and implicit polynomials representation.

3.4.2.2 Tensor Rotation

Upon an orthogonal transformation (e.g., rotation) on a tensor, denoted by an nxn matrix $R = (r_{ij})$, the tensor S_n in the new basis is:

$$s'^{j_1 j_2 \ldots j_n} = \left|J^{-1}\right| \sum_{i_1=1}^{3} \sum_{i_2=1}^{3} \ldots \sum_{i_n=1}^{3} s^{i_1 i_2 \ldots i_n} r_{i_1 j_1} r_{i_2 j_2} \ldots r_{i_n j_n} \tag{3.14}$$

Where J is the Jacobian of the transformation, $|J| = det\,(R)$, ($|J| = 1$ for an orthogonal transformation), and $s'^{j_1 j_2 \ldots j_n}$ is the tensor in the new basis.

3.4.2.3 The Rotation Matrix (Tensor Contraction)

Given a tensor, a contraction with respect to 2 indices (e.g., i_1 and i_2), is defined as a new tensor of order $n - 2$:

$$s'^{i_3 i_4 \ldots i_n} = \sum_{i_1=1}^{3} s^{i_1 i_1 i_3 i_4 \ldots i_n} \tag{3.15}$$

In other words, we set $i_2 = i_1$, and sum over i_1. A total contraction of a tensor gives a zero order tensor (a scalar) which is an invariant.

For example, for a symmetric 3x3 matrix (tensor of order 2), the tensor contraction gives the trace of the matrix, which is known to be an invariant under Euclidean transformations (such as rotation).

A single contraction of a tensor of order n gives a tensor of order $n - 2$, whose components are linear functions of the original tensor. For a tensor of even order $n = 2p$ with 3^n entries, $(p - 1)$ contractions will result in a

3x3 symmetric matrix. When this tensor represents the leading form H_n of an implicit polynomial of degree n, we denote this matrix by $V(H_n)$. Note that this matrix components are linear with respect to the coefficients of the leading form, a fact that contributes to the robustness of the orientation.

The diagonalization of the matrix $V(H_n)$ and the orthonormal eigenvectors basis allow us to estimate the orientation of the polynomial by providing an intrinsic orientation for the polynomial. The geometric interpretation of the eigenvectors of $V(H_n)$ is a 3D ellipsoid which defines the orientation.

3.4.2.4 Implicit Polynomials Pose Estimation Algorithm

The implicit polynomials pose estimation algorithm is as follows:

1. Fit a polynomial of an even degree n=$2p$ to the data points, $f_n(x_1, x_2, x_3)$.
2. Extract the leading form, $H_n(x_1, x_2, x_3)$, and use its coefficients to obtain the tensor representation S_n.
3. Apply $(p-1)$ contractions to S_n, resulting in a 3x3 matrix, $V(H_n)$.
4. Find the the eigenvectors of $V(H_n)$:

$$V(H_n)\, v = \lambda v \tag{3.16}$$

5. Sort the eigenvectors and eigenvalues according to decreasing value of the eigenvalues. We denote the eigenvectors after sorting by v_1, v_2, v_3.
6. Use the eigenvectors matrix as an orthogonal linear transformation to rotate the leading form to a new coordinate system using tensor rota-

tion:

$$R = (r_{ij}) = \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix} \tag{3.17}$$

The rotation matrix can also be applied directly to the data points instead of to the leading form. Note that if we already estimated the translation of the object by locating its center of mass at the origin, we can use any even degree form of the IP for pose estimation (except H_0) instead of just the leading form. However, since the translation estimation might be inaccurate, the pose estimation using the leading form will probably be more accurate than when using a different even form of the IP.

The matrix $V(H_n)$ does not provide a unique solution, but rather 8 solutions due to the symmetries of an ellipsoid (each eigenvector v can be taken as v or $-v$, so there are 2^n orthogonal systems for an nxn matrix).

When using a 2^{nd} degree IP we can see the similarity to PCA by examining $V(H_2)$:

$$H_2(x_1, x_2, x_3) = a_{200}x_1^2 + a_{020}x_2^2 + a_{002}x_3^2 + a_{110}x_1x_2 + a_{101}x_1x_3 + a_{011}x_2x_3 \tag{3.18}$$

$$V(H_2) = \begin{bmatrix} a_{200} & \frac{a_{110}}{2} & \frac{a_{101}}{2} \\ \frac{a_{110}}{2} & a_{020} & \frac{a_{011}}{2} \\ \frac{a_{101}}{2} & \frac{a_{011}}{2} & a_{002} \end{bmatrix} \tag{3.19}$$

$V(H_2)$ has slightly different measurements for the second order moments of the data, but it has a structure which is similar to the scatter matrix.

The difference is that we first fit an IP and then do the pose estimation on the IP coefficients. This can be a disadvantage when the IP coefficients are unstable, since if the IP fitting process results in different IP coefficients (for the same IP degree), we will get a different pose estimation result.

When using higher IP degrees (>2), the pose estimation from $V(H_n)$ is using higher order moments of the data, and is therefore different from PCA.

Examples for IP based pose estimation using a 2^{nd} degree IP leading form are shown in Fig. 3.5 and Fig. 3.6.

In Fig. 3.5 the 2 instances were identical except for synthetic rotation. It can be seen that using a 2^{nd} degree IP with Gradient1 fitting we got identical 8 tensor intrinsic rotations for the 2 instances. In this case changing the IP degree or the fitting method does not affect the results since the rotation is synthetic and the surface is closed.

In Fig. 3.6 the 2 instances were 2 different acquisitions of the same face, but with a different expression and some missing/additional patches of points. It can be seen that this slight deformation together with the re-sampling (i.e., the re-acquisition) resulted in a very different IP fitting (the green ellipsoid in the figure). Since we got different ellipsoids, when the tensor pose estimation rotated the ellipsoids to have their main axes on our coordinate system axes - we got a slightly different pose estimation for each instance. In this case we used a 2^{nd} degree IP with Gradient1 fitting.

Fig. 3.7 and Fig. 3.8 show the different expressions results for a 2^{nd} degree IP with RI-Min-Max and RI-Min-Var, respectively. In both figures we see an IP of a hyperboloid instead of an ellipsoid (in green), and the pose estimation seems more accurate than with Gradient1.

Fig. 3.9 shows the different expressions results for a 4^{th} degree IP with Gradient1 fitting. The IP zero-sets are different and since the tesnor method estimates the pose from the IP leading form (which describes also the spurious zero-sets), different IP fitting result in very different intrinsic orientations (i.e., different pose estimation). The results when using 4^{th} degree IP with RI-Min-Max or RI-Min-Var are very similar.

It can be seen that when we have more than just a synthetic rotation, we get different pose estimation results when using Gradient1 (in a similar manner to the differences we got using PCA, see figure 3.4). However, when using RI-Min-Max or RI-Min-Var, the pose estimation results are more consistent (i.e., we get very similar 8 intrinsic rotations in both cases).

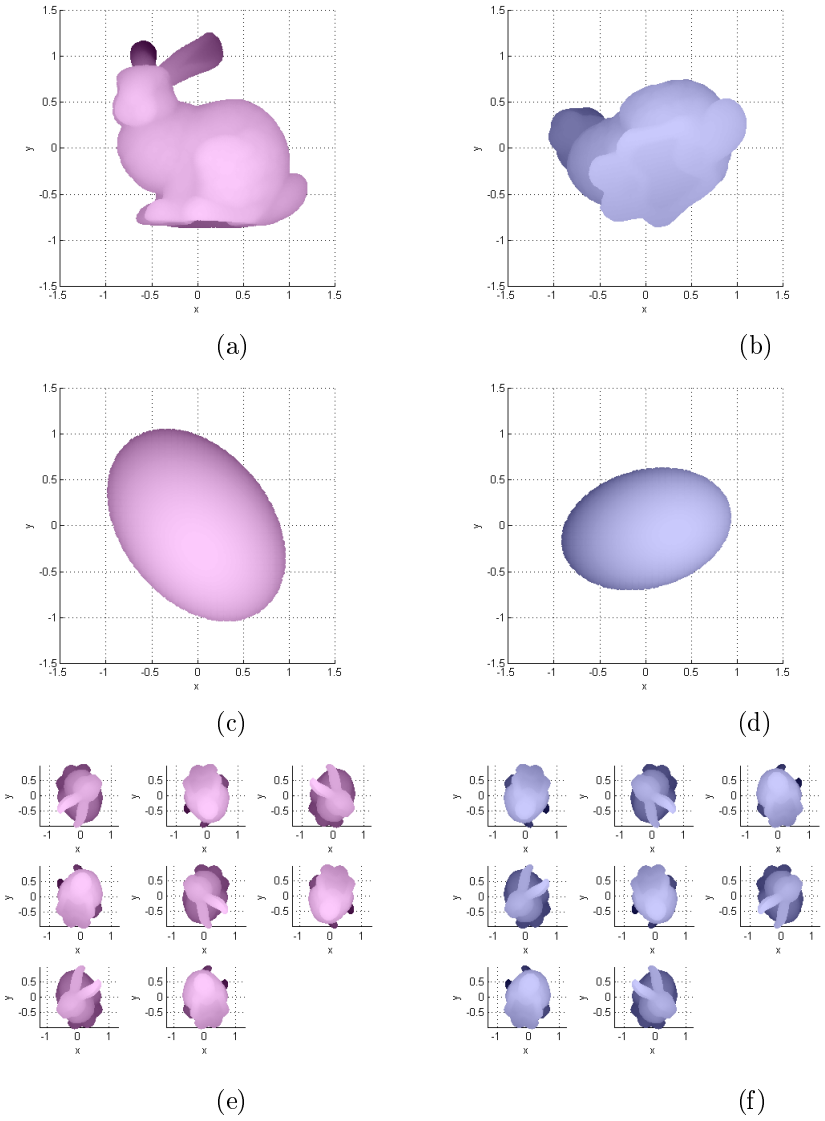


Figure 3.5: An example for the 8 intrinsic rotations of a 2^{nd} degree tensor (Gradient1 IP fitting) of 2 different poses of "The Stanford Bunny": (a) the original data points (b) the data points after rotation (c) a 2^{nd} degree IP fit to the original data points (d) a 2^{nd} degree IP fit to the rotated data points (e) the 8 tensor intrinsic rotations of the original data points (f) the 8 tensor intrinsic rotations of the data points after rotation

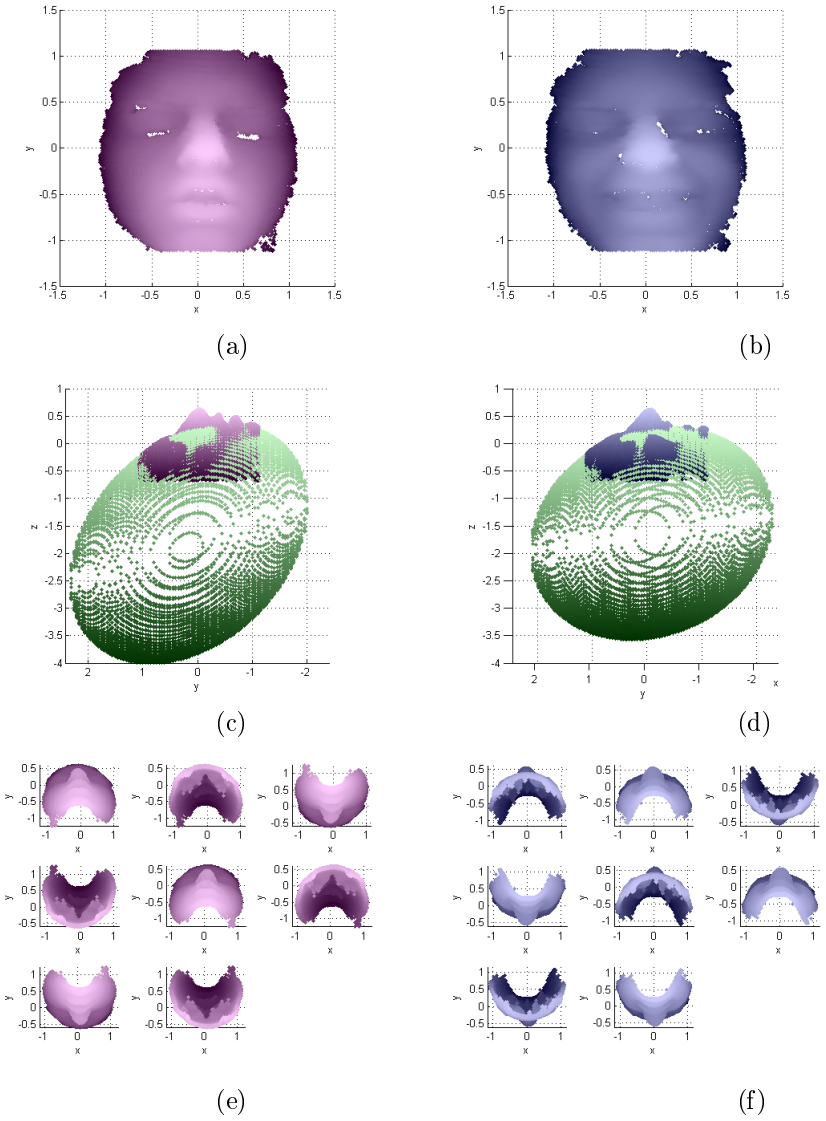


Figure 3.6: An example for the 8 intrinsic rotations of a 2^{nd} degree tensor (Gradient1 IP fitting) of 2 different scans of the same face: (a) neutral expression (b) smiling expression (c) a 2^{nd} degree IP fit to the neutral face (side view, in green) (d) a 2^{nd} degree IP fit to the smiling face (side view, in green) (e) the 8 tensor intrinsic rotations of the neutral expression (f) the 8 tensor intrinsic rotations of the smiling expression

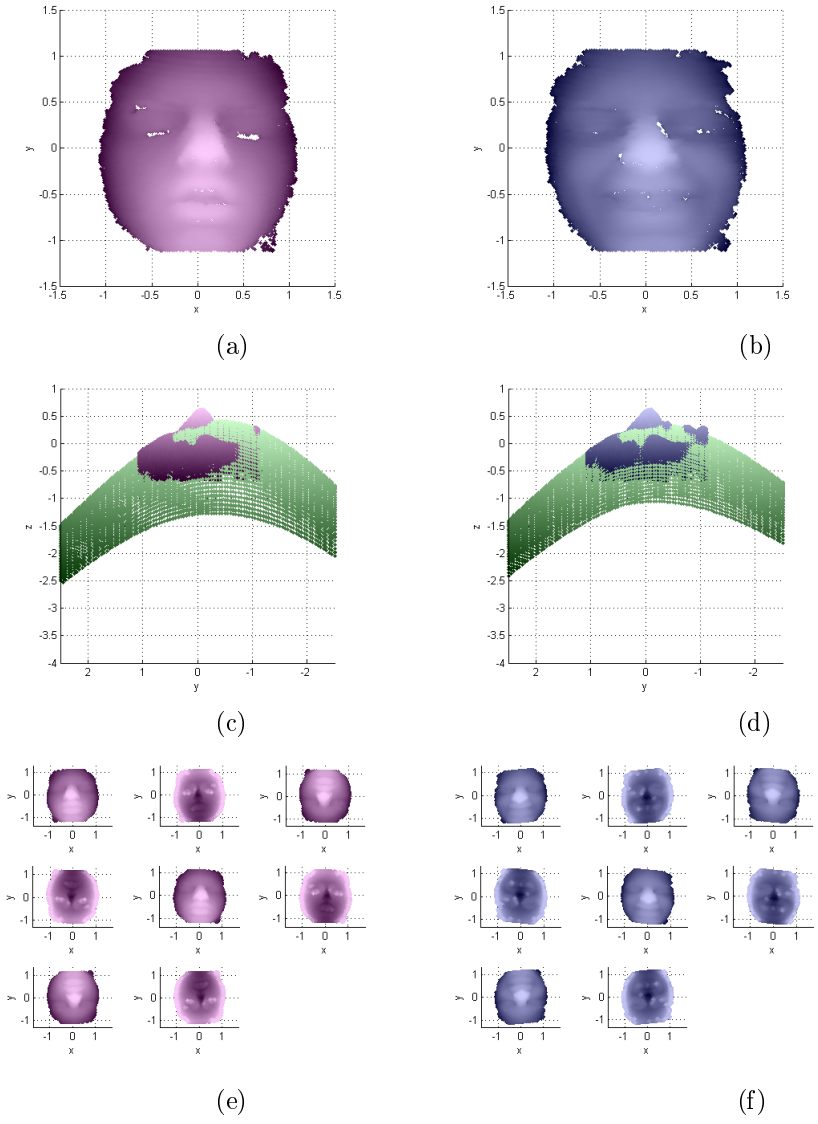


Figure 3.7: An example for the 8 intrinsic rotations of a 2^{nd} degree tensor (RI-Min-Max IP fitting) of 2 different scans of the same face: (a) neutral expression (b) smiling expression (c) a 2^{nd} degree IP fit to the neutral face (side view, in green) (d) a 2^{nd} degree IP fit to the smiling face (side view, in green) (e) the 8 tensor intrinsic rotations of the neutral expression (f) the 8 tensor intrinsic rotations of the smiling expression

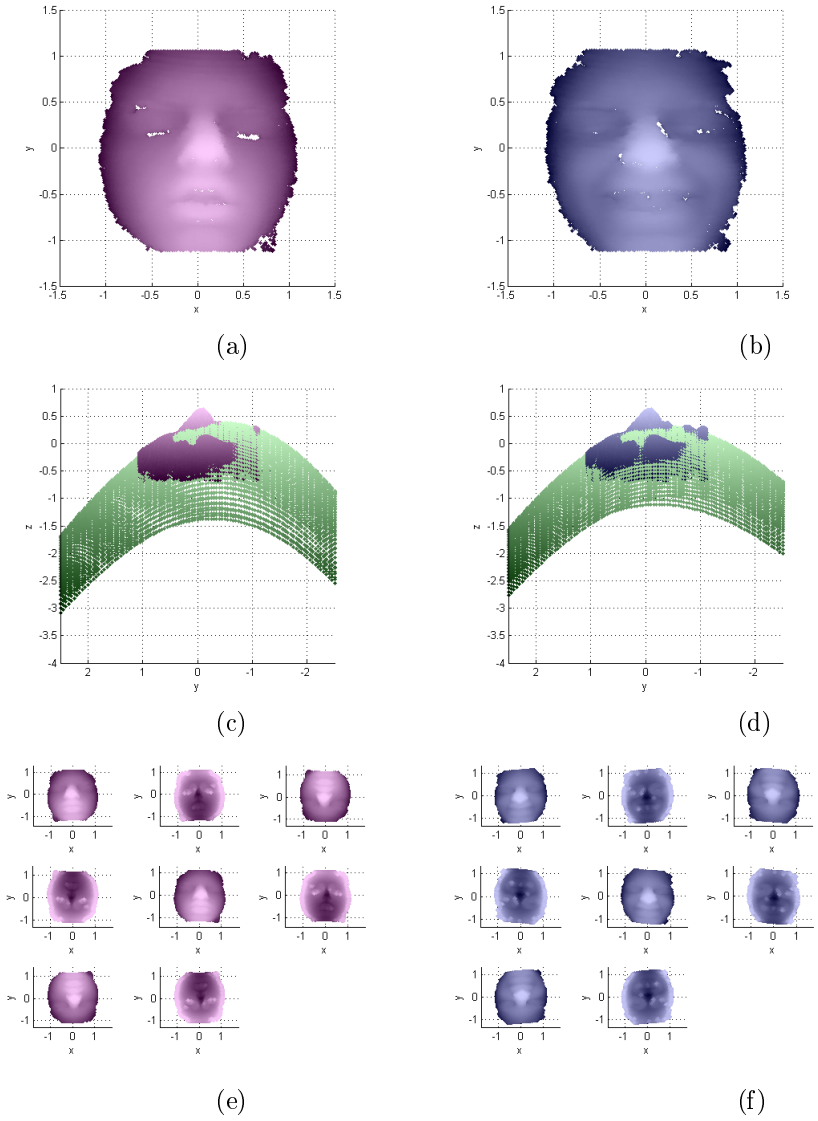


Figure 3.8: An example for the 8 intrinsic rotations of a 2^{nd} degree tensor (RI-Min-Var IP fitting) of 2 different scans of the same face: (a) neutral expression (b) smiling expression (c) a 2^{nd} degree IP fit to the neutral face (side view, in green) (d) a 2^{nd} degree IP fit to the smiling face (side view, in green) (e) the 8 tensor intrinsic rotations of the neutral expression (f) the 8 tensor intrinsic rotations of the smiling expression

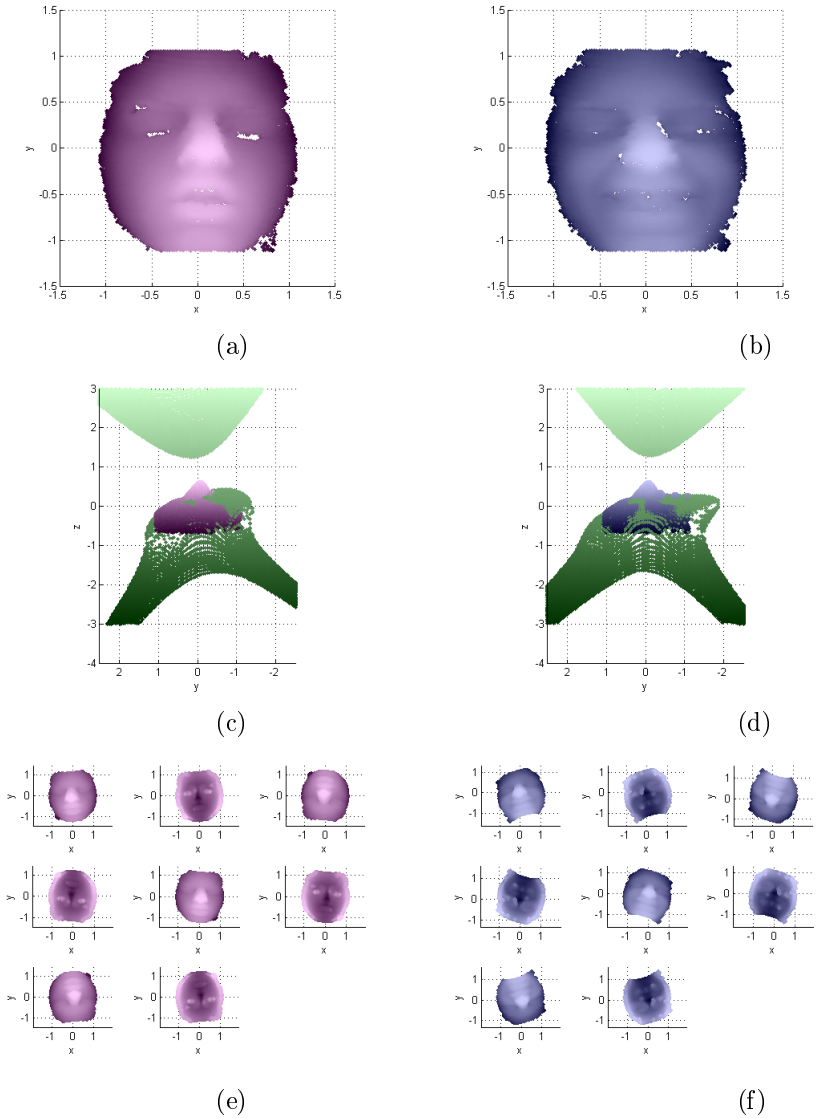


Figure 3.9: An example for the 8 intrinsic rotations of a 4^{th} degree tensor (Gradient1 IP fitting) of 2 different scans of the same face: (a) neutral expression (b) smiling expression (c) a 4^{th} degree IP fit to the neutral face (side view, in green) (d) a 4^{th} degree IP fit to the smiling face (side view, in green) (e) the 8 tensor intrinsic rotations of the neutral expression (f) the 8 tensor intrinsic rotations of the smiling expression

3.5 Methods that Use Point Matching

3.5.1 Quaternion Rotation

The method in this section is described in [12]. The input is two 3D objects, each is represented by a cloud of N data points, $\{r_{1,i} = (x_{1,i}, y_{1,i}, z_{1,i})\}|_{i=1,...,N}$ and $\{r_{2,i} = (x_{2,i}, y_{2,i}, z_{2,i})\}|_{i=1,...,N}$.

We make three assumptions:

1. The center of mass of each object is located at the origin.
2. The 2 objects have the same number of data points.
3. The data points of the 2 objects are *matched*. In other words, we know for sure that each data point $r_{1,i}$ has a corresponding point on the second object, denoted by $r_{2,i}$. This assumption is a major disadvantage, and we will discuss it later.

The objective is to find a rotation matrix R such that $r_2 = R \cdot r_1$ for each data point.

3.5.1.1 Vector Quaternion Representation

A quaternion is a complex number with one real part, q_0, and three different imaginary parts, q_x,q_y and q_z:

$$q = q_0 + q_x i + q_y j + q_z k \tag{3.20}$$

Basic properties of quaternions arithmetic are:

$$\begin{array}{ccc} i^2 = -1 & j^2 = -1 & k^2 = -1 \\ ij = k & jk = i & ki = j \\ ji = -k & kj = -i & ik = -j \end{array} \tag{3.21}$$

It can easily be seen from (3.21) that quaternion multiplication is non-commutative (i.e., $rq \neq qr$).

Let $r = (x, y, z)^T \in \mathbb{R}^3$. Then, it can be represented by a quaternion with a zero real part:

$$r = xi + yj + zk \tag{3.22}$$

This representation is useful when dealing with rotation in $\Re^3$.

3.5.1.2 Quaternions Multiplication

Using the quaternion representation in (3.20), we get that:

$$\begin{aligned} rq = &(r_0 q_0 - r_x q_x - r_y q_y - r_z q_z) + \\ &+i\,(r_0 q_x + r_x q_0 + r_y q_z - r_z q_y) + \\ &+j\,(r_0 q_y - r_x q_z + r_y q_0 + r_z q_x) + \\ &+k\,(r_0 q_z + r_x q_y - r_y q_x + r_z q_0) \end{aligned} \tag{3.23}$$

Since quaternions multiplication is non-commutative, it is often convenient to represent it using matrix multiplication. It can be easily shown that for

rq we get:

$$rq = \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & -r_z & r_y \\ r_y & r_z & r_0 & -r_x \\ r_z & -r_y & r_x & r_0 \end{bmatrix} q = R_r q \tag{3.24}$$

And for qr we get:

$$qr = \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & r_z & -r_y \\ r_y & -r_z & r_0 & r_x \\ r_z & r_y & -r_x & r_0 \end{bmatrix} q = \bar{R}_r q \tag{3.25}$$

Note that $\bar{R}_r$ differs from R_r in that the lower-right-hand 3x3 submatrix is transposed.

3.5.1.3 Vector Rotation Using Unit Quaternions

The rotation of a quaternion r, using a unit quaternion q is described by the product:

$$r' = qr\bar{q} \tag{3.26}$$

Where $\bar{q} = q_0 - q_x i - q_y j - q_z k$ is the conjugate of q. It can be proven (see Appendix C) that if r is purely imaginary, then r' will be purely imaginary as well.

For a rotation around the axis $(\omega_x, \omega_y, \omega_z)$ by an angle of ϕ, the appropriate rotation unit quaternion q is:

$$q = cos\frac{\phi}{2} + sin\frac{\phi}{2}\underbrace{(\omega_x i + \omega_y j + \omega_z k)}_{unit\,vector\,\omega} \tag{3.27}$$

(See Appendix A7 of [12] for a detailed proof).

Using the representation of quaternions multiplication and the quaternions rotation, we can now present the quaternions pose estimation method.

3.5.1.4 Quaternions Rotation Pose Estimation Algorithm

Recall that we are looking for a rotation R such that $r_2 = R \cdot r_1$. Since we know that there exists point matching between the two objects, we can represent the problem as maximization of the following dot product:

$$\underset{R}{max} \sum_{i=1}^{N} (Rr_{1,i}) \cdot r_{2,i} \tag{3.28}$$

Using quaternion representation, we can write:

$$\underset{q}{max} \sum_{i=1}^{N} (qr_{1,i}\bar{q}) \cdot r_{2,i} \tag{3.29}$$

Using the following quaternion dot product property for unit quaternion q and some quaternions p and r [12] (see Appendix C):

$$(pq) \cdot r = p \cdot (r\bar{q}) \tag{3.30}$$

we get:

$$\sum_{i=1}^{N} (qr_{1,i}\bar{q}) \cdot r_{2,i} = \sum_{i=1}^{N} (qr_{1,i}) \cdot (r_{2,i}q) \tag{3.31}$$

If we now replace the quaternions multiplications with matrix multiplications, we get:

$$\sum_{i=1}^{N}(qr_{1,i})\cdot(r_{2,i}q)=\sum_{i=1}^{N}\left(\bar{R}_{r_{1,i}}q\right)\cdot\left(R_{r_{2,i}}q\right)=\sum_{i=1}^{N}q^T\bar{R}_{r_{1,i}}^TR_{r_{2,i}}q=q^T\left(\sum_{i=1}^{N}\bar{R}_{r_{1,i}}^TR_{r_{2,i}}\right)q \tag{3.32}$$

Denoting $L=\sum_{i=1}^{N}\bar{R}_{r_{1,i}}^TR_{r_{2,i}}$, we get the following optimization problem:

$$\max_q q^TLq \tag{3.33}$$

The solution to this problem is known from linear algebra: The unit quaternion q that maximizes q^TLq is the eigenvector of L that corresponds to the most positive eigenvalue of L.

Thus we get the following pose estimation method:

1. Compute $L=\sum_{i=1}^{N}\bar{R}_{r_{1,i}}^TR_{r_{2,i}}$, which is a 4x4 matrix.
2. Find the eigenvector of L that corresponds to the most positive eigenvalue of L.
3. Use this eigenvector as a unit quaternion q to rotate $r_{1,i}$:

$$r'=qr\bar{q} \tag{3.34}$$

The main disadvantage of this method is that it is assumed that the 2 objects have the same number of data points, and that there is point matching. Therefore, we need a pre-processing stage, which can be either:

1. Matching the points of 2 objects with a different number of points. For

example, we can take the object with the smaller number of points as is, and match the points of the object with the larger number of points (find the appropriate point to each of the first object points) .

2. Recognition of key points in each object and using them as representative points for each object for pose estimation.

3.5.2 Iterative Closest Point (ICP)

The method in this section is described in [13]. It utilizes the results of the quaternions rotation method, but it doesn't assume point matching.

The input is two 3D objects. The first object, Q, is represented by a cloud of N_q data points, $\{\vec{q_i} = (x_{1,i}, y_{1,i}, z_{1,i})\}|_{i=1,...,N_q}$, and the second object, P, is represented by a cloud of N_p data points $\{\vec{p_i} = (x_{2,i}, y_{2,i}, z_{2,i})\}|_{i=1,...,N_p}$. The number of points in the general case is different, i.e., $N_q \neq N_p$.

The pose estimation algorithm is as follows:

1. For each point $\vec{p_i}$ in object P, find the closest point in object Q. We denote this point by $\vec{q}_{i,closest}$.

2. Find the registration (i.e., rotation and translation) using the closest points:

   (a) Find the center of mass of the objects: $\vec{\mu}_p = \frac{1}{N_p}\sum_{i=1}^{N_p}\vec{p_i}$ and $\vec{\mu}_q = \frac{1}{N_p}\sum_{i=1}^{N_p}\vec{q}_{i,closest}$.

   (b) Locate the center of mass of each object at the origin.

(c) Apply the quaternions pose estimation described in 3.5.1.4 to object P (with points $\vec{p}_i$, $i = 1, ..., N_p$), with respect to object Q (with points $\vec{q}_{i,closest}$, $i = 1, ..., N_p$).

3. Update the points $\vec{p}_i$ of object P after the registration.

4. Calculate the new sum of errors between closest points: $\sum_{i=1}^{N_p} \vec{p}_i - \vec{q}_{i,closest}$.

5. If the sum of errors is below an empirical threshold - terminate, otherwise: repeat stages 1-4.

In [13] it is proven that this algorithm always converges. In addition, the algorithm is looking for matching points, instead of assuming the objects have the same number of points and that these points are already matched. This algorithm works well when the 2 input objects were already roughly registered (e.g., by PCA), and what is still needed is small angles differences registration. Without an initial registration stage this method may converge to a non-optimal solution. Another disadvantage of this algorithm is that it is iterative, and although not many iterations are needed, each iteration has a high computational cost, since in each iteration, for each point in object P we have to find the closest point in object Q. Therefore, this method has relatively long running times.

An example is shown in Fig. 3.10. After rough registration of 2 instances of the same face using PCA, we still have small angles differences. Applying ICP for registration of the smiling expression, we improved the relative registration between the 2 instances of the same face. As can be seen from this example, the ICP doesn't give perfect results. Over the years, several

variants were suggested to this algorithm, trying to improve its results or to shorten its running time [14].

A recent example for such an improvement is suggested in [14]: when looking for $\vec{q}_{i,closest}$ for each $\vec{p}_i$, the authors suggest to build a look-up matrix with all the distances between all the $\vec{p}_i$ points and the $\vec{q}_j$ points. For each $\vec{q}_j$ that was selected as the closest point to more than one point $\vec{p}_i$, only the closest $\vec{p}_i$ is chosen (closest to $\vec{q}_j$), and the rest of the $\vec{p}_i$ points which were closest to this $\vec{q}_j$ have to choose a different closest point from Q in this iteration. This change is supposed to improve the original ICP results, but the running time is still long.

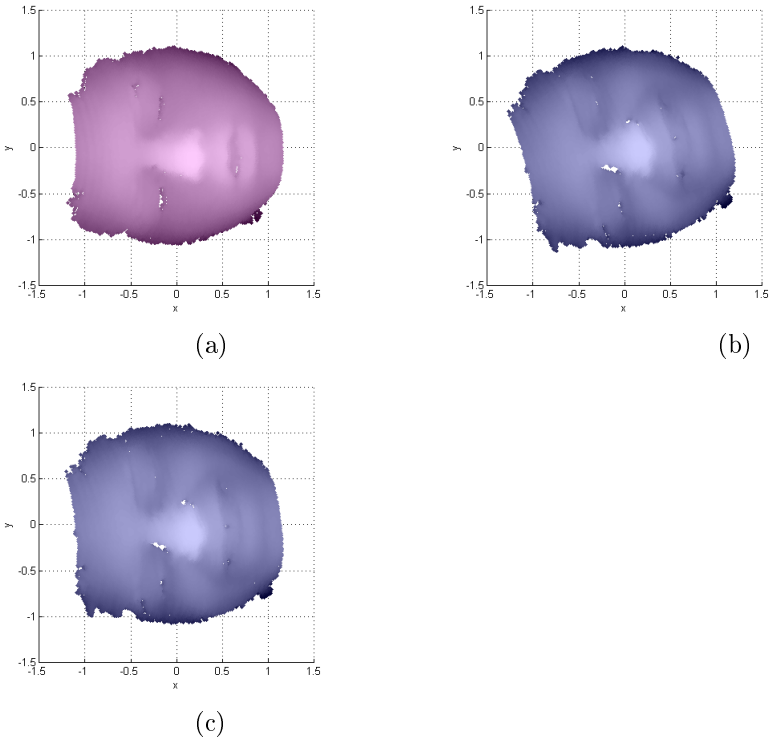


Figure 3.10: An example of ICP pose estimation after PCA using 2 instances of the same face (a) neutral expression after PCA (b) smiling expression after PCA (c) smiling face after PCA and ICP

Chapter 4

Rotation Invariant Representation

In order to avoid pose estimation in the classification process, we need to use expressions that will be invariant to rotation. Such IP based expressions have been developed for the 2D case, both analytically [7], and using symbolic computation [20, 21]. For the 3D case there are only rotation invariants developed using symbolic computation [21]. These invariants contain sum of high degree products of IP coefficients. These products may cause instability due to their high degree, and therefore we prefer to analytically derive invariants similar to those developed in [7].

In this chapter we develop analytically explicit expressions which are rotation invariants based on the IP coefficients. These expressions are linear or quadratic combinations of the IP coefficients.

4.1 Overview of 2D Implicit Polynomials Rotation Invariants

In [7], the authors suggest 2 methods for the derivation of 2D rotation invariants. The first method is based on the rotation matrix and trigonometric identities, and the second method uses a complex representation for 2D implicit polynomial curves.

A 2D implicit polynomial of degree n is represented in the following way:

$$f_n(x,y) = \sum_{0 \le l,m,l+m \le n} a_{lm} x^l y^m \tag{4.1}$$

In the next subsections we will review these 2D invariants derivation methods and will later adjust them for 3D invariants derivation.

4.1.1 Derivation of 2D Invariants Using the Rotation Matrix

In 2D, the new coordinates of an object after rotation by angle θ are:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.2}$$

By inverting the rotation matrix, we get:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{4.3}$$

Substituting (4.3) into (4.1), we can find the relation between the original IP coefficients and the rotated ones. For example, for a 2^{nd} degree IP the original polynomial is:

$$f_2(x,y) = a_{00} + a_{10}x + a_{01}y + a_{20}x^2 + a_{11}xy + a_{02}y^2 \tag{4.4}$$

After substituting the relations between x, y and x', y' we get:

$$\begin{aligned} f_2(x',y') = \; & a_{00} + a_{10}(cx' + sy') + a_{01}(-sx' + cy') + \\ & + a_{20}(cx' + sy')^2 + a_{02}(-sx' + cy')^2 + \\ & + a_{11}(cx' + sy')(-sx' + cy') \end{aligned} \tag{4.5}$$

where $c \triangleq cos\theta$ and $s \triangleq sin\theta$. After expanding and simplifying, we get:

$$\begin{aligned} f_2(x',y') = \; & \underbrace{a_{00}}_{b_{00}} + \underbrace{(ca_{10} - sa_{01})}_{b_{10}}x' + \underbrace{(sa_{10} + ca_{01})}_{b_{01}}y' + \\ & + \underbrace{\left(c^2a_{20} - csa_{11} + s^2a_{02}\right)}_{b_{20}}x'^2 + \underbrace{\left(s^2a_{20} + csa_{11} + c^2a_{02}\right)}_{b_{02}}y'^2 \\ & + \underbrace{\left(2csa_{20} - 2csa_{02} + \left(c^2 - s^2\right)a_{11}\right)}_{b_{11}}x'y' \end{aligned} \tag{4.6}$$

From examination of the new IP coefficients b_{lm} and using the trigonometric identity $c^2 + s^2 = 1$, we get the following linear invariants:

$$L_{2D,0} = b_{00} = a_{00} \tag{4.7}$$

$$L_{2D,2} = b_{20} + b_{02} = a_{20} + a_{02} \tag{4.8}$$

We denote the 2D linear invariants by $L_{2D,k}$, where k is the form degree.

In addition, using some more trigonometric identities, we get the following quadratic invariants:

$$Q_{2D,1} = b_{10}^2 + b_{01}^2 = a_{10}^2 + a_{01}^2 \tag{4.9}$$

$$Q_{2D,2} = (b_{20} - b_{02})^2 + b_{11}^2 = (a_{20} - a_{02})^2 + a_{11}^2 \tag{4.10}$$

We denote the 2D quadratic invariants by $Q_{2D,k}$, where k is the form degree.

Obviously, the complexity of the trigonometric expressions grows considerably with the polynomial degree and therefore this method is useful only for low degree polynomials invariants.

4.1.2 Derivation of 2D Invariants Using a Complex Representation

Given the 2D polynomial:

$$f_n(x,y) = \sum_{0 \le l,m,l+m \le n} a_{lm} x^l y^m \tag{4.11}$$

the main idea in [7] is to rewrite it as a polynomial of complex variables $z = x + iy$ and $\bar{z} = x - iy$:

$$f_n(z) = \sum_{0 \le l,m,l+m \le n} \frac{a_{lm}}{i^m 2^{l+m}} (z + \bar{z})^l (z - \bar{z})^m \tag{4.12}$$

Using binomial expansions for $(z+\bar{z})^l$ and $(z-\bar{z})^m$, we get a new representation for $f_n(z)$ with complex coefficients c_{lm}:

$$f_n(z) = \sum_{0 \le l,m,l+m \le n} c_{lm} \bar{z}^l z^m \tag{4.13}$$

when rotating a polynomial by an angle θ, z transforms as $z' = e^{i\theta} z$, so that $z = e^{-i\theta} z'$. Substituting this relation into the complex polynomial representation, we get:

$$f_n(z') = \sum_{0 \le l,m,l+m \le n} \underbrace{e^{i(l-m)\theta} c_{lm}}_{d_{lm}} \bar{z}'^l z'^m \tag{4.14}$$

Using this method we get a simple relation between the IP coefficients before and after rotation:

$$d_{lm} = c_{lm} e^{i(l-m)\theta} \tag{4.15}$$

which means that when $l = m$ we get invariant rotation IP coefficients. These coefficients always appear before complex monomials of the form: $z^p \bar{z}^p = |z|^{2p}$ where $p \in \mathbb{N}$. Using this method we get one linear invariant from each even degree form $H_{2p}(x,y)$, or $\left\lfloor \frac{n}{2} \right\rfloor + 1$ invariants from an IP of degree n, $f_n(x,y)$. Note that these are the same linear invariants we got using the first method, only this method has the advantage of simplicity.

Also, when $l \neq m$, the magnitude of the complex IP coefficient is still an invariant (i.e., $|d_{lm}| = |c_{lm}|$) since $\left|e^{i(l-m)\theta}\right| = 1$. This leads to a set of quadratic invariants (since $|z|^2 = Re\{z\}^2 + Im\{z\}^2$, so that the invariants are quadratic combinations of the original IP coefficients). In [7], a method for the derivation of the full set of 2D linear and quadratic invariants is

suggested, using the complex representation and a recursive scheme.

4.2 3D Implicit Polynomials Rotation Invariants

A 3D implicit polynomial of degree n is represented in the following way:

$$f_n(x,y,z) = \sum_{0 \le l,m,r,l+m+r \le n} a_{lmr} x^l y^m z^r \tag{4.16}$$

As in 2D, we would now like to derive rotation invariants consisting of combinations of the IP coefficients.

4.2.1 Derivation of 3D Invariants Using the Rotation Matrix

Every 3D rotation can be considered as 3 rotations, one around each axis (x, y and z) with angles α, β and γ respectively.

The new coordinates of an object after each of the rotations will then be:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4.17}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4.18}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4.19}$$

And the general rotation matrix is:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \cdot \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \cdot \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.20}$$

Applying the 2D method described in Section 4.1.1 in the 3D case will require to prove that the expressions are invariant under each of the above rotations.

For example, for a 2^{nd} degree IP the original polynomial is:

$$\begin{aligned} f_2(x,y,z) = \quad & a_{000} + a_{100}x + a_{010}y + a_{001}z + a_{200}x^2 + \\ & + a_{110}xy + a_{101}xz + a_{020}y^2 + a_{011}yz + a_{002}z^2 \end{aligned} \tag{4.21}$$

After substituting the relations between x, y, z and x', y', z' for a rotation around z axis (Eq. (4.19)), we get:

$$\begin{aligned} f_2(x',y',z') = \quad & a_{000} + a_{100}(cx' + sy') + a_{010}(-sx' + cy') + a_{001}z' + \\ & + a_{200}(cx' + sy')^2 + a_{020}(-sx' + cy')^2 + a_{002}z'^2 + \\ & + a_{110}(cx' + sy')(-sx' + cy') + a_{101}(cx' + sy')z' + \\ & + a_{011}(-sx' + cy')z' \end{aligned} \tag{4.22}$$

where $c \triangleq cos\gamma$ and $s \triangleq sin\gamma$. After expanding and simplifying, we get:

$$\begin{aligned} f_2(x', y', z') = & \underbrace{a_{000}}_{b_{000}} + \underbrace{(ca_{100} - sa_{010})}_{b_{100}} x' + \underbrace{(sa_{100} + ca_{010})}_{b_{010}} y' + \underbrace{a_{001}}_{b_{001}} z' + \\ & + \underbrace{\left(c^2 a_{200} - csa_{110} + s^2 a_{020}\right)}_{b_{200}} x'^2 + \underbrace{\left(s^2 a_{200} + csa_{110} + c^2 a_{020}\right)}_{b_{020}} y'^2 + \\ & + \underbrace{\left(2csa_{200} - 2csa_{020} + \left(c^2 - s^2\right) a_{110}\right)}_{b_{110}} x'y' + \\ & + \underbrace{(ca_{101} - sa_{011})}_{b_{101}} x'z' + \underbrace{(sa_{101} + ca_{011})}_{b_{011}} y'z' + \underbrace{a_{002}}_{b_{002}} z'^2 \end{aligned} \tag{4.23}$$

Obviously, we have the following invariant:

$$L_{3D,0} = b_{000} = a_{000} \tag{4.24}$$

We denote the 3D linear invariants by $L_{3D,k}$, where k is the form degree.

From examination of the new IP coefficients b_{lmr} and some trigonometric identities, we get that after z axis rotation, we also get that:

$$b_{200} + b_{020} = a_{200} + a_{020} \tag{4.25}$$

$$b_{002} = a_{002} \tag{4.26}$$

If we follow the same procedure for each of the rotation matrices, we'll get different invariants each time. Rotation around x axis will yield:

$$b_{020} + b_{002} = a_{020} + a_{002} \tag{4.27}$$

$$b_{200} = a_{200} \tag{4.28}$$

and from rotation around y axis:

$$b_{200} + b_{002} = a_{200} + a_{002} \tag{4.29}$$

$$b_{020} = a_{020} \tag{4.30}$$

The conclusion is that if we rotate the polynomial arbitrarily around x, y and z, the general rotation invariant will be

$$L_{3D,2} = b_{200} + b_{020} + b_{002} = a_{200} + a_{020} + a_{002} \tag{4.31}$$

As we mentioned earlier, the complexity of the trigonometric expressions grows considerably with the polynomial degree, and therefore the rotation matrix together with the trigonometric identities are useful only for low degree polynomials. We used this method for deriving 2 more quadratic invariants (see Appendix D):

$$Q_{3D,1} = b_{100}^2 + b_{010}^2 + b_{001}^2 = a_{100}^2 + a_{010}^2 + a_{001}^2 \tag{4.32}$$

$$\begin{aligned} Q_{3D,2} = b_{200}^2 + b_{020}^2 + b_{002}^2 - 2b_{200}b_{020} - 2b_{200}b_{002} - 2b_{020}b_{002} + b_{110}^2 + b_{101}^2 + b_{011}^2 = \\ = a_{200}^2 + a_{020}^2 + a_{002}^2 - 2a_{200}a_{020} - 2a_{200}a_{002} - 2a_{020}a_{002} + a_{110}^2 + a_{101}^2 + a_{011}^2 \end{aligned} \tag{4.33}$$

We denote the 3D quadratic invariants by $Q_{3D,k}$, where k is the form degree.

4.2.2 Derivation of 3D Invariants Using Quaternion Representation

Following the 2D method described in 4.1.2 of complex representation for the 3D case, will require the extension of the regular complex representation to include more dimensions. This can be done using the quaternions representation introduced in Section 3.5.1.1. Quaternions are often used for 3D rotation estimation, and therefore seem appropriate also for the task of rotation invariant representation.

Recall that a vector $r = (x, y, z)^T \in \mathbb{R}^3$ can be represented by a quaternion with a zero real part:

$$q = xi + yj + zk \tag{4.34}$$

For such a quaternion, we can extract the elements x, y and z in the following way:

$$\begin{aligned} x &= \tfrac{1}{2}\left(\bar{i}q + \bar{q}i\right) \\ y &= \tfrac{1}{2}\left(\bar{j}q + \bar{q}j\right) \\ z &= \tfrac{1}{2}\left(\bar{k}q + \bar{q}k\right) \end{aligned} \tag{4.35}$$

Where $\bar{i}, \bar{j}$ and $\bar{k}$ are the conjugates of i, j and k, respectively. Note that in the case of Eq. (4.34), $\bar{q} = -q$.

If we follow the 2D complex method, then given the 3D polynomial:

$$f_n(x, y, z) = \sum_{0 \le l,m,r, l+m+r \le n} a_{lmr} x^l y^m z^r \tag{4.36}$$

we rewrite it as a polynomial of quaternion variables $q = xi + yj + zk$ and

$\bar{q} = -xi - yj - zk$:

$$f_n(q) = \sum_{0 \le l,m,r,l+m+r \le n} \frac{a_{lmr}}{2^{l+m+r}} \left(\bar{i}q + \bar{q}i\right)^l \left(\bar{j}q + \bar{q}j\right)^m \left(\bar{k}q + \bar{q}k\right)^r \qquad (4.37)$$

In order to rotate q, we use the quaternion rotation introduced in Section 3.5.1.3. We denote our rotation unit quaternion by r_q:

$$q' = r_q q \bar{r}_q \qquad (4.38)$$

where q' is the new quaternion after rotation. Since r_q is a unit quaternion, we can write $|r_q| = |\bar{r}_q| = 1$, and therefore $|q'| = |q|$.

Thus, in the same manner as in 2D, coefficients which appear before quaternion monomials of the form $q^p \bar{q}^p = |q|^{2p}$ where $p \in \mathbb{N}$ will be rotation invariant. Using this method we should be able to get one linear invariant from each even degree form $H_{2p}(x, y, z)$, or $\left\lfloor \frac{n}{2} \right\rfloor + 1$ invariants from an IP of degree n, $f_n(x, y, z)$.

The first even degree form is $H_0(x, y, z) = a_{000}$ and the second is $H_2(x, y, z) = a_{200}x^2 + a_{110}xy + a_{101}xz + a_{020}y^2 + a_{011}yz + a_{002}z^2$. When re-writing the IP in quaternion representation, the quaternion IP coefficients of each form are dependent only on the real coefficients of the same degree form of the original IP. Therefore, the first rotation invariant is $L_{3D,0} = a_{000}$, since it is the quaternion coefficient of $|q|^0$ in $H_0(x, y, z)$. Let us now find the second rotation invariant (derived form the 2^{nd} degree form):

$$H_2(x, y, z) = a_{200}x^2 + a_{110}xy + a_{101}xz + a_{020}y^2 + a_{011}yz + a_{002}z^2 \qquad (4.39)$$

we substitute x, y and z by their quaternion representation (Eq. (4.35)) and expand each element separately:

$$a_{200}x^2 = a_{200}\frac{1}{2^2}\left(\bar{i}q + \bar{q}i\right)^2 = \frac{a_{200}}{2^2}(iqiq + 2|q|^2 + qiqi)$$

$$a_{020}y^2 = a_{020}\frac{1}{2^2}\left(\bar{j}q + \bar{q}j\right)^2 = \frac{a_{020}}{2^2}(jqjq + 2|q|^2 + qjqj)$$

$$a_{002}z^2 = a_{002}\frac{1}{2^2}\left(\bar{k}q + \bar{q}k\right)^2 = \frac{a_{002}}{2^2}(kqkq + 2|q|^2 + qkqk)$$

$$a_{110}xy = a_{110}\frac{1}{2^2}\left(\bar{i}q + \bar{q}i\right)\left(\bar{j}q + \bar{q}j\right) = \frac{a_{110}}{2^2}(iqjq - |q|^2k + qkq + qiqj)$$

$$a_{101}xz = a_{101}\frac{1}{2^2}\left(\bar{i}q + \bar{q}i\right)\left(\bar{k}q + \bar{q}k\right) = \frac{a_{101}}{2^2}(iqkq + |q|^2j - qjq + qiqk)$$

$$a_{011}yz = a_{011}\frac{1}{2^2}\left(\bar{j}q + \bar{q}j\right)\left(\bar{k}q + \bar{q}k\right) = \frac{a_{011}}{2^2}(jqkq - |q|^2i + qiq + qjqk)$$

Summing it all up and looking for the coefficient of $|q|^2$ we get the second rotation invariant $\frac{1}{2}L_{3D,2} = \frac{1}{2}(a_{200} + a_{020} + a_{002})$. Note that this is the same invariant as the one we found in (4.31), only multiplied by $\frac{1}{2}$. Also, note that since quaternion multiplication is non-commutative, $iqiq \neq qiqi$ and therefore we can't simplify such expressions or use binomial expansion as in 2D. The multiplications complexity increases with the polynomial degree and becomes impractical when trying to derive quadratic invariants. Therefore, this method as well is useful only for linear invariants of low degree polynomials.

4.2.3 Derivation of 3D Invariants Using Tensor Representation

The third method for 3D rotation invariants derivation, is based on the pose estimation method of [6], which we introduced in Section 3.4.2.

We recall that the implicit polynomial can be separated into homogeneous

parts in the following way:

$$f_n(x_1,x_2,x_3) = \underbrace{a_{000}}_{H_0} + \underbrace{a_{100}x_1 + a_{010}x_2 + a_{001}x_3}_{H_1(x_1,x_2,x_3)} + \\ + \underbrace{a_{200}x_1^2 + +a_{110}x_1x_2 + a_{020}x_2^2 + a_{101}x_1x_3 + a_{011}x_2x_3 + a_{002}x_3^2}_{H_2(x_1,x_2,x_3)} + \\ + ... + \underbrace{a_{n00}x_1^n + ... + a_{0n0}x_2^n + ... + a_{00n}x_3^n}_{H_n(x_1,x_2,x_3)} \tag{4.40}$$

The IP based pose estimation fits an even degree IP with $n = 2p$ and extracts only the leading form ($H_n(x,y,z)$) since this form is invariant to translation. In order to find the intrinsic orientation of the IP, it performs $(p-1)$ tensor contractions on the leading form (see Section 3.4.2), resulting in a 3x3 matrix denoted by $V(H_n)$, whose elements are linear combinations of the leading form coefficients.

We suggest to perform one more tensor contraction (i.e., p contractions for an IP of degree $n = 2p$). This will result in a tensor of order 0 (a scalar) which is the trace of $V(H_n)$. The trace of a matrix is known to be an invariant under Euclidean transformations and therefore should be rotation invariant. We refer only to even degree forms ($n = 2p$) since contraction of an odd degree form results in a vector, which cannot be contracted again into a scalar, and our goal is to derive rotation invariant scalar expressions.

When using a 2^{nd} degree form we get:

$$H_2(x_1,x_2,x_3) = a_{200}x_1^2 + a_{020}x_2^2 + a_{002}x_3^2 + a_{110}x_1x_2 + a_{101}x_1x_3 + a_{011}x_2x_3 \tag{4.41}$$

and:

$$V(H_2) = \begin{bmatrix} a_{200} & \frac{a_{110}}{2} & \frac{a_{101}}{2} \\ \frac{a_{110}}{2} & a_{020} & \frac{a_{011}}{2} \\ \frac{a_{101}}{2} & \frac{a_{011}}{2} & a_{002} \end{bmatrix} \tag{4.42}$$

The rotation invariant we get is $L_{3D,2} = trace\left[V(H_2)\right] = a_{200} + a_{020} + a_{002}$.

The difference from the 2 other methods, is the simplicity in which we can derive an invariant from any even degree form, even for high degrees. A 4^{th} degree form will be:

$$\begin{aligned} H_4(x_1, x_2, x_3) = \quad & a_{400}x_1^4 + a_{040}x_2^4 + a_{004}x_3^4 + \\ & + a_{310}x_1^3x_2 + a_{301}x_1^3x_3 + a_{130}x_1x_2^3 + a_{031}x_2^3x_3 + \\ & + a_{103}x_1x_3^3 + a_{013}x_2x_3^3 + a_{220}x_1^2x_2^2 + + a_{202}x_1^2x_3^2 + \\ & + a_{022}x_2^2x_3^2 + a_{211}x_1^2x_2x_3 + a_{121}x_1x_2^2x_3 + a_{112}x_1x_2x_3^2 \end{aligned} \tag{4.43}$$

and after creating the tensor and one contraction ($p = 2$, $p - 1 = 1$):

$$V(H_4) = \begin{bmatrix} a_{400} + \frac{a_{220}}{6} + \frac{a_{202}}{6} & \frac{a_{310}}{4} + \frac{a_{130}}{4} + \frac{a_{112}}{12} & \frac{a_{301}}{4} + \frac{a_{103}}{4} + \frac{a_{121}}{12} \\ \frac{a_{310}}{4} + \frac{a_{130}}{4} + \frac{a_{112}}{12} & a_{040} + \frac{a_{220}}{6} + \frac{a_{022}}{6} & \frac{a_{031}}{4} + \frac{a_{013}}{4} + \frac{a_{211}}{12} \\ \frac{a_{301}}{4} + \frac{a_{103}}{4} + \frac{a_{121}}{12} & \frac{a_{031}}{4} + \frac{a_{013}}{4} + \frac{a_{211}}{12} & a_{004} + \frac{a_{202}}{6} + \frac{a_{022}}{6} \end{bmatrix} \tag{4.44}$$

and the rotation invariant we get is $L_{3D,4} = trace\left[V(H_4)\right] = a_{400} + a_{040} + a_{004} + \frac{1}{3}(a_{220} + a_{202} + a_{022})$.

In the general case, of a form of degree $n = 2p$, we developed a general expression for the linear invariant we get using p tensor contractions:

$$L_{3D,n} = trace\left[V\left(H_n\right)\right] = \sum_{i_{n-1}=1}^{3} \cdots \sum_{i_3=1}^{3} \sum_{i_1=1}^{3} s^{i_1 i_1 i_3 i_3 \cdots i_{n-1} i_{n-1}} \tag{4.45}$$

and since each tensor element can be expressed by an appropriate polynomial coefficient divided by the number of permutations: $s^{i_1 i_2 \ldots i_n} = \frac{a_{lmr}}{\frac{n!}{l!m!r!}}$, we get :

$$L_{3D,n} = trace\left[V\left(H_n\right)\right] = \sum_{l,m,r\ even, l+m+r=n} \frac{l!m!r!}{n!} \cdot \frac{(n/2)!}{(l/2)!\,(m/2)!\,(r/2)!} \cdot a_{lmr} \tag{4.46}$$

where $\frac{(n/2)!}{(l/2)!(m/2)!(r/2)!}$ is the number of times each tensor element participates in the tensor contractions (the indexes are all divided by 2 since each pair of indexes is set to be equal in the contraction process). Using the tensor approach, we can derive the complete set of $\left\lfloor \frac{n}{2} \right\rfloor + 1$ linear invariants for an IP of degree n. Note that this is an explicit expression for the derivation of 3D invariants, unlike the recursive derivation method given in [7] for 2D invariants. We can extend our explicit 3D invariants expression for the 2D linear invariants computation as well, in the following way:

$$L_{2D,n} = \sum_{l,m\ even, l+m=n} \frac{l!m!}{n!} \cdot \frac{(n/2)!}{(l/2)!\,(m/2)!} \cdot a_{lm} \tag{4.47}$$

Thus, we need less computations than the recursive scheme given in [7] for the 2D linear invariants.

Chapter 5

Feature Extraction

5.1 Background

Recognition or classification of 3D objects is important in various fields, such as medicine, robotics, automatic security systems etc. The recognition problem can be described by the next scheme: Assume that we have L different 3D objects, each is represented by a cloud of points describing its surface, and may have several different representations (for example, the same object was acquired from different viewing angles). Thus, we have a dictionary which contains one or more representations of each of the objects, which can be used as a learning set. Given a new representation of an object, we would like to classify it and determine which object it is out of the L possibilities in our objects set.

Previous approaches usually require a pre-processing stage of pose estimation for the alignment of the new object representation with each of the dictionary objects representations. In many approaches, there is also a need

to identify matching key points between 2 object representations in order to achieve good classification results. These stages have high computational cost, since accurate pose estimation (such as the Iterative Closest Point from Section 3.5.2) and matching key points are usually iterative, and each iteration performs calculations for many data-points. An example for such an approach can be found in [15], which needs a large number of spin images (2D histograms around objects surface points), from different points of view on the object surface, in order to perform surface matching for classification.

The databases which are available for 3D objects classification performance evaluation are either synthetic databases (i.e., computer graphic models, such as [24]) or small real databases (only a relatively small number of different objects, such as [10]). Therefore, we had to create an objects database of our own, which will include real acquisitions of many objects in various viewing angles.

5.2 Rigid Objects Database Creation

This database was acquired using the equipment of the Geometric Image Processing (GIP) lab at the Computer Science Faculty of the Technion.

The acquisition system was developed at the GIP lab and it consists of a camera and a projector. The system is based on the 'structured light' technique, in which different patterns of light stripes are projected on the 3D object, and a depth map is created by analysis of the camera 2D image [18]. The projector's hardware is manipulated so that it triggers the camera. The acquisition analysis software was developed at the GIP lab. The camera

frame rate is 30Hz, and each 3D image is produced from a sequence of 12 consecutive camera frames of 12 different projected patterns. Therefore, the system acquires 2.5 images per second. For the acquisition, the object is placed on a stand, with a black background behind it. After the acquisition, each 3D frame is re-sampled on a uniform xy grid of 320x240 pixels, then filtered in order to smooth noise artifacts, and cropped so that it won't contain background elements. The final number of data-points per object in our database is $\sim 12,000$ on average. Each pixel has 3 coordinates (x, y, z) and represents a point in the 3D space.

We acquired 40 different objects, each was placed on the rotatable stand and acquired in 9 consecutive positions. The difference between 2 consecutive positions is around $10^o - 15^o$ degrees, and we acquired 5 frames in each position of each object. A possible application for such a setup is a factory production line that contains various products which need automatic sorting. The database objects appear in Appendix F. An example for 9 different positions of the object 'fox' is shown in Fig. 5.1.

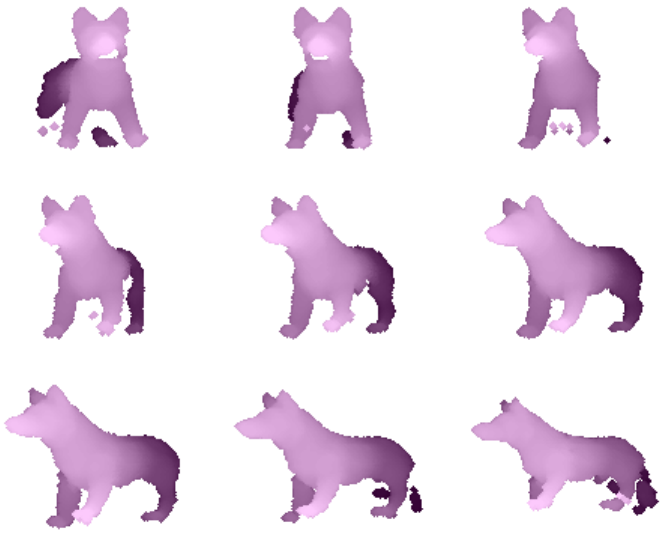


Figure 5.1: 9 different positions of the object 'fox'

5.3 Faces Database

This database is also from the Geometric Image Processing (GIP) lab. The GIP lab gathered acquisitions of many human volunteers using the same system described in the previous section (same frame rate and resolution). The subjects are generally told to look straight at the camera and hold still, since precision is much higher for stationary objects. The result for each subject is a video of $\sim 50 - 70$ 3D frames, during which the subject usually moves his head a little and changes his expression from neutral to happy. Since the expressions are usually moderate, and the subject is looking directly at the camera, this is a cooperative situation. In other words - the subject would like the system to correctly identify him/her. This application

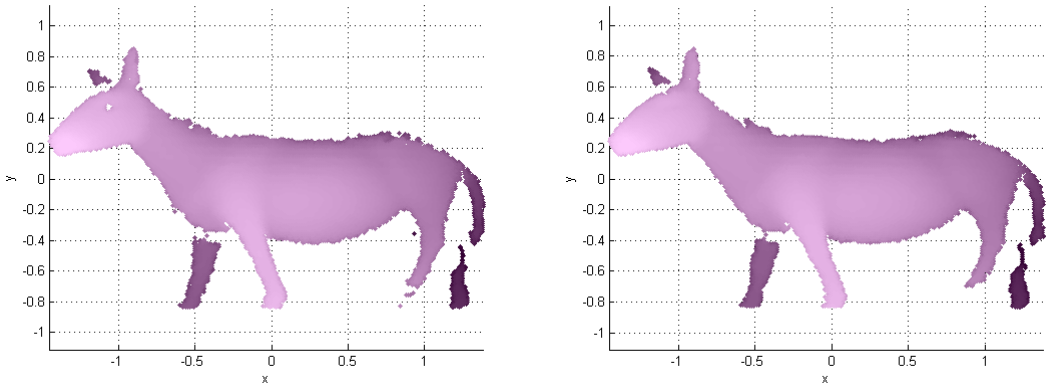
is interesting, for example, for identification at an entrance to restricted facilities. The database faces appear in Appendix G. An example for 6 different frames of the same face is shown in Fig. 5.2.

Figure 5.2: 6 different frames of the same face

When working with faces and implicit polynomials fitting, we tried using high degree polynomials (degree>8). Eventually we chose not to use these IP degrees for classification purposes, since their invariants were not stable enough. A brief overview of high degree IP fitting is brought in Appendix A.

5.4 Noise Analysis

In order to analyze the acquisition noise of the system, we examined 2 consecutive frames of the same position of the object 'mule' (see Figure 5.3).



(a) (b)

Figure 5.3: 2 consecutive frames of the object 'mule'

For each data point in the second frame (x_{f2}, y_{f2}, z_{f2}), we found the closest point in the first frame $(x_{f1,closest}, y_{f1,closest}, z_{f1,closest})$ using l_1 distance. The noise was then estimated for each coordinate separately, using subtraction of coordinates of all the closest points pairs.

$$n_1\left(x_{f2}, y_{f2}\right) = x_{f2} - x_{f1,closest} \tag{5.1}$$

$$n_2\left(x_{f2}, y_{f2}\right) = y_{f2} - y_{f1,closest} \tag{5.2}$$

$$n_3\left(x_{f2}, y_{f2}\right) = z_{f2} - z_{f1,closest} \tag{5.3}$$

where n_1 is the noise of coordinate x, n_2 is the noise of coordinate y and n_3 is the noise of coordinate z. The noise patterns and their histograms are shown in Figure 5.4. It can be seen that the histograms can be approximated by Gaussian distributions and that most of the noise values are in the range $[-0.02, 0.02]$ (in all 3 axes). Using the approximation that $\sim 97\%$ of Gaussian distribution values are in the range $[-3\sigma, 3\sigma]$, where σ is the noise standard

deviation, we estimate $\hat{\sigma} = \frac{0.02}{3} \cong 0.007$ for the noise in each of the 3 axes. The noise auto-correlations in the same coordinate and cross-correlations between different coordinates are shown in Figure 5.5 (only the central area of each correlation matrix is shown). It can be seen that there is a neighborhood of correlated pixels in each direction between noise matrices of the same coordinate (e.g., auto-correlation of n_1), but hardly any correlation between noise matrices of different coordinates (e.g., cross-correlation between n_1 and n_2). If we repeat this analysis using a different object, or one of the faces, we get similar results. We conclude that our noise model is additive colored Gaussian noise in each coordinate. In the same coordinate, the correlation decreases along 5 pixels to each direction, and there is no correlation between different coordinates. The standard deviation of the noise in each axis is $\sigma \cong 0.007$. We will later use this model in order to enrich our learning database to include more synthetic acquisitions.

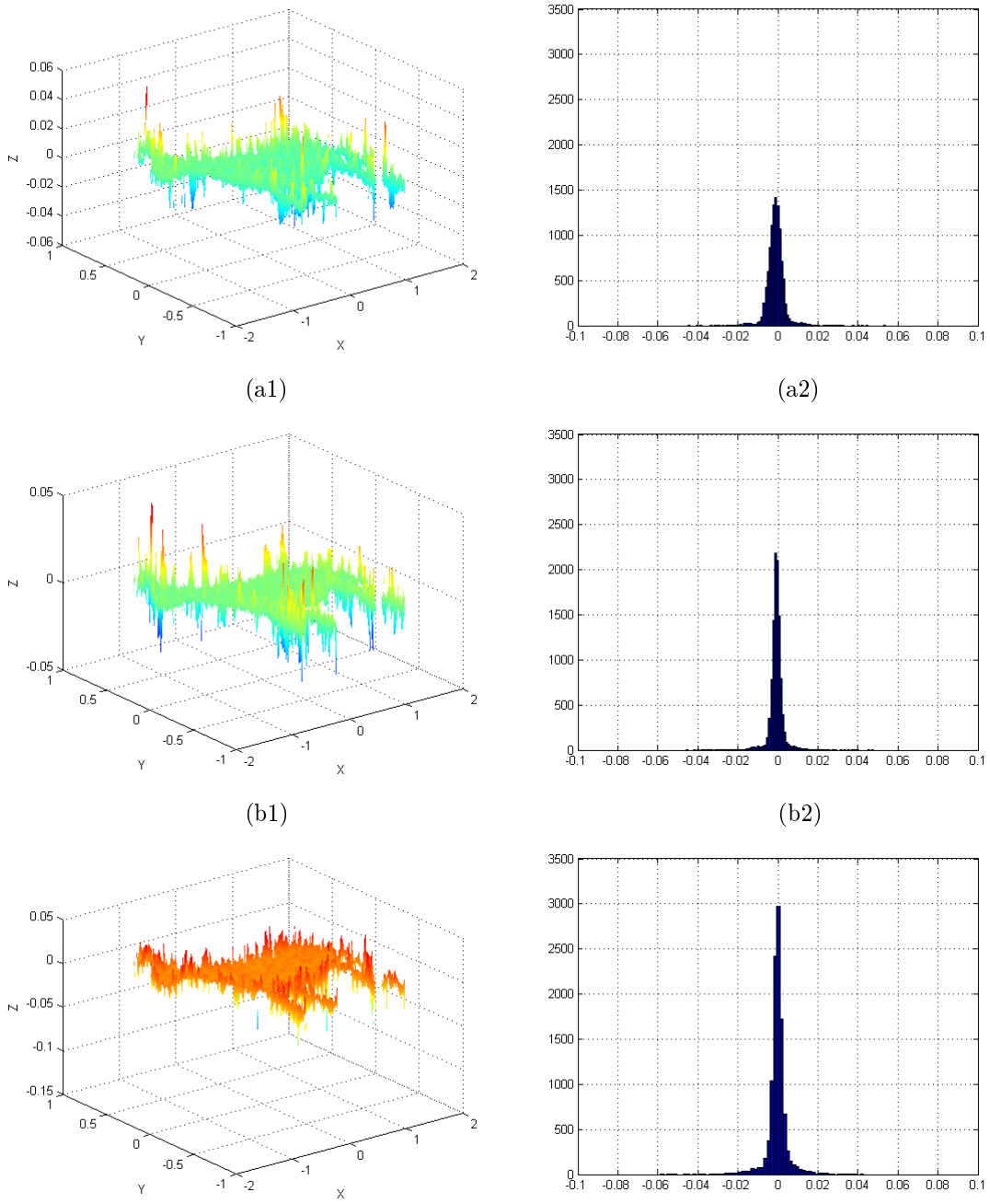
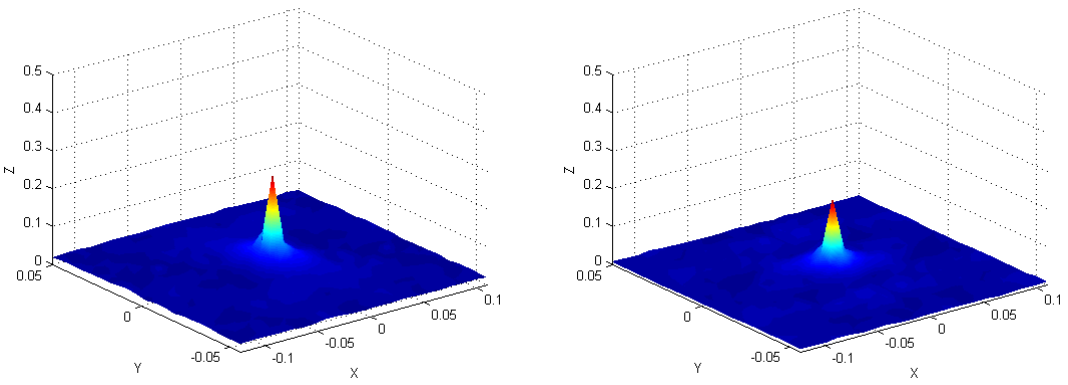
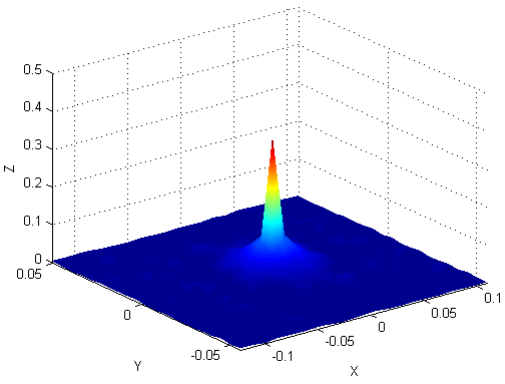


Figure 5.4: Acquisition noise patterns and histograms for each of the coordinates: (a) noise in coordinate x (b) noise in coordinate y and (c) noise in coordinate z

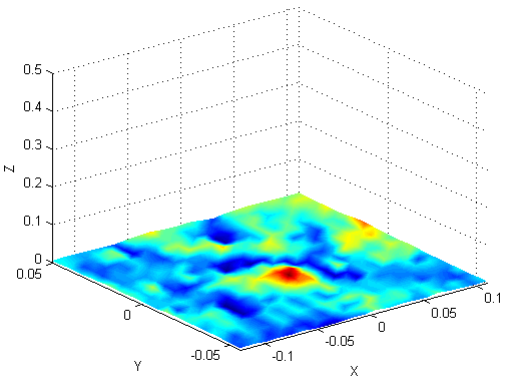


(a)

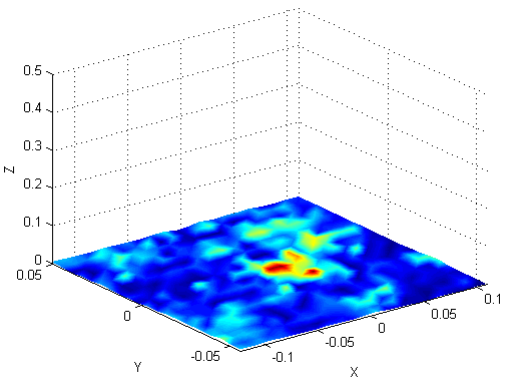
(b)



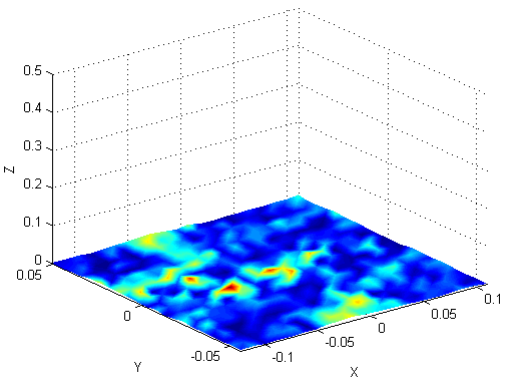
(c)



(d)



(e)



(f)

Figure 5.5: Acquisition noise auto-correlations and cross-correlations: (a) auto-correlation of n_1 (b) auto-correlation of n_2 (c) auto-correlation of n_3 (d) cross-correlation between n_1 and n_2 (e) cross-correlation between n_2 and n_3 (f) cross-correlation between n_1 and n_3

5.5 Data Preprossessing

In order to create a feature vector for classification, we have to apply some pre-processing to each object/face frames. The input to this stage is a cloud of data-points $\{(x_i, y_i, z_i)\}|_{i=1,\ldots,N}$.

5.5.1 Translation

Consider that we have a data-set having a center-of-mass point at (x_{cm}, y_{cm}, z_{cm}), and that the maximum distance between the point (x_{cm}, y_{cm}, z_{cm}) and the data-set is d. If the coordinates (x_{cm}, y_{cm}, z_{cm}) are much larger than d, then the monomials belonging to the different points of the data-set wouldn't differ much. Thus, the corresponding rows of the matrix M (see Sections 2.3.2 and 2.3.3) would be almost the same, and it will be close to singular. Therefore, a reasonable choice of the center-of-mass of the data-set location would be the origin.

Therefore, for translation invariance, we locate the center of mass of the data-points at the origin:

$$x_{i,translated} = x_i - \frac{1}{N}\sum_{k=1}^{N} x_k \tag{5.4}$$

$$y_{i,translated} = y_i - \frac{1}{N}\sum_{k=1}^{N} y_k \tag{5.5}$$

$$z_{i,translated} = z_i - \frac{1}{N}\sum_{k=1}^{N} z_k \tag{5.6}$$

5.5.2 Scaling

In the 2D case [3] it was shown that the scaling factor affects the stability of the polynomial coefficients. If the object is very small, so that the coordinates of the data-set points are very close to zero, then the higher degree monomials will be much smaller than the lower degree ones. The 2D analysis of this case shows that the high degree IP coefficients are much larger than the lower degree ones, and they are very sensitive even to small changes of the data-set. Similarly, if the object is very big, so that the coordinates of the data-set points are much larger than 1, then the higher degree monomials will be much larger than the lower degree ones. The 2D analysis of this case shows that the low degree IP coefficients are much larger than the higher degree ones, and they are very sensitive even to small changes of the data-set. Thus, scaling the data-set so that its coordinates will be close to ± 1 may provide a polynomial with more stable coefficients in the presence of the data-set noise.

The 2D conclusion is applicable also in the 3D case, and so we would like our surface to have coordinates close to ± 1. Applying scaling so that *all* the points will have coordinates in the range $[-1, 1]$ is too sensitive to outliers. As in [3], we chose our scaling factor to be the 75^{th} percentile of the distances of the data-points from the origin. In other words, we find the distances of all the data-points from the origin, sort them and choose the element that is larger than 75% of the other elements as the scaling factor $S_{75\%}$. We then

scale each coordinate using this scaling factor:

$$x_{i,scaled} = x_i / S_{75\%} \tag{5.7}$$

$$y_{i,scaled} = y_i / S_{75\%} \tag{5.8}$$

$$z_{i,scaled} = z_i / S_{75\%} \tag{5.9}$$

5.5.3 Normal Direction Calculation

In order to solve the IP fitting problem, we need the normal direction at each point. In order to find a consistent normal direction in the presence of noise, we find the closest neighbors of the current point and fit to them a local plane. Then we choose the positive direction of the normal to be in the direction of the camera (i.e., so that it has a positive z direction). This has a smoothing effect on the normal direction constraints, since neighboring points will have very similar normals. An example for normal direction estimation was shown in Fig. 2.3.

5.5.4 Mirroring of the Faces

As we saw in Section 3.4.2.4, since a 3D face is an open surface, the IP that we fit to it is unconstrained in areas in which there are no data-points. An example was shown in Fig. 3.6 (repeated here for convenience, see Fig. 5.6), where a 2^{nd} degree IP fit to a face results in an ellipsoid much larger than the face. The second instance of the face (the smiling expression) has a different ellipsoid fit even though the difference between the 2 acquisitions is only a slight deformation.

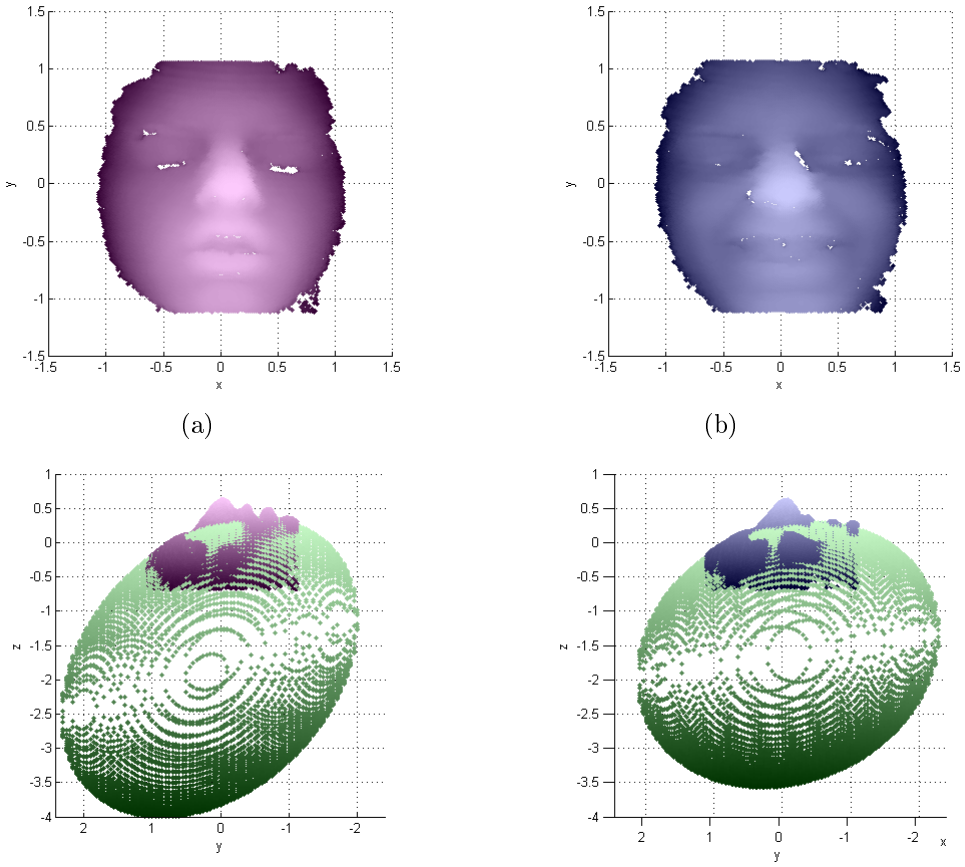


Figure 5.6: An example for different 2^{nd} degree IP fitting (using Gradient1) of 2 different scans of the same face: (a) neutral expression (b) smiling expression (c) a 2^{nd} degree IP fit to the neutral face (side view, in green) (d) a 2^{nd} degree IP fit to the smiling face (side view, in green)

In order to get a stable IP fitting, we suggest to mirror each face in the following way:

We fit a 1^{st} degree IP to the face and place the resulting plane so that it passes through the origin (the center of mass of the face). Then we lower it further in the z direction by an empiric distance (-0.25 for faces after the scaling stage). We dispose of the points below the plane and mirror the rest

of the points with respect to the plane in the following way:

Let $\underline{p}_1$ denote a data point we want to mirror, $\underline{p}_0$ denote some point on the plane and $\underline{n}$ denote the unit normal to the plane. We can find the closest point $\underline{p}_2$ to $\underline{p}_1$ on the plane using:

$$\underline{p}_2 = \underline{p}_1 - \left[\underline{n}^T \left(\underline{p}_1 - \underline{p}_0\right)\right] \underline{n} \tag{5.10}$$

Then, the mirrored point coordinates $\underline{p}_{1m}$ are:

$$\underline{p}_{1m} - \underline{p}_2 = \underline{p}_2 - \underline{p}_1 \tag{5.11}$$

or:

$$\underline{p}_{1m} = 2\underline{p}_2 - \underline{p}_1 \tag{5.12}$$

Figure 5.7 shows the relations between the mirroring plane and the points $\underline{p}_1$, $\underline{p}_0$, $\underline{p}_2$ and $\underline{p}_{1m}$.

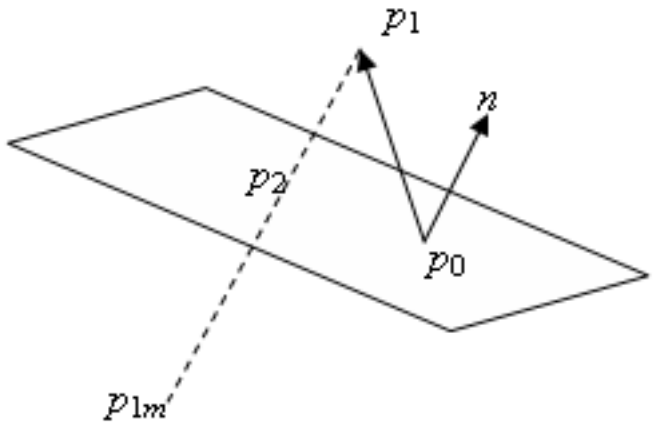


Figure 5.7: Mirroring of a point p_1 with respect to an arbitrary plane

The mirroring results are shown in Fig. 5.8. It can be seen that the ellipsoid is now very similar between the 2 occurrences of the same face, despite the smile deformation. In addition, after the mirroring, the ellipsoid size corresponds to the face size, unlike the large ellipsoid we got before the mirroring (Fig. 5.6).

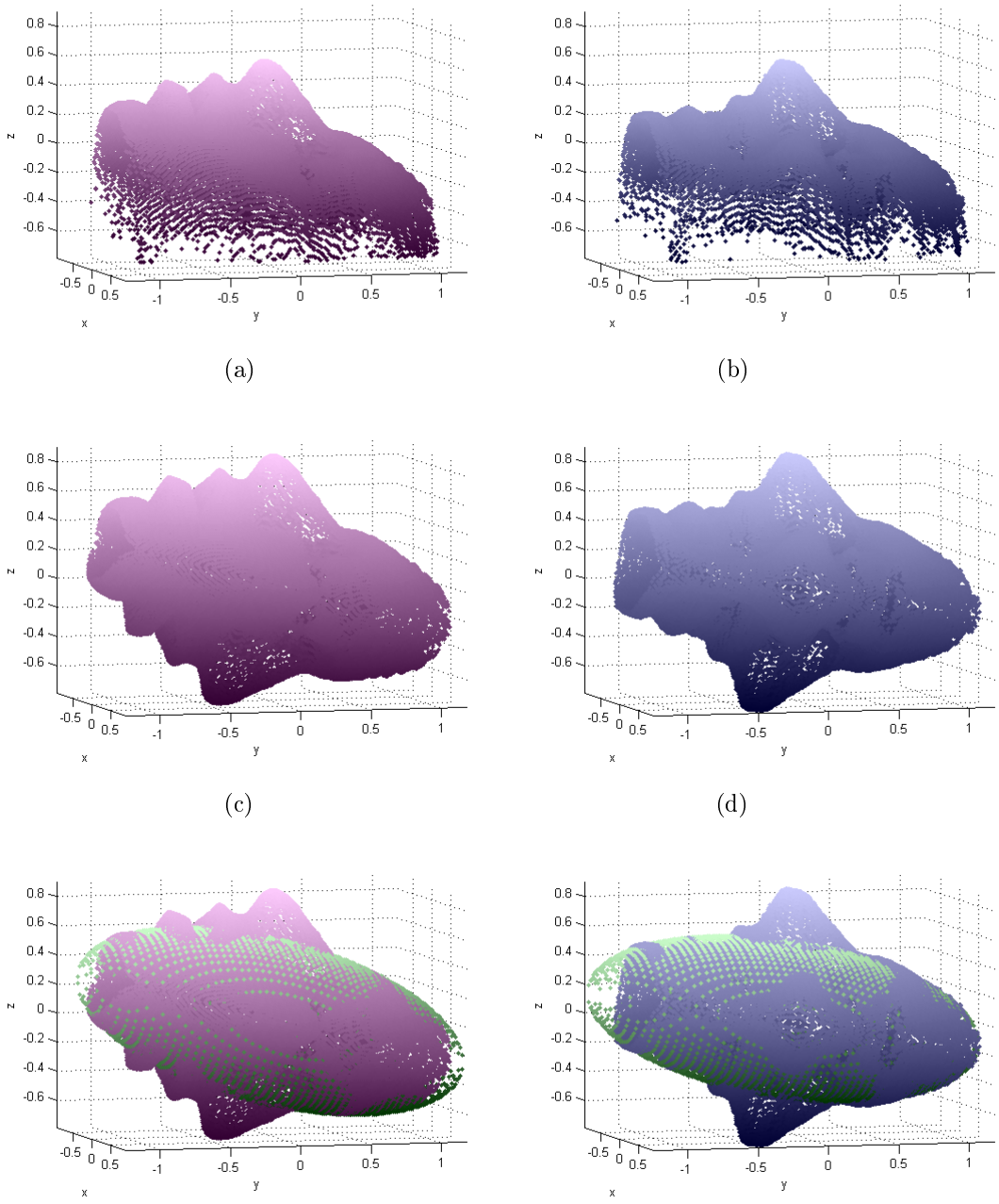


Figure 5.8: An example for the similar 2^{nd} degree IP fitting (using Gradient1) of 2 different scans of the same face when using mirroring: (a) neutral expression (b) smiling expression (c) mirroring of the neutral face (d) mirroring of the smiling face (e) a 2^{nd} degree IP fit to the mirrored neutral face (in green) (f) a 2^{nd} degree IP fit to the mirrored smiling face (in green)

We don't apply mirroring to the rigid objects since they are a heterogeneous group, and the desired location of the mirroring plane varies considerably from object to object.

5.5.5 2D Projections

In order to obtain more classification features using implicit polynomials, we used also 2D projections. We used only projections on the main axes (xy, xz and yz) in order to avoid high complexity calculations.

5.5.5.1 Rigid Objects - xy Projection

For the rigid objects, the most descriptive projection is the projection on xy. The two other projections (xz and yz) were not very informative, and had very noisy contours. An example is shown in Fig. 5.10, note the noisy contour on the lower part of the 2D projections. After the 2D projection, we used the morphological operator of 'closing' for holes filling and then extracted the 2D contour. An example for xy projection of 3 different positions of the object 'fox' is shown in Fig. 5.9.

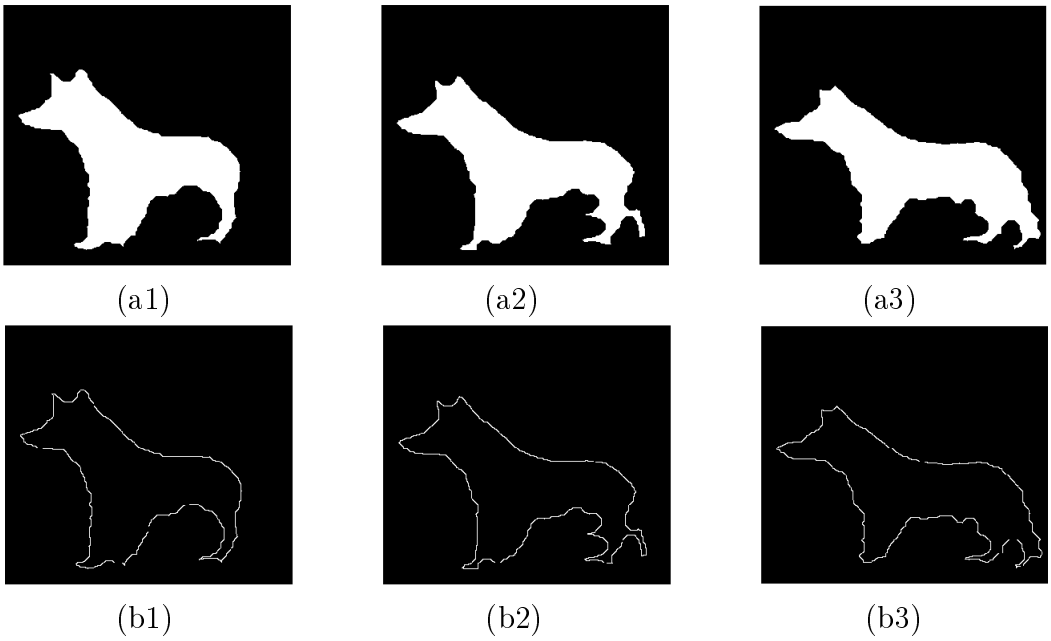


Figure 5.9: (a) xy projections of 3 different positions of the object 'fox' and (b) their 2D contours

Figure 5.10: An example for projections of 'fox' that have noisy contours (a) xz projections of 3 different positions (b) yz projections of 3 different positions. These projections were excluded from our rigid objects classification scheme due to their noisy contours.

5.5.5.2 Faces - xz Projection and yz Projection

In this case, the xy projection is very similar for different faces, and the more descriptive projections are the projections on xz and yz. We perform the 2D projections after the mirroring of the faces, so that they won't have noisy contours as in the objects database (like the example shown in Fig. 5.10). After each 2D projection, we used the morphological operator of 'closing' for holes filling and then extracted the 2D contour. An example for the 3 projections of a face is shown in Fig. 5.11.

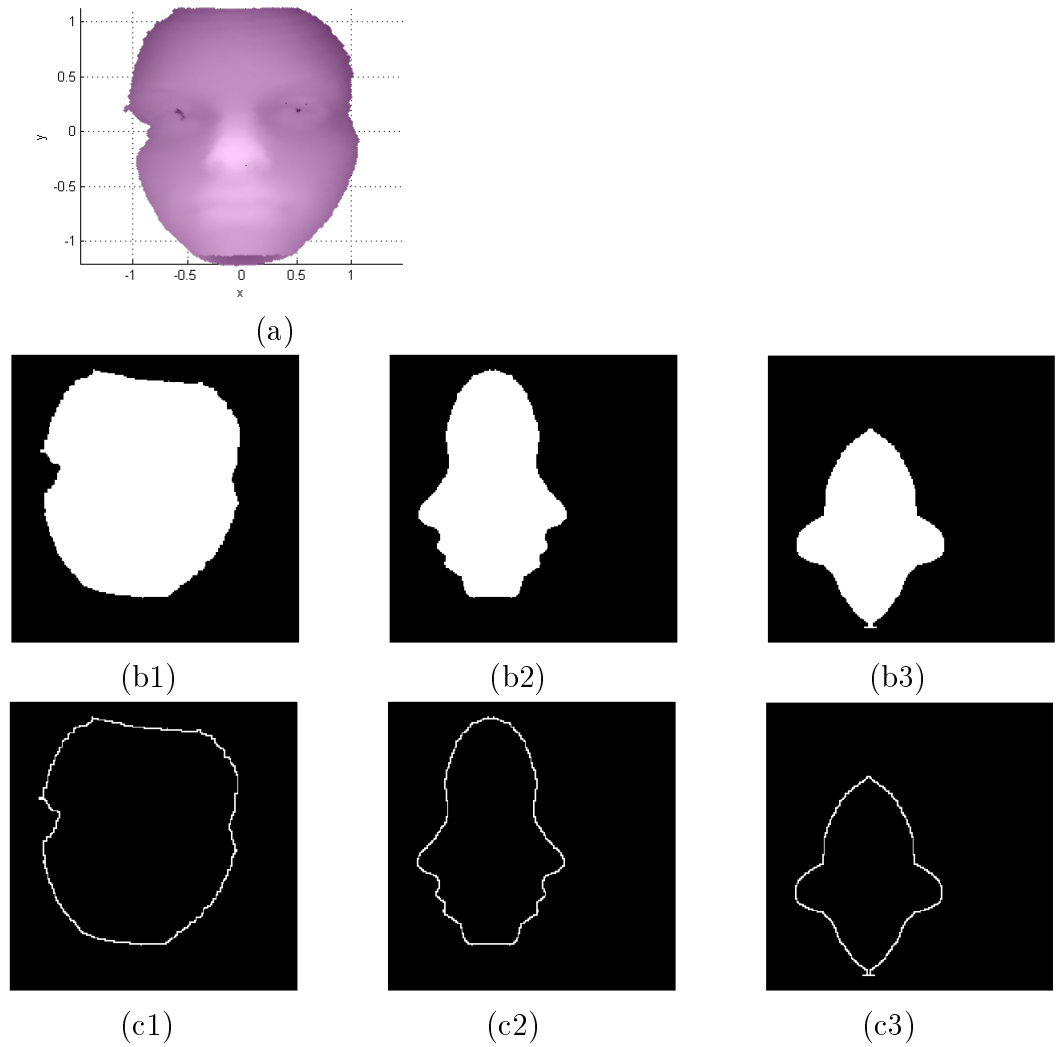


Figure 5.11: (a) a face after mirroring (b) 3 projections (on xy, xz and yz) of this mirrored face and (c) their 2D contours. The xy projection was excluded from our faces classification scheme due to its high similarity between different faces.

5.6 Classification features

5.6.1 Rotation Invariants Derivation

5.6.1.1 Linear Invariants

After all the pre-processing stages, we fit implicit polynomials of degrees 2, 4, 6 and 8 to the 3D objects and faces, and to their 2D projections (xy projection for the objects, and xz and yz projections for the faces). The reason for the multi degree polynomial fitting is that different polynomial degrees describe different features in the object. A simple and smooth object can be well described by a low degree IP, while higher IP degrees fitting will be less stable. However, a more complicated object with many details needs a relatively high degree IP for description, and a low degree IP won't contain enough information about it to be able to classify it correctly. By this approach we follow the Multi-Order and Fitting-Error Technique (MOFET), introduced for 2D contours recognition using implicit polynomials [3]. The reason for the even degree polynomials is that our linear invariants are derived only from even degree forms. Thus, if we fit an IP of an odd degree, we won't use its leading form for invariants derivation, and so we could use a lower (even) degree IP for the same number of invariants derivation.

Using the polynomial representation, we separated the polynomial into forms, and from each even degree form - we derived the linear rotation invariants described in Section 4.2.3 using the explicit expressions we developed. In the 3D case the general invariant expression for a form of degree n (n=2p)

is (Eq. (4.46), repeated here for convenience):

$$L_{3D,n} = \sum_{l,m,r \ even, l+m+r=n} \frac{l!m!r!}{n!} \cdot \frac{(n/2)!}{(l/2)!\,(m/2)!\,(r/2)!} \cdot a_{lmr} \tag{5.13}$$

And in the 2D case, the general invariant expression for a form of degree n (n=$2p$) is (Eq. (4.47), repeated here for convenience):

$$L_{2D,n} = \sum_{l,m \ even, l+m=n} \frac{l!m!}{n!} \cdot \frac{(n/2)!}{(l/2)!\,(m/2)!} \cdot a_{lm} \tag{5.14}$$

In both 3D and 2D cases, we have $\left\lfloor \frac{n}{2} \right\rfloor + 1$ linear invariants for an IP of degree n.

5.6.1.2 Quadratic Invariants

For the 3D case, using the same polynomial representation, we also derived the 2 quadratic rotation invariants described in Section 4.2.1.

$$Q_{3D,1} == a_{100}^2 + a_{010}^2 + a_{001}^2 \tag{5.15}$$

$$Q_{3D,2} = a_{200}^2 + a_{020}^2 + a_{002}^2 - 2a_{200}a_{020} - 2a_{200}a_{002} - 2a_{020}a_{002} + a_{110}^2 + a_{101}^2 + a_{011}^2 \tag{5.16}$$

The quadratic and angular invariants in the 2D case [7] are not stable enough [3], and therefore were not used as classification features.

5.6.2 Additional features

5.6.2.1 Implicit Polynomial Fitting Error

Following the Multi-Order and Fitting-Error Technique (MOFET), introduced for 2D contours recognition using implicit polynomials [3], we also suggest to use the implicit polynomial fitting error as a classification feature. When comparing 2 different objects that have a similar shape, but one is smooth while the other has fine details, the first object will have a lower IP fitting error than the second one, since its smooth IP description will describe it well. Obviously, this feature is an inherent property of the object, and is also invariant to rotation.

Therefore, for each IP fitting degree, after solving the least squares problem for the IP fitting, we calculate the fitting error for each data-point and use the 75^{th} percentile of the errors vector as a feature describing the fitting error. In the 3D case we denote this feature by $E_{75\%,3D,n}$ where n is the IP fitting degree. In the 2D case we denote this feature by $E_{75\%,2D,n}$ where n is the IP fitting degree.

5.6.2.2 Eigenvalues of Principal Component Analysis (PCA)

We obtain 3 more classification features using principal component analysis on the original data points. PCA was described in detail in Section 3.4.1. The PCA eigenvalues are an inherent property of the object, since they describe its data-points scatter on its principal axes, and therefore - they are also invariant to rotation.

In the 3D case, we calculate the 3x3 data scatter matrix (see Eq. (3.4)),

and perform eigenvalue decomposition. We sort the eigenvalues in decreasing order according to their magnitude, and use them as additional classification features, denoted by λ_1, λ_2 and λ_3.

5.6.3 The Classification feature Vector

Assembling all the calculated features, we have:

1. 3D linear invariants from each even degree form of each 3D IP ($\lfloor \frac{n}{2} \rfloor + 1$ linear invariants for an IP of degree n).
2. Two 3D quadratic invariants from each 3D IP.
3. 2D linear invariants from each even degree form of each 2D IP of each projection we chose ($\lfloor \frac{n}{2} \rfloor + 1$ linear invariants for an IP of degree n).
4. 3D fitting error feature from each 3D IP.
5. 2D fitting error feature from each 2D IP of each projection we chose.
6. 3D PCA eigenvalues.

The total number of features depends on the IP degrees we fit to the 3D object and its 2D projections. We chose different feature vectors for objects recognition and for faces recognition according to the differentiation ability of the various features in each recognition problem. For example, the fitting error (in both 3D and 2D cases) is very similar for different faces, and therefore it does not contribute to the classification. On the other hand, the faces have many details and an IP degree of 8 is required in order to achieve good recognition results. The objects are different enough from each other and

so we don't need an IP of degree 8 for classification and use only degrees of 2,4 and 6. The full list of classification features for each database appears in Table 5.1.

	Objects Recognition	Faces Recognition
3D linear invariants	IP degrees 2,4,6 (2+3+4=9 features)	IP degrees 2,4,6,8 (2+3+4+5=14 features)
3D IP fitting error	IP degrees 2,4,6 (1+1+1=3 features)	—
two 3D quadratic invariants	IP degrees 2,4 (2+2=4 features)	IP degrees 2,4 (2+2=4 features)
2D linear invariants	xy - IP degrees 2,4,6 (2+3+4=9 features)	xz - IP degrees 2,4,6 (2+3+4=9 features) yz - IP degrees 2,4 (2+3=5 features)
2D IP fitting error	xy - IP degrees 2,4,6 (1+1+1=3 features)	—
3D PCA eigenvalues	3 eigenvalues (3 features)	3 eigenvalues (3 features)

Table 5.1: Classification features for the objects database and the faces database

The total number of features was 31 in the objects recognition case and 35 in the faces recognition case.

Chapter 6

Classification Process

We chose to use a classifier which is based on the probability density functions (PDFs) of feature vectors. Each PDF is estimated from feature vectors belonging to one or more different views of an object from the dictionary. Thus, each dictionary object is represented by one or more PDFs. In all the simulations we examined the 3 fitting algorithms: Gradient1, RI-Min-Max and RI-Min-Var.

6.1 Objects Database

We divide the objects database into learning and test data sets in the following way: we have 9 different positions for each object ($\sim 10^o - 15^o$ difference between consecutive positions, we denote these positions by #1-#9), and for each position we have 5 different consecutive frames. We use even positions for learning (i.e., positions #2,#4,#6 and #8) and odd positions for testing (i.e., positions #1,#3,#5,#7 and #9).

6.1.1 Learning Positions

For each learning position of each object we do the following: each of the 5 frames is perturbed 10 times by adding colored Gaussian noise with standard deviation of 0.01 (according to the acquisition noise model we analyzed in Section 5.4, after round-off of the estimated standard deviation). Thus, we have $40 \cdot 4 \cdot 5 = 800$ real frames in the learning data set, synthetically enhanced 10 times to be $800 \cdot 10 = 8000$ frames. For each of the 8000 perturbed instances we perform all the pre-processing stages from the previous chapter and calculate the feature vector $\underline{v}$.

Let us denote each object by O_k , $k = 1, ..., 40$, and each learning position (view) by V_n, $n = 2, 4, 6, 8$. We examined several approaches:

Creating a PDF from All Learning Positions of the Same Object

In this approach, for each object we combined all the learning positions and used the $50 frames \cdot 4 positions = 200$ feature vectors of positions #2,#4,#6 and #8 in order to create a vector of means $\underline{\mu}$, and a covariance matrix Σ. These statistical characteristics were the basis for a multi-variable Gaussian probability density function in the following way:

$$P\left(\underline{v}/O_k\right) = \frac{1}{(2\pi)^{d/2}\left|\Sigma\right|^{1/2}} exp\left(-\frac{1}{2}\left(\underline{v}-\underline{\mu}\right)^T \Sigma^{-1}\left(\underline{v}-\underline{\mu}\right)\right) \tag{6.1}$$

Where d is the dimension of the features vector $\underline{v}$ (in case we use the entire feature vector we have $d = 31$), and $k = 1, ..., 40$.

In this approach we have 40 different PDFs for the description of 40 objects (one PDF per object).

Creating a PDF from Each Learning Position of Each Object

In this approach, for each object learning position (#2, #4, #6 and #8), the corresponding 50 feature vectors were used in order to create a separate PDF.

$$P\left(\underline{v}/O_k, V_n\right) = \frac{1}{(2\pi)^{d/2}\left|\Sigma\right|^{1/2}} exp\left(-\frac{1}{2}\left(\underline{v}-\underline{\mu}\right)^T \Sigma^{-1}\left(\underline{v}-\underline{\mu}\right)\right) \qquad (6.2)$$

Where $k = 1, ..., 40$, $n = 2, 4, 6, 8$, and the dimensions of the feature vectors are the same as in the previous approach.

In this approach we have $40 \cdot 4 = 160$ different PDFs for the description of 40 objects (4 PDFs per object).

Creating a PDF from Learning Position Pairs of Each Object

In this approach, we combined pairs of learning positions of each object for PDF creation. Specifically, for each object we used the $50 \cdot 2 = 100$ feature vectors of positions #2 and #4 for creation of one PDF, the 100 feature vectors of positions #4 and #6 for creation of a second PDF, and the 100 feature vectors of positions #6 and #8 for creation of a third PDF.

$$P\left(\underline{v}/O_k, V_n\right) = \frac{1}{(2\pi)^{d/2}\left|\Sigma\right|^{1/2}} exp\left(-\frac{1}{2}\left(\underline{v}-\underline{\mu}\right)^T \Sigma^{-1}\left(\underline{v}-\underline{\mu}\right)\right) \qquad (6.3)$$

Where $k = 1, ..., 40$, $n = 2\&4, 4\&6, 6\&8$, and the dimensions of the feature vectors are the same as in the previous approaches.

In this approach we have $40 \cdot 3 = 120$ different PDFs for the description of 40 objects (3 PDFs per object).

This approach assumes that positions with very different viewing angles

of the same object have very little in common, and therefore it will be better to learn similar positions together, but to separate the learning of different positions.

6.1.2 Testing Positions

We calculate the same feature vector for each of the 5 frames of each object testing position, resulting in $40 \cdot 5 \cdot 5 = 1000$ real frames in the testing data set. We then calculated the probability of it originating from each of the objects. The object which resulted in the highest probability was chosen as the best classification for the test vector.

Bayesian Approach

A test vector $\underline{v}$ is our observation, and we would like to find the probabilities that the object was O_k given that we observed $\underline{v}$:

$$P\left(O_k/\underline{v}\right) \ , \ k = 1, ..., 40 \tag{6.4}$$

The maximum probability will be chosen as the correct classification for the test vector. Using Bayes rule, we get:

$$P\left(O_k/\underline{v}\right) = \frac{P\left(\underline{v}/O_k\right) P\left(O_k\right)}{P\left(\underline{v}\right)} \tag{6.5}$$

Since $P\left(\underline{v}\right)$ is the same for every k, we can ignore it when we compare the different probabilities.

1^{st} Approach

Learning: Creating a PDF from all learning positions of the same object.

Testing: Comparing $P(O_k/\underline{v})$, $k = 1, ..., 40$.

In this approach, we have one PDF to describe each object, and we only need to compare the nominators of the probabilities (i.e., compare $P(\underline{v}/O_k)\,P(O_k)$, $k = 1, ..., 40$). We assume that all the objects have the same probability (i.e., they are uniformly distributed and $P(O_k) = \frac{1}{40}$), and so we can ignore the expression $P(O_k)$ as well. Thus, for the comparison of the probabilities $P(O_k/\underline{v})$, we need to compare only $P(\underline{v}/O_k)$, $k = 1, ..., 40$.

2^{nd} **Approach**

Learning: Creating a PDF from each learning position of each object.

Testing: Comparing $P(O_k/\underline{v})$, $k = 1, ..., 40$.

In this approach, we have 4 PDFs that describe each object. We expand the nominator of the probabilities:

$$P(\underline{v}/O_k)\,P(O_k) = \sum_{n=2,4,6,8} P(\underline{v}/O_k, V_n)\,P(V_n/O_k)\,P(O_k) \qquad (6.6)$$

Where $P(\underline{v}/O_k, V_n)$ is the probability that we observe $\underline{v}$, given that we had object O_k in position V_n, $P(V_n/O_k)$ is the probability that we observed position V_n given that we had object O_k, and $P(O_k)$ is the probability of object O_k. We assume that both $P(V_n/O_k)$ and $P(O_k)$ are uniformly distributed, since each object and each view has the same probability of occurrence, and so we set $P(V_n/O_k) = \frac{1}{4}$ and $P(O_k) = \frac{1}{40}$. Thus, we get that $P(\underline{v}/O_k)\,P(O_k) = \frac{1}{160} \sum_{n=2,4,6,8} P(\underline{v}/O_k, V_n)$. The $\frac{1}{160}$ factor can be ignored when we compare the probabilities, since it has the same value for every k. Therefore, for the comparison of the probabilities $P(O_k/\underline{v})$, we need to

compare only $\sum_{n=2,4,6,8} P\left(\underline{v}/O_k, V_n\right)$, $k = 1, ..., 40$.

3^{rd} **Approach**

Learning: Creating a PDF from each learning position of each object.

Testing: Comparing $P\left(O_k, V_n/\underline{v}\right)$, $k = 1, ..., 40$ and $n = 2, 4, 6, 8$.

In this approach, we have 4 PDFs that describe each object. But instead of calculating the probabilities of $P\left(O_k/\underline{v}\right)$ we calculate the probabilities of $P\left(O_k, V_n/\underline{v}\right)$ for each object and each learning position separately:

$$P\left(O_k, V_n/\underline{v}\right) = \frac{P\left(\underline{v}/O_k, V_n\right) P\left(O_k, V_n\right)}{P\left(\underline{v}\right)} = \frac{P\left(\underline{v}/O_k, V_n\right) P\left(V_n/O_k\right) P\left(O_k\right)}{P\left(\underline{v}\right)} \tag{6.7}$$

Again, $P\left(\underline{v}\right)$ is the same for every k, $P\left(V_n/O_k\right) = \frac{1}{4}$ and $P\left(O_k\right) = \frac{1}{40}$, and so we can ignore these expressions in the probabilities comparison and compare only $P\left(\underline{v}/O_k, V_n\right)$, $k = 1, ..., 40$ and $n = 2, 4, 6, 8$.

4^{th} **Approach**

Learning: Creating a PDF from learning position pairs of each object.

Testing: Comparing $P\left(O_k, V_n/\underline{v}\right)$, $k = 1, ..., 40$ and $n = 2\&4, 4\&6, 6\&8$.

In this approach, we assume that positions with very different viewing angles of the same object have very little in common, and therefore we want to compare the probabilities of each object O_k in each position combination V_n, given the observation $\underline{v}$ (instead of the general probability of O_k given $\underline{v}$):

$$P\left(O_k, V_n/\underline{v}\right),\ k = 1, 2, ..., 40,\ n = 2\&4, 4\&6, 6\&8 \tag{6.8}$$

Using Bayes rule, we get:

$$P\left(O_k, V_n/\underline{v}\right) = \frac{P\left(\underline{v}/O_k, V_n\right) P\left(O_k, V_n\right)}{P\left(\underline{v}\right)} = \frac{P\left(\underline{v}/O_k, V_n\right) P\left(V_n/O_k\right) P\left(O_k\right)}{P\left(\underline{v}\right)} \tag{6.9}$$

$P\left(\underline{v}\right)$ is the same for every k, and we assume that $P\left(O_k\right) = \frac{1}{40}$, and $P\left(V_n/O_k\right) = \frac{1}{3}$. Therefore, we can ignore these expressions in the probabilities comparison and compare only $P\left(\underline{v}/O_k, V_n\right)$, $k = 1, ..., 40$ and $n = 2\&4, 4\&6, 6\&8$.

6.2 Faces Database

We divide the faces database into learning and test databases in the following way: we have about ~ 50 different frames for each face (acquired at a rate of 2.5 frame/sec). We use the first 5 frames as learning frames and 5 other frames are used as test frames (the test frames are linearly spaced along the rest of the movie).

6.2.1 Learning Frames

For each face we do the following: each of the 5 learning frames is perturbed 10 times by adding colored Gaussian noise with standard deviation of 0.01 (according to the acquisition noise model we analyzed in Section 5.4, after round-off of the estimated standard deviation). Thus, we have $41 \cdot 5 = 205$ real frames in the learning data set, synthetically enhanced 10 times to be $205 \cdot 10 = 2050$ frames. For each of the 2050 perturbed instances we perform all the pre-processing stages from the previous chapter and calculate the feature vector $\underline{v}$.

Let us denote each face by O_k , $k = 1, ..., 41$.

For each face we used the 50 feature vectors of the learning frames in order to create a vector of means $\underline{\mu}$, and a covariance matrix Σ. These statistical characteristics were the basis for a multi-variable Gaussian probability density function as shown in Eq. (6.1).

In this case, the dimension of the features vector $\underline{v}$ (if we use the entire feature vector) is $d = 35$, and $k = 1, ..., 41$.

In this approach we have 41 different PDFs for the description of 41 faces (one PDF per face).

6.2.2 Testing Frames

We calculate the same feature vector for each of the 5 testing frames of each face, resulting in $41 \cdot 5 = 205$ frames in the testing data set. We then calculated the probability of it originating from each of the faces. The face which resulted in the highest probability was chosen as the best classification for the test vector.

6.2.2.1 Bayesian Approach

Following the same method as in Section 6.1.2 (only now, $k = 1, ..., 41$ since we have 41 faces), we use Bayes rule (again, ignoring $P(\underline{v})$). Since we use one PDF to describe each face, we only need to compare the nominators of the probabilities (i.e., compare $P(\underline{v}/O_k)\,P(O_k)$, $k = 1, ..., 41$). We assume that all faces have the same probability (i.e., $P(O_k) = \frac{1}{41}$), and so we can ignore $P(O_k)$ as well. Thus, for the comparison of the probabilities $P(O_k/\underline{v})$, we

need to compare only $P(\underline{v}/O_k)$, $k = 1, ..., 41$.

Chapter 7

Experimental Results

7.1 Objects Database Classification Results

Using the entire feature vector ($d = 31$) the results for each of the learning approaches appear in Table 7.1. The 4 different approaches refer to the learning/testing approaches we discussed in Section 6.1.2. It can be seen that the 4^{th} learning approach (creating a PDF from learning position pairs of each object) is the most successful. The advantage of an analysis of pairs of learning positions is that it takes into account the stability of classification features in different (yet close) positions. The 1^{st} approach performs badly since it attempts to learn all 4 positions together. Apparently, almost non of the features are stable after rotation of 90^o of an object (this is the average difference between the first and last positions in our database) since by such a change in the point of view, more than half the samples were replaced by new samples and therefore the joint analysis of the 4 positions is unsuccessful. The different fitting approaches have similar performance, but when using

the 4^{th} approach it can be seen that the Gradient1 fitting results are slightly better than those of the other fitting methods.

	Gradient1	RI-Min-Max	RI-Min-Var
1^{st} approach	17.0%	7.3%	12.6%
2^{nd} approach	75.4%	73.9%	75.4%
3^{rd} approach	96.5%	96.9%	95.7%
4^{th} approach	**98.8%**	98.1%	98.0%

Table 7.1: Results of objects recognition using different learning approaches

Using the 4^{th} learning approach, the analysis of the contribution of classification features based on 3D implicit polynomials only (invariants and fitting errors, $d = 16$) to the classification appears in Table 7.2. It can be seen that without the additional 2D/PCA classification features, the results are a little less good, but we still manage to classify correctly over 90% of the instances. Gradient1 fitting results are better than those of the other fitting methods in this case as well.

Gradient1	RI-Min-Max	RI-Min-Var
96.1%	94.8%	93.2%

Table 7.2: Results of objects recognition using the 4^{th} learning approach and only the classification features based on 3D implicit polynomials (invariants and fitting errors)

7.1.1 Comparison with Pose Estimation Results

We compared our method results with pose estimation classification results. The pose estimation methods we examined are PCA and IP based tensor approach [6]. In case of pose estimation using PCA, after the rotation of the data points, we fit an IP and the entire IP coefficients are our features. In case of pose estimation using IP tensors, we tried two approaches: the first

- rotation of the leading form and using its coefficients as features, and the second - rotation of the data-points, re-fitting an IP and using the entire IP coefficients as features.

We then use an l_2 distance between feature vectors for classification. The IP degree we chose for classification is 4 ($d = 35$), since higher degree polynomial coefficients are less stable and showed poor classification performance. Following the pose estimation results shown in Section 3.4.2, we used IP based pose estimation using a 2^{nd} degree IP and a 4^{th} degree IP.

The results using Gradient1 appear in Table 7.3. Using a different IP fitting algorithm (Min-Max or Min-Var) didn't affect the results. It can be seen that the PCA outperforms the 4^{th} degree IP pose estimation, but has a similar performance to the 2^{nd} degree IP pose estimation. When using the 4^{th} degree IP based pose estimation with the 4^{th} degree leading form coefficients as features, the results are better than when using the entire 4^{th} degree IP coefficients as features with the same pose method. The reason is that the leading form coefficients are invariant to translation (see Section 3.4.2). Therefore, when the pose estimation is based on the leading form coefficients, they are more stable than the rest of the IP coefficients after rotation. However, this is not true when the pose estimation is based on PCA or on a different IP degree, and therefore in these cases the classification is better when using the entire IP coefficients as features.

It can also be seen that our method results in Table 7.1 (98.8%) are better than the PCA/IP based pose estimation methods that appear Table 7.3. We compare the computational complexity of the two methods in Section 7.1.3.

Pose Method	IP Degree Fit	Results (entire IP coef.)	Results (leading form coef.)
PCA	4^{th} degree IP	91.6%	82.8%
4^{th} degree IP	4^{th} degree IP	58.9%	79.4%
2^{nd} degree IP	4^{th} degree IP	90.5%	78.5%

Table 7.3: Results of objects recognition using pose estimation

7.1.2 Comparison with Shape Spectrum Descriptor (SSD)

We compared the performance of our method with the Shape Spectrum Descriptor (SSD) technique [16], which is described briefly in Appendix E. This technique was adopted by the MPEG-7 standard for 3D descriptors and has a relatively low complexity. We used a histogram of 25 bins for the descriptor (the default for MPEG-7 is 100 bins, but its results are a little worse) and we also used the singular and planar descriptors (i.e., we had 27 features, see detailed explanation in Appendix E). We used l_1 norm on the difference between the feature vectors for classification.

The results of this comparison appear in Table 7.4. It can be seen that our method has better performance. We compare the computational complexity of the two methods in the next section.

Our Method Results (Gradient1, $d = 31$)	SSD Results ($d = 27$)
98.8%	90.1%

Table 7.4: Comparison between our method and the SSD method for object recognition

7.1.3 Comparison of Computational Complexity

All 3 methods, our method, pose estimation based techniques and the SSD technique, have *a similar computational complexity*. If we denote the number of object data-points by N, then each of the stages of each method has a complexity of $O(N)$ (The SSD stages are actually dependent on the number of triangles, which in our databases is around $2N$ for an object with N data-points). The constant that multiplies N is large in some stages (such as the Least-Squares IP fit in our case and in the pose estimation case, or the 2^{nd} degree explicit polynomial fit at each point in the SSD case), but is difficult to estimate. Average running times of classification of a single object using matlab implementation appear in Table 7.5. We conclude that our method is both faster and has better results on the objects database than the other two methods.

our method (Gradient1, $d = 31$)	pose estimation (PCA/4^{th}degree IP, $d = 35$)	SSD ($d = 27$)
13Sec	15Sec	55Sec

Table 7.5: Average running times of classification of a single object

7.2 Faces Database Classification Results

Using the entire feature vector ($d = 35$) the results appear in Table 7.6. It can be seen that in this case as well, the Gradient1 fitting based features have the best classification performance.

Gradient1	RI-Min-Max	RI-Min-Var
97.1%	96.6%	93.7%

Table 7.6: Results of faces recognition

The analysis of the contribution of classification features based on 3D implicit polynomials only (invariants, $d = 18$) to the classification appears in Table 7.7. It can be seen that without the additional 2D/PCA classification features, the results are less good, but we still manage to classify correctly around 90% of the instances. Note that Gradient1 fitting results are now a little worse than those of the other fitting methods. We conclude that the 3D classification features are more informative in the cases of Min-Max and Min-Var, compared with Gradient1.

Gradient1	RI-Min-Max	RI-Min-Var
89.3%	91.7%	91.7%

Table 7.7: Results of faces recognition using only the classification features based on 3D implicit polynomials (invariants)

7.2.1 Comparison with Pose Estimation Results

We compared our results with pose estimation classification results. The pose estimation methods we examined are PCA and IP based tensor approach in the same way described in Section 7.1.

The results using Gradient1 appear in Table 7.8. Using a different IP fitting algorithm (Min-Max or Min-Var) didn't affect the results. It can be seen that again, the PCA outperforms the 4^{th} degree IP pose estimation, but has a similar performance to the 2^{nd} degree IP pose estimation.

It can also be seen that our method results using Gradient1 (97.1%) are better than the PCA/IP based pose estimation methods that appear in the table.

Pose Method	IP Degree Fit	Results (entire IP coef.)	Results (leading form coef.)
PCA	4^{th} degree IP	93.2%	88.3%
4^{th} degree IP	4^{th} degree IP	71.2%	81.5%
2^{nd} degree IP	4^{th} degree IP	92.7%	89.3%

Table 7.8: Results of faces recognition using pose estimation

7.2.2 Comparison with Shape Spectrum Descriptor (SSD)

We compared the performance of our method with the Shape Spectrum Descriptor (SSD) technique [16] which is described briefly in Appendix E. We used the same features and norm as in Section 7.1.2.

The results of this comparison appear in Table 7.9. It can be seen that our method has a much better performance for the examined database.

Our Method Results (Gradient1, $d = 35$)	SSD Results ($d = 27$)
97.1%	72.2%

Table 7.9: Comparison between our method and the SSD method for faces recognition

Chapter 8

Summary and Future Work

8.1 Summary

In this work we examined the description and classification abilities of 3D implicit polynomials using existing fitting algorithms (Gradient1, Min-Max and Min-Var). We developed a set of 3D linear rotation invariants which is based on implicit polynomials and their tensor representation, and 2 more quadratic invariants based on implicit polynomials and trigonometric identities. The pre-processing we apply to the data has a considerable influence on the classification performance. Our pre-processing includes translation (locating the center of mass at the origin), scaling (the 75^{th} percentile of the data-points distances from the origin was the scaling factor), robust normal direction calculation (using a local plane fitting), mirroring with respect to an appropriate plane (in case of the faces database) and preparing 2D projections for obtaining extra classification features (xy projection for the rigid objects, $xz\&yz$ projections for the mirrored faces). Following the 2D

IP classification method known as Multi Order and Fitting Error Technique (MOFET) [3] we created a feature vector containing 3D IP rotation invariants and fitting errors, 2D IP rotation invariants and fitting errors (from the most descriptive 2D projections) and the eigenvalues of a PCA decomposition. We used various IP degrees (both for 2D and 3D) in order to utilize both the stability of low degree polynomials and the descriptiveness of high degree polynomials. We designed a probability density function (PDF) based classifier and examined its performance for various learning and testing approaches.

We showed that the 3D IP invariants and fitting error approach has better performance for objects and faces classification compared with pose estimation methods followed by IP fitting [6]. We also found in our tests that our method outperforms the Shape Spectrum Descriptor (SSD) technique [16], which was adopted by the MPEG-7 standard for 3D descriptors.

8.2 Main Original Contributions

In this section we summarize the most significant contributions of this work.

3D Rotation Invariants

- Developing a set of linear rotation invariants based on a tensor representation of the implicit polynomials forms.
- Developing 2 additional quadratic rotation invariants based on implicit polynomials and trigonometric identities.

Rotation Invariant 3D IP Fitting Algorithms

- Adjusting existing IP fitting algorithms (Min-Max and Min-Var) to be rotation invariant.

Mirroring of 3D Objects for IP stabilization

- Suggesting the mirroring of 3D objects and showing that it provides more stable IP coefficients.

3D Objects/Faces Classification Method

- Designing a classifier based on both 3D and 2D implicit polynomials rotation invariant representation and achieving good performance for both rigid objects classification and faces classification (in a cooperative situation).

8.3 Future Work

Dimensionality Reduction for Feature Selection

The classification features were chosen for each application (objects/faces recognition) based on their robustness and informativeness in each case. In order to automatically choose the best classification features for a certain application, it will be interesting to explore dimensionality reduction with known methods (such as PCA or LDA - Linear Discriminant Analysis).

3D Objects Retrieval

The implicit polynomials might be useful for retrieving three-dimensional

objects similar to a given one from a database (classification into categories).

Additional Rotation Invariants

Using quaternions for the derivation of a full set of quadratic rotation invariants, in a similar way to the use of complex representation in the 2D case.

Appendix A

Description of 3D Objects Using High Degree Implicit Polynomials

When the object we would like to describe is complex and has fine details, low degree polynomial fitting will give only a general outline of the object. By using higher degrees, we can get a more accurate description. Previous works dealing with 3D polynomial fitting have used polynomials up to 12^{th} degree [6]. Using each of the Least Squares fitting algorithms, we get a solution of the form as in (2.18), repeated here for convenience:

$$\underline{a}_{LS} = \left(M^T M\right)^{-1} M^T \underline{b} \tag{A.1}$$

When trying to apply polynomial fitting of degrees 18^{th} and higher, the matrix $M^T M$ is usually singular and therefore its inverse $\left(M^T M\right)^{-1}$ does

not exist. In these situations, the least squares solution to the linear system

$$M\underline{a} = \underline{b} \tag{A.2}$$

is not unique. A null vector of M is a nonzero solution to

$$M\underline{x} = \underline{0} \tag{A.3}$$

Any multiple of any null vector can be added to $\underline{a}$ without changing how well $M\underline{a}$ approximates $\underline{b}$.

In order to choose one possible solution, we used the Moore-Penrose pseudoinverse (computed by the matlab command *'pinv'*). This pseudoinverse solution minimizes not only the l_2 norm of $\|M\underline{a} - \underline{b}\|$, but also the Frobenius norm of $pinv\,(M)$, where the Frobenius norm of a matrix is defined by $\|M\|_F = \left(\sum_i \sum_j m_{ij}^2\right)^{1/2}$. These minimization properties define a unique pseudoinverse even if M is rank deficient.

If M is square and non-singular, then the pseudoinverse and the inverse are the same (i.e., $pinv\,(M) = M^{-1}$). If M is not square but has full rank, then the pseudoinverse gives the regular Least Squares solution (i.e., $pinv\,(M) = \left(M^T M\right)^{-1} M^T$). But even if M is not square and singular, we get a unique solution to our least squares problem:

$$\underline{a}_{LS} = pinv\,(M) \cdot \underline{b} \tag{A.4}$$

Fig. A.1 shows various degree polynomials fitting (using Gradient1) applied to a 3D face using Moore-Penrose pseudoinverse. In order to demon-

strate the fitting performance, in these images we show only the zero-set points that are in the face area, and ignore the spurious zero-set points.

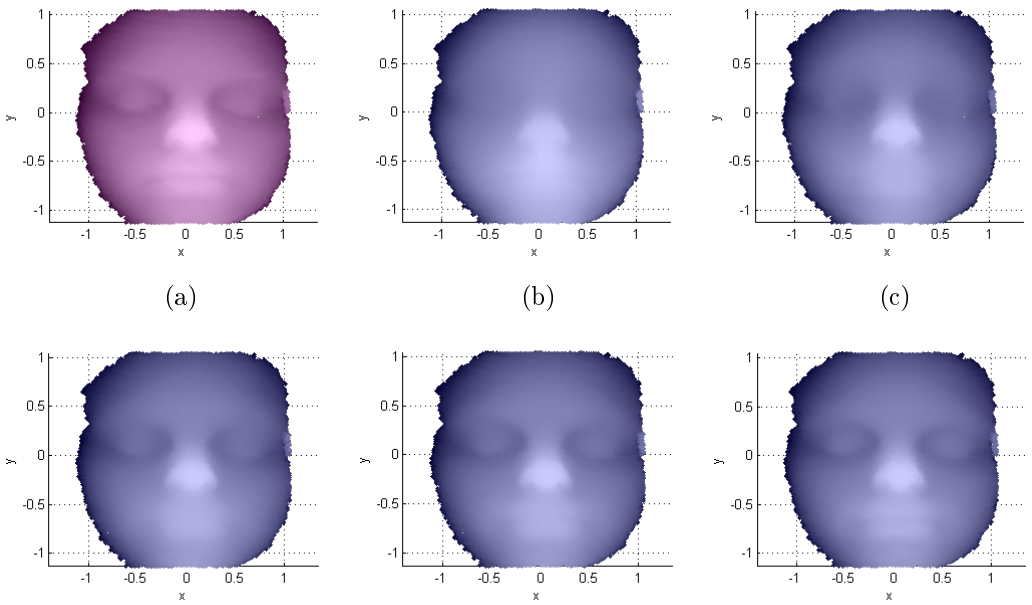


Figure A.1: An example to various degree polynomials fitting (using Gradient1) to a 3D face: (a) the original data points (b) IP fit of 6^{th} degree (c) IP fit of 10^{th} degree (d) IP fit of 14^{th} degree (e) IP fit of 18^{th} degree (f) IP fit of 22^{th} degree

Appendix B

Tensors and Implicit Polynomials Representation

A tensor is a generalized linear representation. It can be considered as a multi-dimensional array, where its rank is the number of array indices required to describe such a quantity. A tensor of rank 0 is a scalar, a tensor of rank 1 is a vector, a tensor of rank 2 is a matrix and so on.

Let us examine an IP form of degree r:

$$H_r\left(x_1, x_2, x_3\right) = \sum_{k+l+m=r} a_{klm} x_1^k x_2^l x_3^m \tag{B.1}$$

where k, l, m are non-negative integers. It includes all the combinations of powers of x_1, x_2 and x_3 which their sum is equal to r, and so it can be expressed as a tensor of rank r:

$$\left(s^{i_1 i_2 \ldots i_r}\right)_{1 \leq i_1, i_2, \ldots, i_r \leq 3} \tag{B.2}$$

and we can represent the IP form in the following way:

$$H_r(x_1, x_2, x_3) = \sum_{i_1=1}^{3} \sum_{i_2=1}^{3} ... \sum_{i_r=1}^{3} s^{i_1 i_2 ... i_r} x_{i_1} x_{i_2} ... x_{i_r} \tag{B.3}$$

Each index i_k, $k = 1, ..., r$, can accept the values 1,2 and 3, each corresponds to one of the variables x_1, x_2 and x_3, respectively. Therefore, we have a multi-dimensional array of 3^r entries.

For $r = 2$ we have the following form:

$$H_2(x_1, x_2, x_3) = a_{200}x_1^2 + a_{020}x_2^2 + a_{002}x_3^2 + a_{110}x_1x_2 + a_{101}x_1x_3 + a_{011}x_2x_3 \tag{B.4}$$

The possible power combinations of the variables x_1, x_2 and x_3 in this case are $x_{i_1}x_{i_2}$ where $1 \leq i_1 \leq 3$ and $1 \leq i_2 \leq 3$.

We can rewrite (B.4) as:

$$\begin{aligned} H_2(x_1, x_2, x_3) = a_{200}x_1^2 + \tfrac{a_{110}}{2}x_1x_2 + \tfrac{a_{101}}{2}x_1x_3 + \tfrac{a_{110}}{2}x_2x_1 + \\ + a_{020}x_2^2 + \tfrac{a_{011}}{2}x_2x_3 + \tfrac{a_{101}}{2}x_3x_1 + \tfrac{a_{011}}{2}x_3x_2 + a_{002}x_3^2 \end{aligned} \tag{B.5}$$

and so we can construct the following tensor of rank 2 with $3^2 = 9$ elements:

$$S_2 = s^{i_1 i_2} = \begin{bmatrix} a_{200} & \frac{a_{110}}{2} & \frac{a_{101}}{2} \\ \frac{a_{110}}{2} & a_{020} & \frac{a_{011}}{2} \\ \frac{a_{101}}{2} & \frac{a_{011}}{2} & a_{002} \end{bmatrix} \tag{B.6}$$

Note that each entry $s^{i_1 i_2}$ corresponds to one of the possible powers combinations of x_1, x_2 and x_3.

A generalization of this case for any form of degree $r \in \mathbb{N}$ is a tensor of

rank r, with entries $s^{i_1 i_2 \ldots i_r} = \frac{a_{klm}}{\frac{r!}{k!l!m!}}$, and we can represent the form in the following way:

$$H_r(x_1, x_2, x_3) = \sum_{i_1=1}^{3} \sum_{i_2=1}^{3} \ldots \sum_{i_r=1}^{3} s^{i_1 i_2 \ldots i_r} x_{i_1} x_{i_2} \ldots x_{i_r} \tag{B.7}$$

Appendix C

Quaternions Properties

We include here 2 proofs for quaternions properties, which appear in [12].

C.1 Vector Rotation Using Unit Quaternions

When referring to Eq. (3.26), repeated here for convenience:

$$r' = qr\bar{q} \tag{C.1}$$

we claimed that when using a unit quaternion q, if r is purely imaginary, then r' will be purely imaginary as well. We now prove this property:

The representation of q for matrix multiplication (discussed in Section 3.5.1.2) is denoted by R_q. It can be readily seen that the appropriate representation for $\overline{q}$ is then R_q^T. Using (3.25) we get:

$r' = qr\bar{q} = (R_q r)\,\bar{q} = \overline{R}_q^T\,(R_q r) = \left(\overline{R}_q^T R_q\right) r$

Expanding $\left(\overline{R}_q^T R_q\right)$ we get:

$$\overline{R}_q^T R_q = \begin{bmatrix} q \cdot q & 0 & 0 & 0 \\ 0 & \left(q_0^2 + q_x^2 - q_y^2 - q_z^2\right) & 2\left(q_x q_y - q_0 q_z\right) & 2\left(q_x q_z + q_0 q_y\right) \\ 0 & 2\left(q_y q_x + q_0 q_z\right) & \left(q_0^2 - q_x^2 + q_y^2 - q_z^2\right) & 2\left(q_y q_z - q_0 q_x\right) \\ 0 & 2\left(q_z q_x - q_0 q_y\right) & 2\left(q_z q_y + q_0 q_x\right) & \left(q_0^2 - q_x^2 - q_y^2 + q_z^2\right) \end{bmatrix} \tag{C.2}$$

When q is a unit quaternion, $q \cdot q = 1$. Thus, when multiplying this matrix by a quaternion r, the first element is not affected by the multiplication. In case of a purely imaginary quaternion r with $r_0 = 0$, after rotation we get $r'_0 = 0$, and so r' is purely imaginary as well.

C.2 Quaternions Dot-Products

Let us first examine the following quaternion dot-product:

$$(qp) \cdot (qr) = (R_q p) \cdot (R_q r) = (R_q p)^T (R_q r) = p^T R_q^T R_q r = p^T (q \cdot q) Ir$$

We conclude that $(qp) \cdot (qr) = (q \cdot q)(p \cdot r)$.

In the special case where q is a unit quaternion, it follows that:

$$(pq) \cdot r = (pq) \cdot (r\overline{q}q) = (p \cdot (r\overline{q}))(q \cdot q) = p \cdot (r\overline{q}).$$

And so we proved Eq. (3.30): $(pq) \cdot r = p \cdot (r\overline{q})$

Appendix D

Proof for the Expressions of the 3D Quadratic Invariants

D.1 2D Quadratic Invariants

2D Quadratic invariants are derived using a complex representation [6]. The first 2 invariants are:

$$Q_{2D,1} = a_{10}^2 + a_{01}^2 \tag{D.1}$$

$$Q_{2D,2} = (a_{20} - a_{02})^2 + a_{11}^2 = a_{20}^2 + a_{02}^2 - 2a_{20}a_{02} + a_{11}^2 \tag{D.2}$$

D.2 Derivation of 3D Quadratic Invariants

When using the same expressions pattern as in 2D, we get the following expressions:

$$Q_{3D,1} = a_{100}^2 + a_{010}^2 + a_{001}^2 \tag{D.3}$$

$$Q_{3D,2} = a_{200}^2 + a_{020}^2 + a_{002}^2 - 2a_{200}a_{020} - 2a_{200}a_{002} - 2a_{020}a_{002} + a_{110}^2 + a_{101}^2 + a_{011}^2 \tag{D.4}$$

We will now use trigonometric identities in order to prove that these 2 expressions are indeed invariant to rotation.

D.2.1 Representation of Rotation

As shown in Section 4.2.1, every 3D rotation can be considered as 3 rotations, one around each axis (x, y and z) with angles α, β and γ respectively.

The rotation matrices, given in (4.19)-(4.18), are repeated here for convenience:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{D.5}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{D.6}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{D.7}$$

If we prove that $Q_{3D,1}$ and $Q_{3D,2}$ do not change under any of the rotations (around each axis), then they are rotation invariants.

D.2.2 Proof for the First 3D Quadratic Invariant

We prove here for rotation around the z axis, the proofs for the other rotations can be done in a similar way.

After rotation around the z axis, the new coordinates are given by (D.7). After inverting the rotation matrix, we get:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \tag{D.8}$$

Where $c \triangleq cos\gamma$ and $s \triangleq sin\gamma$.

The first degree form (from which we would like to derive $Q_{3D,1}$), is:

$$H_1(x, y, z) = a_{100}x + a_{010}y + a_{001}z \tag{D.9}$$

Substituting D.8 into D.9, we get:

$$\begin{aligned} H_1(x', y', z') &= a_{100}(cx' + sy') + a_{010}(-sx' + cy') + a_{001}z' = \\ &= \underbrace{(ca_{100} - sa_{010})}_{b_{100}} x' + \underbrace{(sa_{100} + ca_{010})}_{b_{010}} y' + \underbrace{a_{001}}_{b_{001}} z' \end{aligned}$$

Where b_{100}, b_{010} and b_{001} are the new IP coefficients after rotation around the z axis.

We will now prove that $b_{100}^2 + b_{010}^2 + b_{001}^2$ is equal to $a_{100}^2 + a_{010}^2 + a_{001}^2$,

which means that $Q_{3D,1}$ is invariant to rotation around z axis:

$$\begin{aligned} b_{100}^2 + b_{010}^2 + b_{001}^2 &= (ca_{100} - sa_{010})^2 + (sa_{100} + ca_{010})^2 + a_{001}^2 = \\ &= a_{100}^2 \left(c^2 + s^2\right) + a_{010}^2 \left(s^2 + c^2\right) + 2a_{100}a_{010} \left(-2cs + 2cs\right) + a_{001}^2 = \\ &= a_{100}^2 + a_{010}^2 + a_{001}^2 \end{aligned}$$

The same proof can be repeated with the new coordinates after rotation around x axis, and after rotation around y axis.

D.2.3 Proof for the Second 3D Quadratic Invariant

We prove for rotation around the z axis, the proofs for the other rotations proofs can be done in a similar way.

We use the same notation as in the previous section.

The second degree form (from which we would like to derive $Q_{3D,2}$), is:

$$H_2(x, y, z) = a_{200}x^2 + a_{020}y^2 + a_{002}z^2 + a_{110}xy + a_{101}xz + a_{011}yz \quad \text{(D.10)}$$

Substituting D.8 into D.10, we get:

$$\begin{aligned}
H_2(x',y',z') &= a_{200}(cx'+sy')^2 + a_{020}(-sx'+cy')^2 + a_{002}z'^2 + \\
&+ a_{110}(cx'+sy')(-sx'+cy') + a_{101}(cx'+sy')z' + a_{011}(-sx'+cy')z' = \\
&= \underbrace{\left(c^2a_{200}+s^2a_{020}-csa_{110}\right)}_{b_{200}}x'^2 + \underbrace{\left(s^2a_{200}+c^2a_{020}+csa_{110}\right)}_{b_{020}}y'^2 + \\
&+ \underbrace{a_{002}}_{b_{002}}z'^2 + \underbrace{\left(2csa_{200}-2csa_{020}+\left(c^2-s^2\right)a_{110}\right)}_{b_{110}}x'y' + \\
&+ \underbrace{\left(ca_{101}-sa_{011}\right)}_{b_{101}}x'z' + \underbrace{\left(sa_{101}+ca_{011}\right)}_{b_{011}}y'z'
\end{aligned}$$

Where b_{200}, b_{020}, b_{002}, b_{110}, b_{101} and b_{011} are the new IP coefficients after rotation around the z axis.

We will now prove that $b_{200}^2+b_{020}^2+b_{002}^2-2b_{200}b_{020}-2b_{200}b_{002}-2b_{020}b_{002}+b_{110}^2+b_{101}^2+b_{011}^2$ is equal to $a_{200}^2+a_{020}^2+a_{002}^2-2a_{200}a_{020}-2a_{200}a_{002}-2a_{020}a_{002}+a_{110}^2+a_{101}^2+a_{011}^2$, which means that $Q_{3D,2}$ is invariant to rotation around z axis:

$$\begin{aligned}
b_{200}^2 &= \left(c^2a_{200}+s^2a_{020}-csa_{110}\right)^2 = \\
&= c^4a_{200}^2 + s^4a_{020}^2 + c^2s^2a_{110}^2 + 2c^2s^2a_{200}a_{020} - 2c^3sa_{200}a_{110} - 2cs^3a_{020}a_{110}
\end{aligned}$$

$$\begin{aligned}
b_{020}^2 &= \left(s^2a_{200}+c^2a_{020}+csa_{110}\right)^2 = \\
&= s^4a_{200}^2 + c^4a_{020}^2 + c^2s^2a_{110}^2 + 2c^2s^2a_{200}a_{020} + 2cs^3a_{200}a_{110} + 2c^3sa_{020}a_{110}
\end{aligned}$$

$$b_{002}^2 = a_{002}^2$$

$$\begin{aligned}-2b_{200}b_{020} &= -2\left(c^2a_{200}+s^2a_{020}-csa_{110}\right)\left(s^2a_{200}+c^2a_{020}+csa_{110}\right)= \\ &= -2[c^2s^2a_{200}^2+c^2s^2a_{020}^2-c^2s^2a_{110}^2+\left(c^4+s^4\right)a_{200}a_{020}+ \\ &+\left(c^3s-cs^3\right)a_{200}a_{110}+\left(cs^3-c^3s\right)a_{020}a_{110}]\end{aligned}$$

$$-2b_{200}b_{002} = -2\left(c^2a_{200}+s^2a_{020}-csa_{110}\right)a_{002}$$

$$-2b_{020}b_{002} = -2\left(s^2a_{200}+c^2a_{020}+csa_{110}\right)a_{002}$$

$$\begin{aligned}b_{110}^2 &= \left(2csa_{200}-2csa_{020}+\left(c^2-s^2\right)a_{110}\right)^2= \\ &= 4c^2s^2a_{200}^2+4c^2s^2a_{020}^2+\left(c^2-s^2\right)^2a_{110}^2-8c^2s^2a_{200}a_{020}+ \\ &+4cs\left(c^2-s^2\right)a_{200}a_{110}-4cs\left(c^2-s^2\right)a_{020}a_{110}\end{aligned}$$

$$b_{101}^2 = \left(ca_{101}-sa_{011}\right)^2 = c^2a_{101}^2+s^2a_{011}^2-2csa_{101}a_{011}$$

$$b_{011}^2 = \left(sa_{101}+ca_{011}\right)^2 = s^2a_{101}^2+c^2a_{011}^2+2csa_{101}a_{011}$$

And so we get:

$$b_{101}^2+b_{011}^2 = \left(c^2+s^2\right)a_{101}^2+\left(s^2+c^2\right)a_{011}^2+a_{101}a_{011}\left(-2cs+2cs\right) = a_{101}^2+a_{011}^2$$

$$\begin{aligned}-2b_{200}b_{002}-2b_{020}b_{002} &= -2a_{200}a_{002}\left(c^2+s^2\right)-2a_{020}a_{002}\left(s^2+c^2\right)+a_{110}a_{002}\left(-cs+cs\right)= \\ &= -2a_{200}a_{002}-2a_{020}a_{002}\end{aligned}$$

$$
\begin{aligned}
b_{200}^2 + b_{020}^2 + b_{110}^2 - 2b_{200}b_{020} \quad &= \\
&= a_{200}^2 \left(c^4 + s^4 - 2c^2s^2 + 4c^2s^2\right) + \\
&+ a_{020}^2 \left(s^4 + c^4 - 2c^2s^2 + 4c^2s^2\right) + \\
&+ a_{110}^2 \left(2c^2s^2 + 2c^2s^2 + \left(c^2 - s^2\right)^2\right) - \\
&- 2a_{200}a_{020} \left(-2c^2s^2 + c^4 + s^4 + 4c^2s^2\right) + \\
&+ 2a_{200}a_{110} \left(-c^3s + cs^3 - c^3s + cs^3 + 2c^3s - 2cs^3\right) + \\
&+ 2a_{020}a_{110} \left(-cs^3 + c^3s - cs^3 + c^3s - 2c^3s + 2cs^3\right) = \\
&= \left(a_{200}^2 + a_{020}^2 + a_{110}^2 - 2a_{200}a_{020}\right) \left(c^2 + s^2\right)^2 = \\
&= a_{200}^2 + a_{020}^2 + a_{110}^2 - 2a_{200}a_{020}
\end{aligned}
$$

$$
b_{002}^2 \quad = \quad a_{002}^2
$$

Adding up all the partial expressions we get

$$
\begin{aligned}
&b_{200}^2 + b_{020}^2 + b_{002}^2 - 2b_{200}b_{020} - 2b_{200}b_{002} - 2b_{020}b_{002} + b_{110}^2 + b_{101}^2 + b_{011}^2 = \\
&a_{200}^2 + a_{020}^2 + a_{002}^2 - 2a_{200}a_{020} - 2a_{200}a_{002} - 2a_{020}a_{002} + a_{110}^2 + a_{101}^2 + a_{011}^2.
\end{aligned}
$$

The same proof can be repeated with the new coordinates after rotation around x axis, and after rotation around y axis.

Appendix E

Shape Spectrum Descriptor (SSD)

This is the descriptor that was adopted by MPEG-7 as the 3D descriptor. It is described in detail in [16] and is based on [22].

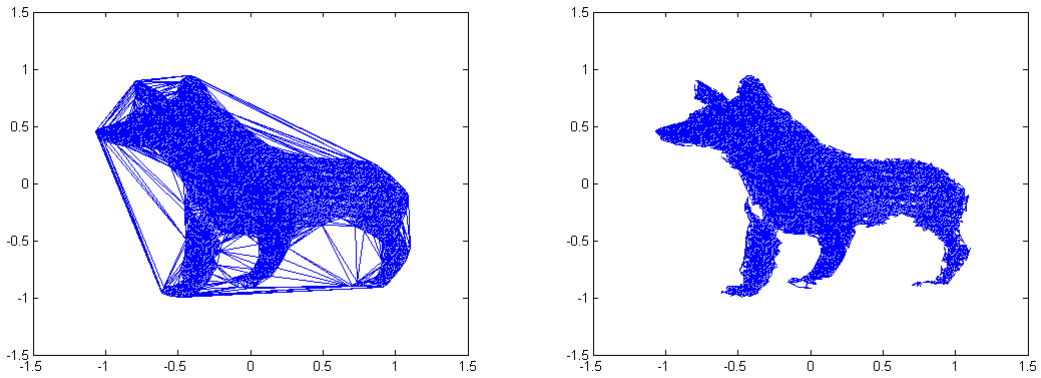
The input is a cloud of points and the triangles T_p, $p = 1, ..., N$ that connect them. For each triangle, we calculate the shape index:

$$I_p = \frac{1}{2} - \frac{1}{\pi} atan \frac{\kappa_1^p + \kappa_2^p}{\kappa_1^p - \kappa_2^p} \tag{E.1}$$

where κ_1^p and κ_2^p are the principal curvatures of the surface at this area and $\kappa_1^p \geq \kappa_2^p$. Note that the shape index is always in the range $[0, 1]$. The Shape Spectrum Descriptor is the histogram of all the shape indices.

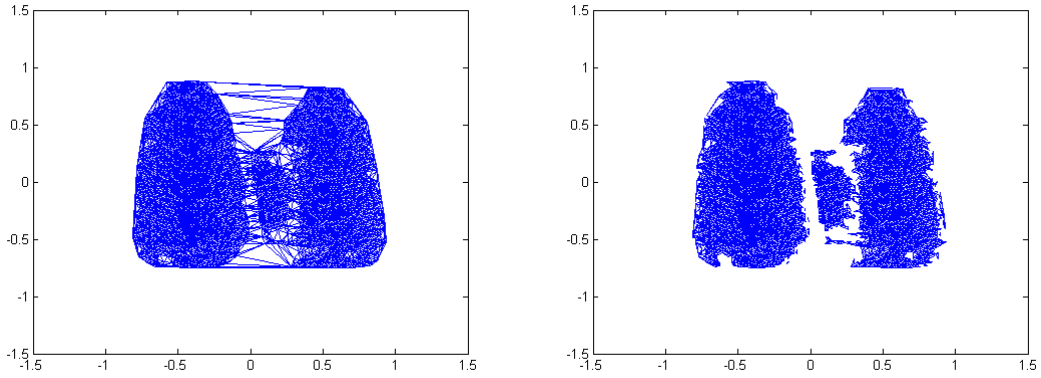
Since each object in our database contains only the cloud of points, we used 2D Delaunay triangulation [23] to create the connecting triangles and disposed of the large triangles which connected far data-points which were

not supposed to be connected (see Fig. E.1 and Fig. E.2).



(a) (b)

Figure E.1: 'Fox' triangulation (a) before and (b) after disposing of large triangles



(a) (b)

Figure E.2: 'Binocular' triangulation (a) before and (b) after disposing of large triangles

We calculated the area of each triangle S_p and its normal n_p, and find the adjacent triangles (Two triangles of the mesh are said to be adjacent if and only if they share a common vertex). We note the set of adjacent triangles by

$$\Lambda_p = \{i | T_i \, adjacent \, to \, T_p\} \tag{E.2}$$

and denote the sum of adjacent triangles area by σ_p:

$$\sigma_p = S_p + \sum_{i \in \Lambda_p} S_i \tag{E.3}$$

If the number of triangles adjacent to T_p (the cardinality of Λ_p) is less than 5, then the area σ_p is regarded as the area of a singular surface. If the triangle is not singular we perform the following:

1. We calculate the mean normal of each triangle using the adjacent triangles:

$$N_p = \sum_{k \in \Lambda_p} w_k n_k \tag{E.4}$$

   where the weights w_k are the areas S_k of these triangles. The mean normal is scaled to have a unit norm:

$$\widetilde{N}_p = \frac{N_p}{\|N_p\|_2} \tag{E.5}$$

2. We find the center of gravity of each triangle $g_p = (x_p, y_p, z_p)$.

3. We define a new Cartesian coordinate system so that origin coincides with the center of gravity g_p of the triangle T_p and the mean normal $\widetilde{N}_p$ is taken as the z axis. We express the centers of gravity of the adjacent triangles in the new coordinate system $G_p = \{g_i' | i \in \Lambda_p\}$.

4. We perform local parametric surface fitting around T_p using a 2^{nd} degree explicit polynomial:

$$z = f(x, y) = a_0 x^2 + a_1 y^2 + a_2 xy + a_3 x + a_4 y + a_5 \tag{E.6}$$

$$a = (a_0, a_1, a_2, a_3, a_4, a_5)^T \tag{E.7}$$

$$b(x, y) = \left(x^2, y^2, xy, x, y, 1\right)^T \tag{E.8}$$

The optimal fit of the quadratic surface through the points $G_p = \{g_i' | i \in \Lambda_p\}$, which minimizes the mean square error, is computed by:

$$\hat{a} = \left[\sum_{i=1}^{M} w_i b(x_i, y_i)\, b^T(x_i, y_i)\right]^{-1} \left[\sum_{i=1}^{M} w_i z_i b(x_i, y_i)\right] \tag{E.9}$$

where the weights w_i are the areas of the triangles. Note that the representation of the gravity centers in local coordinates guarantees the invariance of the approximation with respect to Euclidean transforms.

5. The principal curvatures are defined as the eigenvalues of the Weingarten map (W) given by the following expression:

$$W = I^{-1} II \tag{E.10}$$

where I and II denote the first and the second fundamental differential forms, respectively. If $z = f(x, y)$ is twice differentiable, we get:

$$I = \begin{pmatrix} 1 + f_x^2 & f_x f_y \\ f_x f_y & 1 + f_y^2 \end{pmatrix} \tag{E.11}$$

$$II = \frac{1}{\sqrt{1 + f_x^2 + f_y^2}} \begin{pmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{pmatrix} \tag{E.12}$$

and in case we have a quadratic surface with parameters $a = (a_0, a_1, a_2, a_3, a_4, a_5)^T$,

we can calculate I and II at $(x, y) = (0, 0)$ in the following way:

$$I = \begin{pmatrix} 1 + a_3^2 & a_3 a_4 \\ a_3 a_4 & 1 + a_4^2 \end{pmatrix} \tag{E.13}$$

$$II = \frac{1}{\sqrt{1 + a_3^2 + a_4^2}} \begin{pmatrix} a_0 & a_2 \\ a_2 & a_1 \end{pmatrix} \tag{E.14}$$

6. A surface is declared as planar if the following condition is fulfilled:

$$\sigma_p \sqrt{\left(\kappa_1^p\right)^2 + \left(\kappa_2^p\right)^2} < T \tag{E.15}$$

where T is a given threshold ($0.1 \leq T \leq 0.4$).

After all the shape indices are calculated, we create a histogram of H bins (the default for MPEG7 is $H = 100$) in the range $[0, 1]$ and two more variables: *Singular* and *Planar*. We calculate the sum of all triangle areas: $S = \sum_{p=1}^{N} S_p$.

For each triangle T_p, $p = 1, ..., N$ we perform the following:

1. If it is singular, we increase *Singular* by σ_p / S.

2. Otherwise, if it is planar, we increase *Planar* by σ_p / S.

3. Otherwise, we find the appropriate histogram bin, and increase it by σ_p / S.

Eventually, we normalize the histogram to have a sum of 1. We use the normalized histogram together with the variables *Singular* and *Planar* as our feature vector for classification.

Our SSD matlab implementation is based on [16] and on the freely available MPEG-7 reference software [17].

Examples for Shape Spectrum Descriptors using a histogram of 100 bins (the default for MPEG-7) are shown in Fig. E.3 and Fig. E.4.

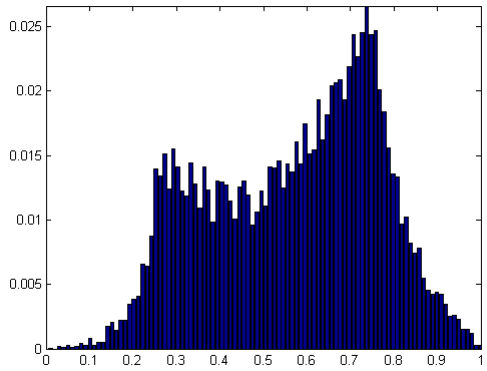


Figure E.3: 'Fox' Shape Spectrum Descriptor

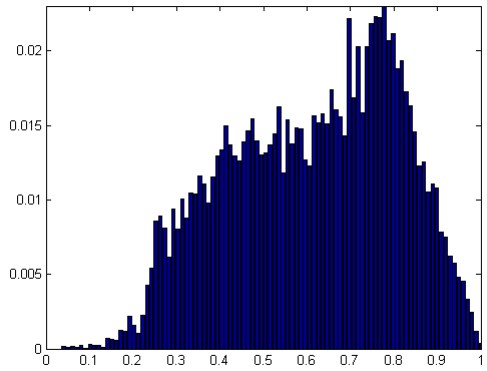


Figure E.4: 'Binocular' Shape Spectrum Descriptor

Appendix F

Rigid Objects Database

Figures F.1 and F.2 show one position for each of the 40 rigid objects of our database.

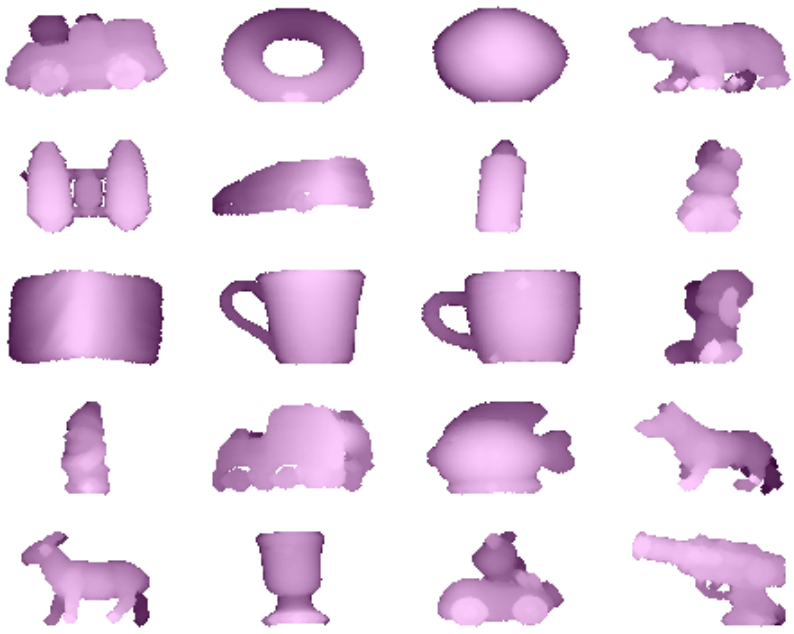


Figure F.1: one position of each of the first 20 rigid objects

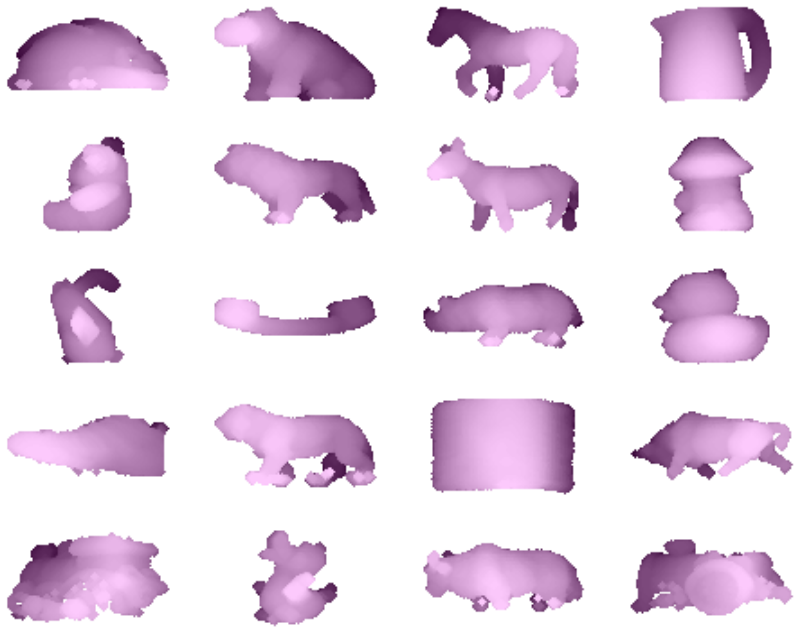


Figure F.2: one position of each of the last 20 rigid objects

Appendix G

Faces Database

Figures G.1 and G.2 show one frame for each of the 41 faces of our database.

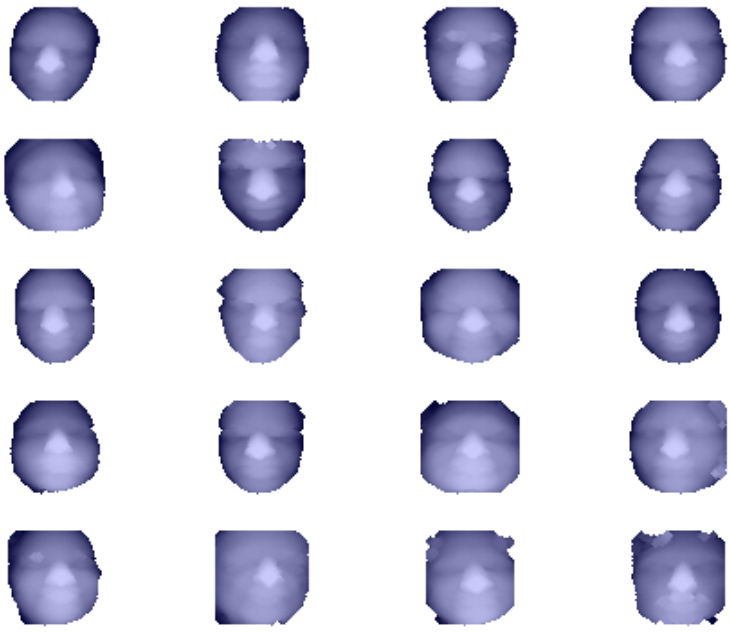


Figure G.1: one frame of each of the first 20 faces

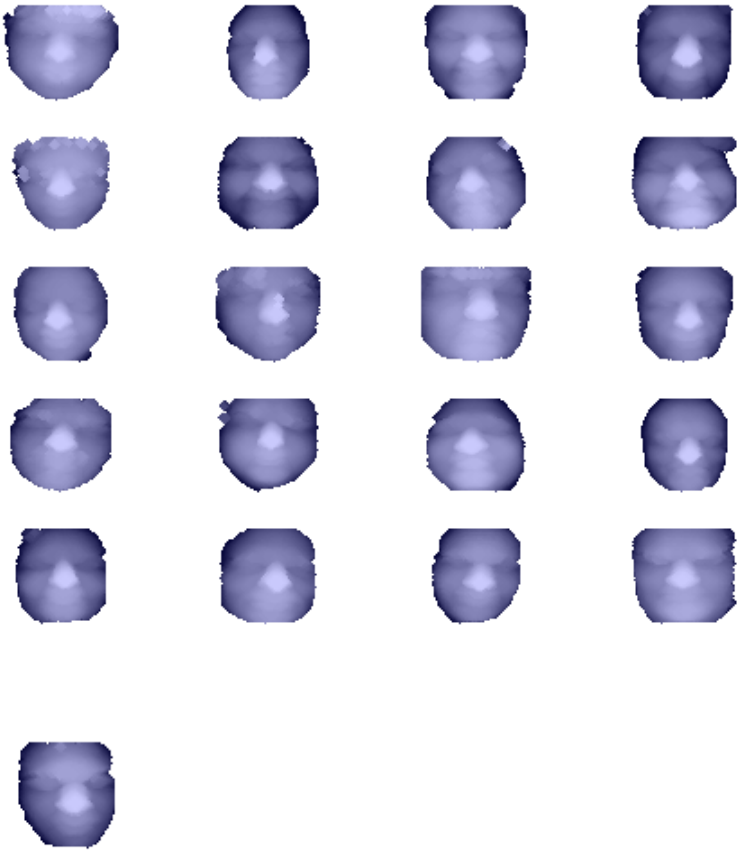


Figure G.2: one frame of each of the last 21 faces

Bibliography

[1] T. Tasdizen, J.P. Tarel and D.B. Cooper, "Improving the Stability of Algebraic Curves for Application", IEEE Trans. on Image Proc., vol.9, No. 3, pp. 405-416, March 2000.

[2] A. Helzer, M. Barzohar and D. Malah, "Stable Fitting of 2D Curves and 3D Surfaces by Implicit Polynomials", IEEE Trans. on PAMI., vol. 26, no. 10, pp. 1283-1294, October 2004.

[3] Z. Landa, "2D Object Description and Classification Based on Contour Matching by Implicit Polynomials", M.Sc. Thesis, The Technion – Israel Institute of Technology, August 2006.

[4] S. Sallivan, L. Sandford and J. Ponce, "Using geometric distance fits for 3-D object modeling and recognition", IEEE Trans. PAMI, vol. 16, pp. 1183-1196, Dec. 1994.

[5] G. Taubin, "Estimation of Planar Curves, Surfaces and Nonplanar Space Curves Defined by Implicit Equations, with Applications to Edge and Range Image Segmentation", IEEE Trans. on PAMI, vol. 13, no. 11, pp. 1115-1138, Nov 1991.

[6] J.P. Tarel, H. Civi and D.B. Cooper, "Pose Estimation of Free-Form 3D Objects without Point Matching using Algebraic Surface Models", Proc. IEEE Workshop Model-Based 3-D Image Analysis, Mumbai, India, pp. 13-21, 1998.

[7] J.P. Tarel and D.B. Cooper, "The Complex Representation of Algebraic Curves and Its Simple Exploitation for Pose Estimation and Invariant Recognition", IEEE Trans. on PAMI., Vol. 22, No. 7, pp. 663-674, July 2000.

[8] B. Zheng, J. Takamatsu, K. Ikeuchi, "3D Model Segmentation and Representation with Implicit Polynomials", IEICE Trans. Inf. & Syst., VOL.E91–D, pp. 1149-1158, no.4 April 2008.

[9] "The Stanford 3D Scanning Repository", Stanford University Computer Graphics Laboratory, http://www-graphics.stanford.edu/data/3Dscanrep/.

[10] S. Rusinkiewicz, D. DeCarlo, A. Finkelstein, and A. Santella, "Suggestive Contour Gallery", http://www.cs.princeton.edu/gfx/proj/sugcon/models/.

[11] R.O. Duda, P.E. Hart and D.G. Stork, "Pattern Classification", Wiley Interscience, November 2000.

[12] B.K.P. Horn, "Closed-Form Solution of Absolute Orientation Using Unit Quaternions", Journal of the Optical Society of America, Vol. 4, pp. 629-642, April 1987.

[13] P.J. Besl, N.D. McKay, "A Method for Registration of 3-D Shapes", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 2, February 1992.

[14] S. Rusinkiewicz, M. Levoy, "Efficient variants of the ICP algorithm", Proc. of the Third Int. Conf. on 3D Digital Imaging and Modeling, Canada, 2001.

[15] A. E. Johnson and M. Hebert, "Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes", IEEE Trans. on PAMI., Vol. 21, No. 5, May 1999.

[16] T. Zaharia and F. Prêteux, "3D Shape-based retrieval within the MPEG-7 framework", Proc. SPIE Conf. on Nonlinear Image Processing and Pattern Analysis XII, Vol. 4304, pp.133-145, San Jose, Etats-Unis, Jan. 2001.

[17] MPEG-7 Implementation Studies Group, "Information Technology - Multimedia Content Description Interface - part 6: Reference Software", ISO/IEC FCD 15938-6 / N4006, MPEG-7, Singapore, March 2001.

[18] D. Scharstein and R. Szeliski, "High-Accuracy Stereo Depth Maps Using Structured Light", Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on , vol.1, pp. I-195-I-202, June 2003.

[19] A. Helzer, "Robust Fitting of Implicit Polynomials with Application to Contour Coding", M.Sc. Thesis, The Technion – Israel Institute of Technology, June 2000.

[20] D. Keren, "Using Symbolic Computation to Find Algebraic Invariants", IEEE Trans. on PAMI., vol. 16, no. 11, pp. 1143-1149, November 1994.

[21] J. Subrahmonia, D. B. Cooper and D. Keren, "Practical Reliable Bayesian Recognition of 2D and 3D Objects Using Implicit Polynomials and Algebraic Invariants", IEEE Trans. on PAMI., vol. 18, no. 5, pp. 505-519, May 1996.

[22] C. Dorai and A. K. Jain, "Shape Spectrum Based View Grouping and Matching of 3D Free-Form Objects", IEEE Transactions on PAMI, , vol.19, no.10, pp.1139-1145, Oct 1997.

[23] G. Paul-Louis and B. Houman, "Delaunay Triangulation and Meshing: Application to Finite Elements", Hermes, Paris 1998.

[24] "Stuttgart Range Image Database", http://range.informatik.uni-stuttgart.de/htdocs/html/.

תיאור וסיווג עצמים תלת מימדיים
באמצעות פולינומים סתומים

הילה בן-יעקב

תיאור וסיווג עצמים תלת מימדיים באמצעות פולינומים סתומים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בהנדסת חשמל

הילה בן-יעקב

הוגש לסנט הטכניון - מכון טכנולוגי לישראל

אלול, תשס"ח חיפה ספטמבר 2008

המחקר נעשה בהנחיית פרופ' דוד מלאך וד"ר מאיר בר-זוהר

בפקולטה להנדסת חשמל

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי

תקציר

פולינומים סתומים משמשים לתיאור עקומות דו-מימדיות ומשטחים תלת-מימדיים המתוארים ע"י מידע בדיד. הקלט למערכת היא רשימת מיקומי נקודות, והפלט הוא מקדמי הפולינום הסתום שהותאם למידע הנתון. במשך הזמן, פותחו אלגוריתמים שונים להתאמת פולינומים סתומים למידע. האלגוריתמים המתקדמים ביותר הם גרדיאנט1 (Gradient1) מינ-מקס (Min-Max) ומינ-וואר (Min-Var), אשר כל אחד מהם משתמש בפתרון בעיית ריבועים פחותים (Least Squares) לינארית, לצורך מציאת מקדמי הפולינום. אלגוריתמים אלה משיגים ביצועים טובים בהרבה בהשוואה לאלגוריתמים ותיקים יותר, אשר משתמשים בשיטות לא לינאריות ואיטרטיביות, גם מבחינת היכולת התיאורית של העצמים, וגם מבחינת יכולת הסיווג בין עצמים שונים. בנוסף, לאלגוריתמים אלו סיבוכיות חישובית נמוכה יותר בהשוואה לאלגוריתמים הוותיקים. בעבר, יכולות התיאור והסיווג של האלגוריתמים להתאמת פולינומים נבחנו בעיקר עבור המקרה הדו-מימדי. מטרת עבודה זו היא לבחון את יכולות התיאור והסיווג במקרה התלת-מימדי.

השלב הראשון המתואר בעבודה זו, הוא חקירת היכולות התיאוריות של אלגוריתמים להתאמת פולינומים סתומים במקרה התלת-מימדי, גם עבור סדרי פולינומים נמוכים וגם עבור סדרים גבוהים. כמו כן, הצענו שינוי לאלגוריתמים מינ-מקס ומינ-וואר כך שהתאמת הפולינום לא תושפע מסיבוב (rotation invariant), כלומר, שהקשר בין מקדמי הפולינומים שהותאמו לאותו אובייקט בתנוחות שונות יהיה קשר של סיבוב בלבד. השינוי שהצענו מבוסס על העבודה שנעשתה במקרה הדו-מימדי עבור אלגוריתמים אלה.

בשלב השני, ביצענו סקירה של אלגוריתמים המבצעים שערוך הכיוון (pose estimation) בו נמצא העצם במקרה התלת-מימדי. באמצעות הרקע המתמטי של אלגוריתמים אלו, חקרנו מספר שיטות להפקת ביטויים שאינם משתנים תחת סיבוב, אשר תלויים במקדמי הפולינום. באמצעות ייצוג טנזורי (tensor representation) של פולינומים סתומים, פיתחנו סדרה של ביטויים שאינם מושפעים מסיבוב, אשר כל אחד מהם הוא צירוף לינארי של מקדמי הפולינום. עבור פולינום תלת-מימדי מסדר זוגי n ניתן להפיק n/2+1 ביטויים כאלה. יישמנו את אותה שיטה על מנת להפיק ביטויים לינארים שאינם מושפעים מסיבוב גם במקרה הדו-מימדי, כיוון שיש לה יתרון חישובי על השיטה הקיימת היום, אשר משתמשת בתהליך נסיגה (רקורסיה).

באמצעות מטריצות הסיבוב בתלת-מימד וזהויות טריגונומטריות, פיתחנו עוד שני ביטויים שאינם מושפעים מסיבוב, אשר כל אחד מהם הוא צירוף של ריבועי מקדמי הפולינום. כמו כן, בדקנו את האפשרות להשתמש בייצוג קווטרניוני (quaternion representation) בתור שיטה חלופית להפקת ביטויים שאינם מושפעים מסיבוב, באופן דומה למה שנעשה במקרה הדו-מימדי באמצעות ייצוג מרוכב. הקווטנרניון מורכב מרכיב אחד ממשי, ומשלושה רכיבים מדומים אשר ניצבים זה לזה. כאשר הרכיב הממשי הוא אפס, ניתן להשתמש בקווטרניון בתור ייצוג חלופי לווקטור במרחב תלת-מימדי. האתגר בחישובים הכוללים קווטרניונים הוא העובדה שלא ניתן ליישם את חוק החילוף בפעולות הכוללות מכפלה של קווטרניונים. לכן שיטה זו אינה ישימה להפקת ביטויים שאינם מושפעים מסיבוב אשר תלויים במקדמי פולינומים מסדרים גבוהים.

בשלב הבא, אנו מתארים את שלבי העיבוד המקדים הנחוצים לצורך קבלת ביצועים טובים בסיווג העצמים לפני התאמת הפולינום ומיצוי המאפיינים. השלבים המתוארים בעבודה הם:

1. הזזת מיקום מרכז המסה של העצם לראשית הצירים על מנת שהתאמת הפולינום לא תושפע מהזזת העצם למיקומים שונים במרחב.
2. שינוי גודל העצם לפי האחוזון ה-75 של מרחקי הנקודות מהראשית, על מנת שמיקומי נקודות המידע שלנו יהיו קרובים ככל האפשר למיקום המעטפת של כדור ברדיוס של אחד. דרישה זו מבוססת על ניתוח של יציבות מקדמי הפולינום עבור עצמים בגדלים שונים שנעשה בעבודה קודמת עבור המקרה הדו-מימדי.
3. שיקוף של פנים תלת-מימדיים סביב מישור על מנת לקבל משטח תלת-מימדי סגור. אנו מראים שפעולה זו משפרת את יציבות מקדמי הפולינום המותאם לפנים.
4. שימוש בהיטלים דו-מימדיים של העצמים על מנת למצות מספר רב יותר של מאפיינים. אנו בוחרים את ההיטלים המתאימים לכל אפליקציה בהתאמה ליכולת התיאורית שלהם בכל מקרה.

בשלב האחרון אנו בונים מסווג לעצמים תלת-מימדיים ומשווים את תוצאות הסיווג בשיטה שאנו מציעים לתוצאות סיווג של אלגוריתמים אחרים.

שיטת הסיווג הממוצעת בעבודה מתבססת על שיטת סיווג של קווי מתאר דו-מימדיים, המשתמשת בהתאמת מספר סדרים של פולינומים וחישוב שגיאת ההתאמה של כל פולינום (MOFET).

הסיבה לכך שנדרשים מספר סדרים של פולינומים היא שאנו רוצים לנצל גם את היציבות של פולינומים בעלי סדר נמוך, אך גם את היכולת התיאורית של פולינומים בעלי סדר גבוה.

היתרון של שיטה זו הוא שאין צורך לבצע שערוך של כיוון העצם (pose estimation) על מנת להשוות בינו לבין העצמים שבמילון. במקום שערוך זה, שהוא בעל סיבוכיות חישובית גבוהה, אנו משתמשים לצורך הסיווג בווקטור מאפיינים המופקים מהפולינומים.

סיווג העצמים התלת-מימדיים נעשה באמצעות המאפיינים הבאים: מאפיינים שאינם מושפעים מסיבוב ואשר תלויים במקדמי הפולינומים שהותאמו לעצם התלת-מימדי, שגיאות ההתאמה של הפולינומים התלת-מימדיים, מאפיינים שאינם מושפעים מסיבוב ואשר תלויים במקדמי הפולינומים שהותאמו להיטלים הדו-מימדיים של העצם, שגיאות ההתאמה של הפולינומים הדו-מימדיים, והערכים העצמיים של

ניתוח הצירים הראשיים של האובייקט באמצעות מטריצת פיזור (scatter matrix), סה"כ 31 עד 35 מאפיינים.

אנו בוחנים את תוצאות הסיווג עבור שלושת האלגוריתמים, גרדיאנט1, מינ-מקס שאינו מושפע מסיבוב ומינ-וואר שאינו מושפע מסיבוב. אנו מפעילים את המסווג על שני מאגרי עצמים: הראשון הוא מאגר של עצמים קשיחים שכל אחד צולם מכמה נקודות מבט, והשני הוא מאגר של פנים של אנשים שונים במצב של שיתוף פעולה (כלומר, צילום בתנאים מבוקרים כאשר הבעות הפנים ותנועות הראש אינן קיצוניות). תוצאות הסיווג בשיטה שלנו מושוות לתוצאות סיווג של אלגוריתמים אחרים בעלי סיבוכיות חישובית דומה:

1. סיווג באמצעות שערוך כיוון העצם שלאחריו מותאם פולינום סתום לעצם ומושווים מקדמי הפולינום עם מקדמי הפולינומים שהותאמו לעצמים שבמילון.
2. סיווג באמצעות סטטיסטיקה של עקמומיות המשטח בכל נקודה (Shape Spectrum Descriptor), שיטה שאומצה ע"י וועדת התקינה של MPEG-7 לצורך סיווג עצמים תלת-מימדיים.

השוואת תוצאות הסיווג על מאגרי עצמים אלה מראה שלשיטה שלנו ביצועים טובים יותר מאשר שתי השיטות האחרות. בנוסף, זמני הריצה של השיטה שלנו קצרים יותר ביחס לזמני הריצה של השיטות האחרות.