Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Master's Program in Computer Science

Max-Planck-Institut für Informatik
Computer Graphics Group
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany

Mathematical Image Analysis Group
Fakultät für Mathematik und Informatik
Saarland University
66041 Saarbrücken, Germany

Silhouette-Based 2D-3D Pose Estimation Using Implicit Algebraic Surfaces

Master's Thesis in Computer Science

Submitted by
Nazar Khan
on 2nd January 2007

Conducted under Advice of
Dr. Bodo Rosenhahn
Prof. Dr. Joachim Weickert

Reviewed by
Dr. Bodo Rosenhahn
Prof. Dr. Joachim Weickert

Begin: August 01, 2006
End: January 02, 2007

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, 1st January 2007

Nazar Khan

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, 1st January 2007

Nazar Khan

Contents

List of Figures

List of Tables

Acknowledgment

I would like to thank my supervisor Dr. Bodo Rosenhahn for all his help, guidance and patience and for proof-reading the text. I also thank Prof. Dr. Joachim Weickert for insightful discussions and proof-reading. I cannot but acknowledge the wonderful and bureaucracy-free environment of the MPII and the assistance provided by the Graphics group secretaries Sabine Budde and Conny Liegl.

I would also like to thank Dr. Robert H. Lewis for considerable help in computing resultants using his computer algebra system `Fermat`. I thank Prof. Dr. Wolfram Decker for introducing me to `SINGULAR` and elimination theory, Arno Eigenwillig for his help with resultants and Ioannis Emiris and George Tzoumas for tips on computing matrix determinants by interpolation.

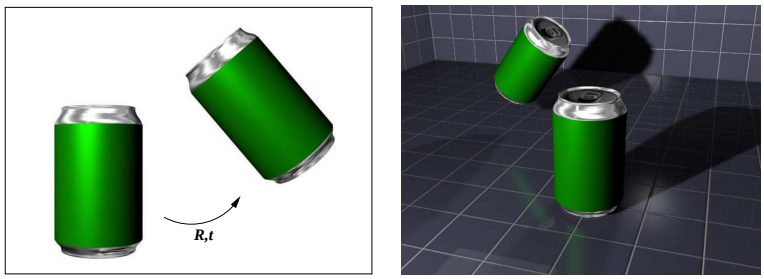
I wish to thank my roommates Sarmad and Nassim for helping me stay sane and for tolerating my alarm clock that always seemd to wake *them* up. Special thanks also to Dominique Beaulieu and Robert Herzog at the Graphics group for making my stay worthwile and helping out with TEXnical stuff now and then.

Last but not least, I wish to thank my family for all their support and I dedicate this thesis to my parents for always being there.

Chapter 1

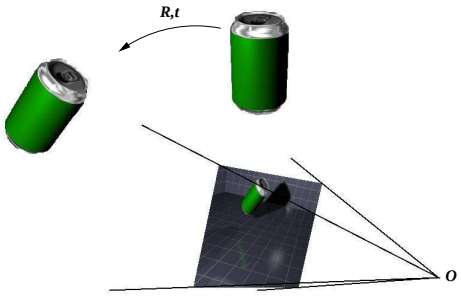
Introduction

2D-3D pose estimation is a crucial task for many problems in areas ranging from robot navigation to medical intervention. It involves estimating the relative position and orientation of a known 3D object with respect to a reference camera system. In other words, we search for a transformation (i.e. the pose) of the 3D object such that the transformed object corresponds to 2D image data. For rigid objects, such a transformation can be the Euclidean transformation consisting of a rotation $\mathbf{R}$ and a translation $\mathbf{t}$. Pose estimation

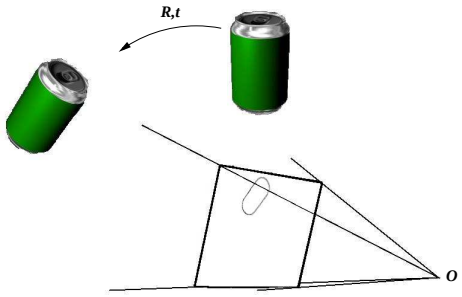


(a) 2D-2D Registration

(b) 3D-3D Registration



(c) 2D-3D Registration



(d) 2D-3D Silhouette-based Registration

Figure 1.1: *The spatial (a), (b) and projective (c), (d) registration domains.*

is a subclass of the more general problem of registration which is one of the key problems

in computer vision. Registration is the process of aligning 2D and/or 3D *objects*. Objects can include images, shapes, curves, surfaces or point sets. Figure 1.1 shows different registration scenarios. Based on the different types of objects being aligned, the main registration domains can be categorised as

1. 2D-2D registration whereby a *source* image object is *transformed* so that it *aligns* with a *target* object,

2. 3D-3D registration whereby a source 3D curve, surface or point set is transformed so that it aligns with a target 3D curve, surface or point set,

3. 2D-3D registration whereby a source 3D curve, surface or point set is transformed so that it aligns with a target 2D image or image curve.

Categories 1 and 2 are instances of *spatial registration* since they involve alignment of spatial data (in 2D image space for category 1 and in 3D space for category 2). The required transformation can then be affine or Euclidean. Category 3 is an instance of *projective registration* since it involves aligning 3D spatial data to 2D projective data [21]. The required transformation in this case is therefore affine or Euclidean *followed* by a projective transformation. Figure 1.2 shows a hierarchy of the different registration domains.

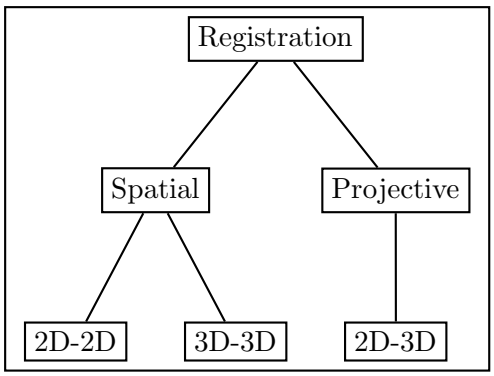


Figure 1.2: *Hierarchy of registration domains*

Pose estimation, as defined in this thesis[1], falls under the category of projective registration. Specifically, it is a subclass of 2D-3D registration where the projective transformation is already known and the required transformation is therefore affine or Euclidean. Since we will be dealing with rigid objects only, we can ignore affine transformations of the 3D objects and we can restrict Euclidean motions to rigid body motions. The pose estimation problem can then be defined as

Definition 1.1 (Pose Estimation). *Estimation of the* 3×3 *rotation matrix* $\mathbf{R}$ *and the* 3×1 *translation vector* $\mathbf{t}$ *that constitute the rigid body motion needed to transform a 3D object into agreement with 2D image data.*

From here onwards, by pose estimation we will mean 2D-3D pose estimation.

[1]The terms pose estimation, registration, motion estimation and tracking are used ambiguously in the literature. We therefore wish to differentiate pose estimation and registration at the outset.

Current approaches to pose estimation (and registration in general) can be divided into two categories:

1. **Explicit** pose estimation whereby the 2D and 3D entities involved are defined explicitly. This includes points, lines and higher order entities such as conics, kinematic chains or higher order 3D curves [58].

2. **Free-form** pose estimation whereby the entities involved are modelled as free-form objects such as parametric curves/surfaces, 3D meshes, active contours and implicit curves/surfaces [58].

Figure 1.3 presents a hierarchy of the different registration techniques.

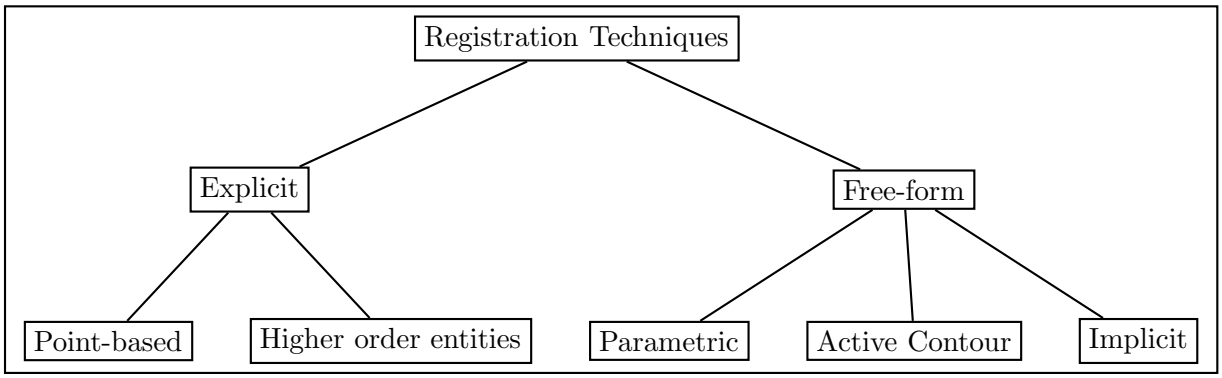


Figure 1.3: *Different registration techniques*

This thesis explores 2D-3D pose estimation using free-form objects. The particular free-form representations we use are implicit surfaces. From the image domain we use only the image silhouettes which are modelled as implicit curves while 3D objects are modelled as implicit surfaces. The particular class of implicit surfaces we use are called algebraic surfaces. The combined approach can therefore be termed as *silhouette-based 2D-3D pose estimation using algebraic surfaces.*

Image *silhouettes*[1] are a rich source of geometric information about the 3D objects being viewed. An image silhouette is the projection of the locus of points on the object – the *contour generator*[2] – which separates the visible object points from the occluded ones [55]. Alternatively, contour generators are 3D curves consisting of points where the surface turns away from the viewer [27]. Mathematically, this implies that the normal to the 3D surface at points on the contour generator is perpendicular to the viewing direction.

1.1 Literature Overview of Silhouette-based 3D Analysis

In this section we present an overview of of both explicit and implicit silhouette-based reconstruction, recognition and tracking techniques.

[1] also called *occluding contours*, *extremal contours*, *apparent contours*
[2] also called *extremal boundary*

Reconstructing shape from silhouettes was introduced by Baumgart [11] more than three decades ago. Cippolla and Blake [55] showed that by analysing silhouette deformations local surface curvature can be computed along the corresponding contour generators. Forsyth [22] showed that outlines of algebraic surfaces completely determine their projective geometry from a single view. Cross et al. [29] studied the projective relationship between the coefficients of quadratic algebraic surfaces and the coefficients of the corresponding 2D algebraic silhouettes. Due to perspective projection, the relationship between algebraic surface and algebraic plane curve coefficients is very complex for higher-order surfaces. Kang et al. [35] reconstructed 3D surfaces from occluding contours of algebraic surfaces using a linear dual-surface approach that makes use of the duality between 3D points and tangent planes.

Taubin and Cooper [66] showed that by representing silhouettes by implicit polynomials, algebraic invariants and geometric information of the objects can be extracted from the polynomial coefficients. Algebraic invariants lead to fast object recognition while geometric information leads to simpler registration. Continuing with Taubin and Coopers' ideas, Lei et al. [43][44] and Siddiqi et al. [34] used 2D object silhouettes to recognise different objects using fast Bayesian recognition based on the algebraic invariants. Tarel et al. [64] and Unel and Wolovich [50] applied the same ideas for aligning 2D shapes.

For 2D-3D pose estimation, Kriegman and Ponce [42] parameterised image silhouette equations by 3D pose parameters and minimised the distance between such equations and pixels representing image outlines to obtain the optimal pose. Rosenhahn [58] used the explicit approach instead and back-projected lines through the silhouette pixels in order to register 3D models with those lines. He extended his approach to human motion tracking in [5]. Ilic et al. [59] and Knossow et al. [41] also used image silhouettes for human motion tracking using implicit equations.

1.2 Literature Overview of Registration Algorithms

We will divide registration literature into spatial and projective registration approaches. Ideas from each approach are, however, used in the other.

1.2.1 Spatial Registration

Considerable work has been done on spatial registration problems. One of the most popular approaches for spatial registration is the Iterative Closest Point (ICP) method simultaneously introduced by several groups [6][16][72][15][48]. In its original form, ICP iteratively finds the closest point pairs between the data sets and tries to find the transformation that minimises the distances between the closest point pairs. The obtained transformation is then applied to the source data set and new closest point pairs are found which are in turn used to find a new distance minimising transformation. This process continues until the registration error between source and target falls below some quality threshold or the solution converges.

The transformation space of rotations and translations might contain many transformations that minimise the point pair distances locally. Such transformations are called

local minima. In contrast, the global minimum is the transformation that minimises the distances globally. The problem with ICP is that it can get stuck in local minima. In order to converge to the global minimum, an initial transformation is required that is close to the true value. ICP also suffers from the problem of slow convergence [63]. Moreover, one data set has to be a proper subset of the other and the method is sensitive to errors in feature extraction and occlusion. There are numerous ICP variants that address these problems.

Zhang [72] improves upon the original ICP by using k-D trees to speed up closest point search and by using a dynamic distance threshold derived through distance histogram statistics. Points further apart than this adaptive threshold are not used for motion computation.

Johnson and Kang [33] improve upon Zhang's work by integrating texture information with the distance measure. Points that have small Euclidean distance but different colours are still treated as having a large distance.

Using Euclidean distance in ICP restricts tangential sliding along aligned lines which causes slow convergence and multiple weak minima near the global minimum. To remedy that problem, Nguyen et al. [51] replace the Euclidean distance by the "normal distance" between patches. It leads to faster convergence. Better registration accuracy, noise tolerance and surface coverage are achieved by using low curvature patches instead of high curvature landmarks such as corners and ridges. Rough pre-alignment is still needed for their approach but their main contribution is speed and accuracy.

Rusinkiewicz and Levoy [63] breakup the ICP algorithm into 6 main stages and examine the effect of choices at each stage on convergence speed. They conclude that projection based correspondence finding and a point-to-plane error metric are most useful for increasing convergence speed. They construct a high-speed ICP by combining different variants and propose the use of variant switching algorithms that switch between appropriate variants depending on local error landscape or the probable presence of local minima.

Stewart [61] proposes using the error covariance matrices of the data to get more accurate estimates of registration since estimates depend on error in data. He formulates registration as a statistical optimisation problem whereby point matching is based on the Mahalanobis distance instead of Euclidean distance and registration therefore involves minimising the combined Mahalanobis distances of all data points.

Boughorbel et al. [10] replace the Euclidean distance of ICP by a Gaussian measure of distance and similarity. Minimising an energy function of the Gaussian measures implies maximisation of both overlap and similarity. The energy function is always differentiable and convex in a large neighborhood of the global solution. This allows a large region of convergence and therefore reduces chances of getting stuck in local minima. ICP, on the other hand, has a non-differentiable cost function that imposes local convergence and hence the need for preliminary matching. The Gaussian energy function provides a fully automatic registration framework. Standard optimisation methods can be used because of the differentiability of the energy function.

Burschka et al. [12] register monocular stereo images to 3D surface models. For efficient search over the transformation space, they use covariance trees instead of k-D trees.

Standard ICP is sensitive to outliers, i.e. a single bad correspondence will affect the sum of squared errors that is to be minimised and hence the transformation estimate is not robust. Standard ICP uses a closed form solution for the transformation estimate (in the minimisation step). The problem is that a *robust* estimate in closed-form is not known. So, researchers have tried robustifying ICP through

1. non-linear or RANSAC-based estimation in the minimisation step, or

2. excluding bad correspondences computed during the matching step.

Both approaches deal with outliers after they have already affected the error/distance values. Furthermore, existing methods incur significant speed loss. Fitzgibbon [26] deals with outliers at an early stage and provides robustness directly in the error function that is used both in the matching and the minimisation steps. The error function is changed to a robust kernel that gives smaller error values for outliers. Fitzgibbon replaces the closed-form standard ICP estimate (which can not incorporate robust statistics) by the estimate achieved through general-purpose iterative, non-linear Levenberg-Marquardt optimisation. This estimate can incorporate robust statistics. The resulting method, called LM-ICP, is faster, simpler to implement and robust.

Chetverikov et al. [17] introduced the Trimmed ICP (TrICP) that excludes bad correspondences. It attempts to reject/trim wrong correspondences by using only those correspondences that have the smallest distances. Correspondences are sorted by squared distance and only a fixed fraction of them are used (Least Trimmed Squares). TrICP is robust to rotations and noisy data and has a proved convergence.

Tarel et al. [32] renounce the correspondence based ICP approach to spatial registration. They represent 2D curve or 3D surface data by a few coefficients of an implicit polynomial. Such implicit polynomial coefficients encode the intrinsic geometry of the entity that they represent. This allows one-shot registration of the entities without the use of correspondences.

1.2.2 Projective Registration

ICP has also been applied in the projective registration domain. Rosenhahn [58] registers 3D model points with 3D lines/planes constructed from 2D image points/lines. The emphasis in that work is on the mathematical modelling of the pose problem using *Conformal algebra*. Specifically, both rigid body motion and 3D objects are modelled using Conformal algebra.

Phong et al. [53] solved a similar problem using 2D-3D line constraints. There it is shown that while point-point constraints can also be used to solve the problem, line-line constraints lead to the decoupling of the rotation and translation estimation problems. Rigid body motions are represented as *dual number quaternions* which are a mathematical representation of screws. Minimisation of the error function was based on a trust-region method instead of ICP.

1.3 Motivation for the Implicit Approach

Phong et al. [53] state: "*Since the object pose from a single view problem is nonlinear, choices for (i) the mathematical representation of the problem, (ii) the error function to be minimized, and for (iii) the optimization method are crucial*". Accordingly, the mathematical representation that they use is explicit. Rigid body motion is modelled using dual number quaternions which leads to an error function that is a sum of squares of quadratic constraints. This, in turn, allows robust optimisation using a trust-region method that avoids local minima and has fast convergence. The result is a robust and fast 2D-3D pose estimation procedure.

Rosenhahn [58] showed that explicit approaches to the pose problem are faster than free-form approaches. But he mentions that in many cases, extracting point-based features from images is not possible. Hence the need for global i.e. free-form descriptors. Furthermore, estimates derived from global descriptors are more accurate and robust.

This work therefore explores the free-form approach whereby objects are modelled as algebraic surfaces. This choice has consequences on the error function to be used and the optimisation procedure employed.

A free-form approach was also used earlier by Kriegman and Ponce [42] who used elimination theory to compute silhouette equations of parametric surfaces. They reported that the free-form approach quickly leads to "unwieldy" silhouette equations. In that paper, they used simple elimination techniques and mentioned that advanced techniques will reduce the complexity of the problem. Silhouette equation computation through elimination has since remained a bottleneck that has restricted further progress in this approach. But due to progress in computational power, advanced elimination heuristics [39][57][45] and the development of specialised computer algebra systems such as `Fermat` [56], implicit silhouette equations have become easier (but not easy) to compute and handle. In this thesis, we therefore re-enter the implicit pose estimation problem.

1.4 Overview

The thesis is divided into 3 main topics:

1. Algebraic surface fitting,
2. Algebraic space-curve projection, and
3. Algebraic pose estimation.

After presenting the mathematical background covering rigid body motions, polynomials and elimination theory in Chapter 2, we review surface representations in Chapter 3 where we provide justification for using algebraic implicit surfaces. The transformation of implicit polyomials, their intrinsic geometry and fitting techniques are described in Chapter 4. Elimination theory is used in Chapter 5 to project algebraic space-curves onto image planes to obtain algebraic plane-curves i.e. silhouette equations. There we describe the non-linear error function and the optimisation method used for pose estimation from algebraic image silhouette equations. The chapter ends with some experimental results. Finally, we present some conclusions and future outlook in Chapter 6.

Chapter 2

Background

In this chapter we will present the mathematical background needed for later chapters. The 3 main mathematical areas employed in this thesis are

1. Rigid body motions
2. Polynomials
3. Elimination theory

Registration of two datasets is the process of identifying a geometrical transformation that aligns the coordinate system of one with that of the other. Since we will be dealing only with rigid objects in this thesis, the required geometrical transformation for our problem is a rigid body transformation. We will describe rigid body motion using *screw theory*.

Polynomials are required both for modelling algebraic curves and surfaces and for obtaining silhouette equations using elimination theory.

2.1 Rigid Body Motions

As the name suggests, a rigid body motion represents the motion of rigid bodies in space. A rigid motion of an object is a motion which preserves distance and orientation between points [49]. This section describes rigid body motion using linear algebra and screw theory. Other representations of rigid body motion use dual-quaternions and Clifford algebras for which we refer the reader to [49] and [30] respectively.

A rigid body can be moved by applying a rotation followed by a translation. The same effect can be achieved by following the motion of a screw as shown in Figure 2.1. Screw motion can be represented by a rotation about a straight line l, called the the screw axis, *combined* with a translation parallel to that line. According to Chasles theorem, every rigid body motion can be realized by a rotation about an axis combined with a translation parallel to that axis. That is, every rigid body motion can be modelled as a screw motion. It should be observed that while the motion of a point on a screw is smooth, the same point will not move smoothly if a rotation followed by a translation is applied. The net *displacement* of a point for both rotation followed by translation and screw motion is the

same but the path taken by the point, i.e. the *motion* is different. Screw theory provides a geometric description of rigid motion that is useful for specific optimisation tasks. We will use it later to linearise pose parameters.

In the following, the exponential representation of rotation in $\mathbb{R}^3$ is described followed by the exponential representation of screw motion.

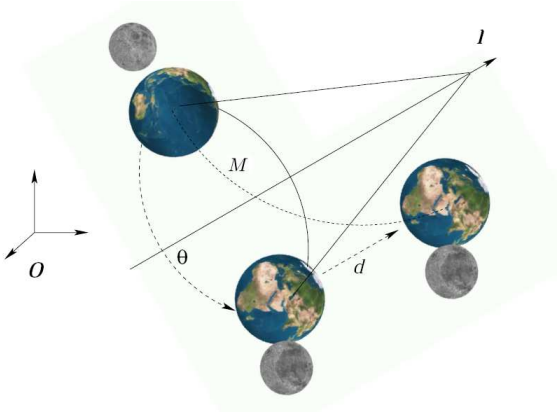


Figure 2.1: *A screw motion M can be achieved by a rotation of θ around the screw axis l combined with a translation d along l. The net effect is the same as applying Euler angle rotation followed by a translation. (Figure courtesy of [58])*

2.1.1 Rotational Motion in $\mathbb{R}^3$

2.1.1.1 Orthogonal Matrix Representation

The classical way of representing rotations in 3-dimensional space $\mathbb{R}^3$ is by using Euler angles. Any rotation $\mathbf{R_3}$ in $\mathbb{R}^3$ can be generated by 3 rotations around the x, y and z axes.

$$\mathbf{R_3} = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

where α, β and γ are the rotation angles around the x, y and z axes respectively. $\mathbf{R}_x, \mathbf{R}_y$ and $\mathbf{R}_z$ are the corresponding rotation matrices. Rotation matrices have the property that they are orthogonal and have a determinant of $+1$. The group of such matrices is called the "special orthogonal" group, written as $SO(3)$. The term "special" signifies the determinant equal to $+1$ property. Since they are members of a group, concatenation results in a matrix that is also a member of the group. Therefore, $\mathbf{R}_3$ is also an orthogonal matrix with determinant $+1$, i.e. a rotation matrix. Specifically,

$$\mathbf{R}_3 \in SO(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}\mathbf{R}^T = I, det(\mathbf{R}) = +1\} \tag{2.2}$$

Using Euler angles, rotation is performed by multiplication of a point $\mathbf{x} \in \mathbb{R}^3$ with the rotation matrix $\mathbf{R}_3$ to get the rotated point $\mathbf{x}'$.

$$\mathbf{x}' = \mathbf{R}_3\mathbf{x} \tag{2.3}$$

2.1.1.2 Canonical Exponential Coordinates Representation

The orthogonal matrix representation is a mathematical treatment of rotation that deals with the initial and final positions of a point and abstracts away the physically relevant *motion* part. Figure 2.2 shows two different motions resulting in the same displacement

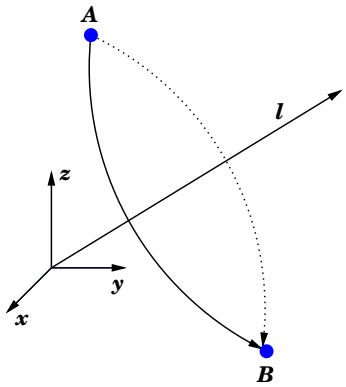


Figure 2.2: *Rotation around axis l by $\pi/2$ and $-\pi/2$ result in the same displacement A to B but different motions.*

from point A to point B. An alternative representation of rotations in $\mathbb{R}^3$ is the exponential representation which can also represent the motion in addition to the displacement captured by the Euler angles approach.

Definition 2.1. ***Skew-symmetric matrix*** *A skew-symmetric matrix $\hat{\omega} \in \mathbb{R}^{n\times n}$ is a square matrix whose transpose is its negative*

$$\hat{\omega}^T = -\hat{\omega} \tag{2.4}$$

The group of all 3×3 skew-symmetric matrices is denoted as $so(3)$ and is written as

$$so(3) = \left\{\hat{\omega} \in \mathbb{R}^{3\times 3} | \omega \in \mathbb{R}^3\right\} \tag{2.5}$$

There is a one-to-one mapping between vectors in $\mathbb{R}^3$ and skew-symmetric matrices in $so(3)$ which implies that

Lemma 2.1.1. *A matrix $\mathbf{M} \in \mathbb{R}^{3\times 3}$ is skew-symmetric if and only if $\mathbf{M} = \hat{\omega}$ for some $\omega \in \mathbb{R}^3$.*

The $\hat{}$ is called the *hat operator* and it constructs a skew-symmetric matrix from the corresponding vector. For example, for $\omega = (\omega_1, \omega_2, \omega_3)$

$$\hat{\omega} = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}$$

The reverse operation of extracting the vector from the skew-symmetric matrix is denoted by the *vee operator* $\vee$.

To see the role played by skew-symmetric matrices in obtaining an exponential representation of rotation, consider a trajectory $\mathbf{R}(t) : \mathbb{R} \to SO(3)$ describing a continuous rotational motion, the rotation must satisfy the orthonormality constraint

$$\mathbf{R}(t)\mathbf{R}^T(t) = I$$

Derivating with respect to time t we get

$$\begin{aligned} &\dot{\mathbf{R}}(t)\mathbf{R}^T(t) + \mathbf{R}(t)\dot{\mathbf{R}}^T(t) = \mathbf{0} \\ \Rightarrow &\dot{\mathbf{R}}(t)\mathbf{R}^T(t) = -\mathbf{R}(t)\dot{\mathbf{R}}^T(t) \\ \Rightarrow &\dot{\mathbf{R}}(t)\mathbf{R}^T(t) = -(\dot{\mathbf{R}}(t)\mathbf{R}^T(t))^T \end{aligned}$$

which shows that $\dot{\mathbf{R}}(t)\mathbf{R}^T(t) \in \mathbb{R}^{3\times 3}$ is a skew-symmetric matrix. Lemma 2.1.1 then implies that there exists a vector $\omega(t) \in \mathbb{R}^3$ such that

$$\dot{\mathbf{R}}(t)\mathbf{R}^T(t) = \hat{\omega}(t)$$

Multiplying both sides from the right by $\mathbf{R}(t)$ gives

$$\dot{\mathbf{R}}(t) = \hat{\omega}(t)\mathbf{R}(t) \tag{2.6}$$

If at some time $t = t_0$, $\mathbf{R}(t_0) = I$, then from (2.6)

$$\dot{\mathbf{R}}(t_0) = \hat{\omega}(t_0) \tag{2.7}$$

This shows that $\hat{\omega}$ is tangent to $\mathbf{R}$ at I. More generally, *the space of skew-symmetric matrices is the* ***tangent space***[1] *at identity of the rotation group* $SO(3)$. Hence the terminology $so(3)$. At $\mathbf{R} \neq I$ the tangent space $\dot{\mathbf{R}}$ is the tangent space at identity i.e. $so(3)$ transformed to $\mathbf{R}$ by multiplication by $\mathbf{R}$ from the right (see (2.6)). From the above observations, we see that around the identity matrix I, a skew-symmetric matrix gives a first-order approximation to a rotation matrix:

$$\mathbf{R}(t_0 + dt) \approx I + \hat{\omega}(t_0)dt \tag{2.8}$$

For the case of rotation around a fixed axis (constant ω) (2.6) becomes

$$\dot{\mathbf{R}}(t) = \hat{\omega}\mathbf{R}(t)$$

which is a time-invariant linear differential equation whose solution is

$$\mathbf{R}(t) = \exp(\hat{\omega}t)\mathbf{R}(0) \tag{2.9}$$

where $\exp(\hat{\omega}t)$ is the matrix exponential

$$\exp(\hat{\omega}t) = \sum_{k=0}^{\infty} \frac{(\hat{\omega}t)^k}{k!} \tag{2.10}$$

$$= I + \hat{\omega}t + \frac{(\hat{\omega}t)^2}{2!} + \cdots + \frac{(\hat{\omega}t)^k}{k!} \tag{2.11}$$

[1] Since $SO(3)$ satisfies the properties of a Lie group, its tangent space at identity i.e. $so(3)$ is its Lie algebra. [31] [49]

Assuming $\mathbf{R}(0) = I$ as the initial condition in (2.9) we get the so-called *exponential map*

$$\mathbf{R}(t) = \exp(\hat{\omega}t) \tag{2.12}$$

that maps $so(3)$ to $SO(3)$. The components of ω, i.e. $(\omega_1, \omega_2, \omega_3)$ are therefore known as *exponential coordinates* of rotation. *Equation (2.12) states that taking the exponential of a skew symmetric matrix $\hat{\omega}t$ results in a matrix $\mathbf{R}$ representing a rotation of t radians around the unit vector ω.*

The exponential map in (2.12) involves the non-trivial task of taking the exponential of the matrix $\exp(\hat{\omega}, t)$. An efficient method for computing matrix exponentials is given by the *Rodrigues' formula* [49]

$$e^{\hat{\omega}t} = I + \hat{\omega}\sin(t) + \hat{\omega}^2(1 - \cos(t)) \tag{2.13}$$

2.1.2 Rigid Motion in $\mathbb{R}^3$

2.1.2.1 Matrix Representation

As stated earlier, rigid body motion is composed of a rotation and translation. For $\mathbf{x}, \mathbf{x}', \mathbf{t} \in \mathbb{R}^3$ and $\mathbf{R} \in SO(3)$ this can be written as

$$\mathbf{x}' = R\mathbf{x} + \mathbf{t} \tag{2.14}$$

Using the homogeneous representation $\overline{\mathbf{x}}$ of $\mathbf{x}$

$$\overline{\mathbf{x}} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \tag{2.15}$$

we can write (2.14) as

$$\overline{\mathbf{x}'} = \begin{pmatrix} \mathbf{x}' \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \overline{\mathbf{x}} \tag{2.16}$$

So in homogeneous space, a rigid body motion is performed by multiplying the homogeneous representation $\overline{\mathbf{x}}$ of a point $\mathbf{x}$ with the 4×4 motion matrix $\mathbf{M}$ where

$$\mathbf{M} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \tag{2.17}$$

All such matrices representing rigid body motions belong to a group denoted by $SE(3)$ or the "special Euclidean" group. Here too, the term "special" signifies that the matrix $\mathbf{R} \in SO(3)$ in $\mathbf{M}$ is a special orthogonal matrix.

$$SE(3) = \left\{ \mathbf{M} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} | \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\} \in \mathbb{R}^{4\times4} \tag{2.18}$$

Just as for the case of rotational motion in $\mathbb{R}^3$, rigid body motion also has an alternate exponential representation.

2.1.2.2 Canonical Exponential Coordinates Representation

Similar to the exponential map from $so(3)$ to $SO(3)$, there exists an exponential map from the tangential space $se(3)$ to the group of rigid transformation matrices $SE(3)$ where

$$se(3) := \{(\mathbf{v}, \hat{\omega}) | \mathbf{v} \in \mathbb{R}^3, \hat{\omega} \in so(3)\}\cdot \tag{2.19}$$

Elements of $se(3)$ are called *twists* and can be written as

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & \mathbf{v} \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{2.20}$$

$\xi := (\mathbf{v}, \omega) \in \mathbb{R}^6$ are called the *twist coordinates* of the twist $\hat{\xi} \in se(3)$. Given a twist $\hat{\xi} \in se(3)$, the exponential map is given by

$$e^{\hat{\xi}} = \begin{bmatrix} e^{\hat{\omega}} & (\mathbf{I} - e^{\hat{\omega}})\hat{\omega}v + \omega\omega^T v \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.21}$$

where $e^{\hat{\omega}}$ is computed using Rodrigues' formula (equation (2.13)).

Twists are infinitesimal generators of screw motions and provide a means for representing *combined* rotation and translation. They can be exploited for numerical pose recovery and for modelling joints in 3D space. For a detailed and accessible introduction to rigid body motions and the proofs and derivations for the exponential representations of rotation and screw motion, the reader is referred to chapter 2 of [49] and [46].

2.2 Polynomials

A polynomial is a mathematical expression involving a sum of powers in one or more variables multiplied by coefficients. A 2D polynomial $f(x_1, x_2)$ of degree 3 can be written as

$$\begin{aligned} f(\mathbf{x}) = &\underbrace{a_{00}}_{H_0} + \underbrace{a_{10}x_1 + a_{01}x_2}_{H_1(x)} \\ &+ \underbrace{a_{20}x_1^2 + a_{11}x_1x_2 + a_{02}x_2^2}_{H_2(x)} \\ &+ \underbrace{a_{30}x_1^3 + a_{21}x_1^2x_2 + a_{12}x_1x_2^2 + a_{03}x_2^3}_{H_3(x)} \\ = &\sum_{r=0}^{3} H_r(\mathbf{x}) \end{aligned} \tag{2.22}$$

The individual summands such as $a_{21}x_1^2x_2$ are called *monomials*, whereas the products of the variables without the coefficients, such as $x_1^2x_2$ are called *terms*[2]. A monomial can be written as $a_\alpha \mathbf{x}^\alpha$ where α is a *multi-index* of the term's powers. For example, for the monomial $a_{21}x_1^2x_2$, the term powers are $(2, 1)$ and hence α can be written as the vector

[2] also called the *monomial basis*

$[2,1]^t$. $|\alpha|$ is the sum of the powers and is called the *degree* of the monomial. So the degree of the monomial $a_{21}x_1^2x_2$ is $2+1=3$. The degree of the polynomial is the highest degree amongst its monomials.

When all monomials in a polynomial are of the same degree r, the polynomial is called a *form* of degree r. Any polynomial of degree d can be decomposed into forms of degree $\leq d$ as shown in equation (2.22) where each $H_r(\mathbf{x})$ is a form of degree r since it contains terms of degree r only. In a form of degree r, there are

$$h_r = \binom{n+r-1}{n-1}$$

monomials (and hence coefficients) and in a polynomial of degree d, there are

$$h = h_d + h_{d-1} + \ldots + h_0 = \binom{n+d}{n}$$

coefficients. The coefficients F_α of the polynomial f of degree d are equal to the partial derivatives of order d evaluated at the origin:

$$F_\alpha = \frac{\partial^{\alpha_1+\ldots+\alpha_n} f}{\partial x_1^{\alpha_1} \ldots \partial x_n^{\alpha_n}}|_{\mathbf{x}=0}$$

In vector notation a polynomial f consisting of k monomials can be written as

$$f = \mathbf{m}^T\mathbf{a} \tag{2.23}$$

where $\mathbf{m}$ is the $k\times 1$ column vector of terms and $\mathbf{a}$ is the $k\times 1$ column vector of polynomial coefficients. For example, a 2D polynomial f of degree 3 consists of $k = \binom{2+3}{2} = 10$ monomials and can be written as

$$f(x,y) = a_{00} + a_{10}x + a_{01}y + a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{30}x^3 + a_{21}x^2y + a_{12}xy^2 + a_{03}y^3$$

$$= \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 & x^3 & x^2y & xy^2 & y^3 \end{bmatrix} \begin{bmatrix} a_{00} \\ a_{10} \\ a_{01} \\ a_{20} \\ a_{11} \\ a_{02} \\ a_{30} \\ a_{21} \\ a_{12} \\ a_{03} \end{bmatrix} = \mathbf{m}^T\mathbf{a}$$

Extension to nD polynomial of degree d is straight-forward.

2.3 Elimination Theory

Elimination theory is a classical branch of mathematics that was introduced for solving systems of polynomial equations [68]. It is a central piece of algebraic geometry that

deals with conditions and methods for eliminating variables to get equivalent problem formulations. *Elimination is the algebraic counterpart of the geometric concept of projection.* By eliminating variables, an algebraic variety is projected onto a lower dimensional subspace [19]. The basic idea is to eliminate variables from a given set of polynomials to obtain a new set of polynomials whose vanishing is a necessary (and sometimes sufficient) condition for solutions to exist.

Elimination of n variables from a system of $n+1$ polynomial equations yields a single polynomial called the *resultant* of the system. For the system to have any solutions, the vanishing of such a resultant is a necessary condition. If a certain monomial ordering is used then the resultant of a polynomial system is an element of the Gröbner basis of the ideal represented by the system [19][38]. However, computing Gröbner bases is an expensive procedure and not feasible for resultant computations in practical problems.

The more effective and popular alternative is a determinantal formulation of elimination. In this approach a matrix is constructed from the coefficients of the polynomials such that the determinant of this matrix is the resultant (or a multiple of the resultant).

2.3.1 Gröbner Bases

The Gröbner bases approach views polynomial systems as algebraic ideals and their common solutions as algebraic varieties. In this section we will follow the treatment of Gröbner basis in [19].

Definition 2.2. *A subset $I \subset k[x_1, \ldots, x_n]$ is an **ideal** if it satisfies:*

1. $0 \in I$.
2. *If $p_1, p_2 \in I$, then $p_1 + p_2 \in I$.*
3. *If $p_1 \in I$ and $p_3 \in k[x_1, \ldots, x_n]$, then $p_3 p_1 \in I$.*

The definition of an ideal is analogous to the concept of a subspace in linear algebra. Both are closed under addition and multiplication with the exception that, for a subspace, we multiply by scalars, whereas for an ideal, we multiply by polynomials. A variety $\mathbf{V}(p_1, \ldots, p_m)$ is the set of all solutions of the system of equations $p_1(x_1, \ldots, x_n) = \cdots = p_m(x_1, \ldots, x_n) = 0$.

Definition 2.3. *Let k be a field and $p_1, \ldots, p_m$ be a finite set of polynomials in $k[x_1, \ldots, x_n]$. Then*

$$\mathbf{V}(p_1, \ldots, p_m) = \{(a_1, \ldots, a_n) \in k^n : p_i(a_1, \ldots, a_n) = 0 \text{ for all } 1 \leq i \leq m\}.$$

*$\mathbf{V}(p_1, \ldots, p_m)$ is the **variety** defined by $p_1, \ldots, p_m$.*

Given a finite set of polynomials $p_1, \ldots, p_m \in k[x_1, \ldots, x_n]$, the **ideal generated by** $p_1, \ldots, p_m$ is written as

$$\langle p_1, \ldots, p_m \rangle = \left\{ \sum_{i=1}^{m} q_i p_i : q_1, \ldots, q_m \in k[x_1, \ldots, x_n] \right\}.$$

Varieties are *determined* by ideals.

Proposition 1. *If* $\mathbf{I} = \langle p_1, \ldots, p_m \rangle$, *then* $\mathbf{V}(I) = \mathbf{V}(p_1, \ldots, p_m)$.

Proof. See [19]. □

This shows that even though a nonzero ideal always contains infinitely many different polynomials, the set $\mathbf{V}(I)$ can still be defined by a finite set of polynomial equations. The polynomials $(p_1, \ldots, p_m)$ therefore form a basis for the ideal $I = \langle p_1, \ldots, p_m \rangle$. However, it is not the unique basis. $\langle p_1, \ldots, p_m \rangle$ can have other bases too. This is where Gröbner bases come into the picture. Gröbner bases are special bases or generators of polynomial ideals that have easier and more useful computational properties compared to the ideals themselves. Hence a variety corresponding to an ideal can be computed more efficiently by replacing the ideal generators by simpler generators, i.e. a Gröbner basis.

If a certain variable ordering, known as the *elimination order*, is used then the resulting Gröbner basis (called the *elimination ideal*) contains the resultant as one of its elements. Example 1 illustrates elimination using a Gröbner basis and also shows that elimination of variables implies projection onto a lower dimensional subspace.

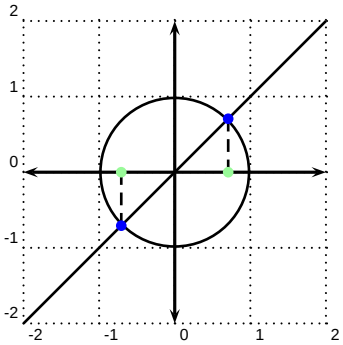


Figure 2.3: *Elimination implies projection. The circle and the line can be represented as a polynomial ideal I whose common solutions (shown in blue) constitute the variety* $\mathbf{V}(I)$ *whose projection onto the x-axis (shown in green) can be obtained by eliminating the variable y from the ideal I.*

Example 1. *Figure 2.3 shows a circle of radius* 1 *centered at the origin and the line* $y = x$. *They can be represented as a polynomial ideal I*

$$I =< x^2 + y^2 - 1, y - x >$$

whose common solutions (shown in blue) constitute the variety $\mathbf{V}(I)$ *whose projection onto the x-axis (shown in green) can be obtained by eliminating the variable y from the ideal I. One can solve the system to get* $\mathbf{V}(I) = \{(-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}), (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})\}$ *and the projection* $\pi(\mathbf{V}(I)) = \{-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\}$. *We can use* SINGULAR[3] *to compute the Gröbner basis for I and show that it contains a polynomial that represents the projection* $\pi(\mathbf{V}(I))$.

```
> ring  r=0,(y,x),lp; //declare polynomial ring Q[x,y] using lex ordering
> ideal I=x^2+y^2-1,x-y; //declare ideal I
> ideal G=groebner(I); //compute the Groebner basis
> G; //display the Groebner basis
G[1]=2x^2-1
G[2]=y-x
```

[3] www.singular.uni-kl.de

The Gröbner basis contains two elements. The first element $2x^2 - 1$ *can be solved for* x *to give* $\{-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\}$ *which is the same as* $\pi(\mathbf{V}(I))$*. The point to be noted is that* **by eliminating** y **from the ideal** I**, we obtained a new polynomial that is the projection of a 2-dimensional variety** $\mathbf{V}(I)$ **onto the 1-dimensional x-axis.**

For this simple example, Gröbner basis computation is not necessary. Instead, one can eliminate y by substituting $y = x$ in the equation of the circle to obtain $2x^2 - 1$ as before. However, for complex examples involving more polynomials and in higher degrees, such elimination by substitution is not feasible.

In this work we treat Gröbner bases as a black-box and hence we do not explain how a Gröbner basis is constructed from a polynomial set. The main algorithm for doing so is the Buchberger algorithm. The interested reader can find an explanation of ideals, varieties, the Buchberger algorithm and the use of Gröbner bases for elimination in [19],[20].

Even though Gröbner bases provide an elegant theory for dealing with geometric objects algebraically, they are not the most favoured choice when it comes to elimination. This is inspite of the increase in computational power over the years and the development of highly tuned Gröbner bases systems like `SINGULAR`. The bottleneck is that Gröbner basis computation may run into a phenomenon called *intermediate expression swell* whereby even though the original polynomial set *and* the final Gröbner basis might have modest size (i.e. few low degree polynomials), the Buchberger algorithm may produce an enormous number of high degree intermediate polynomials with huge coefficients [23]. Mayr and Meyer [47] showed that explosions in polynomial degrees during Gröbner basis computations are inevitable. Generally, Gröbner bases are inferior to resultant based elimination in terms of speed and memory usage [38][36].

Resultant based elimination has proved to be faster and more applicable in problems of practical interest. A major drawback in all current resultant based approaches is the existence of spurious factors which need to be factored out heuristically to obtain the correct irreducible resultant of the system.

2.3.2 Resultants

The resultant of a polynomial system is a polynomial in the coefficients of the original polynomials. The original variables do not appear in the resultant, hence they are *eliminated*. The key idea in resultant based elimination is that a system $\mathbf{S}$ of polynomials has common roots if and only if the resultant $R(\mathbf{S})$ vanishes [54]. In this way, the expression $R = 0$ is an equivalent formulation of the problem expressed by $\mathbf{S} = \mathbf{0}$. A simple example of resultant based elimination is that for a square system of homogeneous linear equations, a necessary and sufficient condition for a non-trivial[4] solution to exist is that the determinant of this system vanishes. Consider the following system of linear equations.

$$\begin{aligned} x - y - 3 &= 0 \\ -x - y &= 0 \\ x - \frac{3}{2} &= 0 \end{aligned}$$

[4]not all zeros.

Its homogeneous representation with homogenising variable w is

$$\begin{aligned} x - y - 3w &= 0 \\ -x - y &= 0 \\ x - \frac{3}{2}w &= 0 \end{aligned}$$

which can be written as

$$\begin{bmatrix} 1 & -1 & -3 \\ -1 & -1 & 0 \\ 1 & 0 & -\frac{3}{2} \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

This system is satisfied when all of x, y and w are 0. This is the trivial solution with no counterpart in affine space. For a non-trivial solution to exist, the necessary and sufficient condition is the vanishing of the determinant of the system, i.e. the determinant of the 3×3 system matrix shown above. Since the determinant of this matrix is 0, we can expect to find a solution for the system. Indeed, the point $(\frac{3}{2}, -\frac{3}{2})$ satisfies all 3 equations of the original system.

This method works for linear equations. For systems of higher-order polynomial equations, the necessary and sufficient condition is vanishing of resultants. There exist many resultant formulations such as Sylvester [36][68], Macaulay [69], Dixon [39][37][38][18] and Sparse [25][24]. Here, we will only describe the Sylvester resultant for introductory purposes and the Dixon resultant since it has been shown to be more efficient than other formulations for a wide variety of applications [18].

Starting with the simplest case of 2 univariate polynomials, a necessary and sufficient condition for their solution to exist is the vanishing of the so-called *Sylvester resultant* presented next in Section 2.3.2.1.

Remark Although current literature employs terms like Sylvester resultant, Macaulay resultant, Dixon resultant, *etc.*, none[5] of them yields the exact resultant in all cases. The determinant for all such methods is a multiple of the exact resultant which then needs to be factored out. For this reason, we will follow [39] in calling the determinants *projection operators*. With this terminology, technically all the following 'resultants' should be called projection operators but we stick with the term resultant to conform with the literature.

2.3.2.1 The Sylvester Resultant

Given a system 2 univariate polynomials $p_1(x), p_2(x) \in \mathcal{Q}[x]$ of degrees n and m respectively,

$$\begin{aligned} p_1(x) &= \sum_{\alpha=0}^{n} p_{1_\alpha} x^\alpha, \\ p_2(x) &= \sum_{\alpha=0}^{m} p_{2_\alpha} x^\alpha \end{aligned}$$

[5] Macaulay resultant is the exact resultant for the case of generic homogeneous polynomials.

the Sylvester resultant, $R(p_1, p_2)$, of the system is given by the determinant of the Sylvester matrix

$$R = \begin{vmatrix} p_{1_0} & 0 & 0 & \cdots & 0 & p_{2_0} & 0 & 0 & \cdots & 0 \\ p_{1_1} & p_{1_0} & 0 & \cdots & 0 & p_{2_1} & p_{2_0} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{1_n} & p_{1_{n-1}} & p_{1_{n-2}} & \cdots & p_{1_{n-m+1}} & p_{2_n} & p_{2_{n-1}} & p_{2_{n-2}} & \cdots & p_{2_0} \\ 0 & p_{1_n} & p_{1_{n-1}} & \cdots & p_{1_{n-m+2}} & p_{2_{n+1}} & p_{2_n} & p_{2_{n-1}} & \cdots & p_{2_1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & p_{1_n} & 0 & 0 & 0 & \cdots & p_{2_m} \end{vmatrix}.$$

The polynomials p_1 and p_2 have common roots when the resultant R vanishes, i.e. $R = 0$. Note that the R does not contain the original variable x which has been eliminated.

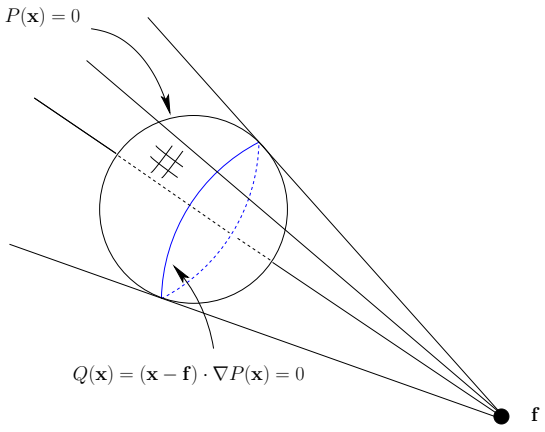


Figure 2.4: *A sphere represented as an implicit algebraic surface $P(\mathbf{x}) = 0$ and 4 lines that are part of its tangent cone. The blue points satisfy the tangency condition $Q(\mathbf{x}) = (\mathbf{x} - \mathbf{f}) \cdot \nabla P(\mathbf{x}) = 0$ with respect to the point $\mathbf{f}$.*

Example 2. *As an example we will use the Sylvester resultant to compute the tangent cone of an arbitrary sphere of radius r centered at (a, b, c) when viewed from the origin. The* tangent cone *of an object viewed from a point $\mathbf{f}$ consists of all lines tangent to the object's surface and passing through $\mathbf{f}$ as shown in Figure 2.4. A sphere of radius r centered at (a, b, c) can be represented as an algebraic surface given by the polynomial*

$$P(\mathbf{x}) = (x - a)^2 + (y - b)^2 + (z - c)^2 - r^2 = 0.$$

Any point $\mathbf{x}$ on the sphere that also lies on a line tangent to the sphere and passing through $\mathbf{f}$ satisfies the tangency condition

$$Q(\mathbf{x}) = (\mathbf{x} - \mathbf{f}) \cdot \nabla P(\mathbf{x}) = 0.$$

The blue points on the sphere in Figure 2.4 satisfy the tangency condition. If we introduce a parameter t and write the sphere equation as $P(\mathbf{x}t) = 0$ and the tangency condition as $Q(\mathbf{x}t) = 0$, then it can be seen that as t is varied, the blue points satisfying the tangency condition move along the tangent lines. Equivalently, t parameterises the tangent cone.

For convenience and w.l.o.g[6]*, we can set* $\mathbf{f}$ *as the origin. So, when viewed from the origin* ($\mathbf{f} = \mathbf{0}$)*, the tangent cone of our sphere satisfies the parametric polynomial system* $\mathcal{S} = \{P(\mathbf{x}t), Q(\mathbf{x}t)\}$ *which can be written as*

$$\begin{aligned} P(\mathbf{x}t) &= (xt-a)^2 + (yt-b)^2 + (zt-c)^2 - r^2 \\ &= (x^2+y^2+z^2)t^2 - 2(ax+by+cz)t + (a^2+b^2+c^2-r^2) = 0, \\ Q(\mathbf{x}t) &= \mathbf{x} \cdot \nabla P(\mathbf{x}t) \\ &= 2(x^2+y^2+z^2)t^2 - 2(ax+by+cz)t = 0\cdot \end{aligned}$$

P and Q can be treated as univariate polynomials in t. Eliminating t from the system gives the equation of the tangent cone. The Sylvester matrix for this system is given by

$$\begin{bmatrix} x^2+y^2+z^2 & -2ax-2by-2cz & a^2+b^2+c^2-r^2 & 0 \\ 0 & x^2+y^2+z^2 & -2ax-2by-2cz & a^2+b^2+c^2-r^2 \\ 2x^2+2y^2+2z^2 & -2ax-2by-2cz & 0 & 0 \\ 0 & 2x^2+2y^2+2z^2 & -2ax-2by-2cz & 0 \end{bmatrix}$$

which has determinant

$$\begin{aligned} &4 \cdot (x^2+y^2+z^2) \cdot (a^2+b^2+c^2-r^2) \\ &\cdot (a^2z^2 + a^2y^2 - 2byax - 2axcz - z^2r^2 - y^2r^2 + z^2b^2 + y^2c^2 - 2bycz + x^2b^2 + x^2c^2 - x^2r^2)\cdot \end{aligned}$$

The resultant of the polynomial system is that factor of the determinant which contains all the variables. *Therefore, it can be seen that in the above expression the last factor*

$$a^2z^2 + a^2y^2 - 2byax - 2axcz - z^2r^2 - y^2r^2 + z^2b^2 + y^2c^2 - 2bycz + x^2b^2 + x^2c^2 - x^2r^2$$

is the resultant. The tangent cone is given by the vanishing resultant. So for a sphere of radius 1 *centered at* $(0,0,5)$ *and viewed from the origin we substitute* (a,b,c,r) *by* $(0,0,5,1)$ *in the expression for the resultant to get the implicit equation for the tangent cone as*

$$C(\mathbf{x}) = 24x^2 + 24y^2 - z^2 = 0\cdot$$

2.3.2.2 The Dixon Resultant

The Sylvester matrix of two polynomials p_1, p_2 of degrees n, m is an $(n+m) \times (n+m)$ matrix. In contrast, the Cayley-Dixon[7] matrix for the same problem is an $n \times n$ matrix. Cayley's formulation for the resultant of two univariate polynomials is as follows. Replace x by α in both $p_1(x)$ and $p_2(x)$ to get the polynomials $p_1(\alpha)$ and $p_2(\alpha)$. Then form the *cancellation matrix* whose determinant

$$\triangle(x,\alpha) = \begin{vmatrix} p_1(x) & p_2(x) \\ p_1(\alpha) & p_2(\alpha) \end{vmatrix}$$

[6] without loss of generality

[7] The method was originally developed by Bezout in 1779, Cayley reformulated it and Dixon extended it to the bivariate case.

is a polynomial in x and α and vanishes when $x = \alpha$. This means that the determinant $\triangle(x, \alpha)$ has $(x - \alpha)$ as a factor. The polynomial

$$\delta(x, \alpha) = \frac{\triangle(x, \alpha)}{(x - \alpha)} = \frac{p_1(x)p_2(\alpha) - p_2(x)p_1(\alpha)}{(x - \alpha)}$$

is then an $n - 1$ degree polynomial in α and is symmetric in x and α. It is called the *Dixon polynomial*. It vanishes at every common zero x_0 of $p_1(x)$ and $p_2(x)$ regardless of the value of α. This implies that at $x = x_0$, the coefficient of every power product of α in $\delta(x, \alpha)$ should be 0. Since $\delta(x, \alpha)$ is an $n - 1$ degree polynomial in α, the number of power products of α is n (i.e. α^i for each $0 \leq i \leqslant n-1$). This gives n equations which are polynomials in x with maximum degree $n - 1$. Any common zero of $p_1(x)$ and $p_2(x)$ is a solution of these n polynomial equations and these n equations have a common solution if the determinant of their coefficient matrix vanishes. The coefficient matrix is called the Dixon matrix since Dixon later generalised Cayley's formulation.

Example 3. *We computed the tangent cone of a sphere using the Sylvester resultant in example 2. We now recompute it using the Dixon resultant. We treat the system as consisting of polynomials in t and introduce α as the new variable to obtain the cancellation matrix*

$$\begin{bmatrix} P(t) & Q(t) \\ P(\alpha) & Q(\alpha) \end{bmatrix}$$

from which we can obtain the following Dixon polynomial:

$$\begin{aligned}
\delta(t, \alpha) =& \frac{\triangle(t, \alpha)}{(t - \alpha)} \\
=& -2x^2ta^2 - 2a^2ty^2 + 2a^3x + 2b^3y + 2c^3z - 2a^2tz^2 + 2a^2yb + 2a^2zc - 2y^2tb^2 \\
& - 2b^2tx^2 - 2b^2tz^2 + 2b^2xa + 2b^2zc - 2z^2tc^2 - 2c^2tx^2 - 2c^2ty^2 + 2c^2xa + 2c^2yb \\
& + 2r^2tx^2 + 2r^2ty^2 + 2r^2tz^2 - 2r^2xa - 2r^2yb - 2r^2zc + 2x^3ta\alpha + 2y^3tb\alpha \\
& + 2z^3tc\alpha - 2a^2\alpha x^2 - 2a^2\alpha y^2 - 2a^2\alpha z^2 - 2b^2\alpha x^2 - 2b^2\alpha y^2 - 2b^2\alpha z^2 \\
& - 2c^2\alpha x^2 - 2c^2\alpha y^2 - 2c^2\alpha z^2 + 2r^2\alpha x^2 + 2xta\alpha y^2 + 2xta\alpha z^2 \\
& + 2ytb\alpha x^2 + 2ytb\alpha z^2 + 2ztc\alpha x^2 + 2ztc\alpha y^2 + 2r^2\alpha y^2 + 2r^2\alpha z^2 \cdot
\end{aligned}$$

Recall that for all common zeros t_0 of P and Q, the Dixon polynomial has to vanish regardless of the value of α. This implies that for all t_0, the coefficients of powers of α become zero. This yields the following 2 $(= n + 1)$ equations (one each for α^0 and α^1):

$$\begin{aligned}
&(-2b^2x^2 - 2c^2x^2 - 2b^2y^2 - 2b^2z^2 + 2r^2y^2 - 2c^2z^2 - 2c^2y^2 + 2r^2z^2 - 2a^2y^2 + 2r^2x^2 - 2a^2z^2 - 2a^2x^2)t \\
&+2a^2zc + 2a^2yb + 2c^2yb + 2c^3z + 2a^3x + 2c^2xa + 2b^3y + 2b^2zc + 2b^2xa - 2r^2yb - 2r^2xa - 2r^2zc = 0
\end{aligned}$$

$$\begin{aligned}
&(2x^3a + 2xaz^2 + 2z^3c + 2y^3b + 2xay^2 + 2zcy^2 + 2ybx^2 + 2ybz^2 + 2zcx^2)t \\
&-2b^2x^2 - 2c^2x^2 - 2b^2y^2 - 2b^2z^2 + 2r^2y^2 - 2c^2z^2 - 2c^2y^2 + 2r^2z^2 - 2a^2y^2 + 2r^2x^2 - 2a^2z^2 - 2a^2x^2 = 0
\end{aligned}$$

which can be written as

$$\begin{bmatrix}
-2b^2x^2 - 2c^2x^2 - 2b^2y^2 - 2b^2z^2 + 2r^2y^2 - 2c^2z^2 & 2a^2zc + 2a^2yb + 2c^2yb + 2c^3z + 2a^3x + 2c^2xa \\
-2c^2y^2 + 2r^2z^2 - 2a^2y^2 + 2r^2x^2 - 2a^2z^2 - 2a^2x^2 & +2b^3y + 2b^2zc + 2b^2xa - 2r^2yb - 2r^2xa - 2r^2zc \\
2x^3a + 2xaz^2 + 2z^3c + 2y^3b + 2xay^2 + 2zcy^2 & -2b^2x^2 - 2c^2x^2 - 2b^2y^2 - 2b^2z^2 + 2r^2y^2 - 2c^2z^2 \\
+2ybx^2 + 2ybz^2 + 2zcx^2 & -2c^2y^2 + 2r^2z^2 - 2a^2y^2 + 2r^2x^2 - 2a^2z^2 - 2a^2x^2
\end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \cdot$$

The 2×2 matrix in the above expression is the Dixon matrix for the polynomial system $\{P, Q\}$. For the Dixon polynomial $\delta(t, \alpha)$ to vanish the determinant of the Dixon matrix (i.e. the Dixon projection operator) has to vanish. The Dixon projection operator is

$$4 \cdot (x^2 + y^2 + z^2) \cdot (a^2 + b^2 + c^2 - r^2)$$
$$\cdot (a^2z^2 + a^2y^2 - 2byax - 2axcz - z^2r^2 - y^2r^2 + z^2b^2 + y^2c^2 - 2bycz + x^2b^2 + x^2c^2 - x^2r^2) \cdot$$

The last factor contains all the variables and is therefore the exact resultant. Note that it is the same as the Sylvester resultant obtained earlier.

Dixon extended Caley's formulation of the two polynomial univariate case to the three polynomial bivariate case. Kapur et al. [39] generalised[8] it further to the multivariate case. We present the multivariate formulation for computing the Dixon projection operator.

Let $X = \{x_1, \ldots, x_n\}$ be a set of n variables and $P = \{p_1, \ldots, p_{n+1}\}$ a set of $n + 1$ polynomials in $k[x_1, \ldots, x_n]$. Form the set $\overline{X} = \{\overline{x}_1, \ldots, \overline{x}_n\}$ of n new variables. Construct the $(n + 1) \times (n + 1)$ cancellation matrix C_P of P:

$$C_P = \begin{bmatrix} p_1(x_1, x_2, \ldots, x_n) & \ldots & p_n(x_1, x_2, \ldots, x_n) \\ p_1(\overline{x}_1, x_2, \ldots, x_n) & \ldots & p_n(\overline{x}_1, x_2, \ldots, x_n) \\ p_1(\overline{x}_1, \overline{x}_2, \ldots, x_n) & \ldots & p_n(\overline{x}_1, \overline{x}_2, \ldots, x_n) \\ \vdots & \ldots & \vdots \\ p_1(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n) & \ldots & p_n(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n) \end{bmatrix}. \tag{2.24}$$

Similar to Cayley's univariate case, $(x_i - \overline{x}_i)$ is a zero of $|C_P|$ for all $1 \leq i \leq n$. As an example, if we set $x_1 = \overline{x}_1$, the first two rows of C_P will become identical, thus causing the determinant to become zero. Therefore $\prod_{i=1}^{n}(x_i - \overline{x}_i)$ divides $|C_P|$ and can be cancelled out. This gives us the Dixon polynomial

$$\delta_P = \frac{|C_P|}{\prod_{i=1}^{n}(x_i - \overline{x}_i)}. \tag{2.25}$$

We can treat δ_P as a polynomial in $\overline{X}$ and extract the set $\overline{\mathcal{F}}$ of its coefficients which are polynomials in X. The Dixon matrix D_P is the coefficient matrix of $\overline{\mathcal{F}}$. Being a coefficient matrix, D_P will not contain any of the variables in X. Hence the variables in X will be eliminated in the expression for the Dixon projection operator $|D_P|$.

[8] Dixon alluded to such a generalisation which is why the formulation is known under Dixon's name

Chapter 3

Surface Representations

3.1 Surface Representations

3D surfaces can be described mathematically in different forms. The most common ones are given in

- Explicit form: $z = f(x, y)$,
- Parametric form: $\mathbf{p} = [x(u, v), y(u, v), z(u, v)]$,
- Implicit form: $f(x, y, z) = 0$,

Table 3.1 shows the explicit, parametric and implicit representations for a sphere of radius r centered at the origin. The explicit representation was used to render the sphere shown in Figure 3.1. In the following, each representation is described in more detail along with a discussion of the advantages and disadvantages of using a particular representation.

Name	**Representation**
Explicit	$z = f(x, y) = \sqrt{r^2 - x^2 - y^2}$ where $x, y \in [-r, r]$ are given as explicit point pairs.
Parametric	$\mathbf{p} = [x(u, \theta), y(u, \theta), z(u, \theta)]$ where $u \in [-r, r]$, $\theta \in [0, 2\pi]$, $x(u, \theta) = \sqrt{r^2 - u^2}\cos(\theta)$, $y(u, \theta) = \sqrt{r^2 - u^2}\sin(\theta)$, $z(u, \theta) = u$
Implicit	$f(x, y, z) = x^2 + y^2 + z^2 - r^2 = 0$

Table 3.1: *The explicit, parametric and implicit representations for the boundary of a sphere of radius r centered at the origin.*

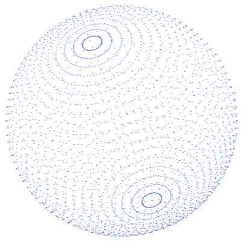


Figure 3.1: *A sphere rendered using the explicit representation.*

3.1.1 Explicit Surfaces

In an explicit representation one explicitly writes down the points that belong to the surface. Popular explicit representations include triangulated surfaces. Graphics designers and animators have tended to prefer explicit models because with them it is relatively easy to control shapes in an intuitively pleasing way. However, they are not well-suited for fitting and automated modelling purposes [60]. Also, it is difficult to confirm if an arbitrary point is on, inside or outside an explicit surface. Since it describes only explicitly where the *available* data points are, interpolation and computation of surface gradients is problematic. Another disadvantage of explicit representations is that tracking of large deformations and topological changes of the surface is quite difficult [73]. For instance, since connectivity of the explicit points determines the triangulation of a 3D surface, changes in connectivity due to surface deformations can pose problems for triangulation algorithms. Figure 3.2 shows an object represented as an explicit set of points and the associated connected mesh.

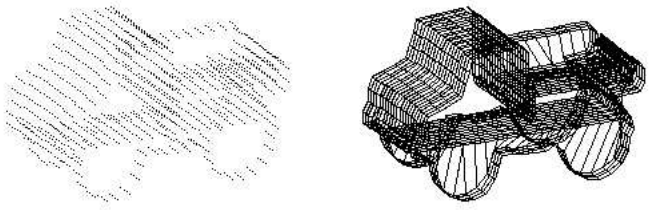


Figure 3.2: *An object represented as an explicit set of points and the associated connected mesh.*

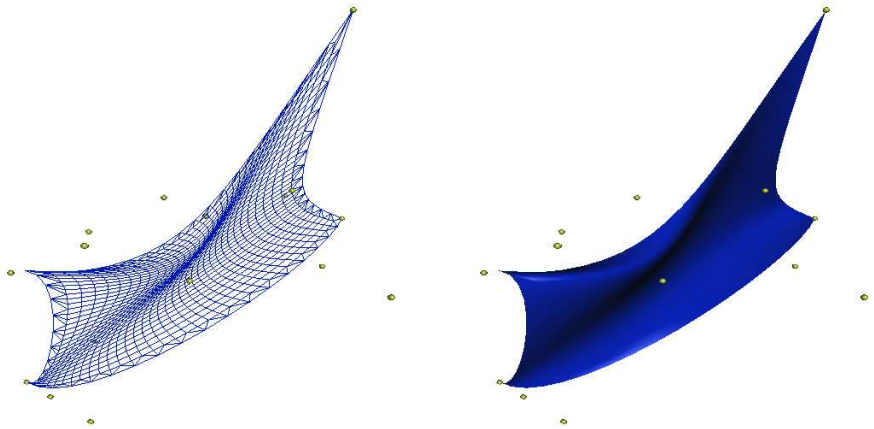
3.1.2 Parametric Surfaces

In parametric form, points on the surface can be specified in terms of the parametric variables (u, v) which can be restricted, without loss of generality, to lie on the unit square [0,1]×[0,1]. Parametric surfaces have the useful ability to identify specific locations on a surface. A common parametric form is the nonuniform rational B-spline (NURBS) (Figure 3.3) defined as

$$S(u,v) = \sum_i \sum_j B^{\mathrm{h}}_{i,j} N_{i,k}(u) M_{i,l}(v) \tag{3.1}$$

where $N_{i,k}(u)$ and $M_{i,l}(v)$ are rational B-spline functions of order k and l, and $B^{\mathrm{h}}_{i,j}$ are the homogeneous coordinates of control points [13].

Parametric forms can be used to represent complex object geometries and generate realistic views. The use of control points allows local finite control of the surface which allows accurate modelling of objects. Complex objects can be modelled using parametric patches that are joined together. However, fitting of parametric patches to arbitrary surface regions is not easy. Furthermore, control points determining the shape of a parametric surface are not easily detectable and non-unique. Campbell and Flynn [13] propose using parametric forms for initial object specification from which a polygonal mesh or other representation can then be generated.



(a) Mesh of a NURBS surface patch. (b) Shaded NURBS surface patch.

Figure 3.3: *A curved NURBS patch depicted as a mesh (a) and as a shaded surface (b). The control points shown in green provide local control over the surface.*

3.1.3 Implicit Surfaces

Implicit representations define objects implicitly as a particular isocontour (usually 0) of some function f. An implicit surface is mathematically defined as the set of points $\mathbf{x} \in \mathbb{R}^3$, satisfying the implicit function $f(\mathbf{x}) = 0$, where $f : \mathbb{R}^3 \to \mathbb{R}$. It is also called the zero set or kernel of f and sometimes written as $f^{-1}(0)$. Points on the inside and the outside of the surface are usually chosen to have opposite signs.

Implicit surface representations include some very powerful geometric properties. For example, it is trivial to determine if an arbitrary point is inside, outside or on the surface. This is difficult to do with explicit representations. Implicit functions make both simple Boolean operations such as surface intersection and union easy to apply. Constructive solid geometry (CSG) reduces to merely looking at the signs of implicit functions without the need to consider geometry or topology [71]. With implicit surfaces, surface gradient, normal and curvature as well as volume and surface integrals can be easily defined. Well-developed level set methods [52] can be used with implicit surfaces to handle dynamic surfaces with changing topology.

An implicit surface is quite different from explicit and parametric surfaces. Whereas explicit surfaces have specific point coordinates and parametric surfaces have specific surface coordinates, implicit surfaces allow no such surface navigation. However, the simple inside/outside tests associated with implicit surfaces make them ideal for shape transformation, collision detection, CSG, and blending. Implicit surfaces are smoother and more compact than explicit polygonal meshes but are harder to deform and visualise in real time since that requires finding the zero set of an implicit function in 3D space. As compared to parametric surfaces, they lack local surface control which makes them less suitable for modelling. However, they are easier to fit, easier to blend, and do not suffer from topology problems. They are well-suited for simulating physically based processes and for modelling smooth objects. They can be collided and deformed, with fusions and separations handled automatically [71].

Implicit surfaces have two important classes depending on the way the implicit function f is defined:

1. Blobby Surfaces
2. Algebraic Surfaces

3.1.3.1 Blobby Surfaces

For blobby or metaball surfaces the implicit function f is a sum of radially symmetric functions with Gaussian profiles. The general form is

$$f(\mathbf{x}) = -t + \sum_{i=1}^{n} h_i(\mathbf{x}) \tag{3.2}$$

where h_i describes the Gaussian profile of a blobby sphere and t represents the isocontour threshold. Typically, the blobby sphere function h_i is of the form

$$h_i(\mathbf{x}) = e^{|\mathbf{x}-\mathbf{c}_i|^2/\sigma_i^2} \tag{3.3}$$

where $\mathbf{c}_i$ is the center of the sphere and σ_i is the standard deviation of the Gaussian which controls the radius of the sphere. Figure 3.4 shows an implicit surface formed by three blobby spheres. One can notice the blobbiness via the smooth blending of the three individual surfaces into each other.

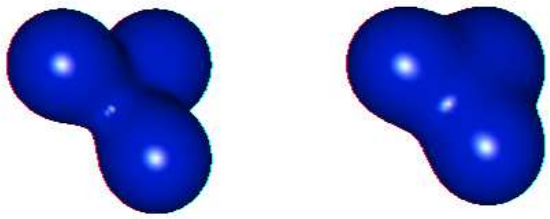


Figure 3.4: *An implicit surface constructed from 3 blobby spheres. The individual surfaces blend smoothly into each other as the spheres are brought closer on the right.*

3.1.3.2 Algebraic Surfaces

For algebraic surfaces the implicit function f is a polynomial expression in x, y and z. Figure 3.5 shows two algebraic surfaces formed using polynomials of degree 4. Algebraic surfaces can interpolate between data points in order to handle missing data and they also have a smoothing property for handling noise and perturbations in data [8][64]. Coefficients of polynomials representing algebraic entities encode useful information about

(a) $f(x, y, z) = x^3y + y^3z + z^3x = 0$ (b) $f(x, y, z) = x^4 + y^4 + z^4 - 200xyz = 0$

Figure 3.5: *Two algebraic implicit surfaces defined by the 4th degree polynomials.*

the represented entity [66]. This information includes

1. Algebraic invariants that help in classifying and recognising different objects [43][44][34][40].

2. Intrinsic surface geometry that helps in registration [32].

Algebraic implicit surfaces are particularly appealing in the context of spatial registration because of the second property. Polynomial coefficients of an algebraic entity can be used to estimate its center and orientation vectors which together make up the entity's *intrinsic reference frame*. This is explained in Section 4.2. Given such intrinsic information, the problem of registering two objects reduces to alignment of their intrinsic reference frames. This obviates the need for iteratively finding correspondences needed by explicit registration approaches such as ICP [6].

However, the property of algebraic surfaces that is useful in the projective registration domain is that image outlines of algebraic surfaces completely determine their projective geometry [22]. This implies that algebraic curve equations representing the image outlines of projections of algebraic surfaces can be computed given the algebraic surface equation and the camera projection matrix. Given an algebraic surface, elimination theory can be used to compute the surface's tangent cone [54] as viewed from a certain point. Figure 3.6 shows the tangent cone of a torus when viewed from the origin. Elimination

theory can also be used to compute the occluding contour of an algebraic surface's projection [42]. The silhouette-based 2D-3D pose estimation method in this thesis is based on this property.

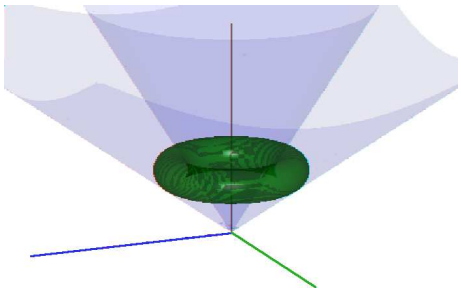


Figure 3.6: *Tangent cone of a torus viewed from the origin.*

To summarise, the main advantages of implicit surfaces include topological flexibility, compact representation and efficient computation of various surface operations. Moreover, algebraic implicit surfaces are particularly suited for the pose recovery problem. The interested reader is referred to [9] for a nice introduction to implicit surfaces. Table 3.2 presents a comparison of the different surface representation techniques.

	Explicit	**Parametric**	**Implicit**
Data Interpolation	Difficult	Easy	Easiest
Smoothness	No	Yes	Yes
Compact Representation	No	Yes	Yes
Visualisation	Easy	Easy	Difficult
Surface Operations	Bad	Good	Very Good
Topology Preserving Deformation	Difficult	Easy	Easiest
Local Shape Control	Yes	Yes	No
Gradient Computation	Bad	Good	Good

Table 3.2: *Comparison of surface representations*

3.1.3.3 Literature Overview of Implicit Surfaces

Even within the domain of implicit surface representations, various approaches exist in order to satisfy application specific requirements ranging from modelling to deforming.

Modelling and Fitting Ilic and Fua [60] present a technique for switching from explicit to implicit surfaces that takes advantage of the strengths of both approaches. Each explicit facet in a triangular mesh is replaced by a metaball whose parameters depend on the geometry of the explicit facet. In this way, explicit and implicit surface deformations can be performed in-tandem and implicit surfaces can share the advantage of explicit surfaces having more intuitive deformations. In contrast, variational [67] and algebraic [32] surfaces are controlled not only by the positions of the radial basis functions but also their weights which do not have a geometric interpretation. This makes in-tandem deformation of explicit and implicit surfaces difficult.

For fast surface reconstruction from 3D point sets, Zhao et al. [73] enclose 3D data by an implicit surface and then shrink it to wrap the data. For this they use a surface energy functional and level set methods.

Turk and O'Brien [67] introduced a different kind of surface creation method through variational interpolation of scattered data. The resulting surfaces are called variational implicit surfaces. A function f is represented through a linear combination of radial basis functions. The weights of the linear combination are computed by solving a scattered data interpolation problem. If the radial basis functions are chosen appropriately, then the computed weights also minimise an energy functional that penalises high curvature. This results in everywhere smooth surfaces. Variational implicit surfaces can easily be created from polygonal meshes and deformation is also easy.

Variational implicit surfaces as introduced by Turk and O'Brien are not suitable for large data sets ($N \gtrsim 2000$) since the fitting procedure involves a non-sparse system matrix and evaluation involves computing radial basis functions in the order of N. Carr et al. [14] improve upon Turk and O'Brien's variational implicit surfaces by using a reduced number of radial basis functions for fitting and using fast multipole methods for faster evaluation. They also improve upon fitting accuracy by using dynamic projection to generate valid off-surface points used in the fitting procedure.

In the domain of implicit polynomials, Taubin [66] reduced the non-linear least-squares problem of fitting implicit algebraic curves and surfaces to data to a generalised eigenvector problem. Blane et al. [8][7] introduced the 3L fitting algorithm that converted this non-linear problem to a linear least-squares problem. Kang et al. [35] reconstruct 3D surfaces from occluding contours of algebraic surfaces using a linear dual-surface approach that makes use of the duality between 3D points and tangent planes.

Applications The most popular applications of algebraic surfaces are in projective registration and invariant-based object recognition. They have already been presented in Section 1.1.

For spatial 3D-3D registration, Tarel et al. [32] fit implicit algebraic surfaces to 3D surface data using the 3L fitting method. This allows fast extraction of geometric surface information from the polynomial coefficients. Robustness of polynomial coefficients to occlusion and their consistency under rigid body motions that comes with 3L fitting, is important for the registration problem. The use of implicit algebraic surfaces allows them to control shape resolution by varying the degree of the polynomials. It also allows them to represent and register both open and closed shapes.

Chapter 4

Implicit Algebraic Curves and Surfaces

An implicit algebraic surface consists of the set of points where a polynomial f takes on the value 0. Specifically, it is the set of zeros $Z(f)$ of a smooth[1] polynomial function $f : \mathbb{R}^3 \to \mathbb{R}$ of three variables:

$$Z(f) = \{(x, y, z) : f(x, y, z) = 0\}\cdot$$

Similarly, an implicit algebraic 2D curve is the zero-set

$$Z(f) = \{(x, y) : f(x, y) = 0\}$$

of a smooth polynomial function $f : \mathbb{R}^2 \to \mathbb{R}$ of two variables and an implicit algebraic 3D curve is the intersection of two surfaces f_1 and f_2, i.e. the zero-set

$$Z(f) = \{(x, y, z) : f(x, y, z) = \mathbf{0}\}$$

of a two dimensional vector function $f : \mathbb{R}^3 \to \mathbb{R}^2$ of three variables

$$f(x, y, z) = \begin{bmatrix} f_1(x, y, z) \\ f_2(x, y, z) \end{bmatrix}.$$

More generally, curves and surfaces can be represented by functions belonging to a family parametrised by a finite number r of parameters. Let $\phi : \mathbb{R}^{r+n} \to \mathbb{R}^k$ be a smooth function and consider the maps $f : \mathbb{R}^n \to \mathbb{R}^k$ which can be written as

$$f(\mathbf{x}) \equiv \phi(\mathbf{u}, \mathbf{x})$$

for certain $\mathbf{u} = (u_1, \cdots, u_r)^T$. f can be written as $f = \phi_{\mathbf{u}}$. The $u_1, \cdots, u_r$ are called the *parameters* while the $x_1, \cdots, x_n$ are called the *variables*. The family of all such maps can be written as

$$\mathbb{F} = \{f : \exists \mathbf{u} \;\; f = \phi_{\mathbf{u}}\} \tag{4.1}$$

where ϕ is the *parametrisation* of the family $\mathbb{F}$. It can be seen that for $f \in \mathbb{F}$, $Z(f)$ is a 2D curve when $n = 2$ and $k = 1$, a surface when $n = 3$ and $k = 1$ and a 3D curve when $n = 3$ and $k = 2$.

[1] having continuous first- and second-order derivatives at every point

In the following, we explain how algebraic entities represented by implicit polynomials can be transformed by manipulating the polynomial coefficients. Then we explain how intrinsic geometry of $Z(f)$ can be extracted from the coefficients and how this leads to a simple method of implicit spatial registration that is a correspondence-free alternative to explicit registration methods such as ICP [6]. We end this chapter by presenting techniques for fitting implicit polynomials to data where we describe the 3L fitting algorithm.

4.1 Polynomial Transformation

The zero-set $Z(f)$ of a polynomial f can be transformed by manipulating the polynomial coefficients. There are two ways of manipulating polynomial coefficients to affect zero-set transformations, the matrix-based approach and the tensor-based approach. In the rest of this section, polynomials will be represented by lowercase letters of the Latin alphabet such as f whereas forms will be represented by Greek symbols such as ϕ.

4.1.1 Matrix-based Approach

If we define $F_\alpha = \alpha! a_\alpha$ where $\alpha!$ is a *multi-factorial*, then without loss of generality, a polynomial of degree, say 3, $f = \phi_0 + \phi_1 + \phi_2 + \phi_3$ can be rewritten as

$$\begin{aligned} f(\mathbf{x}) = & \underbrace{\frac{F_{00}}{0!0!}}_{\phi_0} + \underbrace{\frac{F_{10}}{1!0!}x_1 + \frac{F_{01}}{0!1!}x_2}_{\phi_1(x)} \\ & + \underbrace{\frac{F_{20}}{2!0!}x_1^2 + \frac{F_{11}}{1!1!}x_1x_2 + \frac{F_{02}}{0!2!}x_2^2}_{\phi_2(x)} \\ & + \underbrace{\frac{F_{30}}{3!0!}x_1^3 + \frac{F_{21}}{2!1!}x_1^2x_2 + \frac{F_{12}}{1!2!}x_1x_2^2 + \frac{F_{03}}{0!3!}x_2^3}_{\phi_3(x)} \\ = & \underbrace{F_{00}}_{\phi_0} + \underbrace{F_{10}x_1 + F_{01}x_2}_{\phi_1(x)} \\ & + \underbrace{\frac{1}{2}F_{20}x_1^2 + F_{11}x_1x_2 + \frac{1}{2}F_{02}x_2^2}_{\phi_2(x)} \\ & + \underbrace{\frac{1}{6}F_{30}x_1^3 + \frac{1}{2}F_{21}x_1^2x_2 + \frac{1}{2}F_{12}x_1x_2^2 + \frac{1}{6}F_{03}x_2^3}_{\phi_3(x)} \\ = & \sum_{r=0}^{3} \phi_r(\mathbf{x}). \end{aligned}$$

More generally, an nD polynomial in variables $\mathbf{x} = (x_1, \ldots, x_n)$ can be written as

$$f(\mathbf{x}) = \sum_\alpha \frac{1}{\alpha!} F_\alpha \mathbf{x}^\alpha$$

where the vector of non-negative integers $\alpha = (\alpha_1, \ldots, \alpha_n)$ is a multi-index of size $|\alpha| = \alpha_1 + \ldots + \alpha_n$, $\alpha! = \alpha_1! \ldots \alpha_n!$ is a multi-index factorial (or multi-factorial), F_α is a coefficient of degree $|\alpha|$, and $\mathbf{x}^\alpha = x_1^{\alpha_1} \ldots x_n^{\alpha_n}$ is a term of degree $|\alpha|$.

A 3D quadratic form can then be written as

$$\begin{aligned}\phi(x_1, x_2, x_3) &= a_{200}x_1^2 + a_{110}x_1x_2 + a_{101}x_1x_3 + a_{020}x_2^2 + a_{011}x_2x_3 + a_{002}x_3^2 \\ &= \frac{\Phi_{(2,0,0)}}{2}x_1^2 + \Phi_{(1,1,0)}x_1x_2 + \Phi_{(1,0,1)}x_1x_3 + \frac{\Phi_{(0,2,0)}}{2}x_2^2 + \Phi_{(0,1,1)}x_2x_3 + \frac{\Phi_{(0,0,2)}}{2}x_3^2 \\ &= \frac{1}{2}\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}\begin{pmatrix} \Phi_{(2,0,0)} & \Phi_{(1,1,0)} & \Phi_{(1,0,1)} \\ \Phi_{(1,1,0)} & \Phi_{(0,2,0)} & \Phi_{(0,1,1)} \\ \Phi_{(1,0,1)} & \Phi_{(0,1,1)} & \Phi_{(0,0,2)} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.\end{aligned}$$

Taubin and Cooper [66] represent the 3×3 symmetric matrix as $\Phi_{[1,1]}$. The purpose of constructing $\Phi_{[1,1]}$ is that under a linear transformation $\mathbf{x}' = A\mathbf{x}$, this matrix of coefficients transforms as

$$\Phi'_{[1,1]} = A^{-T}\Phi_{[1,1]}A^{-1}. \tag{4.2}$$

Hence, the coefficients of the transformed quadratic form can be computed using (4.2). Two questions that arise here concern the construction of the coefficient matrix and the transformation of forms of degree $r > 2$.

4.1.1.1 Construction of Coefficient Matrices

For a form of degree j, let the coefficient vector $\Phi_{[j]}$ be defined as

$$\frac{\Phi_\alpha}{\sqrt{\alpha!}} : |\alpha| = j \tag{4.3}$$

where Φ_α are the *symbols* instead of the values representing the coefficients. Let h_j be the number of monomials in a form of degree j. Then for forms of degree $j + k = r$ the $h_j \times h_k$ coefficient matrix $\Phi_{[j,k]}$ can be constructed from the vectors of lower degree forms $\Phi_{[j]}$ and $\Phi_{[k]}$ using

$$\Phi_{[j,k]} = \Phi_{[j]} \star \Phi_{[k]}^T \tag{4.4}$$

where $\star$ represents the classic matrix multiplication with the difference that the individual element-wise multiplications are performed according to the rule $\Phi_\alpha\Phi_\beta = \Phi_{\alpha+\beta}$. Such a symbolic multiplication gives the set of coefficients

$$\{\frac{\Phi_{\alpha+\beta}}{\sqrt{\alpha!\beta!}} : |\alpha| = j, |\beta| = k\}.$$

As an example, to form $\Phi_{[1,1]}$ as above, first the vector $\Phi_{[1]}$ is formed:

$$\Phi_{[1]} = \begin{pmatrix} \frac{\Phi_{(1,0,0)}}{\sqrt{1!0!0!}} \\ \frac{\Phi_{(0,1,0)}}{\sqrt{0!1!0!}} \\ \frac{\Phi_{(0,0,1)}}{\sqrt{0!0!1!}} \end{pmatrix} = \begin{pmatrix} \Phi_{(1,0,0)} \\ \Phi_{(0,1,0)} \\ \Phi_{(0,0,1)} \end{pmatrix}.$$

Then $\Phi_{[1,1]}$ is formed by $\Phi_{[1]} \star \Phi_{[1]}^T$ with the element-wise multiplications performed as $\Phi_\alpha \Phi_\beta = \Phi_{\alpha+\beta}$:

$$\begin{aligned}
\Phi_{[1,1]} &= \Phi_{[1]} \star \Phi_{[1]}^T \\
&= \begin{pmatrix} \Phi_{(1,0,0)} \\ \Phi_{(0,1,0)} \\ \Phi_{(0,0,1)} \end{pmatrix} \begin{pmatrix} \Phi_{(1,0,0)} & \Phi_{(0,1,0)} & \Phi_{(0,0,1)} \end{pmatrix} \\
&= \begin{pmatrix} \Phi_{(1,0,0)+(1,0,0)} & \Phi_{(1,0,0)+(0,1,0)} & \Phi_{(1,0,0)+(0,0,1)} \\ \Phi_{(0,1,0)+(1,0,0)} & \Phi_{(0,1,0)+(0,1,0)} & \Phi_{(0,1,0)+(0,0,1)} \\ \Phi_{(0,0,1)+(1,0,0)} & \Phi_{(0,0,1)+(0,1,0)} & \Phi_{(0,0,1)+(0,0,1)} \end{pmatrix} \\
&= \begin{pmatrix} \Phi_{(2,0,0)} & \Phi_{(1,1,0)} & \Phi_{(1,0,1)} \\ \Phi_{(1,1,0)} & \Phi_{(0,2,0)} & \Phi_{(0,1,1)} \\ \Phi_{(1,0,1)} & \Phi_{(0,1,1)} & \Phi_{(0,0,2)} \end{pmatrix}.
\end{aligned}$$

4.1.1.2 Transformation of Forms

Using the form representation matrix $\Phi_{[j,k]}$, Taubin and Cooper [66] show that under a non-singular coordinate transformation A, the transformed coefficient matrix is given by

$$\Phi'_{[j,k]} = A_{[j]}^{-T} \Phi_{[j,k]} A_{[k]}^{-1} \tag{4.5}$$

where $A_{[j]}$ is a non-singular $h_j \times h_j$ matrix and $A_{[k]}$ is a non-singular $h_k \times h_k$ matrix.

4.1.1.3 Transformation of Polynomials

By introducing homogeneous coordinates, an algebraic curve or surface described in Euclidean space by a polynomial in n variables can be described in projective space by a corresponding homogeneous polynomial in $n+1$ variables. To convert a ternary (i.e. 3D) polynomial of degree d

$$f_d(x_1, x_2, x_3) = \sum_{0 \le i,j,k; i+j+k \le d} a_{ijk} x_1^i x_2^j x_3^k$$

into its homogeneous representation, a new component $x_4 = 1$ is added to every 3D point (x_1, x_2, x_3) to get the same polynomial written in homogeneous representation

$$f_d(x_1, x_2, x_3, x_4) = \sum_{0 \le i,j,k,l; i+j+k+l = d} a_{ijkl} x_1^i x_2^j x_3^k x_4^l. \tag{4.6}$$

A homogeneous polynomial corresponding to a polynomial of degree d in n variables is a form of degree d in $n+1$ variables and hence the procedure mentioned in the last section can be used to transform the homogeneous polynomial. Similar to the coefficient representation matrix $\Phi_{[j,k]}$ for a form, we can make the coefficient representation matrix $F_{[j,k]}$ for a homogeneous polynomial. As an example, for a 2D polynomial of degree 4, the homogeneous form is a 3D polynomial of degree 4 and we can setup the coefficient representation

matrix as

$$F_{[2,2]} = F_{[2]} \star F_{[2]}^T$$
$$= \begin{bmatrix} \frac{1}{2}F_{(4,0,0)} & \frac{1}{\sqrt{2}}F_{(3,1,0)} & \frac{1}{\sqrt{2}}F_{(3,0,1)} & \frac{1}{2}F_{(2,2,0)} & \frac{1}{\sqrt{2}}F_{(2,1,1)} & \frac{1}{2}F_{(2,0,2)} \\ \frac{1}{\sqrt{2}}F_{(3,1,0)} & F_{(2,2,0)} & F_{(2,1,1)} & \frac{1}{\sqrt{2}}F_{(1,3,0)} & F_{(1,2,1)} & \frac{1}{\sqrt{2}}F_{(1,1,2)} \\ \frac{1}{\sqrt{2}}F_{(3,0,1)} & F_{(2,1,1)} & F_{(2,0,2)} & \frac{1}{\sqrt{2}}F_{(1,2,1)} & F_{(1,1,2)} & \frac{1}{\sqrt{2}}F_{(1,0,3)} \\ \frac{1}{2}F_{(2,2,0)} & \frac{1}{\sqrt{2}}F_{(1,3,0)} & \frac{1}{\sqrt{2}}F_{(1,2,1)} & \frac{1}{2}F_{(0,4,0)} & \frac{1}{\sqrt{2}}F_{(0,3,1)} & \frac{1}{2}F_{(0,2,2)} \\ \frac{1}{\sqrt{2}}F_{(2,1,1)} & F_{(1,2,1)} & F_{(1,1,2)} & \frac{1}{\sqrt{2}}F_{(0,3,1)} & F_{(0,2,2)} & \frac{1}{\sqrt{2}}F_{(0,1,3)} \\ \frac{1}{2}F_{(2,0,2)} & \frac{1}{\sqrt{2}}F_{(1,1,2)} & \frac{1}{\sqrt{2}}F_{(1,0,3)} & \frac{1}{2}F_{(0,2,2)} & \frac{1}{\sqrt{2}}F_{(0,1,3)} & \frac{1}{2}F_{(0,0,4)} \end{bmatrix}.$$

4.1.2 Tensor-based Approach

Tarel et al. [32] show that for a homogeneous polynomial of degree d (4.6) can be written in a unique way as

$$f_d(x_1, x_2, x_3, x_4) = \sum_{i_1=1}^{4} \sum_{i_2=1}^{4} \cdots \sum_{i_d=1}^{4} b^{i_1 i_2 \ldots i_d} x_{i_1} x_{i_2} \ldots x_{i_d} \tag{4.7}$$

where

$$b^{i_1 i_2 \ldots i_d} = a_{ijkl} \frac{i!j!k!l!}{n!} \tag{4.8}$$

and the multi-index $ijkl$ is derived from the multi-index $i_1 i_2 \ldots i_d$ as follows:

i = number of times 1 appears in $i_1 i_2 \ldots i_d$,

j = number of times 2 appears in $i_1 i_2 \ldots i_d$,

k = number of times 3 appears in $i_1 i_2 \ldots i_d$,

l = number of times 4 appears in $i_1 i_2 \ldots i_d$.

Note that this construction is for a 3D polynomial whose homogeneous representation requires the 4 indices $ijkl$. In general, an nD polynomial would require $n+1$ indices $ij\ldots$. The length of the other multi-index $i_1 i_2 \ldots i_d$ is independent of the number of polynomial variables. It depends only on (and is equal to) the degree d. However, the value of each index i_k for $k = 1 \ldots d$ ranges from 0 till the number of variables. The different $b^{i_1 i_2 \ldots i_d}$ form the rank d covariant tensor B_d

$$B_d = (b^{i_1 i_2 \ldots i_d})_{1 \le i_1 i_2 \ldots i_d \le 4} \tag{4.9}$$

that is the tensor representation the polynomial f_d. Note that B_d is symmetric and therefore highly redundant. The mapping from $ijkl$ to $i_1 i_2 \ldots i_d$ is one-to-many. Rank d tensors are implemented as d-dimensional arrays. Since the homogeneous representation of f_d is a polynomial in 4 variables, B_d is a d-dimensional array with 4^d entries.

To consider transformation of the tensor, let $M = (m_i^j)$ be a 4×4 matrix representing a linear transformation of the *world coordinate system*[2] where m_i^j represents the entry at

[2]Transforming the world coordinate system by M is equivalent to transforming objects in the world coordinate system by M^{-1}.

row i and column j. Upon such a transformation, the transformed tensor entries in the new basis are given by

$$b'^{j_1 j_2 \ldots j_d} = |J^{-1}| \sum_{i_1=1}^{4} \sum_{i_2=1}^{4} \cdots \sum_{i_d=1}^{4} b^{i_1 i_2 \ldots i_d} m_{i_1}^{j_1} m_{i_2}^{j_2} \ldots m_{i_d}^{j_d} \tag{4.10}$$

where J is the Jacobian matrix of M. For orthogonal (and hence Euclidean) transformations $|J| = det(M) = 1$. So to transform a 3D polynomial by a linear transformation M the following steps should be followed:

1. Transform the polynomial to its homogeneous form by adding homogenising variable x_4.

2. Construct a tensor from the homogeneous form using (4.8).

3. Transform the tensor elements by M^{-1} using (4.10).

4. Convert the transformed tensor back to homogeneous polynomial form by making use of (4.8). Since the mapping from $i_1 i_2 \ldots i_d$ to $ijkl$ is many-to-one, any one of the multi-indices $i_1 i_2 \ldots i_d$ that map to $ijkl$ can be used.

5. Convert the homogeneous polynomial to its normal representation by substituting $x_4 = 1$.

The transformation shown in Figure 4.2 was achieved using this approach.

4.2 Intrinsic Geometry of Implicit Algebraic Curves and Surfaces

4.2.1 Computation of Euclidean Center

Taubin and Cooper [66] show that the Euclidean center $\mathbf{c}$ of an implicit polynomial $f_d = 0$ of degree d can be computed as

$$\mathbf{c} = -F_{[d-1,1]}^{\dagger} F_{[d-1]} \tag{4.11}$$

where $F_{[d-1,1]}$ is a symmetric matrix constructed from the coefficients of the highest degree form, $F_{[d-1]}$ is a vector constructed from the coefficients of the form of the second highest degree and $\dagger$ implies taking the pseudoinverse which is given as

$$F_{[d-1,1]}^{\dagger} = [F_{[1,d-1]} F_{[d-1,1]}]^{-1} F_{[1,d-1]}$$

provided that $F_{[1,d-1]}F_{[d-1,1]}$ is non-singular. For a 2D polynomial of degree 4, (4.11) expands to

$$\begin{aligned}\begin{bmatrix}c_1\\c_2\end{bmatrix} &= -\begin{bmatrix}\frac{1}{\sqrt{6}}F_{(4,0)} & \frac{1}{\sqrt{6}}F_{(3,1)}\\ \frac{1}{\sqrt{2}}F_{(3,1)} & \frac{1}{\sqrt{2}}F_{(2,2)}\\ \frac{1}{\sqrt{2}}F_{(2,2)} & \frac{1}{\sqrt{2}}F_{(1,3)}\\ \frac{1}{\sqrt{6}}F_{(1,3)} & \frac{1}{\sqrt{6}}F_{(0,4)}\end{bmatrix}^{\dagger}\begin{bmatrix}F_{(1,0)}\\F_{(0,1)}\end{bmatrix}\\ &= -\begin{bmatrix}\frac{4!0!}{\sqrt{6}}a_{40} & \frac{3!1!}{\sqrt{6}}a_{31}\\ \frac{3!1!}{\sqrt{2}}a_{31} & \frac{2!2!}{\sqrt{2}}a_{22}\\ \frac{2!2!}{\sqrt{2}}a_{22} & \frac{1!3!}{\sqrt{2}}a_{13}\\ \frac{1!3!}{\sqrt{6}}a_{13} & \frac{0!4!}{\sqrt{6}}a_{04}\end{bmatrix}^{\dagger}\begin{bmatrix}1!0!a_{10}\\0!1!a_{01}\end{bmatrix}.\end{aligned}$$

Moreover, the center is a covariant. That is, if the coordinate axes transform by $\mathbf{x}' = A\mathbf{x} + \mathbf{b}$ then the center also transforms by $\mathbf{c}' = A\mathbf{c} + \mathbf{b}$.

Lemma 4.2.1. *Let $f(\mathbf{x})$ be a polynomial of degree d, $\mathbf{x}' = A\mathbf{x} + b$ a Euclidean coordinate transformation, $f'(\mathbf{x}') = f(A^T(\mathbf{x}' - b))$, $\mathbf{c} = -F^{\dagger}_{[d-1,1]}F_{[d-1]}$ and $\mathbf{c}' = -F'_{[d-1,1]} \dagger F'_{[d-1]}$. Then $\mathbf{c}' = A\mathbf{c} + \mathbf{b}$.*

4.2.2 Computation of Euclidean Orientation

Continuing from the approach for Euclidean center computation, Taubin and Cooper [66] show that for an nD polynomial f_d of degree d, the n eigenvectors of the symmetric matrix

$$F^T_{[d-1,1]}F_{[d-1,1]} \tag{4.12}$$

form an orthogonal coordinate system that defines the intrinsic orientation of the implicit polynomial $f_d = 0$ (i.e. $Z(f_d)$). Let us denote by $\mathbf{V}_f$ the $n \times n$ orthogonal matrix formed by placing the n eigenvectors column-wise. The geometric interpretation of the orthogonal matrix $\mathbf{V}_f$ is that it transforms the standard Cartesian axes to the intrinsic axes of the polynomial f_d. Alternatively, $\mathbf{V}_f^{-1}$ transforms the polynomial's intrinsic axes into alignment with the standard Cartesian axes. For a 2D polynomial of degree 4, the symmetric matrix whose 2 eigenvectors define the implicit polynomial's orientation is given by

$$\begin{aligned}&F^T_{[d-1,1]}F_{[d-1,1]} = F^T_{[3,1]}F_{[3,1]}\\ &= \begin{bmatrix}\frac{F^2_{(4,0)}}{6} + \frac{F^2_{(3,1)}}{2} + \frac{F^2_{(2,2)}}{2} + \frac{F^2_{(1,3)}}{6} & A\\ A & \frac{F^2_{(3,1)}}{6} + \frac{F^2_{(2,2)}}{2} + \frac{F^2_{(1,3)}}{2} + \frac{F^2_{(0,4)}}{6}\end{bmatrix}\end{aligned}$$

where

$$A = \frac{F_{(4,0)}F_{(3,1)}}{6} + \frac{F_{(3,1)}F_{(2,2)}}{2} + \frac{F_{(2,2)}F_{(1,3)}}{2} + \frac{F_{(1,3)}F_{(0,4)}}{6}$$

A tensor-based extension of this approach can be found in [32]. The particular advantage of that approach is that it leads to a matrix with entries that are *linear* in the coefficients of the highest degree form of f_d. This linearity leads to more robust orientation estimation. Figure 4.1 shows the intrinsic reference frames extracted from coefficients of a 4th-degree algebraic curve fitted to 2D data undergoing linear transformations consisting of rotation

followed by translation. Figure 4.2 shows the intrinsic reference frame extracted from coefficients of a 4th-degree algebraic surface representing a paper puncher and for the same puncher rotated by 45° around the x-axis.

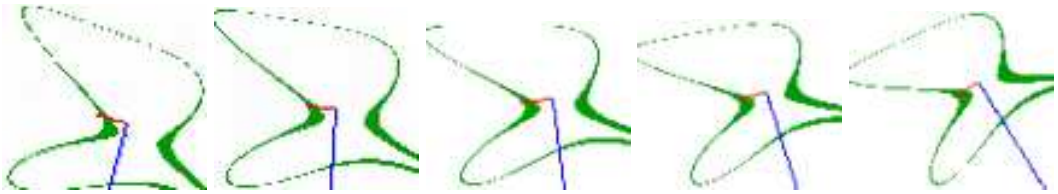


Figure 4.1: ***Left to right:*** *Intrinsic reference frames (center+orientation) extracted from coefficients of 4th-degree implicit polynomials fitted to 2D data. First image shows the algebraic curve fitted to data in the original position and its intrinsic reference frame. The rest show the same information for data rotated by* $10, 20, 30$ *and* 45 *degrees and then translated by* $10, 10$*. It can be observed that the computed intrinsic reference frames are covariant w.r.t. the transformations.*

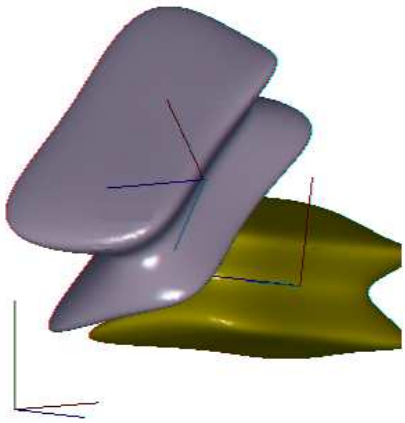


Figure 4.2: *Intrinsic reference frames extracted from coefficients of a 4th-degree algebraic surface representing a paper puncher and for the same puncher rotated by* $45°$ *around the* x*-axis.*

4.3 Spatial Registration of Implicit Polynomials

The intrinsic geometry of algebraic entities presented in the last section provides an easy correspondence-free method of spatial registration.

4.3.1 Translation Estimation

Let $f_d = 0$ be an implicit polynomial of degree d and g_d be a translated version of f_d, i.e.

$$g_d(\mathbf{x}) = f_d(\mathbf{x} + \mathbf{t})$$

where $\mathbf{t}$ is the translation vector. Using (4.11) the centers of both polynomials are given by

$$\mathbf{c}_f = -F^{\dagger}_{[d-1,1]} F_{[d-1]},$$
$$\mathbf{c}_g = -G^{\dagger}_{[d-1,1]} G_{[d-1]}.$$

Now, translation $\mathbf{t}$ is given as the difference between the centers:

$$\begin{aligned}\mathbf{t} &= \mathbf{c}_g - \mathbf{c}_f \\ &= -G^{\dagger}_{[d-1,1]} G_{[d-1]} + F^{\dagger}_{[d-1,1]} F_{[d-1]}.\end{aligned}$$

Since translation leaves the highest degree form unchanged [66], $G_{[d-1,1]} = F_{[d-1,1]}$ as both matrices are constructed from the respective highest degree forms only. This implies that

$$\mathbf{t} = G^{\dagger}_{[d-1,1]}(F_{[d-1]} - G_{[d-1]}). \tag{4.13}$$

4.3.2 Rotation Estimation

Let $f_d = 0$ be an implicit nD polynomial of degree d and g_d be a rotated version of f_d, i.e.

$$g_d(\mathbf{x}) = f_d(\mathbf{R}\mathbf{x})$$

where $\mathbf{R}$ is the $n \times n$ rotation matrix. The intrinsic orientations $\mathbf{V}_f$ and $\mathbf{V}_g$ can be computed as explained in Section 4.2.2. Since the rotation $\mathbf{V}_f^{-1}$ aligns the polynomial f_d with the world coordinate system and $\mathbf{V}_g$ aligns the world coordinate system with the intrinsic axes of polynomial g_d, the rotation $\mathbf{R}$ that transforms the implicit polynomial f_d to g_d is given as

$$\mathbf{R} = \mathbf{V}_g \mathbf{V}_f^{-1} = \mathbf{V}_g \mathbf{V}_f^{T}. \tag{4.14}$$

It should be noticed, however, that any intrinsic orientation matrix such as $\mathbf{V}_f$ does not provide a unique transformation between the intrinsic axes and the standard Cartesian frame. For a 2D polynomial there are 4 pairs of intrinsic eigenvectors (corresponding to the symmetry of an ellipsoid) only one of which is the correct transformation. For a 3D polynomial, there are 8 such groups of 3 eigenvectors (corresponding to the symmetry of a hyperboloid). A method for disambiguating the rotation estimates is presented in the next section.

4.3.3 Rotation + Translation Estimation

When the transformation between f_d and g_d is Euclidean (i.e. rotation + translation), a method for disambiguation of the rotation estimate can be found in [32] where the correct rotation is found by combining the different rotation estimates with the translation estimates and picking the transformation $(\mathbf{R}, \mathbf{t})$ that minimises the distance between the coefficient vectors of $f_d(\mathbf{R}\mathbf{x}+\mathbf{t})$ and g_d. It can be seen that the transformation $(\mathbf{R}, \mathbf{t})$ leads to spatial registration of $Z(f_d)$ with $Z(g_d)$. This implicit registration is an alternative to explicit correspondence-based registration such as the ICP method [6].

4.4 Implicit Algebraic Curve and Surface Fitting

Given a finite set of n-dimensional data points $\mathcal{D} = \{p_1, \cdots, p_q\}$, *fitting* an implicit curve or surface $Z(f)$ to the data set $\mathcal{D}$ means computing a minimiser $\hat{f} \in \mathbb{F}$ of the mean square distance

$$\frac{1}{q}\sum_{i=1}^{q} \mathrm{dist}(p_i, Z(f))^2 \tag{4.15}$$

from the data points to the curve or surface $Z(f)$. The polynomial f is also called the *interpolating polynomial*. Such a minimisation is complicated by the fact that there is no closed form expression for $\text{dist}(\mathbf{x}, Z(f))$, the distance from a point $\mathbf{x} \in \mathbb{R}^n$ to a generic implicit curve or surface $Z(f)$ [66]. Therefore iterative methods are required which make the minimisation of equation (4.15) computationally impractical.

Taubin [65] derives an *approximation* of the point to implicit entity distance as follows. A closed form expression for $\text{dist}(\mathbf{x}, Z(f))$ *does* exist when f is linear, i.e. a first degree polynomial. In that case, the Jacobian matrix $Df(\mathbf{x})$ is constant and the following identity is satisfied:

$$f(\mathbf{y}) \equiv f(\mathbf{x}) + Df(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) \tag{4.16}$$

for a point $\mathbf{y} \in Z(f)$ and $\mathbf{x}$. The unique point $\hat{\mathbf{y}}$ that minimises the distance $\|\mathbf{y} - \mathbf{x}\|$ to $\mathbf{x}$, constrained by $f(\mathbf{y}) = 0$, is given by

$$\hat{\mathbf{y}} = \mathbf{x} - [Df(\mathbf{x})]^{\dagger} f(\mathbf{x}) \tag{4.17}$$

where $[Df(\mathbf{x})]^{\dagger} = Df(\mathbf{x})^T (Df(\mathbf{x}) Df(\mathbf{x})^T)^{-1}$ is the *pseudoinverse* of $Df(\mathbf{x})$. So the squared distance from $\mathbf{x}$ to $Z(f)$ is given by

$$\text{dist}(\mathbf{x}, Z(f))^2 = f(\mathbf{x})^T [Df(\mathbf{x}) Df(\mathbf{x})^T]^{-1} f(\mathbf{x}) \cdot \tag{4.18}$$

For the general case when f is non-linear, the distance between $\mathbf{x}$ and $Z(f)$ can be approximated by the distance between $\mathbf{x}$ and a *linear model* $\bar{f}$ of f at $\mathbf{x}$ such that

$$f(\mathbf{y}) - \bar{f}(\mathbf{y}) = O(\|\mathbf{y} - \mathbf{x}\|^2) \cdot$$

$\bar{f}$ is taken to be the truncated Taylor series expansion of f:

$$\bar{f}(\mathbf{y}) = f(\mathbf{x}) + Df(\mathbf{x})(\mathbf{y} - \mathbf{x}) \cdot$$

Since $\bar{f}(\mathbf{x}) = f(\mathbf{x})$ and $D\bar{f}(\mathbf{x}) = Df(\mathbf{x})$, an *approximate squared distance* is given by

$$\text{dist}(\mathbf{x}, Z(f))^2 \approx f(\mathbf{x})^T [Df(\mathbf{x}) Df(\mathbf{x})^T]^{-1} f(\mathbf{x}) \cdot \tag{4.19}$$

We will be primarily interested in planar curves and surfaces in which case $Df(\mathbf{x}) = \nabla f(\mathbf{x})^T$ and equation (4.19) reduces to

$$\text{dist}(\mathbf{x}, Z(f))^2 \approx \frac{f(\mathbf{x})^2}{\|\nabla f(\mathbf{x})\|^2} \cdot \tag{4.20}$$

Taubin [65] has shown that the approximate distance function is independent of the representation of $Z(f)$, is invariant to rigid body transformations and is proportional to scale transformations. Using equation (4.20) we can write the *approximate mean square distance* from the data set $\mathcal{D}$ to the set of zeros of $f = \phi_u \in \mathbb{F}$ as

$$\Delta^2_{\mathcal{D}}(u) = \frac{1}{q} \sum_{i=1}^{q} \frac{f(p_i)^2}{\|\nabla f(p_i)\|^2} \cdot \tag{4.21}$$

4.4.1 Generalised Eigenvector Fitting

Minimisation of equation (4.21) is a non-linear least squares problem and can be solved by iterative techniques such as the Levenberg-Marquardt algorithm. However, such a minimisation is local. To obtain a global initial estimate that can be used in subsequent local minimisation, Taubin [65] replaces the approximate mean square distance, equation (4.21), by the following approximation that reduces the problem of finding the minimising parameters to a generalised eigenproblem. For certain families of implicit curves and surfaces, $\|\nabla f(\mathbf{x})\|$ is constant on $Z(f)$. Hence equation (4.21) can be written as

$$\Delta^2_{\mathcal{D}}(u) = \frac{1}{q}\sum_{i=1}^{q}\frac{f(p_i)^2}{\|\nabla f(p_i)\|^2} \approx \frac{\frac{1}{q}\sum_{i=1}^{q} f(p_i)^2}{\frac{1}{q}\sum_{i=1}^{q}\|\nabla f(p_i)\|^2}.$$

The right hand side can be written as

$$\frac{\frac{1}{q}\sum_{i=1}^{q} f(p_i)^2}{\frac{1}{q}\sum_{i=1}^{q}\|\nabla f(p_i)\|^2} = \frac{FMF^T}{FNF^T} \tag{4.22}$$

where $M = \frac{1}{q}\sum_{i=1}^{q}[X(p_i)X(p_i)^T]$, $N = \frac{1}{q}\sum_{i=1}^{q}[DX(p_i)DX(p_i)^T]$, $F = (F_1, \cdots, F_h)$ is a row vector of coefficients, i.e. the required minimising parameters, and $X = (X_1, \cdots, X_h)^T : \mathbb{R}^n \to \mathbb{R}^h$ is a fixed map such that $f(\mathbf{x}) = F_1X_1(\mathbf{x}) + \cdots + F_hX_h(\mathbf{x}) = FX(\mathbf{x})$. M and N are non-negative definite, symmetric matrices that are functions of only the data points. Equation (4.22) is minimised by the eigenvector corresponding to the minimum eigenvalue of the pencil $F(M - \lambda N) = 0$ and hence the minimisation problem has been reduced to a generalised eigenvalue problem. Such fitting is termed *generalised eigenvector fitting* by Taubin [65] where it is proposed as a fitting method even for the general case when $\|\nabla f(\mathbf{x})\|$ is *not* constant on $Z(f)$.

4.4.2 The 3L Fitting Algorithm

Blane et al. [7][8] showed that generalised eigenvector fitting produces zero-sets that are not faithful to the data. Instead, they introduced a completely different approach for fitting implicit polynomials to data. Their 3L fitting algorithm converts the non-linear minimisation problem of equation (4.15) to a linear least-squares *explicit* polynomial fitting problem. The basic idea is to generate additional points at distance levels +c and -c from the data set in the normal direction. This gives 3 levels of constraints on the interpolating polynomial; the 0 level consisting of the original data points, the +c level consisting of interior points and the -c level consisting of the exterior points. When the number of data points (original plus generated) are greater than the number of polynomial coefficients to solve for, the constraint equations can be written as an over-constrained linear system $\mathbf{Ax} = \mathbf{b}$. For example, for a 2D quadratic polynomial 3L fitting solves the

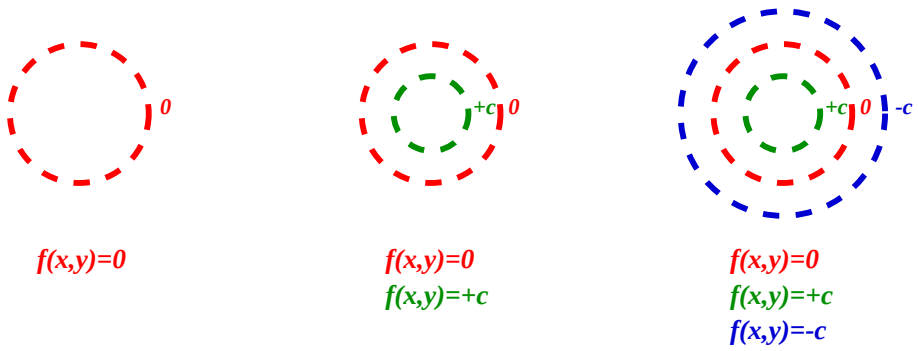


Figure 4.3: *The three levels used in 3L fitting.*

following over-constrained linear system:

$$\underbrace{\begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ & & & \vdots & & \\ 1 & x_i & y_i & x_i^2 & x_iy_i & y_i^2 \\ 1 & x_{i+1} & y_{i+1} & x_{i+1}^2 & x_{i+1}y_{i+1} & y_{i+1}^2 \\ & & & \vdots & & \\ 1 & x_j & y_j & x_j^2 & x_jy_j & y_j^2 \\ 1 & x_{j+1} & y_{j+1} & x_{j+1}^2 & x_{j+1}y_{j+1} & y_{j+1}^2 \\ & & & \vdots & & \\ 1 & x_n & y_n & x_n^2 & x_ny_n & y_n^2 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} a_{00} \\ a_{10} \\ a_{01} \\ a_{20} \\ a_{11} \\ a_{02} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} +c \\ \vdots \\ +c \\ 0 \\ \vdots \\ 0 \\ -c \\ \vdots \\ -c \end{bmatrix}}_{\mathbf{b}} \cdot \tag{4.23}$$

Extension to higher dimensional, higher degree polynomials is straight-forward as can be seen from the following system for a 3D polynomial of degree d:

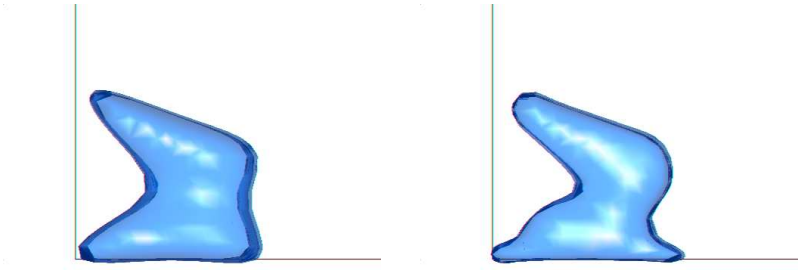
$$\underbrace{\begin{bmatrix} 1 & x_1 & y_1 & z_1 & x_1^2 & \dots & y_1z_1^{d-1} & z_1^d \\ & & & \vdots & & & & \\ 1 & x_i & y_i & z_i & x_i^2 & \dots & y_iz_i^{d-1} & z_i^d \\ 1 & x_{i+1} & y_{i+1} & z_{i+1} & x_{i+1}^2 & \dots & y_{i+1}z_{i+1}^{d-1} & z_{i+1}^d \\ & & & \vdots & & & & \\ 1 & x_j & y_j & z_j & x_j^2 & \dots & y_jz_j^{d-1} & z_j^d \\ 1 & x_{j+1} & y_{j+1} & z_{j+1} & x_{j+1}^2 & \dots & y_{j+1}z_{j+1}^{d-1} & z_{j+1}^d \\ & & & \vdots & & & & \\ 1 & x_n & y_n & z_n & x_n^2 & \dots & y_nz_n^{d-1} & z_n^d \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} a_{000} \\ a_{100} \\ a_{010} \\ a_{001} \\ a_{200} \\ \vdots \\ a_{01(d-1)} \\ a_{00d} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} +c \\ \vdots \\ +c \\ 0 \\ \vdots \\ 0 \\ -c \\ \vdots \\ -c \end{bmatrix}}_{\mathbf{b}} \cdot \tag{4.24}$$

The polynomial that represents the 2D algebraic curve for system (4.23) and the 3D algebraic surface for system (4.24) is then given by the coefficient vector

$$\mathbf{x} = \mathbf{A}^{\dagger}\mathbf{b} \tag{4.25}$$

where $\mathbf{A}^{\dagger}$ is the pseudoinverse $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ of matrix $\mathbf{A}$. For increased numerical stability, the Householder algorithm [2] can also be used for obtaining the coefficient vector $\mathbf{x}$.

Figure 4.4 shows implicit polynomials of degree 4 and 8 fitted to a puncher model. For the fittings shown every $4th$ point in the original data set was selected, and additional points were generated in the positive and negative normal directions at distances $0.5, 1.0, 1.5, 2.0$. The better fitting due to additional level-sets is *not* in agreement with [8] who state that conceptually and empirically additional level-sets will not improve fitting results. We also found that fitting error increased when every point in the original data set was used. This was because at regions with high concentration of points, self-intersecion of normals can occur. Also, using level-sets in addition to the 3 proposed by 3L fitting improved the quality of the fits and more accurately controlled the behaviour of the resulting polynomials far away from the data set. This is in agreement with the observations and suggestions made in [14]. For a detailed treatment of 3L fitting, its comparison with other fitting techniques and justification of the fact that $\mathbf{A}^T\mathbf{A}$ is non-singular, we refer the reader to [8].



(a) 4th degree fit (b) 8th degree fit

Figure 4.4: *Profile views of a puncher fitted with polynomials of degree 4 and 8 using 3L fitting.*

Chapter 5

Occluding Contours and Pose Estimation

In Section 2.3 we described how to obtain tangent cones of algebraic surfaces using resultants. In this chapter we describe how to obtain occluding contour equations from equations of algebraic surfaces and the camera projection matrix using the Dixon resultant. We also describe how such occluding contours maybe used for pose estimation of algebraic surfaces. We end by presenting a hybrid implicit-explicit approach to pose estimation.

5.1 Occluding Contours

Figure 5.1 illustrates the formation of occluding contours from algebraic surfaces viewed through a perspective projection. As explained earlier, the red contour in the image is

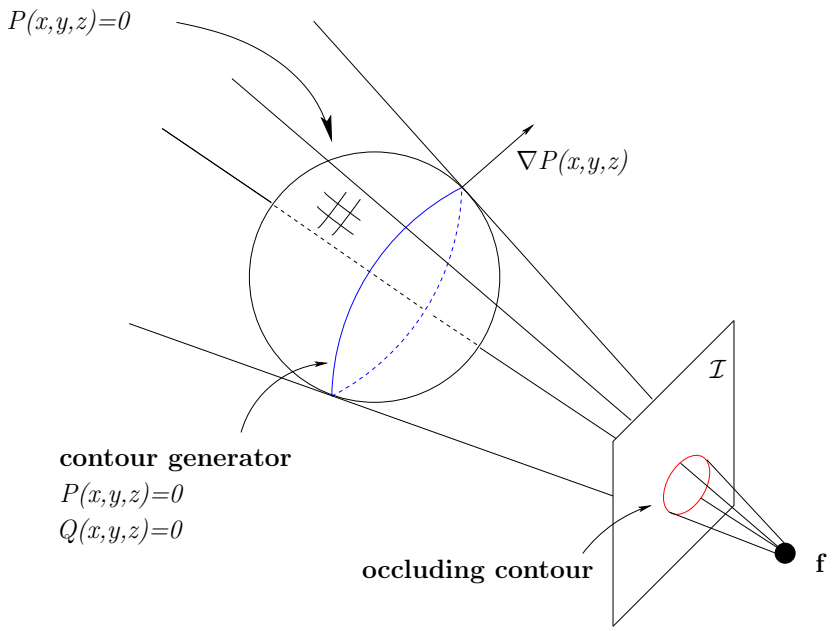


Figure 5.1: *A sphere boundary represented as an implicit algebraic surface* $P(x, y, z) = 0$*, a contour generator and the corresponding occluding contour with respect to the focal point* $\mathbf{f}$ *of the camera.*

formed by the projection of the blue points on the sphere that lie on tangential rays emanating from the focal point $\mathbf{f}$. The 3D points constitute the contour generator and their projections constitute the occluding contour. All points $\mathbf{x}$ on the contour generator satisfy

$$\begin{aligned} P(\mathbf{x}) &= 0, \\ Q(\mathbf{x}) &= (\mathbf{x} - \mathbf{f}).\nabla P(\mathbf{x}) = 0. \end{aligned}$$

$Q(\mathbf{x}) = 0$ is called the *tangency condition*. Let focal point $\mathbf{f}$ be at $(0, 0, -400)$ and the projection matrix $\mathbf{M}$ encoding the image formation process be

$$\mathbf{M} = \begin{bmatrix} 100 & 0 & 80 & 32000 \\ 0 & -100 & 60 & 24000 \\ 0 & 0 & 1 & 400 \end{bmatrix}.$$

$\mathbf{M}$ represents projection onto a 120×160 image plane parallel to the xy-plane and placed at $z = -300$ with focal point $(0, 0, -400)$. Projection equations can be written as

$$\begin{aligned} s &= \frac{100x + 80z + 32000}{z + 400}, \\ t &= \frac{-100y + 60z + 24000}{z + 400} \end{aligned}$$

where the first equation defines projection onto the horizontal axis of the image plane and the second equation defines the vertical projection. The top-left corner of the image is the image origin $(0, 0)$. We can write the projection equations as implicit polynomials

$$\begin{aligned} H(\mathbf{x}, s) &= 100x - s(z + 400) + 80z + 32000 = 0, \\ V(\mathbf{x}, t) &= 100y + t(z + 400) - 60z - 24000 = 0. \end{aligned}$$

Since the occluding contour is the projection of the contour generator, it satisfies the following polynomial system:

$$\begin{aligned} P(\mathbf{x}) &= 0 \\ Q(\mathbf{x}) &= 0 \\ H(\mathbf{x}, s) &= 0 \\ V(\mathbf{x}, t) &= 0 \end{aligned} \tag{5.1}$$

5.1.1 Occluding Contour of a Sphere

For a sphere of radius 70 centered at $(0, 0, 0)$ and viewed via projection matrix $\mathbf{M}$, the system (5.1) becomes

$$\begin{aligned} x^2 + y^2 + z^2 - 70^2 &= 0 \\ 2x^2 + 2y^2 + 2z^2 + 800z &= 0 \\ 100x - s(z + 400) + 80z + 32000 &= 0 \\ 100y + t(z + 400) - 60z - 24000 &= 0 \end{aligned}$$

To obtain the equation of the occluding contour in terms of the image coordinate system (s,t), we need to eliminate the variables x, y, z from the system. Although the Dixon resultant can be used at this stage to eliminate x, y, z simultaneously, one can observe that for our simple projection matrix $\mathbf{M}$, $H(\mathbf{x}, s)$ and $V(\mathbf{x}, t)$ are linear in x and y respectively. This allows us to solve them for x and y and substitute the resulting expressions in the first two equations of the polynomial system to get two equations in z, s, t. Now, the Dixon resultant or any other bivariate resultant can be used to eliminate z from the two polynomials. Such elimination by substitution significantly reduces the size of resultant matrices.

For the purposes of illustrating elimination through Gröbner bases, we next compute the same occluding contour as above but this time the sphere is parametrised by the 3 translation parameters (v_1, v_2, v_3). We use the Groebner package in MAPLE.

```
with(Groebner):
fp:=[0,0,-400]:
P:=(x+v1)^2+(y+v2)^2+(z+v3)^2-70^2:
Q:=(x-fp[1])*diff(P,x)+(y-fp[2])*diff(P,y)+(z-fp[3])*diff(P,z):
H:=100*x-s*(z+400)+80*z+32000:
V:=100*y+t*(z+400)-60*z-24000:
idl:=[P,Q,H,V]:
tt:=time(): gb:=Basis(idl,plex(x,y,z,s,t,v1,v2,v3)): time()-tt;
OC:=op(1,gb):
OC:=op(-1,factor(OC));
```

which gives the occluding contour as

$$
\begin{aligned}
&OC(s,t) = \\
&(s^2 - 160s + 10000 + t^2 - 120t)v_3^2 \\
&+ (-800t^2 - 800s^2 - 200sv_1 + 200tv_2 + 16000v_1 + 128000s - 8000000 - 12000v_2 + 96000t)v_3 \\
&+ s^2v_2^2 - 24816000s + 2stv_1v_2 + t^2v_1^2 + 4800000v_2 + 155100t^2 - 120sv_1v_2 - 120tv_1^2 \\
&- 160sv_2^2 + 9600v_1v_2 + 155100s^2 - 160tv_1v_2 + 13600v_1^2 - 80000tv_2 + 80000sv_1 \\
&- 6400000v_1 + 16400v_2^2 - 18612000t + 1502000000 \qquad (5.2)
\end{aligned}
$$

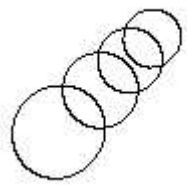


Figure 5.2: *Some outlines of a sphere moving downwards, from left to right and towards the camera.*

5.1.2 Occluding Contour of a Quartic

While the occluding contour of a quadratic surface is a plane curve that is still quadratic in s, t, the occluding contour of a quartic surface is of degree 12 in s, t. This follows from lemma (5.1.1) that we reproduce from [27].

Lemma 5.1.1. *For a generic surface of degree d viewed from focal point $\mathbf{f}$, the tangent cone, and hence its outline, has degree $d(d-1)$.*

Proof. In projective space, consider a generic focal point $\mathbf{f} = (f_0, f_1, f_2, f_3)$. The surface $P(x, y, z, w)$ is of degree d and the tangency condition requires that planes tangent to the surface at contour generator points must pass through the focal point. This point-plane duality can be expressed as

$$f_0\frac{\partial P}{\partial w} + f_1\frac{\partial P}{\partial x} + f_2\frac{\partial P}{\partial y} + f_3\frac{\partial P}{\partial z} = 0$$

which is of degree $d-1$. Since the contour generator is the complete intersection of these two hypersurfaces, it is of degree $d(d-1)$ in the generic case. □

Herein lies a practical limitation of studying the projective geometry of algebraic surfaces. Contour generators and hence occluding contours very quickly become high degree polynomials that are hard to handle. Table 5.1.2 illustrates how the degree of the occluding contour in the image coordinates (s, t) varies as a function of the degree of the algebraic surface in 3D coordinates (x, y, z). We therefore limit ourselves to studying only

Degree of Surface	2	4	6	8
Degree of Occluding Contour	2	12	30	56

Table 5.1: *Degree of occluding contour in (s, t) as a function of the degree of the algebraic surface in (x, y, z).*

4th order algebraic surfaces. They can represent many useful 2D shapes and 3D real world objects [8] as can be seen in Figure 4.4.

We work in the same environment as for the sphere in the last section but replace the sphere by the quartic surface

$$P(\mathbf{x}) = x^4 + y^4 + z^4 - 200xyz = 0\cdot$$

As before we construct the polynomial system P, Q, H, V and eliminate x, y, z to obtain the occluding contour. Attempting such an elimination using Gröbner bases exhibits the severe limitation of Gröbner bases. On a 3.06 GHz n-Series Dual Intel XEON with 2 GB RAM, the method failed to produce the occluding contour for the quartic parametrised by *only a single translation parameter* v_1 even after 1 hour. In comparison, the Dixon resultant based approach described next computed the occluding contour in less than 4 seconds. Figure 5.4 shows the occluding contours obtained with v_1 set to $-200, -100, 0, 100, 200$ which corresponds to a horizontal movement of the quartic surface from left to right in 3D.

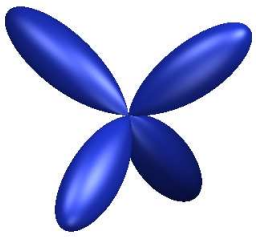


Figure 5.3: *Quartic surface defined by* $x^4 + y^4 + z^4 - 200xyz = 0$.

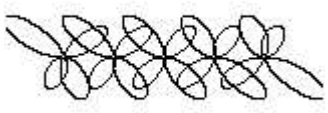


Figure 5.4: *Some outlines of a quartic surface moving from left to right.*

5.1.2.1 Dixon Resultant and Spurious Factors

We will go through the actual MAPLE code to explain how we use the Dixon resultant and remove spurious factors to reduce computation time. To actually compute the Dixon resultant we use the Dixon/Bézout resultant package for MAPLE developed by Arthur D. Chtcherba [3]. To begin, we load the LinearAlgebra package for basic linear algebra operations that come with MAPLE. Then we read in the Dixon/Bézout resultant package stored on the local disk and start a timer to see how long the computation runs.

```
with(LinearAlgebra):
read("~/myCode/Dixon.mpl"):
tt1:=time():
```

We then define the quartic surface P and the tangency condition Q.

```
P:=x^4+y^4+z^4-200*x*y*z:
P:=subs([x=x+v1],P):
fp:=[0,0,-400]:
Q:=expand((x-fp[1])*(diff(P, x))+(y-fp[2])*(diff(P, y))+(z-fp[3])*(diff(P, z))):
```

Since the projection matrix $\mathbf{M}$ leads to linear projection equations in x and y, we can eliminate them (i.e. x and y) by substitution. Therefore we substitute the expressions for x and y in P and Q.

```
P:=primpart(subs([x=(s*(z+400)-80*z-32000)/100, y=(60*z+24000-t*(z+400))/100],P)):
Q:=primpart(subs([x=(s*(z+400)-80*z-32000)/100, y=(60*z+24000-t*(z+400))/100],Q)):
```

Now we need to eliminate z only. We first compute the Dixon matrix of P and Q w.r.t. z.

```
mtx:=Dixon:-DixonMatrix([P,Q],[z],'rowH','colH'):
```

The determinant of this matrix *contains* the Dixon resultant *alongwith* spurious factors. We make use of the fact that the true Dixon resultant is irreducible and we use the following rule from linear algebra:

$$det \begin{bmatrix} a_1\text{row}_1 \\ a_2\text{row}_2 \\ \vdots \\ a_n\text{row}_n \end{bmatrix} = a_1 a_2 \ldots a_n det \begin{bmatrix} \text{row}_1 \\ \text{row}_2 \\ \vdots \\ \text{row}_n \end{bmatrix}.$$

This tells us that any row factors $a_1, a_2 \ldots, a_n$ are spurious factors because the Dixon resultant is irreducible. They can be removed from the Dixon matrix before computing the determinant. If they are not removed, they will end up as spurious factors in a comparatively bigger determinant expression. In our experiments, factoring out such factors from very large determinant expressions was often not possible in MAPLE. Early factoring out of spurious factors leads us closer to the exact resultant and significantly reduces the time for determinant computation. Therefore, as the next step, we compute the GCD of each row and divide it out to obtain a factored-out Dixon matrix. The process can be repeated for the columns also.

```
FMtx:=Matrix(Dimensions(mtx)):
for i from 1 to RowDimension(mtx) do
 row_gcd:=mtx[i,1]:
 for j from 2 to ColumnDimension(mtx) do
  row_gcd:=gcd(row_gcd,mtx[i,j]):
 od:
 for j from 1 to ColumnDimension(mtx) do
  FMtx[i,j]:=simplify(mtx[i,j]/row_gcd):
 od:
od:
```

Finally, we use fraction-free Gaussian elimination to compute the determinant of the factored-out Dixon matrix. The determinant can be factored again to remove any remaining spurious factors. The resultant is the only factor that contains all the variables (i.e. s, t, v_1 in this case) and is usually returned as the last factor by the factor() command. We end by stopping the timer to see how long the computation took.

```
tt:=time(): detFMtx:=Determinant(FMtx,'method'='fracfree'): time()-tt;
tt:=time(): fdetFMtx:=factor(detFMtx): time()-tt;
if (nops(fdetFMtx)>1) then OC:=op(-1,fdetFMtx): else OC:=fdetFMtx: fi:
time()-tt1;
```

As stated before, computing $OC(s, t, v_1)$ took less than 4 seconds. However, computing $OC(s, t, v_1, v_2)$ took 257 seconds, out of which 255 were spent on computing the determinant using fraction-free Gaussian elimination. Computation of $OC(s, t, v_1, v_2, v_3)$ failed to

give an answer even after 5 and a half hours. To reduce determinant computation time, one alternative is to use multivariate Lagrange interpolation using `Fermat` [56]. We did so to compute $OC(s, t, v_1, v_2, v_3)$ which took 1 hour 15 minutes. Incorporating the rotation

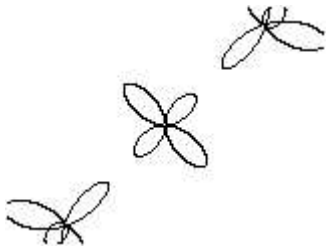


Figure 5.5: *Some outlines of a quartic surface moving from left to right and downwards.*

parameters $(\omega_1, \omega_2, \omega_3)$ leads to more complex polynomials and the corresponding occluding contour computations always took more time than for cases involving the $v_i's$. As an illustration, computation of $OC(s, t, \omega_1, \omega_2, \omega_3)$ took 3 hours 34 minutes.

5.1.3 Real Projections

In the previous sections, we used a very simple projection matrix $\mathbf{M}$ which lead to projection equations that were linear in x and y. Consequently, we eliminated x and y by substitution and saved computation time. In practice, the projection matrix obtained after calibrating a real camera is more complex and leads to non-linear projection equations in x, y, z. For such cases the following heuristic can be applied. We consider the case of $OC(s, t, v_1)$. It should be recalled from Chapter 2 that the Dixon resultant can eliminate n variables from $n + 1$ equations.

1. Eliminate x and y from P, Q, H to get a polynomial $A(s, t, z, v_1)$
2. Eliminate x and y from P, Q, V to get a polynomial $B(s, t, z, v_1)$
3. Eliminate z from A, B to get the occluding contour $OC(s, t, v_1)$

It should be observed that here too, for each elimination step,

1. factoring out the Dixon matrix removes spurious factors and reduces determinant computation time, and
2. using `Fermat's` multivariate Lagrange interpolation is always better than computing the determinant in `MAPLE`.

Infact, for cases involving more pose parameters, factored-out Dixon matrices and multivariate Lagrange interpolation were the only feasible option. As an example, we consider the following real projection matrix obtained after camera calibration:

$$\mathbf{M}_r = \begin{bmatrix} 0.759472 & -0.0899725 & -0.566994 & 200.1 \\ -0.264048 & -0.863091 & -0.234716 & 297.341 \\ 0.000507187 & -0.000370792 & 0.000518946 & 1.0 \end{bmatrix}. \tag{5.3}$$

The image in Figure 5.6 was taken using a camera whose projection matrix was $\mathbf{M}_r$.

Figure 5.6: *An image taken using a camera with projection matrix* $\mathbf{M}_r$.

We also replace the quartic surface used earlier by another quartic surface that represents the paper puncher shown in Figure 5.6 obtained through 3L fitting as shown in Figure 4.4(a). Therefore we now have a scenario with a real 3D object and a real camera. $OC(s, t, v_1, v_2, v_3)$ for this scenario was computed on a 2.4 GHz Opteron machine with 8 GB RAM using the above heuristics in 9 hours 37 minutes using `Fermat`. The polynomial contained 88179 monomials storing of which required 22.2 MB. The partial derivatives w.r.t. s and t contained 75582 monomials each while those w.r.t. to the $v_i's$ contained 74256 monomials each. Evaluation of $OC(s, t, v_1, v_2, v_3)$ on a 288×384 image took 9 seconds. Figure 5.7 shows the zero-sets of $OC(s, t, v_1, v_2, v_3)$ obtained for arbitrary values of the $v_i's$. Since the size of the occluding contour parametrised by the $v_i's$ is already hard to handle for the pose estimation procedure (explained later), we did not attempt to compute $OC(s, t, \omega_1, \omega_2, \omega_3)$.

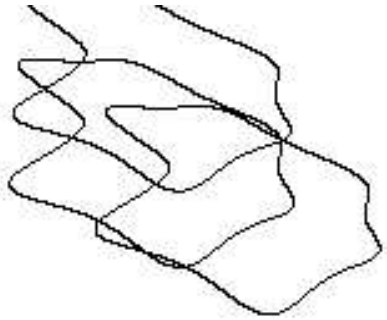


Figure 5.7: *Some outlines of a real-world puncher viewed through a real camera projection* $\mathbf{M}_r$.

Remark For polynomial evaluation on images, a naive approach would be to evaluate the polynomial for each pixel by subsituting the pose values and the pixel coordinates. However, this is extremely time-consuming for large polynomials. Evaluation can be optimised as follows

1. Substitute the numerical values of the pose parameters on the polynomial expression $OC(s, t, \mathbf{v}, \omega)$ to obtain $OC(s, t)$.

2. Add-up all coefficients of $OC(s,t)$ corresponding to the same power-pair $s^\alpha t^\beta$ to obtain $OC^*(s,t)$.

3. Evaluate $OC^*(s,t)$ on each pixel.

If the degree of the polynomial in s and t is d, then step 2 reduces the size of the polynomial to $\binom{2+d}{2}$ monomials. So for 4th-order algebraic surfaces, since $OC(s,t,\mathbf{v},\omega)$ is of degree 12 in s and t, step 2 will reduce the size to 91 monomials regardless of how large the polynomial originally was. To illustrate the effect of such an optimisation, evaluation of 88179 term polynomial $OC(s,t,v_1,v_2,v_3)$ from the last section on a 288×384 image took 1 hour 40 minutes using the naive approach. In contrast, evaluation of the optimised 91 term polynomial $OC^*(s,t)$ took only 9 seconds.

Polynomial	**# Monomials**	**# Variables**	**Evaluation Time on 288×384 image**
Original	88179	5	1 hour 40 minutes
Optimised	91	2	9 seconds

Table 5.2: *Effect of optimising polynomials before evaluation on images. Substituting pose values in $OC(s,t,v_1,v_2,v_3)$ and then reducing it to a minimal sized polynomial can significantly reduce evaluation time.*

5.2 Pose Estimation

Let $\mathcal{D} = \{p_1, \ldots, p_q\}$ be a set of image pixels representing a binary image outline of the object in an arbitrary position. Given $\mathcal{D}$, we can minimise the *distance* between outline pixels and the zero-set of $OC(s,t,\mathbf{v},\omega)$ to obtain the optimal twist coordinates $\mathbf{v},\omega$ from which the rigid body motion can be computed that transforms the 3D model into alignment with the image data. As discussed in Chapter 3, there is no closed-form expression for the exact Euclidean distance between a point and an implicitly defined curve or surface. Alternatives are:

1. First-order approximations of exact Euclidean distance (see Section 4.4) such as

$$\frac{|OC(s,t,\mathbf{v},\omega)|}{\|\nabla OC(s,t,\mathbf{v},\omega)\|}. \tag{5.4}$$

2. Algebraic distance $|OC(s,t,\mathbf{v},\omega)|$.

For first-order approximation, the *approximate mean square distance* $\Delta^2_{\mathcal{D}}(\mathbf{v},\omega)$ between the set of zeros of $OC(s,t,\mathbf{v},\omega)$ and the image point set $\mathcal{D}$ is given by

$$\Delta^2_{\mathcal{D}}(\mathbf{v},\omega) = \frac{1}{q}\sum_{i=1}^{q} \frac{OC(p_i,\mathbf{v},\omega)^2}{\|\nabla OC(p_i,\mathbf{v},\omega)\|^2} \tag{5.5}$$

whereas, for the algebraic distance, it becomes

$$\Delta^2_{\mathcal{D}}(\mathbf{v},\omega) = \frac{1}{q}\sum_{i=1}^{q} OC(p_i,\mathbf{v},\omega)^2 \tag{5.6}$$

5.2.1 Approximate vs. Algebraic Distance

We now compare the approximate and algebraic distance measures. We start with the simple case of a sphere viewed through projection matrix $\mathbf{M}$ as described at the beginning of this chapter. For illustrative purposes, we assume that our model sphere can only move in the horizontal direction and is therefore parametrised only by v_1. We substitute $v_2 = 0, v_3 = 0$ in (5.2) to to obtain

$$\begin{aligned} OC(s,t,v_1) =& (13600 + t^2 - 120t)v_1^2 + (-6400000 + 80000s)v_1 \\ &+ 155100t^2 + 1502000000 - 24816000s - 18612000t + 155100s^2 \end{aligned}$$

Experiment 5.2.1. *To verify that the mean square distances (5.5) and (5.6) are indeed minimum at the correct value of v_1, we perform the following steps*

1. *Extract the set of zeros of $OC(s,t,20)$ which means the outline for the sphere moved horizontally by -20 units. This gives us the set of image points $\mathcal{D} = \{p_1, \ldots, p_q\}$ as shown in Figure 5.8.*
2. *Plot (5.5) for varying v_1.*
3. *Plot (5.6) for varying v_1.*

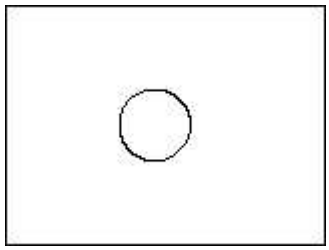
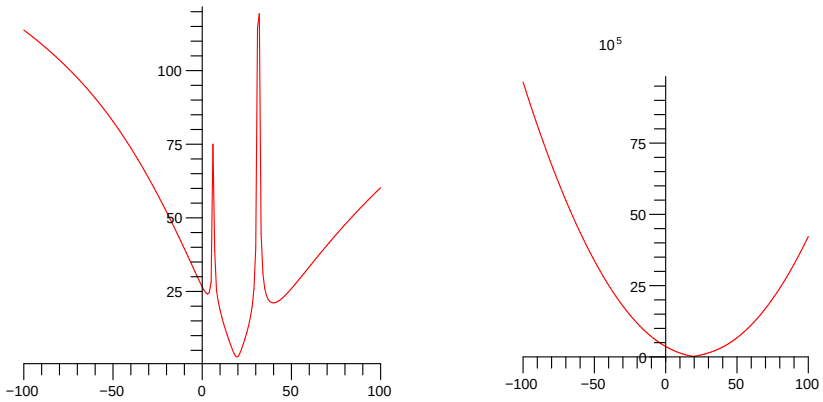


Figure 5.8: *Outline of sphere translated horizontally by -20 units and viewed through* $\mathbf{M}$.

Figure 5.2.1 shows the resulting plots for values of v_1 around the true minimum of 20. As can be seen, both plots are minimum at the correct value of $v_1 = 20$. However, for the case of approximate distance there are also local minima surrounding the global minimum while the plot for algebraic distance is smoother. It can also be observed that the approximate distance really does approximate the actual Euclidean distance while algebraic distance is far-off. However, since we are interested in the minimum only, using the algebraic distance allows us to avoid the local minima. Some more insight can be gained by writing out the residual functions for the approximate and algebraic distances. For clarity, we will abbreviate $OC(s,t,v_1)$ by simply OC. For the approximate distance, the residual function and its partial derivative w.r.t. v_1 become

$$R = \sqrt{\frac{OC^2}{OC_s^2 + OC_t^2}},$$

$$\frac{\partial}{\partial v_1} R = \frac{1}{\sqrt{\frac{OC^2}{OC_s^2+OC_t^2}}} \left(\frac{OC \frac{\partial}{\partial v_1} OC}{{OC_s}^2 + {OC_t}^2} - \frac{OC^2 \left(OC_s \frac{\partial}{\partial v_1} OC_s + OC_t \frac{\partial}{\partial v_1} OC_t \right)}{\left(OC_s^2 + OC_t^2\right)^2} \right)$$



(a) Using first-order approximation of exact distance.

(b) Using algebraic distance.

Figure 5.9: *The mean squared distance $\Delta^2_{\mathcal{D}}(v_1)$ using (a) approximate distance and (b) algebraic distance as functions of the sphere translation parameter v_1. The global minimum at 20 is surrounded by local minima on each side for the case of approximate distance while algebraic distance leads to a smoother residual function.*

where OC_s and OC_t are the appropriate partial derivatives. This requires evaluating the 6 polynomials $OC, OC_{v_1}, OC_s, OC_t, OC_{s_{v_1}}, OC_{t_{v_1}}$. In contrast, when algebraic distance is used the same functions become

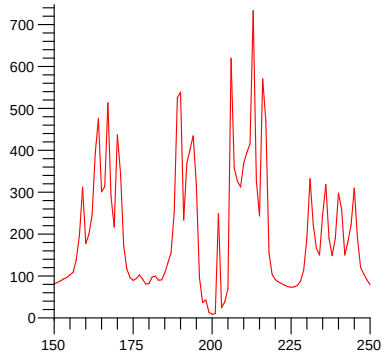
$$R = |OC|,$$
$$\frac{\partial R}{\partial v_1} = sign(OC)\frac{\partial}{\partial v_1}OC$$

which requires evaluating only 2 polynomials OC and OC_{v_1}. Figure 5.10 shows the result of Experiment (5.2.1) repeated for the quartic surface from Figure 5.3 using horizontal translation of -200.

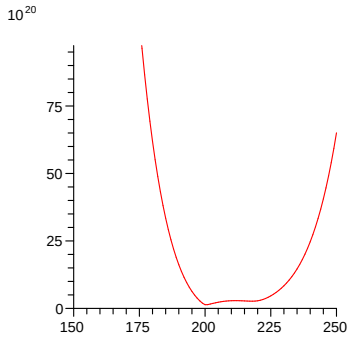
For the case of horizontal as well as vertical translation of the standard model, similar to Experiment (5.2.1), we set both v_1 and v_2 to 0 in $OC(s, t, v_1, v_2)$, extract the zero-set and then plot the magnitude of the residual vector for $-10 \leq v_1 \leq 10$ and $-10 \leq v_2 \leq 10$. For the case of approximate distance, Figure 5.11 shows that the residual vector magnitude is minimum at the correct values 0 and 0 and that the function is convex far away from the global minimum. However, this global minimum is surrounded by local minima. For the case of algebraic distance, Figure 5.12 shows that the residual vector magnitude is minimum at the correct values 0 and 0 and that the function is convex with no local minima occurring in a considerable region around the global minimum.

In summary, we conclude that while approximate distance might be more accurate it leads to residual functions that

1. are larger in size and therefore computationally more expensive, and
2. have more local minima around the global minimum.

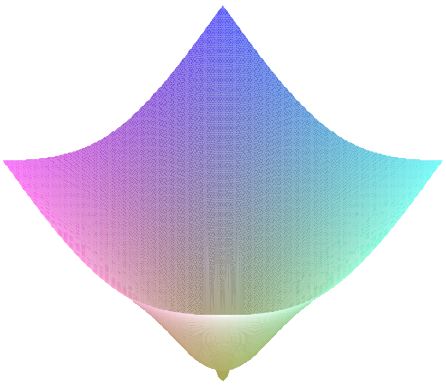


(a) Using first-order approximation of exact distance.

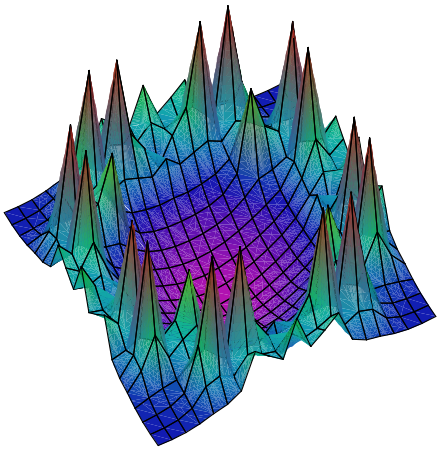


(b) Using algebraic distance.

Figure 5.10: *The mean squared distance* $\Delta^2_{\mathcal{D}}(v_1)$ *using (a) approximate distance and (b) algebraic distance as functions of a quartic's pose parameter* v_1*. The global minimum at* 200 *is surrounded by numerous local minima on each side for the case of approximate distance while algebraic distance leads to a smoother residual function.*

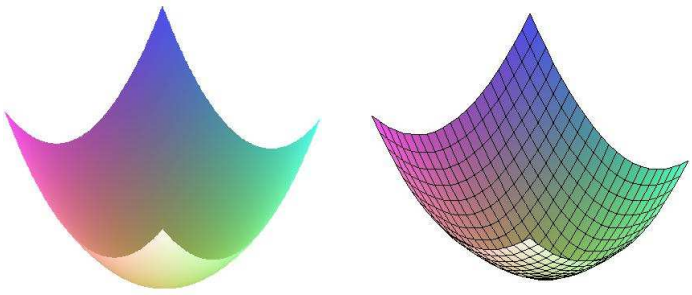


(a) Residual is convex far away from the global minimum. Local minima are close to the global minimum.



(b) A close-up.

Figure 5.11: *The sum of squared approximate distances as a function of the sphere translation parameters* v_1 *and* v_2*. The global minimum at* $0, 0$ *is surrounded by local minima.*



(a) Residual is convex around the global minimum.

(b) A close-up.

Figure 5.12: *The sum of squared algebraic distances as a function of the sphere translation parameters v_1 and v_2. The function is smooth and convex around the global minimum at $0, 0$.*

Using the approximate distance measure becomes practically infeasible when real projection matrices are used since they lead to occluding contours that are very large polynomials as discussed in Section 5.1.3. On the other hand, while algebraic distance is nowhere near an exact distance measure, it leads to residual functions that

1. are smaller in size and therefore computationally less expensive, and
2. have fewer local minima.

5.2.2 Experiments

As can be seen from Figure 5.2.1, the optimal v_1 minimises $\Delta^2_{\mathcal{D}}(v_1)$. Since $\Delta^2_{\mathcal{D}}(v_1)$ is non-linear in v_1, a non-linear minimisation technique such as the Levenberg-Marquardt algorithm is required to find the optimal v_1. For the general case of estimating $(\mathbf{v}, \omega)$, the non-linear least squares problem involves minimising the length of the residual vector $R = (R_1, \ldots, R_q)$:

$$\| R(\mathbf{v}, \omega) \|^2 = \sum_{i=1}^{q} R_i(\mathbf{v}, \omega)^2$$

where

$$R_i(\mathbf{v}, \omega) = |OC(p_i, \mathbf{v}, \omega)^2|$$

is the algebraic distance from point p_i to $Z(OC(\mathbf{v}, \omega))$. Once R is defined, we can start the minimisation (see details in Appendix A.1). In the following we will check robustness of the estimation procedure in the presence of increasing values of radial noise added to the image silhouette pixels. By radial noise, we mean that each pixel is translated by a normally distributed random variable lying between 0 and a maximum radius value and then rotated around the original position by an angle uniformly distributed between 0° and 360°. For the noise experiments, maximum noise radius was increased from 0 to 12 pixels in increments of 4 pixels. Standard deviation was always kept at half the maximum noise radius. We will explore both the monocular and stereo cases. For the monocular

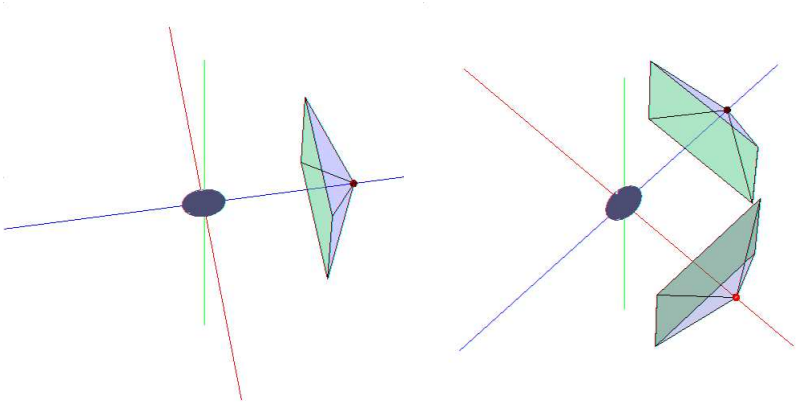
case, we use projection matrix $\mathbf{M}$ as described earlier. For the stereo setup, we use an additional projection matrix $\bar{\mathbf{M}}$:

$$\bar{\mathbf{M}} = \begin{bmatrix} -80 & 0 & 100 & 104000 \\ -60 & -100 & 0 & 48000 \\ -1 & 0 & 0 & 800 \end{bmatrix}.$$

$\bar{\mathbf{M}}$ represents projection onto a 120×160 image plane parallel to the yz-plane and placed at $x = 300$ with focal point $(400, 0, 0)$.

Scenario 1

1. Surface: An ellipsoid[1] $\frac{x^2}{5^2} + \frac{y^2}{5^2} + \frac{z^2}{8^2} - 7^2 = 0$
2. Projection: $\mathbf{M}$
3. Pose Parameters: v_1, v_2, v_3.
4. Ground-truth: $50, 50, 250$.



(a) Scenario 1: An ellipsoid viewed through projection matrix $\mathbf{M}$.

(b) Scenario 2: An ellipsoid viewed through projection matrices $\mathbf{M}$ and $\bar{\mathbf{M}}$.

Figure 5.13: *The monocular and stereo setups.*

Figure 5.14 shows how the minimisation proceeds in the image plane starting from an initial estimate of $20, 20, 360$ for which the corresponding outline is shown in green. The blue outlines illustrate the convergance to the ground-truth of $50, 50, 250$ shown in red.

While we use algebraic error for minimisation purposes it does not give much help in determining the accuracy of the estimated pose. Therefore we use the Euclidean norm between the estimated pose and ground-truth to check accuracy. For further analysis, we also compute an error between outline images as follows

1. Compute Euclidean distance transform of ground-truth outline.

[1]This surface is also called a prolate spheroid

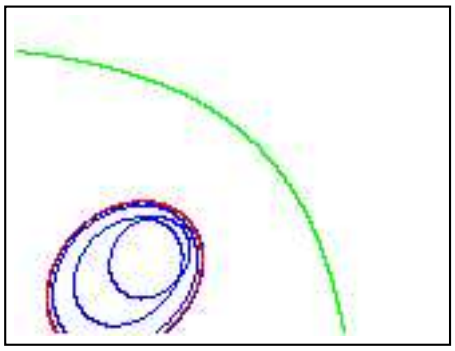


Figure 5.14: *Minimisation in the image plane. The green outline corresponds to the model in the initial estimated position. The blue outlines are obtained during successive iterations of the minimisation. Red outline is the true position.*

2. Compute Euclidean distance transform of estimated outline.

3. Compute image error as the mean absolute difference of the two distance transform images.

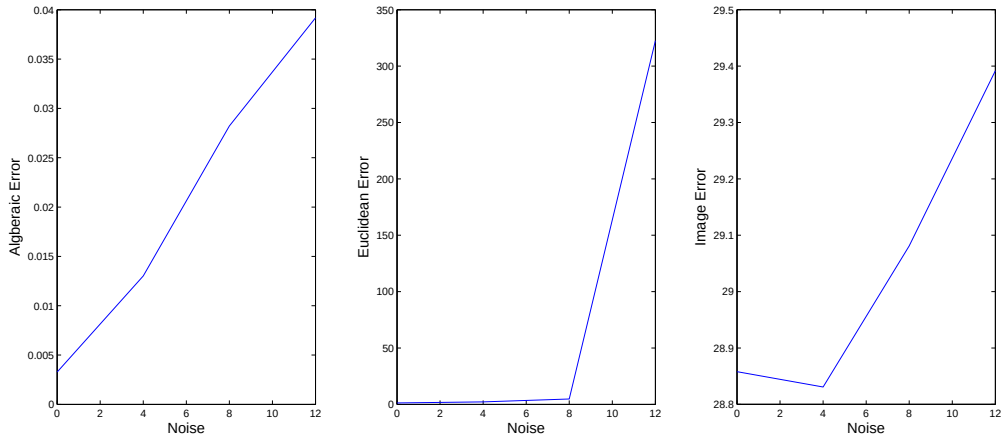


Figure 5.15: *Variation of algebraic, Euclidean and image errors with increasing noise for scenario 1.*

Figure 5.16 shows image outlines affected by radial noise. Robustness to such noise is illustrated in Figure 5.15. It should be observed that Euclidean error rapidly increases for a noise radius of 12 pixels while both algebraic and image errors do not suffer from such a drastic change. The reason for this is illustrated by plotting the occluding contours for the estimated pose $-47, -44, -542$ for noise radius 12 and the ground-truth as shown in Figure 5.17. As can be seen, both occluding contours are very similar despite the difference in poses. This is a clear example of non-uniqueness of the pose in the image plane. The reason for this particular ambiguity is that the tangent cone extends to both sides of the focal point and hence there are 2 poses for each outline, one real and one virtual. For instance, with focal point $0, 0, -400$ of projection matrix $\mathbf{M}$, the virtual counterpart of the pose $50, 50, 250$ is $-50, -50, 550$. Using a stereo setup such as the one shown in Figure 5.13(b) allows us to reduce the effect of such ambiguities.

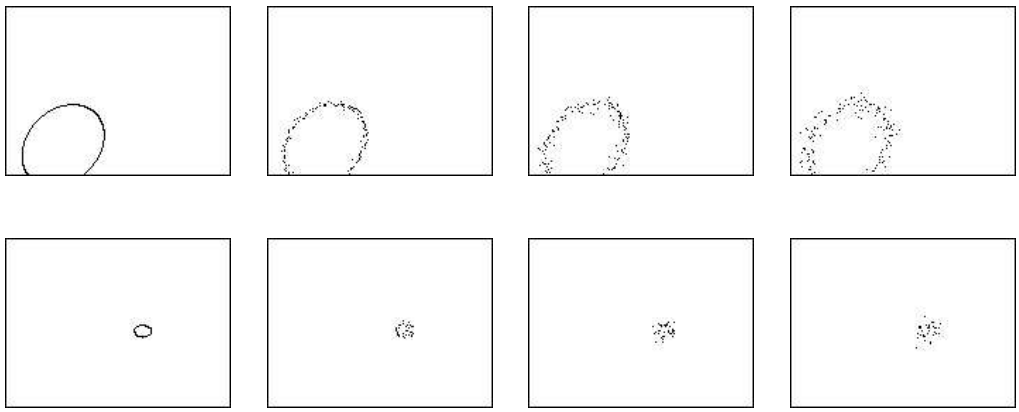
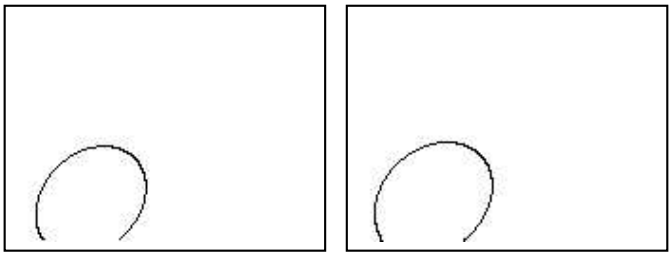


Figure 5.16: ***Left to right:*** *Image outlines affected by radial noise of radius 0,4,8 and 12 pixels. Top row is the outline as seen from projection matrix* $\mathbf{M}$ *while bottom row is for the outline as seen from projection matix* $\bar{\mathbf{M}}$.



(a) $OC(50, 50, 250)$ (b) $OC(-47, -44, 542)$

Figure 5.17: *Two distinct poses can reveal very similar outlines. The second pose* $-47, -44, 542$ *is very close to the 'virtual counterpart'* $-50, -50, 550$ *of the first pose* $50, 50, 250$.

Scenario 2:

1. Scenario 1 combined with projection matrix $\bar{\mathbf{M}}$.

For a stereo setup with occluding contours OC_1 and OC_2, the combined residual function using algebraic distance can be defined as

$$R = \sqrt{OC_1^2} + \sqrt{OC_2^2} \tag{5.7}$$

whose partial derivative w.r.t. v_i

$$\frac{\partial}{\partial v_i} R = sign(OC_1)\frac{\partial}{\partial v_i} OC_1 + sign(OC_2)\frac{\partial}{\partial v_i} OC_2 \tag{5.8}$$

Variation of errors with increasing noise for the stereo setup is shown in Figure 5.18. It can be observed that accuracy improved since Euclidean error did not jump drastically for a noise radius of 12 pixels.

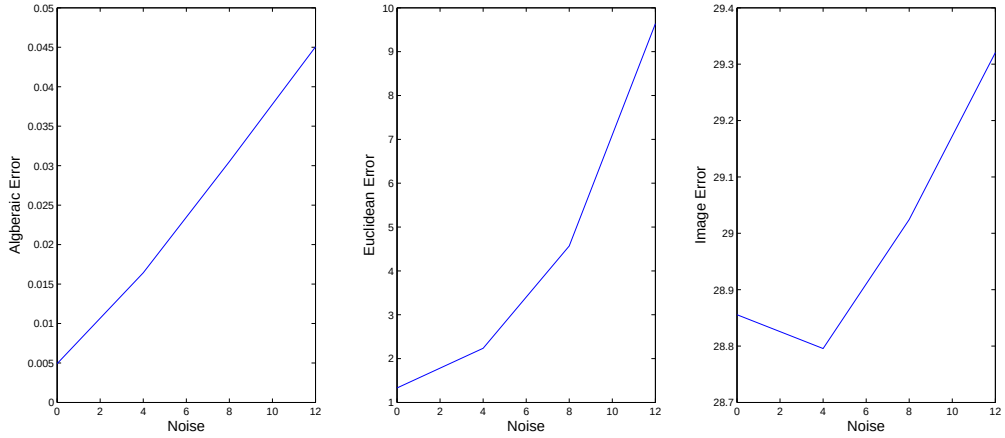


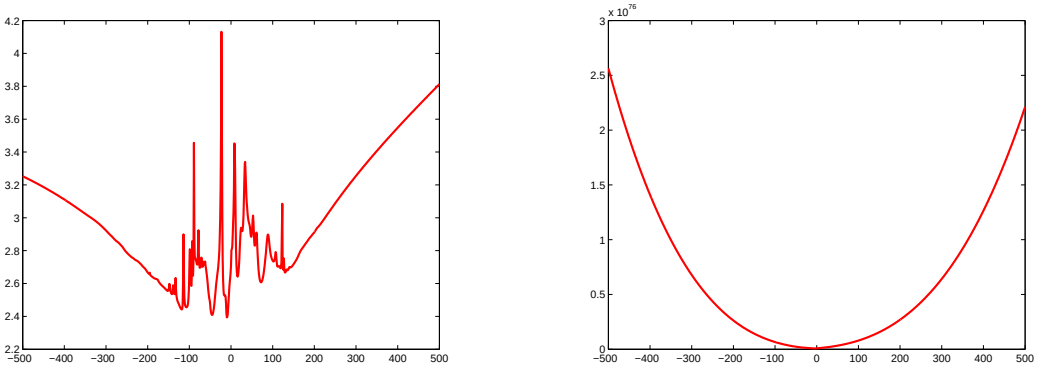
Figure 5.18: *Variation of algebraic, Euclidean and image errors with increasing noise for stereo setup of scenario 2.*

Scenario 3

1. Surface: A real-world puncher represented as a 4th-order algebraic surface as shown in Figure 4.4(a) in Chapter 4.

2. Projection: Real-world projection matrix $\mathbf{M_r}$ from Section 5.1.3

We start with the case of just 1 pose parameter v_1. Here too, the choice between algebraic and approximate distance is made easier by observing the residual plots for different values of v_1 around the ground-truth in Figure 5.19. It can be seen that the approximate distance leads to an error profile with many local minima while algebraic distance leads to a smooth error profile. Given an initial pose estimate for the puncher in Figure 5.6,



(a) Using first-order approximation of exact distance.

(b) Using algebraic distance.

Figure 5.19: *The mean squared distance $\Delta^2_{\mathcal{D}}(v_1)$ using (a) approximate distance and (b) algebraic distance as functions of a puncher's pose parameter v_1. The global minimum at 0 is surrounded by numerous local minima on each side for the case of approximate distance while algebraic distance leads to a smoother residual function.*

we try to estimate the correct pose. Figure 5.20 shows the convergence behaviour of our algorithm when approximate distance is used. Starting from an initial estimate of 300, the minimisation gets stuck in a local minimum at 136. This behaviour conforms to the residual profile in Figure 5.19(a). For the case of algberaic distance, Figure 5.21 shows

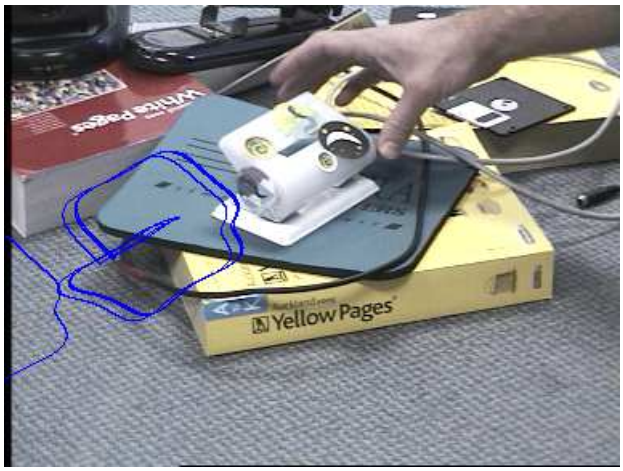


Figure 5.20: *Convergence behaviour when approximate distance is used. Starting from an initial estimate of 300, the minimisation gets stuck in a local minimum at 136. This behaviour conforms to the residual profile in Figure 5.19(a).*

the convergence behaviour of our algorithm when the image outline is affected by missing data and radial noise. It can be seen that in this particular instance, results are quite robust. Robustness to noise and missing data is also exhibited in the error graphs shown in Figure 5.23. Figure 5.2.2 shows the convergence behaviour when 2 pose paramters v_1 and v_2 are estimated.

5.3 A Hybrid Approach

In this section we present a hybrid explicit-implicit approach. It is basically an extension of the explicit approach used by Rosenhahn in [58] which can be summarised as

1. Reconstruct projection rays from the image points
2. Find correspondences between the projection rays and the 3D model points.
3. Estimate the pose using this correspondence set.
4. Goto 2.

The only change we make to this approach is that instead of using all 3D model points in step 2, we can use only those points that form the contour generator. To find such points we use the implicit algebraic representation $P(\mathbf{x}) = 0$ of the 3D model and perform the following test on each point $\mathbf{x}$ of the 3D model

2a. Compute the normal $\nabla P(\mathbf{x})$ at $\mathbf{x}$. Normalise it.

2b. Compute the ray $\mathbf{r} = \mathbf{x} - \mathbf{f}$ passing through $\mathbf{x}$ and the focal point f. Normalise it.

2c. If $\mathbf{r} \cdot \nabla P(\mathbf{x}) < \epsilon$ for some threshold ϵ close to 0, x lies on the contour generator.

Figure 5.25 shows two views of the contour generator of the puncher model with respect to a fixed focal point. These results can be improved by using the approach used by Ilic in [59] whereby the search for the contour generator is formulated as the solution to an ordinary differential equation (ODE). Let $\mathbf{x}(t), t \in [0, 1]$ be the contour generator on the algebraic surface $P(\mathbf{x}) = 0$. Then $\mathbf{x}(t)$ is a solution of the ODE

$$\frac{\partial \mathbf{x}(t)}{\partial t} = \frac{(H(\mathbf{x}(t)))(\mathbf{x}(t) - \mathbf{f}) \times \nabla P(\mathbf{x}(t))}{\|(H(\mathbf{x}(t)))(\mathbf{x}(t) - \mathbf{f}) \times \nabla P(\mathbf{x}(t))\|} \tag{5.9}$$

where $H(\mathbf{x}(t))$ is the Hessian matrix of $P(\mathbf{x}(t))$, $\nabla P(\mathbf{x}(t))$ is the gradient vector and $\mathbf{f}$ is the focal point of the camera. Solving this ODE requires an initial contour generator point $\mathbf{x}(0)$.

Since in [58] it is possible to weight the correspondences in step 3, we can also use the other (non-contour generating) 3D model points with lesser weights. The value of the dot product $\mathbf{r} \cdot \nabla P(\mathbf{x})$ can also be used to choose appropriate weights. Figure 5.26 shows the convergence behaviour of this hybrid approach for the puncher model. Given an initial pose estimate of the model, the contour generator (shown in yellow) is computed and registered with the reconstructed projection rays to give a new pose estimate. The model is transformed by this pose estimate and the process is repeated until convergence.

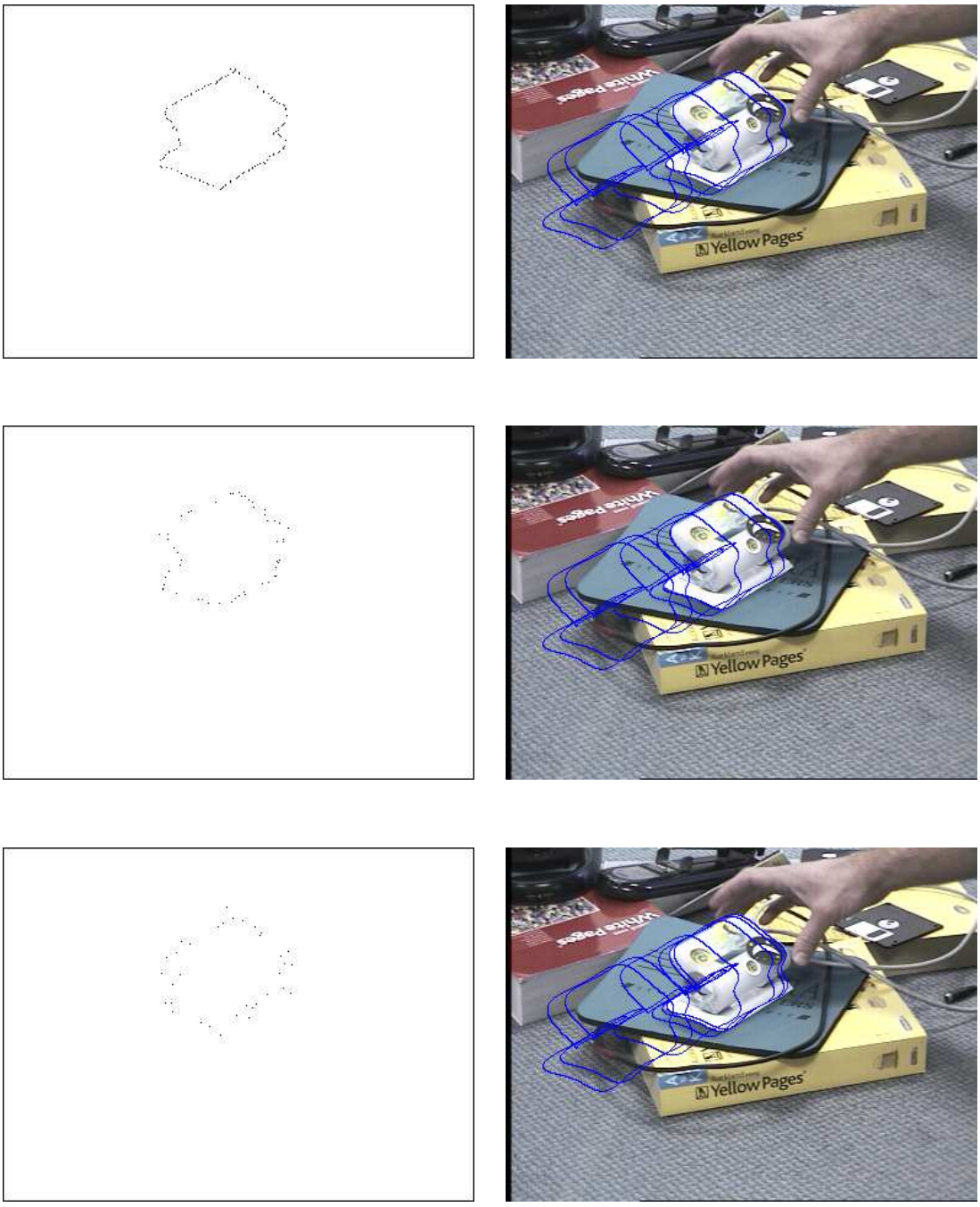


Figure 5.21: *Convergence behaviour using algebraic distance in presence of noise and missing data.*

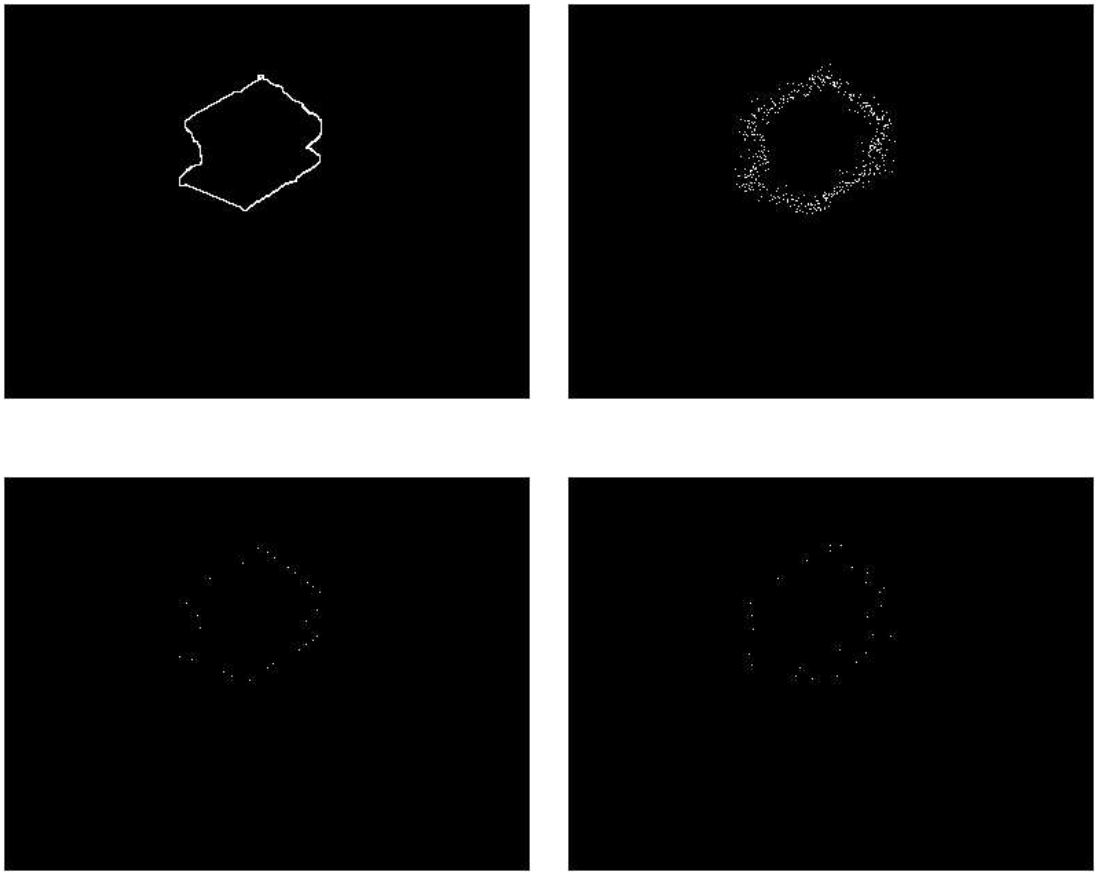


Figure 5.22: *Image outline affected by missing data and radial noise. Top to bottom:* $100\%, 5\%$ *data. Left to right:* $0, 12$ *pixels radial noise.*

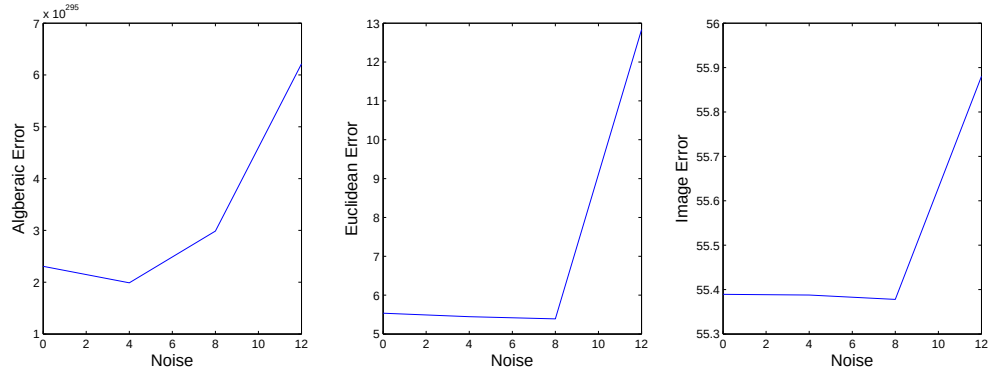


Figure 5.23: *Variation of algebraic, Euclidean and image errors with increasing noise for scenario 3 using* 5% *image data.*

Figure 5.24: *Convergence behaviour using algebraic distance when estimating 2 pose parameters* v_1, v_2.

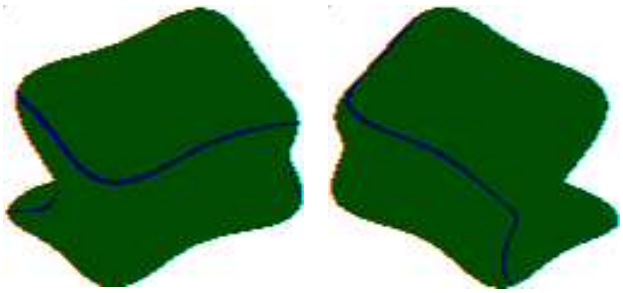


Figure 5.25: *2 views of the occluding contour (in blue) of the puncher model with respect to a fixed focal point.*

(a) Reconstructed rays, initial pose and its corresponding contour generator (in yellow).

(b) Convergence behaviour.

Figure 5.26: *Given an initial pose estimate of the model, the contour generator (in yellow) is computed and registered with the reconstructed projection rays to give a new pose estimate. The model is transformed by this pose estimate and the process is repeated until convergence.*

Chapter 6

Conclusion

We have explored the problem of silhouette-based 2D-3D pose estimation using implicit algebraic surfaces. Our approach consisted of 3 phases.

1. Modelling of 3D objects using implicit polynomials.
2. Projection of algebraic contour generators onto algebraic occluding contours.
3. Pose estimation using image silhouette data and occluding contour equations.

The first two phases involve offline symbolic and numeric computation. Their result is then used in the final online pose estimation phase.

Our work highlights the fundamental problem of implicit formulations of projective registration, namely the rapidly increasing degree of projected polynomials. This leads to two difficulties:

1. How can such high-degree projected polynomials be computed?
2. Once computed, how can we use them for pose estimation purposes?

To address the first difficulty, we have used advanced heuristics for the Dixon resultant developed by Kapur et al. [38] combined with the specialised computer algebra system `Fermat`. We have been able to compute real-world projections of 4th-order algebraic surfaces parametrised by 3D translation parameters. To the best of our knowledge, this is a small step forward in algebraic surface projections.

The second difficulty, however, still remains. Since it is problematic to compute projections of 4th-order algebraic surfaces using all 6 pose parameters, the estimation problem needs to be decoupled into separate translation and rotation estimation. Furthermore, we have used linearised twist parameters to represent pose which was succesfully used for a fast implementation of explicit pose estimation in [58]. While this leads to significantly smaller polynomials, the approach can only work when projection of algebraic surfaces is performed online. This calls for exploring polynomial algebra libraries such as SYNAPS [62] and those developed under the FRISCO project [28] such as *Algébre Linéaire pour les Polynômes* (ALP) [4] and MARS [1] which have some multivariate resultant implementations.

These difficulties beg the question, "Why bother with implicit polynomials when they are hard to compute and then hard to use?". The reason for exploring implicit polynomials is that they offer a global description of objects and hence can potentially succeed in situations where methods based on local descriptions fail.

To completely avoid dealing with high-degree projected polynomials, we have also presented a hybrid implicit-explicit approach similar to the one presented by Ilic et al. [59]. Staying purely in the implicit domain, we can attempt to reduce polynomial degrees by representing 3D objects using Fourier coefficients which can then be implicitised [70].

Appendix A

Appendix

A.1 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm is a popular heuristic for finding the minimum of a function $F(\mathbf{u})$ that is a sum of squares of *nonlinear* functions of $\mathbf{u} = (u_1, \cdots, u_r)$,

$$F(\mathbf{u}) = \sum_{i=1}^{q} R_i(\mathbf{u})^2 \tag{A.1}$$

where the R_i are called *residuals*, $q \geq r$ is the number of data points. The goal of the Levenberg-Marquardt algorithm is to find the parameter vector $\mathbf{u}$ for which (A.1) is minimised. Alternatively, to find the $\mathbf{u}$ for which the magnitude of the *residual vector* $R(\mathbf{u}) = (R_1(\mathbf{u}), \ldots, R_q(\mathbf{u}))$ is minimised. The algorithm starts with an initial estimate $\mathbf{u}^0$ and iteratively converges to a local minimum based on the following iteration step:

$$\mathbf{u}^{k+1} = \mathbf{u}^k - (H(\mathbf{u}^k) + \mu_k diag(H))^{-1} J(\mathbf{u}^k)^t R(\mathbf{u}^k) \tag{A.2}$$

where $J(\mathbf{u})$ is the Jacobian of R with respect to $\mathbf{u}$

$$J_{ij}(\mathbf{u}) = \frac{\partial R_i}{\partial u_j}(\mathbf{u}),$$

$H(\mathbf{u})$ is the Hessian of R with respect to $\mathbf{u}$

$$H(\mathbf{u}) = J(\mathbf{u}) J(\mathbf{u})^t,$$

and μ is a small nonnegative number varied during iterations to *maneuver* the minimisation between gradient descent and Gauss-Newton iteration depending on how far the estimate is from a local minimum.

Bibliography

[1] Aaron Wallack and Ioannis Z. Emiris and Dinesh Manocha. MARS: A MAPLE/-MATLAB/C Resultant-Based Solver. In *International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 244–251, 1998.

[2] Alston S. Householder. Unitary Triangularization of a Nonsymmetric Matrix. *Journal ACM*, 5(4):339–342, 1958.

[3] Arthur D. Chtcherba. Maple Package for Dixon / Bézout Resultant Computation. http://www.cs.panam.edu/∼cherba/Projects/maple-dixon/index.html.

[4] B. Mourrain. Algébre Linéaire pour les Polynômes. http://www−sop.inria.fr/saga/logiciels/ALP/.

[5] B. Rosenhahn and U. Kersting and D. Smith and J. Gurney and T. Brox and R. Klette. A System for Marker-less Human Motion Estimation. In W. Kropatsch, R. Sablatnig, and A. Hanbury, editors, *Pattern Recognition 2005, DAGM, Wien*, LNCS 3663, pages 230–237, Berlin Heidelberg, 2005. Springer-Verlag.

[6] Paul Besl and Neil McKay. A method for registration of 3-D shapes. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 14(2), pages 239–256, 1992.

[7] M. M. Blane, Z. Lei, and David B. Cooper. The 3L Algorithm for Fitting Implicit Polynomial Curves and Surfaces to Data. Lems tr-160, LEMS, Brown University, Providence, RI, 1996. http://www.lems.brown.edu/∼jpt/lems160.html.

[8] Michael M. Blane, Zhibin Lei, Hakan Civi, and David B. Cooper. The 3L Algorithm for Fitting Implicit Polynomial Curves and Surfaces to Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(3):298–313, 2000.

[9] Bloomenthal, Jules, editor. *Introduction to Implicit Surfaces.* The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Fransisco, California, 1997.

[10] F. Boughorbel, A. Koschan, B. Abidi, and M. Abidi. Gaussian Energy Functions for Registration without Correspondences. In *17th International Conference on Pattern Recognition (ICPR)*, August 2004.

[11] Bruce Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, Stanford University, 1974.

[12] Darius Burschka, Ming Li, Russell Taylor, and Gregory D. Hager. Scale-Invariant Registration of Monocular Stereo Images to 3D Surface Models. In *Proceedings of International Conference on Intelligent Robots and Systems (IROS)*, pages 2581–2586, 2004.

[13] Richard J. Campbell and Patrick J. Flynn. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding*, 81(2):166–210, 2001.

[14] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, W. Richard Fright, Bruce C. McCallum, and Tim R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In Eugene Fiume, editor, ***S**pecial **I**nterest **G**roup for Computer **GRAPH**ics (SIGGRAPH) 2001, Computer Graphics Proceedings*, pages 67–76. ACM Press / ACM SIGGRAPH, 2001.

[15] G. Champleboux, S. Lavallee, R. Szeliski, and L. Brunie. From Accurate Range Imaging Sensor Calibration to Accurate Model-Based 3D Object Localization. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 83–89, June 1992.

[16] Yang Chen and Gérard Medioni. Object Modelling by Registration of Multiple Range Images. *Image and Vision Computing*, 10(3):145–155, 1992.

[17] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek. The trimmed iterative closest point algorithm. In *International Conference on Pattern Recognition, Quebec, Canada, 2002. IEEE Comp. Soc.*, 2002.

[18] Arthur D. Chtcherba and Deepak Kapur. On the Efficiency and Optimality of Dixon-based Resultant Methods. In *ISSAC '02: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 29–36, New York, NY, USA, 2002. ACM Press.

[19] David A. Cox, John B. Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms*. Undergraduate Texts in Mathematics. Springer, Berlin, 2005.

[20] David A. Cox, John B. Little, and Donal O'Shea. *Using Algebraic Geometry*, volume 185 of *Undergraduate Texts in Mathematics*. Springer, New York, NY, 2. ed. edition, 2005.

[21] C.M. Cyr, A.F. Kamal, T.B. Sebastian, and B.B. Kimia. 2D-3D Registration Based on Shape Matching. In *IEEE Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA)*, pages 198–203, June 2000.

[22] D. A. Forsyth. Recognizing Algebraic Surfaces From Their Outlines. *International Journal of Computer Vision*, 18(1):21–40, 1996.

[23] Wolfram Decker and Christoph Lossen. *Computing in Algebraic Geometry - A Quick Start Using SINGULAR* , volume 16 of *Algorithms and Computation in Mathematics*. Springer, Berlin, 2006.

[24] I. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, University of California at Berkeley, 1993.

[25] Ioannis Z. Emiris and Bernard Mourrain. Matrices in Elimination Theory. *Journal of Symbolic Computation*, 28(1–2):3–43, 1999.

[26] A. Fitzgibbon. Robust Registration of 2D and 3D Point Sets. In *British Machine Vision Conference*, volume II, pages 411–420, Manchester, UK, 2001. citeseer.ist.psu.edu/fitzgibbon01robust.html.

[27] David Forsyth, Joseph L. Mundy, Andrew Zisserman, and Charles Rothwell. *Symbolic and Numerical Computation for Artificial Intelligence*, chapter 7: Applications of Invariant Theory in Computer Vision. Computational Mathematics and Applications. Academic Press, 1992.

[28] FRISCO. http://www.nag.co.uk/projects/FRISCO.html.

[29] G. Cross and A. Zisserman. Quadric Surface Reconstruction from Dual-Space Geometry. In *Proceedings of the 6th International Conference on Computer Vision, Bombay, India*, pages 25–31, January 1998.

[30] G. Sommer, editor. *Geometric Computing with Clifford Algebras*. Springer-Verlag, Heidelberg, 2001.

[31] Jean Gallier. *Geometric Methods and Applications*, volume 38 of *Texts in Applied Mathematics*. Springer-Verlag, 2000.

[32] Jean-Philippe Tarel and Hakan Civi and David B. Cooper. Pose Estimation of Free-Form 3D Objects without Point Matching using Algebraic Surface Models. In *Proceedings of IEEE Workshop Model Based 3D Image Analysis*, pages 13–21, Mumbai, India, 1998. http://www-rocq.inria.fr/∼tarel/mb3ia.html.

[33] A. Johnson and S. Kang. Registration and Integration of Textured 3-D Data. Technical Report CRL 96/4, Digital Equipment Corp. Cambridge Research Lab. 25, 1996. citeseer.ist.psu.edu/johnson96registration.html.

[34] K. Siddiqi and J. Subrahmonia and D. Cooper and B. Kimia. Part-Based Bayesian Recognition Using Implicit Polynomial Invariants. In *IEEE International Conference on Image Processing, (Washington, D. C.), October 1995.*, 1995.

[35] Kang, K. and Tarel, J.-P. and Fishman, R. and Cooper, D. B. A Linear Dual-Space Approach to 3D Surface Reconstruction from Occluding Contours using Algebraic Surfaces. In *IEEE International Conference on Computer Vision (ICCV'01)*, volume I, pages 198–204, Vancouver, Canada, 2001. http://www-rocq.inria.fr/∼tarel/iccv01.html.

[36] Deepak Kapur and Yagiti N. Lakshman. *Symbolic and Numerical Computation for Artificial Intelligence*, chapter 2: Elimination Methods: an Introduction. Computational Mathematics and Applications. Academic Press, 1992.

[37] Deepak Kapur and Tushar Saxena. Comparison of Various Multivariate Resultant Formulations. In *ISSAC '95: Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, pages 187–194, New York, NY, USA, 1995. ACM Press.

[38] Deepak Kapur and Tushar Saxena. Extraneous Factors in the Dixon Resultant Formulation. In *ISSAC '97: Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, pages 141–148, New York, NY, USA, 1997. ACM Press.

[39] Deepak Kapur, Tushar Saxena, and Lu Yang. Algebraic and Geometric Reasoning Using Dixon Resultants. In *ISSAC '94: Proceedings of the 1994 International Symposium on Symbolic and Algebraic Computation*, pages 99–107, 1994.

[40] D. Keren. Using Symbolic Computation to Find Algebraic Invariants. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(11):1143–1149, 1994.

[41] Knossow, David and Ronfard, Remi and Horaud, Radu P. Human Motion Tracking with a Kinematic Parameterization of Extremal Contours. Research Report RR-6007, INRIA, 655 avenue de l'Europe, 38330 Montbonnot St Martin, October 2006.

[42] David J. Kriegman and Jean Ponce. On Recognizing and Positioning Curved 3-D Objects from Image Contours. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 12, pages 1127–1137, Dec 1990.

[43] Z. Lei, D. Keren, and D. B. Cooper. Computationally Fast Bayesian Recognition of Complex Objects Based on Mutual Algebraic Invariants. In *Proceedings of IEEE International Conference on Image Processing (ICIP'95)*, volume 3, pages 635–638, 23-26 Oct. 1995, Washington, DC, USA, October 1995.

[44] Z. Lei, T. Tasdizen, and D.B. Cooper. PIMs and Invariant Parts for Shape Recognition. In *Proceedings of Sixth International Conference on Computer Vision (ICCV'98)*, pages 827–832, Mumbai, India, 1998. Also as LEMS Tech. Report 163, Brown University.

[45] Robert H. Lewis and Peter F. Stiller. Solving the Recognition Problem for Six Lines Using the Dixon Resultant. *Mathematics and Computers in Simulation*, 49(3):205–219, 1999.

[46] Yi Ma, Stefano Saotta, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3-D Vision, From Images to Geometric Models*. Springer, New York, 2003.

[47] E.W. Mayr and A.R. Meyer. The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. *Advances in Mathematics*, 46:305–329, 1982.

[48] C.H. Menq, H.T. Yau, and G.Y. Lai. Automated Precision Measurement of Surface Profile in CAD-Directed Inspection. In *IEEE Transactions on Robotics and Automation*, volume 8, pages 268–278, April 1992.

[49] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1994.

[50] Mustafa Unel and William A. Wolovich. Pose Estimation and Object Identification Using Complex Algebraic Representations. citeseer.ist.psu.edu/233217.html.

[51] Van-Duc Nguyen, Victor Nzomigni, and Charles V. Stewart. Fast and Robust Registration of 3D Surfaces Using Low Curvature Patches. In *Second International Conference on 3-D Imaging and Modeling (3DIM '99)*, page 201, 1999.

[52] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer, New York, 2003.

[53] Phong, T.Q. and Horaud, Radu P. and Yassine, A. and Tao, P.D. Object Pose from 2D to 3D Point and Line Correspondences. *International Journal of Computer Vision*, 15(3):225–243, July 1995.

[54] J. Ponce and D.J. Kriegman. *Symbolic and Numerical Computation for Artificial Intelligence*, chapter 5: Elimination Theory and Computer Vision. Computational Mathematics and Applications. Academic Press, 1992.

[55] Robert Cipolla and Andrew Blake. Surface Shape from the Deformation of Apparent Contours. *International Journal of Computer Vision*, 9(2):83–112, 1992.

[56] Robert H. Lewis. Computer Algebra System Fermat. http://www.bway.net/∼lewis (also at http://www.fordham.edu/lewis/).

[57] Robert H. Lewis. Improving the Dixon Resultant for Polynomial Systems. Submitted for publication, 2006.

[58] B. Rosenhahn. *Pose Estimation Revisited*. PhD thesis, Inst. für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, 2003. http://www.ks.informatik.uni-kiel.de/∼vision/doc/Dissertationen/Bodo_Rosenhahn/tr0308.pdf.

[59] S. Ilic and M. Salzmann and P. Fua. Implicit Surfaces Make for Better Silhouettes. In *Conference on Computer Vision and Pattern Recognition, San Diego, CA*, volume 1, pages 1135–1142, June 2005.

[60] S. Ilic and P. Fua. From Explicit to Implicit Surfaces for Visualization, Animation and Modeling. In *ISPRS Workshop on Visualization and Animation of Reality-based 3D Models, Vulpera, Switzerland*, February 2003.

[61] C. Stewart. Covariance-based registration. Technical Report RPI-CS-TR 02-8, Department of Computer Science Rensselaer Polytechnic Institute, June 2002.

[62] SYNAPS. (Symbolic Numeric ApplicationS).
http://www−sop.inria.fr/galaad/software/synaps/.

[63] Szymon Rusinkiewicz and Marc Levoy. Efficient Variants of the ICP Algorithm. In *3rd International Conference on 3D Digital Imaging and Modeling (3DIM 2001), 28 May - 1 June, Quebec City, Canada*, 2001.

[64] Jean-Philippe Tarel and David B. Cooper. The Complex Representation of Algebraic Curves and Its Simple Exploitation for Pose Estimation and Invariant Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):663–674, 2000.

[65] G. Taubin. Estimation of Planar Curves, Surfaces, and Nonplanar Space Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(11):1115–1138, 1991.

[66] G. Taubin and D.B. Cooper. *Symbolic and Numerical Computation for Artificial Intelligence*, chapter 6: 2D and 3D Object Recognition and Positioning with Algebraic Invariants and Covariants. Computational Mathematics and Applications. Academic Press, 1992.

[67] Greg Turk and James F. O'Brien. Variational implicit surfaces. Technical Report GIT-GVU-99-15, Graphics, Visualization, and Useability Center, Georgia Institute of Technology, 1999. 9 pages.

[68] Chionh Eng Wee and Ronald N. Goldman. Elimination and Resultants - Part 1: Elimination and Bivariate Resultants. *IEEE Computer Graphics and Applications*, 15(1):69–77, January 1995.

[69] Chionh Eng Wee and Ronald N. Goldman. Elimination and Resultants - Part 2: Multivariate Resultants. *IEEE Computer Graphics and Applications*, 15(2):60–69, March 1995.

[70] Hulya Yalcin, Mustafa Unel, and William Wolovich. Implicitization of parametric curves by matrix annihilation. *International Journal of Computer Vision*, 54(1-3):105–115, 2003.

[71] Gary Yngve and Greg Turk. Creating smooth implicit surfaces from polygonal meshes. Technical Report GIT-GVU-99-42, Graphics, Visualization, and Useability Center, Georgia Institute of Technology, 1999. 9 pages.

[72] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. In *International Journal of Computer Vision*, volume 13(2), pages 119–152, 1994. Also Research Report No. 1658, INRIA Sophia-Antipolis, 1992.

[73] H. K. Zhao, S. Osher, and R. Fedkiw. Fast Surface Reconstruction Using the Level Set Method. In *Proceedings of IEEE Workshop on Variational and Level Set Methods in Computer Vision (VLSM 2001), Vancouver, CANADA*, July 2001.