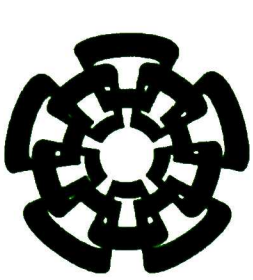
xx(116564.1)



CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Métodos Geométricos para Visión Robótica

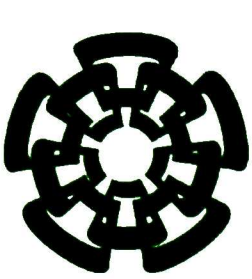
Tesis que presenta:
Leo Hendrik Reyes Lozano

para obtener el grado de:
Doctor en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Director de Tesis
Dr. Eduardo José Bayro Corrochano

Guadalajara, Jal., Julio del 2004.



CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Geometric Methods for Robot Vision

Thesis submitted by:
Leo Hendrik Reyes Lozano

for the degree of:
Doctor in Sciences

in the specialty of:
Electrical Engineering

Thesis Advisor
Dr. Eduardo José Bayro Corrochano

Guadalajara, Jal., July 2004.

Métodos Geométricos para Visión Robótica

Tesis de Doctorado en Ciencias
Ingeniería Eléctrica

Por:
Leo Hendrik Reyes Lozano
Ingeniero en Computación
Universidad de Guadalajara 1995-1999

Maestro en Ciencias
Ingeniería Eléctrica
CINVESTAV Unidad Guadalajara 1999-2001

Becario del CONACyT, expediente No. **143765**

Director de Tesis
Dr. Eduardo José Bayro Corrochano

CINVESTAV del IPN Unidad Guadalajara, Julio del 2004.

Geometric Methods for Robot Vision

Tesis of Doctor in Sciences
Electric Engineering

By:
Leo Hendrik Reyes Lozano
Computer Engineer
University of Guadalajara 1995-1999

Master of Sciences in
Electric Engineering
CINVESTAV Unidad Guadalajara 1999-2001

CONACyT Scholarship Recipient, file No. **143765**

Thesis Advisor
Dr. Eduardo José Bayro Corrochano

CINVESTAV del IPN Unidad Guadalajara, July 2004.

Resumen

En este documento, se toma el tema de reconstrucción de mapas para navegación robótica mediante cámaras. Dos casos principales deben distinguirse: cámaras calibradas y no calibradas. En esta tesis se presentan algoritmos que pueden ser usados en ambos casos.

Para el caso calibrado, se asume que una reconstrucción métrica del entorno del robot puede ser producida fácilmente. Por lo tanto, nuestra principal preocupación en este tema es la calibración externa de las cámaras con respecto al robot. Este problema es más comúnmente conocido como el problema de Calibración Mano-Ojo (Hand-Eye Calibration). Sin embargo, las soluciones actuales asumen que la cámara esta fija con respecto al marco de referencia. Estas soluciones no son prácticas para el caso cuando la cámara está montada en una unidad Pan-Tilt y su relación con respecto al marco de referencia varía constantemente. En este documento se propone un método, llamado Calibración Cuerpo-Ojo (Body-Eye Calibration), para resolver este problema sin la necesidad de re-calibrar constantemente. Este método también puede ser empleado para calibrar otros sensores.

En el caso no calibrado, se le ha dado considerable atención a la reconstrucción de movimiento y estructura a partir de una serie de vistas. Sin embargo, todo este trabajo se ha enfocado principalmente en la reconstrucción de puntos. Aquí se presenta un método que produce reconstrucciones simultáneas de varios tipos de primitivas (puntos, líneas, cuádricas, cónicas planas y cuádricas degeneradas) y cámaras por medio del *Bundle Adjustment*, usando un método de inicialización simple pero efectivo. De forma adicional, se presenta una restricción extra que debe ser incluida dentro de las iteraciones del Bundle Adjustment para mantener la topología de las cuádricas que se están reconstruyendo. De igual forma, se presenta un método original para hacer la corrección epipolar de las siluetas de cónicas en n vistas.

Finalmente, en la última parte de esta tesis, se ataca el problema de encontrar correspondencias de puntos en 2D y 3D, cuando los puntos han sufrido una transformación rígida. Usando el Álgebra Geométrica, hemos podido cambiar el enfoque del problema a encontrar un par de líneas, y un plano, respectivamente. Al usar la Votación Tensorial para encontrar estas entidades, se han podido detectar aún en la presencia de grandes cantidades de *outliers* (para el caso 2D, las líneas se pueden detectar aún cuando se tiene 1000% de outliers). Este método también ha demostrado ser útil para detectar movimiento no-rígido y elástico en 2D y 3D.

II

Abstract

In this document we address the problem of map reconstruction using cameras for robotic navigation. Two main cases must be distinguished: calibrated and uncalibrated cameras. We present algorithms that can be used in either case.

For the calibrated case, we assume that a metric reconstruction of the surroundings of the robot can be produced easily. Therefore, our main concern in this subject is the external calibration of the camera with respect to the robot. This problem is more commonly known as the Hand-Eye Calibration problem. However, current approaches assume that the camera is fixed with respect to the reference frame. These solutions are impractical for the case when the camera is mounted in a Pan-Tilt Unit and its relationship with the reference frame varies constantly. We have proposed a method called Body-Eye Calibration to cope with this problem without constant re-calibration. This method can also be employed to calibrate other sensors.

In the uncalibrated case, considerable attention has been given to the reconstruction of motion and structure from a series of views. However, all this work has focused mainly on the reconstruction of points. We present an approach that simultaneously performs reconstruction of various types of primitives (points, lines, quadrics, plane conics and degenerate quadrics) and cameras by means of Bundle Adjustment using a simple but effective initialization method. Additionally, we provide an extra constraint that must be included in the Bundle Adjustment iterations in order to maintain the topology of the quadrics being reconstructed. We also present an original method to do epipolar correction of conic outlines in n views.

Finally, in the last part of this thesis we tackle the problem of finding point correspondences in in 2D and 3D, when the points have undergone a rigid transformation. By using Geometric Algebra, we have been able to re-cast this problem as a problem of finding a pair of lines, and a plane, respectively. By using Tensor Voting to find these entities, we have been able to detect them even in the presence of large amounts of outliers (as much as 1000% outliers in the 2D case). This approach has also proven to be useful to detect non-rigid and elastic motion in 2D and 3D.

Agradecimientos

Como en todo trabajo de investigación, esta tesis no habría sido posible sin la ayuda y apoyo de numerosas personas. Esta lista es un intento fallido de agradecer a todos aquellos que hicieron que esta pesadilla fuera más tolerable. Sólo puedo esperar que no haya olvidado demasiados nombres. Antes de continuar, necesito hacer una aclaración. Al compilar esta lista me he dado algo de licencia para usar el lenguaje de forma coloquial para preservar la intención y significado original de ciertas frases. Espero que el lector sea comprensivo y perdone este pequeño detalle.

Primero, quiero agradecer a mi familia por su apoyo y paciencia. Este agradecimiento va en particular para mis padres Carlos y María de Lourdes y mi hermana Cinthia; pero también para mi abuelita Victoria Reyes. Mis disculpas van para ella también, lo único que puedo hacer ahora es dedicarle esta tesis a ella.

El Prof. Eduardo Bayro también debe ser agradecido por su paciencia infinita y apoyo. El trabajo presente no habría sido posible si no fuera por su dedicación al Laboratorio GEOVIS. Es gracias a él que contamos con un robot completamente equipado para nuestros trabajos de investigación.

Un agradecimiento también para los numerosos amigos que he hecho en la escuela: Sandino Flores (Hay quienes niegan su existencia, pues ya ha entrado en el campo de la leyenda. ¡Pero es real!), Ángel Ávalos (La Bondad Infinita en persona), Rubén Morquecho (¿Quien más se *atrevería* a escribir un determinante de 7×7 ¡*Sin iteraciones*!), Jaime Ortegón (Gracias por inventar el sistema de culpas), Juan Francisco Alvarado (Fundador del Burdismo), Germán Pérez (Al Universo nunca se le acabarán los apodos), José Gazga (Descubrió que las Fresas son niñas pero que los Limones son niños), Carlos López (También conocido como el "Lunamielero"), Julio Zamora (Fuente ilimitada de noticias inútiles), Miguel Bernal (Gurú y Vaca Sagrada de Linux. Sin embargo yo puedo mugir mejor que el), Miguel de la Torre (Amante de la música. Esperemos que algún día aprenda a tocar esa condenada trompeta), Jorge Rivera (El protegido de Dios), Nancy Arana (La reina del laboratorio, y eso incluye a "La Culpa de Nancy" también), J Refugio Vallejo (Así es, su nombre es simplemente "J"), Luis Eduardo Falcón (¡Mateyogui!), David González (El Amable), Héctor Orozco, Rubén Machucho y "La Tía de Nancy". Gracias por todo su apoyo, esos fueron buenos tiempos.

También me gustaría agradecer a todo el personal administrativo del CINVESTAV por su ayuda. En particular, quiero agradecer a Lupita Ciprián, Aracely Calzado, Aurora Ramírez, Francisco de la Torre, Antonio Rodríguez y José Pintor Jaramillo. Gracias por toda su ayuda y paciencia.

También hay muchos amigos fuera de la escuela a los cuales me gustaría agradecer: Vanessa Taylor, gracias por tu gran amistad y apoyo en todos estos años. Jorge Gutiérrez (El Trompe. Gurú de computadoras), Rodolfo de la Torre (¡Niggah!), Salvador Magaña (Mi hermano gemelo perdido), Jorge Arias ("El Buen George"), Alfonso Bizarro (Prueba viviente de que las momias existen), y Mónica Vázquez (Amante de los cachorros). Finalmente, pero no menos importante, se encuentra Paulina Machuca. Simplemente no existen palabras en el Multiverso para expresar mi gratitud hacia ella.

Gracias muy especiales para la familia Careaga de Los Angeles. Este trabajo no habría podido terminarse si no me hubieran recibido (¡Y soportado!) tan amablemente en su casa. Gracias muy especiales para el Señor Gerardo Careaga y la Señora Angelina Velasco. También para Gerardo Jr. y Loretta, Joshua, Verónica y sus hijos, Angel y Destiny; y para Rosa y Erika. Gracias por soportarme y apoyarme. Les estoy eternamente agradecido.

Durante mi visita de investigación doctoral, tuve el placer de trabajar bajo la guía del Prof. Gerard Medioni del Instituto para Robótica y Sistemas de Inteligencia en la Universidad del Sur de California. La última parte de esta tesis no habría sido posible sin sus profundas discusiones y sugerencias. La ayuda de Philippos Mordohai también resultó ser invaluable.

Finalmente deseo agradecer a CONACYT por su apoyo monetario. Yo tuve el privilegio de ser el becario CONACYT No. 143765.

Acknowledgments

As with most research, the present work would not have been possible without the help and support of numerous people. This list is but a failed attempt to thank all those who have made this nightmare a little more tolerable. I can only hope I have not forgotten too many names. Before continuing, a brief disclaimer is in order, though. While compiling this list I have given myself some license to misuse the language so as to preserve the original intent of certain phrases. May the reader forgive this small detail.

First and foremost, I must thank my family for their support and patience. This goes to my parents Carlos and María de Lourdes and my sister Cinthia; but also to my Grandmother, Victoria Reyes. My apologies for her too, the only thing I can do now is to dedicate this thesis to her.

Prof. Eduardo Bayro must be thanked too for his infinite patience and support. The present work would not have been possible if it were not for his dedication to the GEOVIS Laboratory. Because of him we have a fully-equipped robot to do our research work in this laboratory.

I must thank the numerous friends I made at school too: Sandino Flores (There are some who deny his existence, for he has now entered the realm of legend, but he's real folks! To those naysayers I say: marshmallows on you! :P) Ángel Ávalos (Infinite Goodness), Rubén Morquecho (Who else would even *dare* to type a 7×7 determinant *without iterations*!), Jaime Ortegón (Thanks for coming up with the blame-system. Now we know who is to blame every day of the week!), Juan Francisco Alvarado (Founder of the Burdism), Germán Pérez (The Universe will never run out of nicknames), José Gazga (Discovered that Strawberries were girls but Lemons were boys), Carlos López (A.K.A. "The Honeymooner" Wee!), Julio Zamora (Unlimited source of useless news), Miguel Bernal (Linux guru, master and sacred cow. However, I can moo better than him), Miguel de la Torre (Music lover. Hopefully he'll learn to play that damn trumpet some day :P), Jorge Rivera (God's protégée), Nancy Arana (The Lab's Queen, and that includes "Nancy's fault" too), J Refugio Vallejo (That's right, ladies and gentlemen, his name is simply "J"), Luis Eduardo Falcón (Mateyogui!), David González (Das habe Ich nicht verstanden), Héctor Orozco Rubén Machucho and "La Tía de Nancy" Thank you all for your support, those were fun times.

I would also like to thank the administrative staff of the CINVESTAV for their support. In particular, I would like to thank Lupita Ciprián, Aracely Calzado, Aurora Ramírez, Francisco de la Torre, Antonio Rodríguez and José Pintor Jaramillo. Thank you for all your help and patience.

There are also many friends outside school I would like to thank: Vanessa Taylor, thanks for your long-lasting friendship and support. Jorge Gutiérrez (Trompe! Computer guru), Rodolfo de la Torre (Niggah!), Salvador Magaña (My lost twin brother), Jorge Arias (Four disks fit in my box, five if I force them. Sometimes I get read errors, but I assume it's normal), Alfonso Bizarro (Living proof that mummies do exist!) and Mónica Vázquez (Puppy lover). Last, but not least, very special thanks to Paulina Machuca. There are no words in the Multiverse to express my gratitude towards her.

Very special thanks to the Careaga family from Los Angeles. This work would not have been completed if they had not received me so kindly in their home. Special thanks to Gerardo Careaga and Angelina Velasco, Gerardo Jr. and Loretta, Joshua, Verónica and her children, Angel and Destiny; Rosa and Erika. Thanks for putting up with me. I am forever indebted with you.

During my Ph.D. research visit, I had the pleasure to work under the guidance of Prof. Gerard Medioni of the Institute for Robotics and Intelligence Systems, in the University of Southern California. The last part of this thesis would not have been possible without his insightful discussions and guidance. The help of Philippos Mordohai proved to be invaluable too.

My alter-ego Skraag would also like to thank my on-line friends and clanmates from [DeX], Destruction Elite of Xian's/Xtreme and SiN, Strength in Numbers. Special thanks to Xian for starting the great community that is now ND80.net. Very special thanks to Dennis Hughes (ND80) for creating the greatest on-line gaming community ever. A very big MOO! and Thank you to my clanmates and friends: Merrill Bowers "DarthSerious" (The best tactic, when playing as axis, is to rush the North Alley. Then, the guy who leads the group must stop just short of the exit and refuse to move. The guys following him should bunch together and start pushing back and forth yelling "Clear the path!". Once everyone is dead, rinse and repeat), Amy Wightman "Ivy" (If you /kill yourself, she can revive you faster than you can tap out!), Chris Wightman "X51Luv/Terror/OrgazmoMecha" (Flame-master. You still scare meh!), Rodney Scharich "Yendor" (Someone needs to increase his appreciation of the beauty of silence! :P), Todd Corley "Doc NH". (Next door friendly neighborhood reactor builder!), Joe Minewiser "Stormcrow" (He went down faster than a $2...), James Isaac Lambert "Softbatch" (All your cookie are belong to me!), Doug Shapero "Stoner/Pvt. Joker" (For the n-th time, he does not do drugs), Andrew Walsh "Pha|Thrax" (Mr. Venom), Ryan Park "Deadmeat" (Moos well, but I can still moo better!), Tony "Mafioso" (Panzer master), Ripley (And son!), Dots A.K.A. .+. (Destruction master), ROR (BS!), Valk (Will he ever learn how to spell?), V!P3R (Old leader), BIG_JIM_SLADE ("BIG JIM DOWN!!!"), Tyrone "Sloth" (First human aimbot), Mike Bunka "Lakron" (Second human aimbot), DeathWolf(CHINA) ("OH NO! I DEAD. WE LOSED"), Luigi Sedda "NickFury(ITA)" (Nade-war master), Bill Allison ("Die Bill Die!"), DarkPenguin (His knife still scares me!), Superdome (Call in artillery!), The Lucky Family (Dynamite Planted!), Kaiser Sose (He's the one who actually got me in DeX. Thanks!), SupremeCTN (Great Lieutenant), Wolverine (The Universe will run out of numbers trying to count his points), Kringle (Special-Edition), Sterno (White is a color too), Captain Obvious (He constantly does honor to his name), BumpyKnuckles/Shockshockey/Celtic (Make up your mind already!), Goldfinger (Barry White is an amateur compared to him), Wazdakka (Mortar/Panzer master), Adamski (Great soldier), Bubba (Herald of the Apocalypse), ProAssKicker (Kicking ass is his profession), APissedGuy (He isn't always pissed), Brookss (Jo Lo Lo. Don't ask), Stoney (Medic master), Dran ("It's your Barbie-doll undies I'm after". Again, don't ask!), Karkin Soulstab (Medic master), BeerMan (Self-explanatory), Jack (Held radar for one hour in UA *all by himself*), Budman, Ryegar, Frau Blugher (Tram addict), Quanah Blaine "Parker" (Hugs! Do not confuse with Roger MagicBus), Friendly Fire (Or is it paul?), Pulsar (Do *you* know what is a pulsar?), Angry Inch (He whined about his title), Major Lee Slow (He *earned* his name), Demoman (Not a cheater, a *sniper-onanist*), Tigran (I was one of his sycophants); and Bloated Dead Cow and friends from MAC (These are the guys who taught me how to MOO!), Marion, Cow_God, Devil_Cow, Pushups, Dano, Mr. Cranky and ProAssLicker, keep mooing along guys!

Thanks also to all the administrative staff at ND80's for helping maintain the community. Thanks to SiliconSlick (System lord), Marqaha (Ze Coffee Connoisseur/The abusive admin), Dogmatix (Believe me, you don't want to know the real causes of WWII), Ranger, Icculus (Thanks for the picture!), Miserichick, +The Medic+, CodFish ("If they build it, I can bork it"), DarKon (Third human aimbot), dbldn11, Dingo (Great artist), DiW/HoNoR, D_N_G (Chronic cheater), Gretz, Guilty, HEADSHOT, Ifurita (Quake 3 engine guru), MaynardsDick (Disappointed by Windows' inbred text-to-speech system), PreciousDeath, Rhys (Sneaky bastard!) and Sphinx (Mucho caliente! Muy bien!).

I owe many hours of fun and joy to these guys (thanks for helping with the translation too!). Also, thanks to Delta9ine for dragging me into Xian's forums. I would have never met them if

it weren't for him. Finally, thanks to the HoG, CRR/TFO, MoD, DWI, E/TS, XDS clans and the Mad American Cows. GG's guys.

I also wish to thank CONACYT for their financial support. I was awarded CONACYT's Ph.D. scholarship number 143765.

List of Formulae

Projective Geometry	
Symbol	Meaning
$\mathbf{X}$	Homogeneous 3D point.
$\mathbf{x}$	Homogeneous 2D point.
$\mathbf{a}$	A vector.
A, a, k, α	Scalars.
λ	A scale factor.
$\mathtt{P}$	The projection matrix.
$\mathbf{C}$	The center of projection.
$\mathtt{H}$	A general homography.
$\mathtt{L}$	A line in Plücker Matrix form.
$\mathcal{L}$	A line in Plücker 6-vector form.
$\mathcal{P}$	The projection matrix for lines in Plücker 6-vecto form
$\mathbf{l}$	A homogeneous 2D line.
$\mathtt{C}$	A conic.
$\mathtt{Q}$	A quadric.
$\mathbf{c}$	A conic in 6-vector form.
$\mathbf{q}$	A quadric in 10-vector form.
$\mathtt{F}$	The Fundamental Matrix.
$\mathcal{T}$	The Trifocal Tensor.
$\mathcal{Q}$	The Quadrifocal Tensor.
$\mathtt{I}$	The Identity Matrix.
$\mathbf{e}$	The epipole.
$\mathtt{X}^*$	The dual of $\mathtt{X}$.
$\text{adj}(\mathtt{X})$	The adjoint of $\mathtt{X}$.
$\text{diag}(x_1, ..., x_n)$	A matrix where the elements $x_1, ..., x_n$ appear in the main diagonal.
$\text{null}(\mathtt{X})$	The null space of $\mathtt{X}$.
$\det(\mathtt{X})$	The determinant of $\mathtt{X}$.
$\mathtt{X}^\mathsf{T}$	The transpose of $\mathtt{X}$.
$\mathbf{X}\ \mathbf{Y}$	The *inner product* of $\mathbf{X}$ and $\mathbf{Y}$.
$\mathtt{X} \wedge \mathtt{Y}$	The *meet* of $\mathtt{X}$ and $\mathtt{Y}$.
$\mathbf{X} \vee \mathbf{Y}$	The *join* of $\mathbf{X}$ and $\mathbf{Y}$.
$[\mathbf{x}]_\times$	A skew-symmetric matrix such that $[\mathbf{x}]_\times \mathbf{y} = \mathbf{x} \times \mathbf{y}$.
$\|\|\mathbf{x}\|\|$	The magnitude of $\mathbf{x}$.

Geometric Algebra	
Symbol	Meaning
e_i	The i-th basis of a Geometric Algebra.
e_+	The $n+1$ basis of an nD Conformal Algebra.
e_-	The $n+2$ basis of an nD Conformal Algebra.
e_0	The null vector representing the origin in Conformal Algebra.
e_∞	The null vector representing the point at infinity in Conformal Algebra.
e	The null vector representing the origin in the Affine Plane.
$\bar{e}$	The null vector representing the point at infinity in the Affine Plane.
I	The Pseudoscalar.
E	The Pseudoscalar representing the Minkowski plane.
R	A Rotor.
M	A Motor.
T	A Translator.
$\langle \mathbf{A} \rangle_r$	The component r-vector of $\mathbf{A}$.
$\mathbf{X} \wedge \mathbf{Y}$	The *wedge product* of $\mathbf{X}$ and $\mathbf{Y}$.
$\mathbf{X} \cup \mathbf{Y}$	The *join* of $\mathbf{X}$ and $\mathbf{Y}$.
$\mathbf{X} \cap \mathbf{Y}$	The *meet* of $\mathbf{X}$ and $\mathbf{Y}$
$\hat{\mathbf{A}}$	The Grade Involution of $\mathbf{A}$.
$\widetilde{\mathbf{A}}$	The Reversion of $\mathbf{A}$.
$\bar{\mathbf{A}}$	The Clifford Conjugate of $\mathbf{A}$.
$\mathbf{x}^*$	The dual of $\mathbf{x}$.
$\mathbf{x}_e$	A generic Euclidean point.
x_c	A generic Conformal point.
x_a	A generic point in the Affine Plane.
s	A generic sphere.
z	A generic circle.
l	A generic line.
π	A generic plane.
P_E	The projection operator in Conformal Geometry.
$P_E^\perp$	The rejection operator in Conformal Geometry.

Contents

Chapter 1

Introduction

1.1 Motivation

A recurring problem in visually-guided robotic navigation is that of the construction of a map of the surrounding environment. The final goal in this approach would be the fully-automatic building of the map. This implies that the robot should be able to calibrate its various coordinate systems with respect to each other (i.e. calibrate all its sensors and effectors), and hopefully build an internal representation of the world without any external assistance. In this document we will focus both on the calibrated and uncalibrated reconstruction of maps.

In the calibrated case, we assume a stereo system where the cameras' intrinsic parameters have been computed beforehand. Binocular systems of this type are usually mounted on pan-tilt units to allow a greater viewing range. However, the exact positioning between the stereo system and the pan-tilt unit are seldom known. When one realizes that the cameras' coordinate system is usually placed at the left camera center, which in turn is a point placed *somewhere inside* the case of the stereo system, it becomes evident that a simple external measurement is impractical and inaccurate. Therefore, in the first part of this document, we address this problem, commonly known as *Hand-Eye Calibration*, and propose a new method that enables the fusion of measurements from different sensors.

In the uncalibrated case, we are presented with n uncalibrated views of a scene. The problem is to find a projective reconstruction that produces the same images. In this subject, extensive work has been done for the case of 2 to 4 views, but relatively few algorithms exist for the general n-view case. Furthermore, almost all the work produced so far is exclusively concerned with the reconstruction of cameras and points. Therefore, we have addressed the more general case of reconstructing cameras, points, lines, quadrics, plane conics and degenerate quadrics using n uncalibrated views. In the process, we also produced a novel algorithm for conic outline correction that enables the preservation of topology in quadrics.

Finally, we also address the problem of finding the correspondences of two point sets in 2D or 3D assuming that a rigid transformation has taken place without further information. These correspondences enable us to compute the motion between these sets to perform registration. The problem of registering data sets is common in the computer vision literature. Applications range from the alignment of range measurements for the automatic re-construction of maps for robotic navigation [31], [42]; registration of CT and MR images for medical purposes [24], [51], [20], [37], [22]; computer graphics and CAD modeling [58], [19] and recognition of objects [11], [59].

1.2 State of the Art

1.2.1 Calibrated case

As stated previously, we will focus on two different aspects of map reconstruction. In the case of the Hand-Eye Calibration problem, one of the first closed-form solutions was given Tsai and Lenz in [57] where a matrix system of the form $\mathtt{AX} = \mathtt{XB}$ was solved. More recently, this basic approach was modified by Dornaika and Horaud in [18] to account for the Robot-World calibration too. Andreff further improved the basic scheme in [1] to enable on-line Hand-Eye Calibration using a Kalman Filter. In a similar way, Angeles et al used recursive least squares estimation in [2] to enable on-line calibration. Wei et al proposed a nonlinear approach in [23] that not only solves the hand-eye calibration problem, but also accounts for camera calibration (with lens distortion) and path planning for the calibration itself. In [8] Bayro et al approach the problem of Hand-Eye Calibration in the context of The Motor Algebra. In that paper, a different formulation is presented based on the motion of lines. Finally, Malm and Heyden propose a method based on normal derivatives of the image flow field in [45] to achieve Hand-Eye Calibration.

1.2.2 Uncalibrated case

For the case of multiple view uncalibrated reconstruction. Hartley and Zisserman [27] have proposed the use of n-view tensors which can be computed directly from image features. This method has the advantage that no constraints are imposed on the scene (in general), and the correspondences of points and lines can be mixed to add robustness. However, the computation of the tensors is a complex task and can only be applied to four views at most. The points and lines used to compute the tensors must also be visible in all the views.

Carlsson and Weinshall [12] have used an approach based on the duality principle between points and camera centers. This approach has the advantage that it can be used with n views and the equations are rather straightforward (compared to the tensors). However, line correspondences cannot be used and the points are required to be visible in all views. Rother et al further simplified this scheme [49], by relying on the a priori knowledge of planarity in the scene. The disadvantage of this method is that it imposes a constraint on the scene, namely that four points visible in all views lie on a plane. Planarity has also been used by Kaucic et al to simplify the computation of n-view tensors [36]. This method requires features to be visible in all views and imposes a constraint on the scene.

Projective factorization alone has also been used by Sturm and Triggs in [55]; however, this approach is only valid if the projective depths are known or can be reasonably guessed, and all the features must be present in all the views. Another approach proposed by Bayro and Banarer [5] produces a projective reconstruction using the n-view tensors and the invariants theory. This method has the same advantages and disadvantages of the tensor computation methods. In [33] Kahl and Heyden propose a method to compute structure and motion from points, lines and conics. This method has the advantage of simultaneously estimating several different kinds of primitives. However, quadrics are not reconstructed and the epipolar constraint is not taken into account. On the other hand, this method was developed for the case of affine cameras using factorization. Nevertheless, a method for handling missing data is introduced.

Bundle Adjustment (BA) [56] can be used to simultaneously estimate multiple cameras and 3D primitives. Classical Bundle Adjustment [27] can cope with missing data, but current work

specializes on the reconstruction of points and cameras. Another disadvantage is that the parameterization is not minimal leading to a large minimization problem. This basic approach has been improved by McLauchlan [46] to account for gauge invariance. Bartoli and Sturm [4] developed a minimal parameterization scheme and Malis and Bartoli [44] further improved this to produce an Euclidean reconstruction. All this work is based in the estimation of cameras and 3D points. Berthilsson et al [9] used BA for the estimation of general space curves. Shan et al applied BA [50] to model human faces from a prototype neutral face model and a series of images.

In [3] Åström proposed a method based on Bundle Adjustment for the computation of structure and motion from points, lines and conics. This method can deal with projective cameras, in contrast to [33]. However, quadrics are not reconstructed and the problem of epipolar consistency is also dismissed.

On the reconstruction of quadrics, Cross and Zisserman [16] work with previously known camera matrices to develop an algorithm to do outline correction of the conics and perform the reconstruction from two views. No clear details are given on how to extend the algorithm for 3 or more views, and the problem of noise in the process of the estimation of the cameras is not addressed. In [15], Cross describes a similar method for outline correction and reconstruction of quadrics from n views. However, the details are only given for 2 and 3 views and, as will be proven in this report, the algorithm for 4 or more views cannot be directly applied as stated by Cross. Again, camera matrices are assumed to be known beforehand and the problem of noise in the camera matrices is disregarded.

1.2.3 Point registration

The classical solution to this problem was given by Besl and McKay with the Iterative Closest Points algorithm (ICP) [10]. Several improvements have been made to the basic scheme starting with a more efficient way to compute the distance and thresholding of the points [61] by the use of K-D trees and statistical analysis, respectively; the use of extra cues to improve the matching of points like texture [32], [25]; the implementation of a *soft-assign* scheme to allow for varying degrees of certainty in the matches [59]; and the use of more robust iteration techniques like Levenberg-Marquardt [13], [21].

The advantages of the ICP are that it is simple, fast, and hence can be used for real-time applications. However, in general, all the ICP-based approaches refine an initial guess of the registration by iteratively updating the parameters of the transformation. If the initialization is poor, the method will not converge to the desired solution. Because of this, ICP is generally used when the difference between the model and the data is small, constraining the range of possible applications to those where this constraint can be met. One way to overcome this problem is by initializing the transformation with other methods like the Procrustes [43] algorithm. Another drawback of this method is that it requires a model and a data set, so that the data set is a subset of the model. That is, at least one of the data sets *must not contain outliers* (the model) and must be relatively noise-free. Also, the ICP in general is not robust against the presence of large amounts of outliers, and is limited to registering points via a rigid transformation. Another disadvantage is that the resulting mapping may not be one-to-one.

Another method widely used to solve the registration problem in 2D is Chamfer matching [11]. In this paper, Borgefors improves the original chamfer matching algorithm by using a resolution pyramid in a hierarchical matching procedure. Chamfer matching can also be implemented in a parallel fashion [60] to improve its performance. Again, the limitation of this method is that the

motion between the images must be small or the process might converge to a local minimum. The images must also be free of outliers.

Methods that do not require initialization and are robust to outliers are generally based on voting schemes, like the Hough Transform. In this respect, Hu [30] solved the registration problem in 3D by employing a Hough-based voting scheme. The advantages of this method are that it does not require an initialization and works equally well independently of the "size" of the transformation. However, the translation must lie within the working range of the voting space. As will be proven in this paper, limiting and quantizing the range of possible transformations has an effect on the accuracy and ranges that can be reliably detected by this type of algorithms. Another disadvantage is that this method is model-based, meaning that the model must be relatively free of noise and must not contain outliers.

Following the same voting scheme, Kalviainen et al use the Randomized Hough Transform [34] to obtain the 2D motion between two point sets. The problem with this approach is, again, that the voting space introduces a quantization that degenerates the accuracy and possible ranges of the solution.

Finally, in the voting-based algorithms. Kang et al [35] demonstrated how Tensor Voting can be used to detect multiple affine motions in 2D. It should be noted that all the algorithms mentioned so far only work when a single global motion is present in the data. Kang's algorithm is largely independent of initialization (it allows for multiple candidate matches and has a robust way of rejecting the false matches), and is robust to outliers. This paper is inspired by the work of Kang, therefore, we will discuss the similarities and differences of both algorithms in a subsequent section.

There are also a few algorithms that handle non-rigid transformations like [14] which uses *soft-assign* and an iterative minimization method (deterministic annealing) to produce a non-rigid registration of points. The advantage of this method is that it guarantees a one-to-one mapping when the algorithm finishes, but the disadvantage is that it requires a good initialization and only withstands a small amount of outliers in the input data. Another ICP-based algorithm that performs non-rigid (affine and spline) registration is given in [20], the same disadvantages of initialization and outliers apply. Finally, in [37] B-splines are used to perform the registration and gradient descent technique is used to find the solution. This approach is dependent on a good initialization. Generally, these methods find a greater application for medical imaging.

Another type of problem is the registration of multiple data sets. Solutions to this problem have been reviewed in [17]. One solution consists mainly of expressing the problem as an optimization problem where the parameters are all the transformations needed to register the multiple data sets [22] and then a standard minimization algorithm is used. Another approach relies on modeling the problem as a dynamic spring system to register the multiple sets together [19], [54]. These algorithms suffer from the same problems that have been already mentioned: they require an initialization and, depending on its quality, may or may not converge to the desired solution, and they are not robust against the presence of outliers. However, we will restrict ourselves to the two-frame case only.

1.3 Chosen Problems

As we can see, the Hand-Eye Calibration problem is mainly used when the camera is rigidly attached to the end effector of a robotic arm. These approaches can also be used when the camera

is mounted on a pan-tilt unit to find the transformation linking the camera to the robot's main framework; however they become impractical if the pan-tilt unit is moved constantly, requiring re-calibration for each motion. Therefore, we concentrated on the problem of Hand-Eye Calibration when the relationship between the camera and the robot is not fixed. Indeed, one of the main goals of the work was to produce an algorithm that was capable of computing the transformation between camera and robot without re-calibration. Furthermore, we also extended this algorithm to account for other sensors and thus achieve full Body-Eye calibration too.

For the case of uncalibrated reconstruction, we noted that most of the methods published concentrate on a specific type of primitive. To the best of our knowledge, there is not an approach for projective reconstruction, that takes into account all the types of primitives at the same time while keeping the topology of quadrics. Hence, we decided to tackle the problem of simultaneous estimation of points, lines, quadrics (including plane conics and degenerate quadrics) and cameras from a series of uncalibrated views. We also dedicated special attention to the problem of conic outline correction in 4 or more views (required to preserve the topology of the reconstructed quadrics). The advantages of our method are:

- Works even when there is missing data.
- Applicable to n views.
- Uses projective cameras.
- Does not require a priori knowledge.
- Does not impose constraints on the scene.
- Simultaneous estimation of multiple primitives.
- The problem of epipolar-consistency (topology) is addressed in the presence of noisy camera matrices.

Finally, for the case of point registration. We found that in general, the ICP and Chamfer matching based algorithms suffer from the following disadvantages.

- A good initialization must be provided. The quality of the initial correspondences impacts the performance of the algorithms.
- The range of possible motions is restricted (they work better for small motions).
- No outliers are permitted in the model and few to none in the data.
- Require preprocessing of the data to reject outliers (if present).
- Only work when a single global rigid motion is present.
- For the gradient-descent and Levenberg-Marquardt implementations, a computation of the derivatives is needed.
- The resulting mapping is not guaranteed to be one-to-one.

For the Hough-based algorithms, the disadvantages are

- The range of the possible transformations is limited due to the size of the voting space.
- The voting space introduces a quantization of the parameters which compromises the accuracy of the solution.

In this chapter we propose a novel algorithm that provides the following advantages.

- Works in the presence of large numbers of outliers.
- No initialization required.
- No thresholding to reject incorrect pairings needed (as is the case for most ICP implementations).
- Works equally well for large and small motions.
- No quantization is introduced for the parameters of the transformation, the accuracy of the solution is therefore not compromised.
- Does not require any preprocessing of the data.
- Does not require estimation of derivatives.
- Guarantees a one-to-one correspondence.
- Multiple overlapping motions can be detected.

1.4 Main Contributions

To summarize, three main problems have been tackled. In the Hand-Eye calibration problem, an algorithm was proposed to account for the varying positions of a stereorig mounted on a Pan-Tilt Unit without constant re-calibration. This algorithm was also extended to comprise other sensors, like the laser, and thus find the correspondences between all these systems and the robot main framework. Therefore, we have called this algorithm the Body-Eye Calibration.

The second problem was the simultaneous projective reconstruction of various primitives such as points, lines, quadrics, plane conics and degenerate quadrics. We found that in order to preserve the topology of the quadrics, an extra constraint was needed during the iterations of Bundle Adjustment. Also, a novel algorithm for the epipolar correction of conic outlines in n views was proposed.

Finally, in the point correspondence and registration problem, two novel algorithms have been proposed. These algorithms are not iterative and do not require an initialization. All transformations are equally detectable regardless of their magnitude. Our algorithms perform even in the presence of large amounts of outliers (on the order of 1000%). And, finally, these algorithms can be extended to account for multiple overlapping motions and non-rigid or elastic motion.

Publications

1. E. Bayro-Corrochano y L. Reyes. Body-eye calibration using the Geometric Algebra Framework. *Journal of the Royal Society*, UK, 2004 (to appear).

2. E. Bayro-Corrochano, P. Lounesto y L. Reyes. Applications of Algebra of Incidence in Visually Guided Robotics. Chapter 32, pp. 361-372 in the book *Applications of Geometric Algebra in Computer Science and Engineering*, Eds Leo Dorst, Chris Doran, and Joan Lasenby, Birkhuser 2002, ISBN 0-8176-4267-6.

3. L. Reyes and E. Bayro. Simultaneous Uncalibrated Multiple View Reconstruction of Points, Lines, Quadrics and Cameras, *in Proceedings of the 3rd International Conference on Computer Vision, Pattern Recognition & Image Processing*, Cary, North Carolina, USA, September 2003.

4. Leo Reyes Lozano and Eduardo Bayro-Corrochano. The Projective Reconstruction of Points, Lines, Quadrics, Plane Conics and Degenerate Quadrics Using Uncalibrated Cameras, *Journal of Image and Vision Computing*, 2004, Elsevier (to appear).

5. L. Reyes and E. Bayro. Geometric Approach for Simultaneous Projective Reconstruction of Points, Lines, Planes, Quadrics, Plane Conics and Degenerate Quadrics, *in Proceedings of the 17th International Conference on Pattern Recognition*, Cambridge UK, August 2004 (in press).

Chapter 2

The Mathematical Models of Computer Vision

In this chapter we will present two different mathematical models that are commonly used to solve computer vision problems. The first section of this chapter is devoted to the most widely used method, called *Projective Geometry*. This mathematical model is based on vector calculus, matrix algebra and tensors.

The second part of this chapter presents a different approach based on Clifford's *Geometric Algebra*. In particular, we will concentrate on the *Geometric Conformal Algebra* and one of its related spaces: *The Affine Plane*.

Each model has its own set of advantages and drawbacks. Some problems are more easily solved in one framework or the other. Therefore, it is necessary to have a good understanding of both methods in order to select the best one for each particular problem.

2.1 Part I: Projective Geometry

The classical approach to computer vision consists in modeling the camera as a 3×4 projection matrix. A review on the projection of points, lines and quadrics will be given in this section. The interested reader may look in [27] for further details.

2.1.1 Projection of points

In projective geometry, all nD points are represented by (n+1)D column vectors using *homogeneous coordinates*. Thus, for instance, the general format of a 3D point $\mathbf{X}$ is

$$\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \tag{2.1}$$

$\mathbf{X}$ represents the Euclidean point $(X/W, Y/W, Z/W)$. Under the pinhole camera model, the projection of a homogeneous 3D point is written as

$$\lambda \mathbf{x} = \mathbf{P}\mathbf{X}, \tag{2.2}$$

$$= \begin{bmatrix} p_{1,1} & \cdots & p_{1,4} \\ \vdots & \ddots & \vdots \\ p_{3,1} & \cdots & p_{3,4} \end{bmatrix} \mathbf{X},$$

$$= \begin{bmatrix} \mathbf{p}_1^\mathsf{T} \\ \mathbf{p}_2^\mathsf{T} \\ \mathbf{p}_3^\mathsf{T} \end{bmatrix} \mathbf{X}.$$

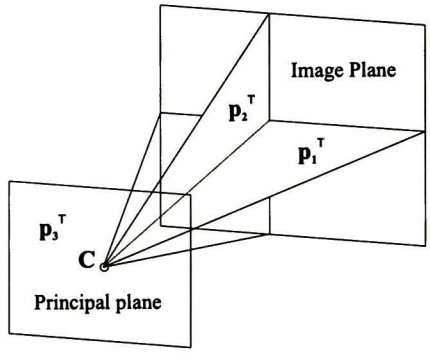


Figure 2.1: The optical planes $\mathbf{p}_i^\mathsf{T}$ in a camera with center C.

Where, again, $\mathbf{x}$ is a 3-vector that represents a 2D point in projective coordinates, λ is an arbitrary nonzero scale factor known as projective depth and P is a 3×4 matrix representing the camera. Note that the rows $\mathbf{p}_i^\mathsf{T}$ of P can be seen as three intersecting optical planes (see Fig. 2.1). For the remainder of this paper, we will assume that all points are represented in homogeneous coordinates unless otherwise stated. Scalar factors will also be omitted when not necessary.

The set of all 3D homogeneous points $\mathbf{X}$ is called *The projective space of 3D* $\mathbb{P}^3$ In this set, the null point $\mathbf{0} = [0, 0, 0, 0]^\mathsf{T}$ is excluded. Similarly, the set of all 2D homogeneous points $\mathbf{x}$ is called *The projective space of 2D* $\mathbb{P}^2$

2.1.2 Projection of lines

There are a few ways of representing 3D lines, but we chose the Plücker representation. A line in 3D can be represented either by a Plücker matrix, which is a 4×4 skew-symmetric matrix, or a Plücker vector, which is a 6×1 column vector containing the entries of the Plücker matrix. The line joining two points $\mathbf{A}$ and $\mathbf{B}$ is the Plücker matrix given by

$$\mathtt{L} = \mathbf{A}\mathbf{B}^\mathsf{T} - \mathbf{B}\mathbf{A}^\mathsf{T} \tag{2.3}$$

A Plücker line matrix can be transformed into a Plücker vector by the rule

$$\mathcal{L} = [l_{12}\ l_{13}\ l_{14}\ l_{23}\ l_{42}\ l_{34}]^\mathsf{T} \tag{2.4}$$

where l_{ij} represents the element from the i^{th} row and j^{th} column from matrix L. Additionally, a 6-vector $\mathcal{L}$ only represents a line if and only if it satisfies the *internal orthogonality constraint* whereas

$$l_{12}l_{34} + l_{13}l_{42} + l_{14}l_{23} = 0. \tag{2.5}$$

For the projection of the lines in Plücker coordinates, we need to define a different projection matrix $\mathcal{P}$ as follows

$$\mathcal{P} = \begin{bmatrix} \mathtt{P}^2 \wedge \mathtt{P}^3 \\ \mathtt{P}^3 \wedge \mathtt{P}^1 \\ \mathtt{P}^1 \wedge \mathtt{P}^2 \end{bmatrix} \tag{2.6}$$

Where $\mathtt{P}^{i\mathsf{T}}$ is the i^{th} row of the point-projection camera matrix $\mathtt{P}$ (given in Eq. 2.2), and $\mathtt{P}^i \wedge \mathtt{P}^j$ is the *meet* or intersection between the optical planes $\mathtt{P}^i$ and $\mathtt{P}^j$, implemented with matrices (we will be using the symbol $\wedge$ to denote the intersection or *meet* of geometric objects throughout this report, though the actual implementation may vary). For an extended discussion and further insight into the geometric meaning of the coefficients of the Plücker coordinates, and the *meet* and *join* operations in the Geometric Algebra framework, see [5]. The *meet* between the planes $\mathtt{P}$ and $\mathtt{Q}$ can be computed by

$$\mathtt{P} \wedge \mathtt{Q} \equiv \mathtt{P}\mathtt{Q}^{\mathsf{T}} - \mathtt{Q}\mathtt{P}^{\mathsf{T}} = \mathtt{L}^*, \tag{2.7}$$

where $\mathtt{L}^*$ is the dual Plücker line matrix. The relationship between a line and its dual is given by the rule

$$[l_{12}\ l_{13}\ l_{14}\ l_{23}\ l_{42}\ l_{34}]^{\mathsf{T}} = [l^*_{34}\ l^*_{42}\ l^*_{23}\ l^*_{14}\ l^*_{13}\ l^*_{12}]^{\mathsf{T}} \tag{2.8}$$

Using the camera matrix as defined by Eq. 2.6, the projection of the line may now be written as

$$\mathtt{l} = \mathcal{P}\mathcal{L} = \begin{bmatrix} (\mathtt{P}^2 \wedge \mathtt{P}^3)|\mathcal{L} \\ (\mathtt{P}^3 \wedge \mathtt{P}^1)|\mathcal{L} \\ (\mathtt{P}^1 \wedge \mathtt{P}^2)|\mathcal{L} \end{bmatrix} = \begin{bmatrix} \mathcal{L}^{23}|\mathcal{L} \\ \mathcal{L}^{31}|\mathcal{L} \\ \mathcal{L}^{12}|\mathcal{L} \end{bmatrix} \tag{2.9}$$

where the product $\mathcal{L}|\hat{\mathcal{L}}$ is defined as

$$\mathcal{L}|\hat{\mathcal{L}} = l_{12}\hat{l}_{34} + l_{34}\hat{l}_{12} + l_{13}\hat{l}_{42} + l_{42}\hat{l}_{13} + l_{14}\hat{l}_{23} + l_{23}\hat{l}_{14}. \tag{2.10}$$

Which is the projection of $\mathcal{L}$ into the three optical basis rays.

2.1.3 Projection of quadrics, plane conics and degenerate quadrics

The inhomogeneous equation of a quadric in $\mathbb{R}^3$

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j = 0. \tag{2.11}$$

Can be transformed into a homogeneous expression in $\mathbb{P}^3$ by substitution, yielding the following matrix equation

$$\mathbf{X}^{\mathsf{T}}\mathtt{Q}\mathbf{X} = 0, \tag{2.12}$$

where

$$\mathtt{Q} = \begin{bmatrix} a & d/2 & e/2 & g/2 \\ d/2 & b & f/2 & h/2 \\ e/2 & f/2 & c & i/2 \\ g/2 & h/2 & i/2 & j \end{bmatrix} \tag{2.13}$$

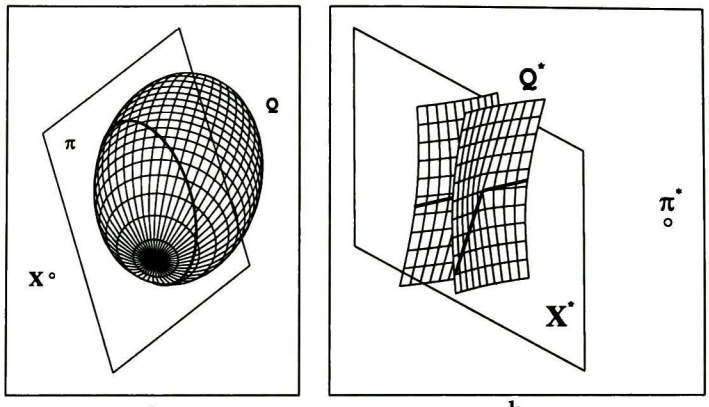


Figure 2.2: a) The polar π generated by a point $\mathbf{X}$ on the quadric $\mathtt{Q}$ and their duals b).

The projection of a quadric from $\mathbb{P}^3$ to $\mathbb{P}^2$ produces a conic; in fact, only the *outline* of the quadric is projected. The outline is generated by the intersection of the *polar* of the quadric with respect to the camera center (see Fig. 2.2). The polar of a quadric is a plane π generated by

$$\pi = \mathtt{Q}\mathbf{X}. \tag{2.14}$$

The intersection of the polar with the quadric is a set of points whose tangent planes pass through the camera center. Therefore the projection is simplified when the dual quadric is used instead. Just like the normal quadric is an equation on points, the dual quadric is an equation on planes (which are the dual of the points). The dual quadric $\mathtt{Q}^*$ is computed by

$$\mathtt{Q}^* = \mathrm{adj}(\mathtt{Q}). \tag{2.15}$$

Finally under the point projection matrix $\mathtt{P}$ (Eq. 2.2) the quadric $\mathtt{Q}$ is mapped as follows

$$\mathtt{C}^* = \mathtt{P}\mathtt{Q}^*\mathtt{P}^\mathsf{T} \tag{2.16}$$

where $\mathtt{C}^*$ is a dual (line) conic, and $\mathtt{C} = \mathrm{adj}(\mathtt{C}^*)$ (see Fig. 2.3). The equation of the conic reads

$$ax^2 + bxy + cy^2 + dx + ey + f = 0. \tag{2.17}$$

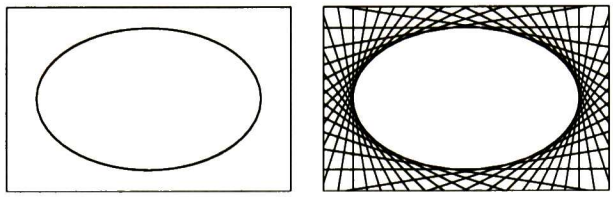
which can be represented in matrix form using homogeneous coordinates as

$$\mathbf{x}^\mathsf{T}\mathtt{C}\mathbf{x} = 0, \tag{2.18}$$

where

$$\mathtt{C} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \tag{2.19}$$

As explained in detail later, a plane conic is represented as a dual cone $\mathtt{Q}_c^*$, therefore, the plane conics are projected using Eq. 2.16 too. In contrast, degenerate quadrics cannot be projected using Eq. 2.16 since the adjoint of a rank-deficient matrix $\mathtt{Q}$ is undefined. Therefore, it is necessary



a b

Figure 2.3: A conic a) and its dual b).

to use a different projection equation that works in real-space (see [15] for the full details on the derivation of this formula)

$$\mathtt{C} = \mathtt{A}^{-\mathsf{T}} \left[\mathtt{Q}_{33}\mathbf{c}\mathbf{c}^{\mathsf{T}}\mathtt{Q}_{33} + 2\mathbf{q}\mathbf{c}^{\mathsf{T}}\mathtt{Q}_{33} + \mathbf{q}\mathbf{q}^{\mathsf{T}} - \left(\mathbf{c}^{\mathsf{T}}\mathtt{Q}_{33}\mathbf{c} + 2\mathbf{c}^{\mathsf{T}}\mathbf{q} + \mathtt{q}\right)\mathtt{Q}_{33} \right] \mathtt{A}^{-1} \tag{2.20}$$

where

$$\mathtt{Q} = \begin{bmatrix} \mathtt{Q}_{33} & \mathbf{q} \\ \mathbf{q}^{\mathsf{T}} & q \end{bmatrix} \tag{2.21}$$

is the real-space quadric we want to project $\mathtt{P} = [\mathtt{A}\ \mathbf{a}]$, is the camera matrix and $\mathbf{c} = -\mathtt{A}^{-1}\mathbf{a}$ is the optical center of $\mathtt{P}$.

2.1.4 Projection of planes

A plane π does not produce an image on a camera but rather induces a 2D homography between itself and the image plane. This homography can be computed by features extracted from the plane or by the limits of a plane segment like its edges or its corners. However, both edges and corners fall into the case of lines and points respectively which have already been considered. Furthermore, to compute this homography, at least four points (or its equivalent in lines) lying on the plane are needed. If these points are reconstructed, then the plane can be obtained by simply *joining* three of them (i.e. $\pi = \mathbf{P}_1 \vee \mathbf{P}_2 \vee \mathbf{P}_3$, see [5]) or by performing a linear regression with the whole set. Plane reconstruction is, therefore, a subproblem of point and line computation, so we will not consider the reconstruction of 3D planes in our approach.

2.2 Projective Geometry of N Views

We have just reviewed the basic projection equations for points, lines and quadrics. In order to be able to reconstruct these primitives from image measurements, however, the projective geometry of the scene must be first estimated. Depending upon the number of views available, this geometry can be expressed by the bifocal, trifocal or quadrifocal tensor. If more than four views are available, combinations of the tensors just mentioned can be used. We will now describe the algorithms we implemented to compute these tensors in the present work.

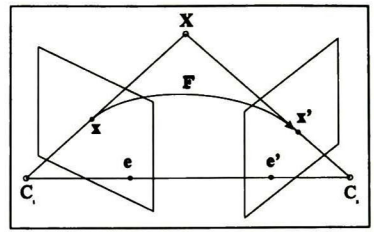


Figure 2.4: The Fundamental Matrix F.

2.2.1 The bifocal tensor

The projective geometry defined by two views can be described by the Bifocal Tensor (also called The Fundamental Matrix, see Fig. 2.4). This matrix is defined by the equation

$$\mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = 0, \tag{2.22}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are corresponding points in the first and second views respectively. The 3×3 matrix F has rank two with 7 degrees of freedom (dof), and can be computed with 8 (or more) point correspondences. If $\mathbf{x} = [x, y, 1]^{\mathsf{T}}$ and $\mathbf{x}' = [x', y', 1]^{\mathsf{T}}$, then Eq. 2.22 can be rewritten as

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0. \tag{2.23}$$

The entries of F can be rearranged in a column vector $\mathbf{f}$ such that the previous equation can be rewritten as

$$\begin{bmatrix} x'x & x'y & x' & y'x & y'y & y' & x & 1 \end{bmatrix} \mathbf{f} = 0. \tag{2.24}$$

From a set of n point correspondences, we can make a system of the form

$$\begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & 1 \end{bmatrix} \mathbf{f} = \mathtt{Z}\mathbf{f} = 0. \tag{2.25}$$

From which $\mathbf{f}$ can be solved by taking the null vector of Z. In practice, due to noise in the measurements, the best null vector is obtained by taking the Singular Value Decomposition (SVD, see [48] for details) of Z and extracting the singular vector that corresponds to the smallest singular value.

Finally, recall that F must be a rank-2 matrix. In order to enforce this constraint we must decompose $\mathtt{F} = \mathtt{U}\mathtt{D}\mathtt{V}^{\mathsf{T}}$ using the SVD decomposition, where $\mathtt{D} = \mathrm{diag}(r, s, t)$ is a diagonal matrix satisfying $r \geq s \geq t$. Then the new rank-2 matrix we need is $\mathtt{F} = \mathtt{U}\mathrm{diag}(r, s, 0)\mathtt{V}^{\mathsf{T}}$

To further improve this algorithm, the set of points can be normalized by applying homographies H and H′ such that the centroid of the set of points lies at the origin and their mean distance to it is $\sqrt{2}$ (see [27]). In that case, the resulting F is further de-normalized according to $\mathtt{F}' = \mathtt{H}'\mathtt{F}\mathtt{H}$.

Once the matrix F is known, the first pair of camera matrices can be calculated as

$$\mathtt{P} = [\mathtt{I}|\mathbf{0}] \tag{2.26}$$

$$\mathtt{P}' = [[\mathbf{e}']_\times \mathtt{F}|\mathbf{e}'], \tag{2.27}$$

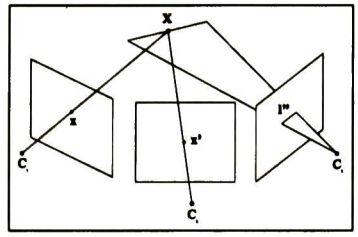


Figure 2.5: The point-point-line trilinear relation.

where $\mathbf{e}' = \text{null}(\mathbf{F}^\mathsf{T})$ and

$$[\mathbf{e}']_\times = \begin{bmatrix} 0 & -e'_3 & e'_2 \\ e'_3 & 0 & -e'_1 \\ -e'_2 & e'_1 & 0 \end{bmatrix} \tag{2.28}$$

with $\mathbf{e}' = [e'_1\ e'_2\ e'_3]^\mathsf{T}$. Note that the matrix $\mathbf{P}'$ in Eq. 2.27 has the left 3×3 sub-matrix $[\mathbf{e}']_\times \mathbf{F}$ which has rank 2. Therefore, this submatrix has a null vector $\mathbf{n}$. If the extra column of $\mathbf{e}'$ is added, we arrive at the conclusion that the null vector for $\mathbf{P}'$ has the form $[\mathbf{n}^\mathsf{T}, 0]^\mathsf{T}$ Hence, the center of $\mathbf{P}'$ is at infinity. Note however, that this does not necessarily imply that the principal plane lies at infinity too. The general formula to compute $\mathbf{P}'$ is in fact

$$\mathbf{P}' = [[\mathbf{e}']_\times \mathbf{F} + \mathbf{e}'\mathbf{v}^\mathsf{T} | \lambda \mathbf{e}'], \tag{2.29}$$

where $\mathbf{v}$ is any 3-vector. However, we have not noticed any significant difference between computing $\mathbf{P}'$ using Eq. 2.27 or Eq. 2.29, since the cameras are further refined through Bundle Adjustment and the projective reconstruction is then rectified to a metric framework through the application of a homography. This rectifying homography has the effect of mapping the center of the second camera from infinity to a real point. The details of the Bundle Adjustment stage and the metric rectification will be given in the Chapter 4.

2.2.2 The trifocal tensor

The Trifocal tensor defines the projective geometry of three views. Whereas the Fundamental matrix only comprised relations between points, the trifocal tensor $\mathcal{T}$ relates points, lines and their combinations as shown in Table 2.1 (for details about these relations see [27]). The point-point-line trilinear relation is illustrated in Fig. 2.5.

Correspondence Type	Equation
point-point-point	$x^i x'^j x''^k \epsilon_{jqu} \epsilon_{krv} \mathcal{T}_i^{qr} = 0_{uv}$
point-point-line	$x^i x'^j l''_r \epsilon_{jqu} \mathcal{T}_i^{qr} = 0_u$
point-line-line	$x^i l'_q l''_r \mathcal{T}_i^{qr} = 0$
line-line-line	$l_p l'_q l''_r \epsilon^{piw} \mathcal{T}_i^{qr} = 0^w$

Table 2.1: The trilinear relations.

In this, and the following section, 2D points $\mathbf{x}$ are represented as column vectors of the form $\mathbf{x} = [x^1, x^2, x^3]^\mathsf{T}$ and lines are represented as row vectors $\mathbf{l} = [l_1, l_2, l_3]$. Therefore, a matrix

$\mathtt{H}_{i,j}$ is represented as the tensor $\mathtt{H}^i_j$. According to the way they transform, points are said to be *contravariant* tensors and lines are *covariant* tensors.

Unlike the Fundamental matrix, the Trifocal tensor is not easy to compute. This is because the Trifocal tensor has a greater number of internal constraints (the tensor has 27 entries but only 18 dof). Thus, even though any relation shown in Table 2.1 can be rewritten in the form $\mathtt{At} = \mathtt{0}$ where $\mathtt{t}$ is a 27-vector representing the trifocal tensor, simply extracting the null vector of $\mathtt{A}$ will not produce a geometrically valid tensor satisfying all the internal constraints, in general. Therefore, a more sophisticated algorithm is needed. However, the method just mentioned serves as a starting point for other algorithms that further refine the trifocal tensor. The full normalized linear algorithm (taken from [27]) is presented in Algorithm 2.1.

1. Find transformation matrices $\mathtt{H}$, $\mathtt{H}'$, and $\mathtt{H}''$ to apply to the three images, such that the centroid of the points/lines lies at the origin and their average distance to it is $\sqrt{2}$.
2. Transform points according to $x^i \mapsto \hat{x}^i = \mathtt{H}^i_j x^j$, and lines according to $l_i \mapsto \hat{l}_i = (\mathtt{H}^{-1})^j_i l_j$. Points and lines in the second and third view transform in the same way.
3. Compute the trifocal tensor $\hat{\mathcal{T}}$ linearly in terms of the transformed points and lines using the equations of Table 2.1 by solving a set of equation of the form $\mathtt{At} = \mathtt{0}$ (see Algorithm A.1).
4. Compute the trifocal tensor corresponding to the original data according to $\mathcal{T}_i^{jk} = \mathtt{H}^r_i (\mathtt{H}'^{-1})^j_s (\mathtt{H}''^{-1})^k_t \hat{\mathcal{T}}_r^{st}$.

Algorithm 2.1 *The normalized linear algorithm for the computation of* $\mathcal{T}$

In general, the trifocal tensor that results from applying Algorithm 2.1 is not geometrically valid. In order to enforce the internal constraints, the tensor must be parameterized in terms of the camera matrices. Recall that in a framework with canonical cameras $\mathtt{P} = [\mathtt{I}|\mathtt{0}]$, $\mathtt{P}' = [\mathtt{A}|\mathtt{a}]$ and $\mathtt{P}'' = [\mathtt{B}|\mathtt{b}]$, the trifocal tensor may be computed as

$$\mathcal{T}_i^{jk} = a_i^j b_4^k - a_4^j b_i^k, \tag{2.30}$$

where a^i_j and b^i_j represent the entry at row i, column j from the second and third camera matrices, respectively. The last columns of $\mathtt{P}'$ and $\mathtt{P}''$ are the epipoles (the center of $\mathtt{P}$ projected on $\mathtt{P}'$ and $\mathtt{P}''$). These epipoles can also be computed from the trifocal tensor as follows:

1. For $i = 1, .., 3$, find the unit vector $\mathtt{v}_i$ that minimizes $\|\mathtt{T}_i\mathtt{v}_i\|$, where $\mathtt{T}_i = \mathcal{T}_i^{\cdot\cdot}$ (see Algorithm A.1). Form the matrix $\mathtt{V}$, the i^{th} row of which is $\mathtt{v}_i^\mathsf{T}$
2. Compute the epipole $\mathtt{e}''$ as the unit vector that minimizes $\|\mathtt{V}\mathtt{e}''\|$.

The epipole $\mathtt{e}'$ is computing similarly, using $\mathtt{T}_i^\mathsf{T}$ instead of $\mathtt{T}_i$. In terms of the epipoles, the trifocal tensor can be computed as

$$\mathcal{T}_i^{jk} = a_i^j e''^k - e'^j b_i^k. \tag{2.31}$$

Which provides a parameterization to compute a geometrically valid trifocal tensor. The full procedure is described in Algorithm 2.2.

The trifocal tensor resulting from the application of Algorithm 2.2 is geometrically valid; however it can further be improved by minimizing a more meaningful cost function based on the geometric distance. One such algorithm (The Gold Standard Method) is described in Algorithm 2.3.

1. Compute an initial estimate of $\mathcal{T}_i^{jk}$ using Algorithm 2.1.
2. Find the two epipoles $\mathbf{e}'$ and $\mathbf{e}''$ from $\mathcal{T}_i^{jk}$ as the common perpendicular to the left (respectively right) null-vectors of the three $\mathtt{T}_i$.
3. Construct the 27×8 matrix $\mathtt{E}$ such that $\mathbf{t} = \mathtt{E}\mathbf{a}$ where $\mathbf{t}$ is the vector of entries of $\mathcal{T}_i^{jk}$, $\mathbf{a}$ is the vector representing entries of a_i^j and b_i^k, and $\mathtt{E}$ expresses the linear relationship $\mathcal{T}_i^{jk} = a_i^j e''^k - e'^j b_i^k$.
4. Solve the minimization problem: minimize $\|\mathtt{AEa}\|$ subject to $\|\mathtt{Ea}\| = 1$ (see Algorithm A.2). Compute the error vector $\epsilon = \mathtt{AEa}$.
5. **Iteration:** The mapping $(\mathbf{e}', \mathbf{e}'') \mapsto \epsilon$ is a mapping from $\mathbb{R}^6$ to $\mathbb{R}^{27}$ Iterate on the last two steps with varying $\mathbf{e}'$ and $\mathbf{e}''$ using the Levenberg-Marquardt algorithm to find the optimal $\mathbf{e}'$ and $\mathbf{e}''$ Hence find the optimal $\mathbf{t} = \mathtt{Ea}$ containing the entries of $\mathcal{T}_i^{jk}$.

Algorithm 2.2 *Computation of a geometrically valid trifocal tensor. The data must be normalized as in Algorithm 2.1. The last step is optional.*

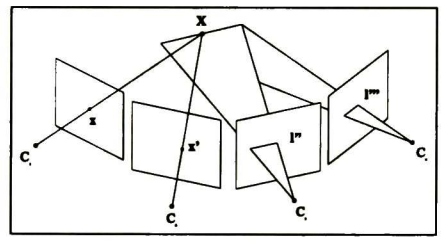


Figure 2.6: The point-point-line-line quadrilinear relation.

2.2.3 The quadrifocal tensor

The Quadrifocal tensor defines the projective geometry of four views. Like the trifocal tensor, the quadrifocal tensor $\mathcal{Q}$ relates points, lines and their combinations. The equations are in Table 2.2 (details on the derivation of these relations can be found in [27]). The point-point-line-line quadrilinear relation is illustrated in Fig. 2.6.

The quadrifocal tensor has 81 entries but only depends on 29 parameters. The equations from Table 2.2 can be rearranged to produce a matrix equation of the form $\mathtt{Aq} = \mathbf{0}$, where $\mathbf{q}$ is an 81-vector containing all the entries of the quadrifocal tensor. However, the vector obtained from this equation will not satisfy the constraints of a *geometrically valid* quadrifocal tensor. Therefore a more sophisticated estimation method must be used. We will be using the method presented in [26], which we describe next.

In [26], an initial estimation of the quadrifocal tensor is obtained by solving an equation of the form $\mathtt{Aq} = \mathbf{0}$ built from the quadrilinear relations. However, the matrix $\mathtt{A}$ is first replaced by a *reduced measurement matrix* $\hat{\mathtt{A}}$ such that $\|\mathtt{Aq}\| = \|\hat{\mathtt{A}}\mathbf{q}\|$. This matrix is computed using the QR decomposition where $\mathtt{A} = \mathtt{Q}\hat{\mathtt{A}}$.

Then, in order to reduce the number of unsatisfied constraints, a *reduced quadrifocal tensor* is computed. This tensor arises from cameras consisting of zeros everywhere except for the main diagonal and the last column. In particular the camera matrices $\mathtt{P}$, $\mathtt{P}'$, $\mathtt{P}''$ and $\mathtt{P}'''$ have the form:

1. Compute an initial geometrically valid estimate of $\mathcal{T}$ using Algorithm 2.2.
2. Compute an initial estimate of the subsidiary variables $\{\hat{\mathbf{x}}_i, \hat{\mathbf{x}}'_i, \hat{\mathbf{x}}''_i\}$ as follows:
   a) Retrieve the camera matrices $\mathtt{P}'$ and $\mathtt{P}''$ from $\mathcal{T}$
   b) From the (given) point correspondence $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i \leftrightarrow \mathbf{x}''_i$ and $\mathtt{P} = [\mathtt{I}|\mathbf{0}]$, $\mathtt{P}'$, $\mathtt{P}''$ determine an estimate of $\hat{\mathbf{X}}_i$ using triangulation.
   c) The correspondence consistent with $\mathcal{T}$ is obtained as $\hat{\mathbf{x}}_i = \mathtt{P}\hat{\mathbf{X}}_i$, $\hat{\mathbf{x}}'_i = \mathtt{P}\hat{\mathbf{X}}_i$ and $\hat{\mathbf{x}}''_i = \mathtt{P}\hat{\mathbf{X}}_i$.
3. Minimize the cost function

$$\sum_i d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 + d(\mathbf{x}'_i, \hat{\mathbf{x}}'_i)^2 + d(\mathbf{x}''_i, \hat{\mathbf{x}}''_i)^2$$

over $\mathcal{T}$ and $\hat{\mathbf{X}}_i$, $i = 1, .., n$. The cost is minimized using the Levenberg-Marquardt algorithm over $3n + 24$ variables: $3n$ for the n 3D points $\hat{\mathbf{X}}_i$ and 24 for the elements of the camera matrices $\mathtt{P}'$, $\mathtt{P}''$.

Algorithm 2.3 *The Gold Standard Algorithm for estimating the trifocal tensor* $\mathcal{T}$

Correspondence Type	Equation
four points	$x^i x'^j x''^k x'''^l \epsilon_{ipw}\epsilon_{jqx}\epsilon_{kry}\epsilon_{lsz}\mathcal{Q}^{pqrs} = 0_{wxyz}$
three points, one line	$x^i x'^j x''^k l'''^l_s \epsilon_{ipw}\epsilon_{jqx}\epsilon_{kry}\mathcal{Q}^{pqrs} = 0_{wxy}$
two points, two lines	$x^i x'^j l''^l_r l'''^l_s \epsilon_{ipw}\epsilon_{jqx}\mathcal{Q}^{pqrs} = 0_{wx}$
three lines	$l_p l'^l_q l''^l_r \mathcal{Q}^{pqrs} = 0^s$
four lines	$l_p l'^l_q l''^l_r \mathcal{Q}^{pqrs} = 0^s$, $l_p l'^l_q l''^l_s \mathcal{Q}^{pqrs} = 0^r$

Table 2.2: The quadrilinear relations.

$$\mathtt{P} = [\mathtt{I}|\mathbf{0}], \quad \mathtt{P}' = \begin{bmatrix} a_1 & & & a'_1 \\ & a_2 & & a'_2 \\ & & a_3 & a'_3 \end{bmatrix}$$

$$\mathtt{P}'' = \begin{bmatrix} b_1 & & & b'_1 \\ & b_2 & & b'_2 \\ & & b_3 & b'_3 \end{bmatrix} \quad \mathtt{P}''' = \begin{bmatrix} c_1 & & & c'_1 \\ & c_2 & & c'_2 \\ & & c_3 & c'_3 \end{bmatrix} \tag{2.32}$$

This, in turn produces a large number of zeros in the entries of the quadrifocal tensor, thus reducing the number of significant entries. In practice, this means that three correspondences $\mathbf{u}_i \leftrightarrow \mathbf{u}'_i \leftrightarrow \mathbf{u}''_i \leftrightarrow \mathbf{u}'''_i$ for $i = 1, .., 3$ must be chosen so that these points are not collinear in any image. Then, the canonical points $\mathbf{e}_1 = [1, 0, 0]^\mathsf{T}$, $\mathbf{e}_2 = [0, 1, 0]^\mathsf{T}$ and $\mathbf{e}_3 = [0, 0, 1]^\mathsf{T}$ must be mapped to the points $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ via the mappings of the form

$$\mathtt{T} = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix} \tag{2.33}$$

where $[u_i, v_i, w_i]^\mathsf{T}$ are the coordinates of $\mathbf{u}_i$.

The rule to transform from the reduced quadrifocal tensor $\hat{\mathcal{Q}}$ to the full quadrifocal tensor $\mathcal{Q}$ is

$$\mathcal{Q}^{ijkl} = \hat{\mathcal{Q}}^{abcd} \mathtt{T}_a^i \mathtt{T}'^j_b \mathtt{T}''^k_c \mathtt{T}'''^l_d. \tag{2.34}$$

The reduced quadrifocal tensor $\hat{\mathcal{Q}}$ has only 36 non-zero entries and thus can be represented by a 36-vector $\hat{\mathbf{q}}$. The preceding equation can thus be expressed as a 81×36 matrix T such that $\mathbf{q} = \mathtt{T}\hat{\mathbf{q}}$, where $\mathbf{q}$ is an 81-vector representing the full quadrifocal tensor.

Once the reduced quadrifocal tensor is computed, the diagonals of the camera matrices can be extracted from it by solving

$$\begin{bmatrix} 0 & \mathcal{Q}^{2311} & \mathcal{Q}^{3211} \\ \mathcal{Q}^{1322} & 0 & \mathcal{Q}^{3122} \\ \mathcal{Q}^{1233} & \mathcal{Q}^{2133} & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \mathbf{0}$$
$$\begin{bmatrix} 0 & \mathcal{Q}^{2131} & \mathcal{Q}^{3121} \\ \mathcal{Q}^{1232} & 0 & \mathcal{Q}^{3212} \\ \mathcal{Q}^{1323} & \mathcal{Q}^{2313} & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \mathbf{0} \tag{2.35}$$
$$\begin{bmatrix} 0 & \mathcal{Q}^{2113} & \mathcal{Q}^{3112} \\ \mathcal{Q}^{1223} & 0 & \mathcal{Q}^{3221} \\ \mathcal{Q}^{1332} & \mathcal{Q}^{2331} & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \mathbf{0}$$

After the values of $[a_1, a_2, a_3]^\mathsf{T}$, $[b_1, b_2, b_3]^\mathsf{T}$ and $[c_1, c_2, c_3]^\mathsf{T}$ are computed, the values of the epipoles of the reduced camera matrices (the last columns) can be computed too, since the entries of $\hat{\mathcal{Q}}$ are a linear combination of the epipoles. Recall that the quadrifocal tensor can be computed from

$$\mathcal{Q}^{pqrs} = \det \begin{bmatrix} \mathbf{a}^p \\ \mathbf{b}^q \\ \mathbf{c}^r \\ \mathbf{d}^s \end{bmatrix}, \tag{2.36}$$

where $\mathbf{a}^p$ is the p^{th} of the first camera matrix, $\mathbf{b}^q$ is the q^{th} row of the second camera matrix and so on. Thus, the exact form of this relation between $\hat{\mathcal{Q}}$ and the epipoles can be computed by cofactor expansion down the last columns of the reduced camera matrices. This relation has the form $\hat{\mathbf{q}} = \mathtt{M}\mathbf{a}'$ where M is a 36×9 matrix and $\mathbf{a}'$ is the vector

$$\mathbf{a}' = [a'_1, a'_2, a'_3, b'_1, b'_2, b'_3, c'_1, c'_2, c'_3]^\mathsf{T} \tag{2.37}$$

containing the entries of the epipoles. The full procedure to compute the quadrifocal tensor is described in Algorithm 2.4.

The quadrifocal tensor computed by Algorithm 2.4 is valid and minimizes the algebraic error subject to two conditions:

1. The camera matrix diagonals have the value given by the 9-vector $\mathbf{a}$, computed at step 5 of the algorithm.
2. The three points used to compute transforms $\mathtt{T}, .., \mathtt{T}'''$ correspond precisely.

The solution from Algorithm 2.4 can be further improved by a Levenberg-Marquardt nonlinear minimization. The parameters for this minimization are $\mathbf{a}$ (from the mapping $\mathbf{a} \mapsto \mathtt{A}\mathbf{q}$) and points $\mathbf{u}'_i$, $\mathbf{u}''_i$, $\mathbf{u}'''_i$ (producing, in turn, the transformation matrices $\mathtt{T}', .., \mathtt{T}'''$). This produces a total of 27 parameters for the minimization.

1. Form the reduced measurement matrix $\hat{\mathbf{A}}$ from the original data (according to Table 2.2).
2. Obtain the transformation matrices $\mathtt{T}, .., \mathtt{T}'''$ from three of the correspondences using Eq. 2.33.
3. Compute the 81×36 transformation matrix T such that $\mathbf{q} = \mathrm{T}\hat{\mathbf{q}}$ from the transformation rule 2.34.
4. Minimize $\|\mathtt{A}\mathrm{T}\hat{\mathbf{q}}\|$ subject to $\|\mathrm{T}\hat{\mathbf{q}}\| = 1$ to find $\hat{\mathbf{q}}$, an initial estimate of the reduced quadrifocal tensor.
5. Find the diagonal elements (vector $\mathbf{a}$) of the reduced camera matrices by solving Eq. 2.35.
6. Compute the 36×9 matrix $\mathtt{M}$ such that $\hat{\mathbf{q}} = \mathtt{M}\mathbf{a}'$, where $\mathbf{a}'$ is the 9-vector containing the elements of the last columns of the reduced camera matrices.
7. Minimize $\|\mathtt{A}\mathrm{T}\mathtt{M}\mathbf{a}'\|$ subject to $\|\mathrm{T}\mathtt{M}\mathbf{a}'\| = 1$ (see Algorithm A.2). From this we may derive the vector $\mathbf{q} = \mathrm{T}\mathtt{M}\mathbf{a}'$ corresponding to a full quadrifocal tensor.
8. If desired, we may compute the reduced camera matrices from the vectors $\mathbf{a}$ and $\mathbf{a}'$. These may be transformed to camera matrices for the original set of data by left-multiplication by the transforms $\mathtt{T}, .., \mathtt{T}'''$.

Algorithm 2.4 *Computation of a geometrically valid quadrifocal tensor.*

2.3 Part II: Clifford Geometric Algebra, Conformal Geometry and the Affine Plane

The main alternative to the classical approach to computer vision is Clifford *Geometric Algebra*. This algebra system was invented by the English mathematician William Kingdom Clifford (1845-1879) who combined the ideas introduced by the German mathematician Hermann Günther Grassmann (1809-1877) and Sir William Rowan Hamilton (1805-1865). Since the 1960s, David Hestenes has been working on developing his version of Clifford Algebra [28]. We will now present a brief introduction to Geometric Algebra.

Geometric Algebra is similar to the Vector Calculus but it is enabled with a new product (the *Clifford product*) that has inverse (in general) and combines the properties of the interior and exterior products. That is to say, for two vectors $\mathbf{a}$ and $\mathbf{b}$, their Clifford product is expressed as

$$\mathbf{ab} = \mathbf{a} \quad \mathbf{b} + \mathbf{a} \wedge \mathbf{b}, \tag{2.38}$$

where the *wedge product* $\wedge$ is similar to the cross product; but instead of producing a vector, a new entity, called a *bivector* is rendered. The bivector $\mathbf{a} \wedge \mathbf{b}$ can be visualized as the oriented plane spanned by $\mathbf{a}$ and $\mathbf{b}$ (the sense of the normal to the plane can be obtained by the right-hand rule, see Fig. 2.7).

The Clifford product is linear, associative and anticommutative, that is

$$\mathbf{ab} = -\mathbf{ba}, \tag{2.39}$$

from whence

$$\mathbf{a} \wedge \mathbf{b} = -\mathbf{b} \wedge \mathbf{a} \tag{2.40}$$

can be easily derived. From Eq. 2.38, we can see that the Clifford product of a vector with itself produces a scalar. In general, this scalar *will not be necessarily positive*. By definition, in a

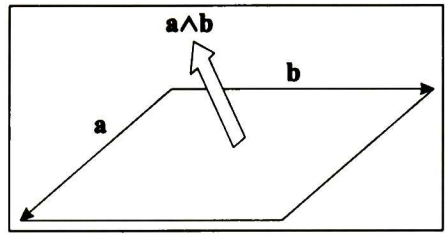


Figure 2.7: The bivector $\mathbf{a} \wedge \mathbf{b}$.

geometric algebra $\mathcal{G}_{p,q,r}$, the first p basis vectors $e_1, .., e_p$ will square to 1, the next q basis vectors $e_{p+1}, .., e_{p+q}$ will square in -1 and the last r basis vectors will square to 0.

The common geometric algebra of 3D space $\mathcal{G}_{3,0,0}$ is spanned by

$$\left\{ \underbrace{1}_{scalar}, \underbrace{e_1, e_2, e_3}_{vectors} \underbrace{e_{23}, e_{31}, e_{12}}_{bivectors}, \underbrace{e_{123} \equiv I}_{trivectors} \right\}. \tag{2.41}$$

The elements that span a given geometric algebra are called *blades*. The *grade* of a blade indicates the number of basis vectors used to form it. That is to say, a scalar is a zero-grade blade and the element e_{123} is a three-grade blade; however, in general, blades will be multiplied by a scalar factor. The blade with the greatest grade in any given algebra is called the *pseudoscalar* and is commonly represented by the letter I. A *k-vector* is a linear combination of blades of grade k.

A *multivector* is a linear combination of blades of different grades. For any given multivector $\mathbf{A}$, the notation $\langle \mathbf{A} \rangle_r$ indicates the component r-vector of $\mathbf{A}$ (if the subindex is omitted, the scalar part is assumed). The grade of a multivector is the highest grade of its component blades. A *homogeneous multivector* is a multivector that only contains blades of the same grade. Using the blade grade, we can express the Clifford product of any two given multivectors $\mathbf{A}_r$ and $\mathbf{B}_s$ of grade r and s, respectively, as

$$\mathbf{A}_r \mathbf{B}_s = \langle \mathbf{A}_r \mathbf{B}_s \rangle_{r+s} + \langle \mathbf{A}_r \mathbf{B}_s \rangle_{r+s-2} + ... + \langle \mathbf{A}_r \mathbf{B}_s \rangle_{|r-s|}. \tag{2.42}$$

From this equation, we can derive the general definition for the interior and exterior product for multivectors as

$$\mathbf{A}_r \;\; \mathbf{B}_s = \langle \mathbf{A}_r \mathbf{B}_s \rangle_{|r-s|}, \tag{2.43}$$

$$\mathbf{A}_r \wedge \mathbf{B}_s = \langle \mathbf{A}_r \mathbf{B}_s \rangle_{r+s}. \tag{2.44}$$

Finally, for an r-grade multivector $\mathbf{A}_r = \sum_{i=0}^{r} \langle \mathbf{A}_r \rangle_i$, the following operations are defined

$$\text{Grade Involution: } \hat{\mathbf{A}}_r = \sum_{i=0}^{r} (-1)^i \langle \mathbf{A}_r \rangle_i, \tag{2.45}$$

$$\text{Reversion: } \widetilde{\mathbf{A}}_r = \sum_{i=0}^{r} (-1)^{\frac{i(i-1)}{2}} \langle \mathbf{A}_r \rangle_i, \tag{2.46}$$

$$\text{Clifford Conjugation: } \bar{\mathbf{A}}_r = \widetilde{\hat{\mathbf{A}}}_r = \sum_{i=0}^{r} (-1)^{\frac{i(i+1)}{2}} \langle \mathbf{A}_r \rangle_i. \tag{2.47}$$

The grade involution simply negates the odd-grade blades of a multivector. The reversion can also be obtained by reversing the order of basis vectors making up the blades in a multivector and then rearranging them to their original order using the anticommutativity of the Clifford product. The Clifford conjugation can be used to compute the inverse of a vector **a** as

$$\mathbf{a}^{-1} = \frac{\bar{\mathbf{a}}}{\mathbf{a}\bar{\mathbf{a}}}. \tag{2.48}$$

This formula for the inverse can also be applied for homogeneous multivectors, but cannot be used for all multivectors in general.

Before finishing this small introduction, we would like to advance a few words about an entity called Rotor. A rotor is a bivector that represents a rotation of a vector about the origin. The rotor can be expressed in exponential form as

$$R = e^{\frac{1}{2}\theta B} \tag{2.49}$$

where B is a unit bivector that represents the axis of rotation and θ is the angle of rotation. This representation of the rotor can be developed to yield the following formula

$$R = \cos(\tfrac{\theta}{2}) + B\sin(\tfrac{\theta}{2}) = \cos(\tfrac{\theta}{2})\left(1 + B\frac{\sin(\frac{\theta}{2})}{\cos(\frac{\theta}{2})}\right) \tag{2.50}$$

This definition of the rotor will prove to be quite useful when we discuss the problem of 2D point registration. We will present more details on the rotor in subsequent sections.

We shall conclude our introduction to Clifford Algebra here. We refer the interested reader to [28], [29], [40], [6] and [7] for more details. We also recommend the use of CLICAL [39], [41] to help in familiarizing with the Clifford Algebras.

2.4 Conformal Geometry

The Geometric Algebra can be used to express Conformal Geometry in a very elegant way. To see how this is possible, we follow the same formulation presented in [53] and show how the Euclidean vector space $\mathbb{R}^n$ is represented in $\mathbb{R}^{n+1,1}$ This space has an orthonormal vector basis given by $\{e_1, .., e_n, e_+, e_-\}$ with the properties

$$e_i^2 = 1, \quad i = 1.., n; \tag{2.51}$$

$$e_\pm^2 = \pm 1, \tag{2.52}$$

$$e_i \cdot e_+ = e_i \cdot e_- = e_+ \cdot e_- = 0, \quad i = 1, .., n. \tag{2.53}$$

A *null basis* $\{e_0, e_\infty\}$ can be introduced by

$$e_0 = \tfrac{1}{2}(e_- - e_+), \tag{2.54}$$

$$e_\infty = e_- + e_+. \tag{2.55}$$

with the properties

$$e_0^2 = e_\infty^2 = 0,\ e_\infty \cdot e_0 = -1. \tag{2.56}$$

A unit pseudoscalar $E \in \mathbb{R}^{1,1}$ which represents the Minkowski plane is defined by

$$E = e_\infty \wedge e_0 = e_+ \wedge e_- = e_+ e_-, \tag{2.57}$$

having the properties

$$E^2 = 1, \quad \widetilde{E} = -E \tag{2.58}$$

$$Ee_\pm = -e_\mp, \tag{2.59}$$

$$Ee_\infty = -e_\infty E = -e_\infty, \quad Ee_0 = -e_0 E = e_0, \quad \text{(absorption property)} \tag{2.60}$$

$$1 - E = -e_\infty e_0, \quad 1 + E = -e_0 e_\infty. \tag{2.61}$$

The dual of E is given by

$$E^* = EI^{-1} = -E\widetilde{I}, \tag{2.62}$$

where I is the pseudoscalar for $\mathbb{R}^{n+1,1}$.

Euclidean points $\mathbf{x}_e \in \mathbb{R}^n$ can be represented in $\mathbb{R}^{n+1,1}$ in a general way as

$$x_c = \mathbf{x}_e + \alpha e_0 + \beta e_\infty. \tag{2.63}$$

where α and β are arbitrary scalars. A conformal point $x_c \in \mathbb{R}^{n+1,1}$ can be divided into its Euclidean and conformal part by an operation called the *conformal split*. This split is defined by the projection operators P_E (projection) and $P_E^\perp$ (rejection) as follows

$$P_E(x_c) = (x_c \cdot E)E = \alpha e_0 + \beta e_\infty \in \mathbb{R}^{1,1}, \tag{2.64}$$

$$P_E^\perp(x_c) = (x_c \ E^*)\widetilde{E^*} = (x_c \wedge E)E = \mathbf{x}_e \in \mathbb{R}^n \tag{2.65}$$

The names "projection" and "rejection" stem from the geometrical meaning of these operators. The first returns the component of x_c which is parallel to E by a projection (dot product). The latter produces the component of x_c which is orthogonal to E, hence the name (see Figure 2.8).

To improve our model, we would like to use *homogeneous coordinates* as in the case of projective geometry. In homogeneous coordinates, all points are equal up to a scale factor. Therefore, we need to define some way to fix the scale of the points. A point $x_c \in \mathbb{R}^{n+1,1}$ is normalized or in standard form when

$$x_c \cdot e_\infty = -1. \tag{2.66}$$

Now, recall that a hyperplane $\mathbb{P}(n, a) \in \mathbb{R}^{n+1,1}$ with normal n and passing though the point a is the solution to the equation

$$n \cdot (x - a) = 0,\ x \in \mathbb{R}^{n+1,1} \tag{2.67}$$

The normalization condition $x_c \cdot e_\infty = e_\infty \cdot e_0 = -1$ is equivalent to the equation

$$e_\infty \cdot (x_c - e_0) = 0, \tag{2.68}$$

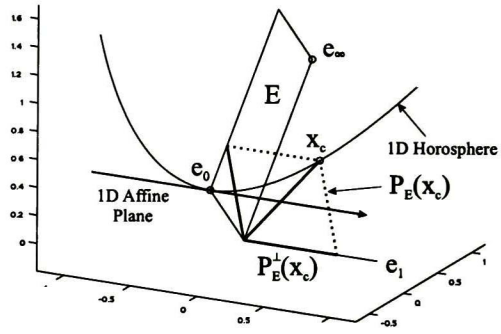


Figure 2.8: The projection and rejection of vector $x_c \in \mathbb{R}^{2,1}$ from the E plane. The operators are illustrated for the case of 1-D.

which is the equation of a hyperplane $\mathbb{P}(e_\infty, e_0)$. Thus the normalization condition of Eq. 2.66 constrains the points x_c to lie in a hyperplane passing though e_0 with normal e_∞. Eq. 2.66 fixes scale; however for the conformal model, another constraint is needed to fix x_c as a unique representation of $\mathbf{x}_e \in \mathbb{R}^n$.

To complete the definition of *generalized homogeneous coordinates* for points in $\mathbb{R}^{n+1,1}$ the last constraint is that $x_c^2 = 0$. The set $\mathbb{N}^{n+1}$ of vectors that square to zero is called the *null cone*. Therefore, conformal points are required to lie in the intersection of the null cone $\mathbb{N}^{n+1}$ with the hyperplane $\mathbb{P}(e_\infty, e_0)$, the resulting surface $\mathbb{N}_e^n$ is called the *Horosphere*:

$$\mathbb{N}_e^n = \mathbb{N}^{n+1} \cap \mathbb{P}(e_\infty, e_0) = \{x_c \in \mathbb{R}^{n+1,1} | x_c^2 = 0, x_c \cdot e_\infty = -1\} \tag{2.69}$$

These two constraints finally define the mapping from Euclidean space to conformal space. To see how this mapping is obtained, first we see that any point $x_c = \mathbf{x}_e + \alpha e_0 + \beta e_\infty \in \mathbb{N}_e^n$ can be expressed as $x_c = \mathbf{x}_e + k_1 e_+ + k_2 e_-$, for some scalars k_1, k_2, since e_0 and e_∞ are linear combinations of the basis vectors e_+ and e_-. Then, by applying the conformal split to x_c we get

$$x_c = x_c E^2 = (x_c \wedge E + x_c \ \lrcorner E)E = (x_c \wedge E)E + (x_c \ \lrcorner E)E, \tag{2.70}$$

since $E^2 = 1$. Now, recall that $(x_c \wedge E)E = \mathbf{x}_e$ is the rejection (see Eq. 2.65). The expression $(x_c \cdot E)E$ can be expanded as

$$\begin{aligned}
(x_c \cdot E)E &= (x_c \cdot (e_\infty \wedge e_0))E, \\
&= ((x_c \cdot e_\infty) \wedge e_0 - e_\infty \wedge (x_c \cdot e_0))E, \\
&= (-e_0 - e_\infty \wedge (x_c \cdot e_0))E \qquad \text{(applying constraint 2.66)}, \\
&= e_0 + \tfrac{1}{2}(k_1 + k_2)e_\infty.
\end{aligned} \tag{2.71}$$

Now, applying the condition that $x_c^2 = 0$, we find from 2.70

$$\begin{aligned}
x_c^2 &= ((x_c \wedge E)E + (x_c \ \lrcorner E)E)^2, \\
0 &= (\mathbf{x}_e + e_0 + \tfrac{1}{2}(k_1 + k_2)e_\infty)^2.
\end{aligned}$$

$$\begin{aligned} 0 &= \mathbf{x}_e^2 - (k_1 + k_2), \\ \mathbf{x}_e^2 &= (k_1 + k_2). \end{aligned} \tag{2.72}$$

Finally, using 2.70 and substituting 2.72 into 2.71 we get

$$x_c = (x_c \wedge E)E + (x_c \cdot E)E = \mathbf{x}_e + e_0 + \frac{1}{2}(k_1 + k_2)e_\infty = \mathbf{x}_e + \frac{1}{2}\mathbf{x}_e^2 e_\infty + e_0. \tag{2.73}$$

An illustration of the null cone, the hyperplane and the horosphere can be seen in Figure 2.9

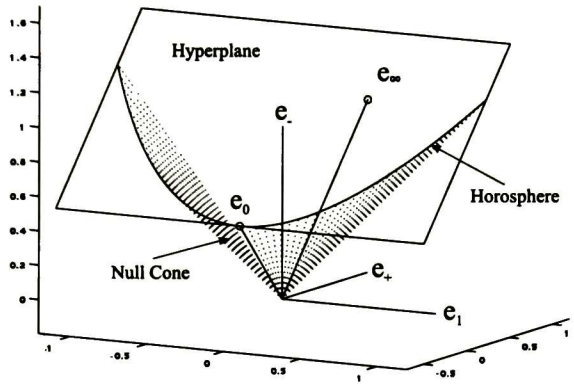


Figure 2.9: The Null Cone (dotted lines), the Hyperplane $\mathbb{P}(e_\infty, e_0)$ and the Horosphere for 1-D. Note that even though the normal of the hyperplane is e_∞ the plane is actually *geometrically parallel* to this vector.

We can gain further insight into the geometrical meaning of the null vectors by analyzing Eq. 2.73. For instance by setting $\mathbf{x}_e = 0$ we find that e_0 represents the origin of $\mathbb{R}^n$ (hence the name). Similarly, dividing this equation by $x_c \cdot e_0 = -\frac{1}{2}\mathbf{x}_e^2$ gives

$$\frac{x_c}{x_c \cdot e_0} = -\frac{2}{\mathbf{x}_e^2}(\mathbf{x}_e + \frac{1}{2}\mathbf{x}_e^2 e_\infty + e_0), \tag{2.74}$$

$$= -2(\frac{1}{\mathbf{x}_e} + \frac{1}{2}e_\infty + \frac{e_0}{\mathbf{x}_e^2}) \underset{\mathbf{x}_e \to \infty}{\longrightarrow} e_\infty. \tag{2.75}$$

Thus we conclude that e_∞ represents the point at infinity.

2.5 The Stereographic Projection

The Conformal Geometry is equivalent to a Stereographic Projection in Euclidean space. Generally speaking, a stereographic projection is a mapping taking points lying on a hypersphere to points lying on a hyperplane following a simple geometric construction. This projection is used in cartography to make maps of the earth. In this case, the projection plane passes through the equator and the sphere is centered at the origin. To make a projection, a line is drawn from the

north pole to each point on the sphere and the intersection of this line with the projection plane constitutes the stereographic projection.

For simplicity, we will illustrate the equivalence between Stereographic Projections and Conformal Geometric Algebra in $\mathbb{R}^1$. We will be working in $\mathbb{R}^{2,1}$ with the basis vectors $\{e_1, e_+, e_-\}$ having the usual properties. The projection plane will be the x-axis and the sphere will be a circle centered at the origin with unitary radius.

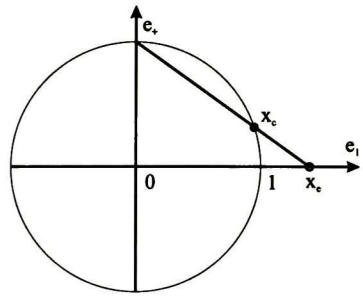


Figure 2.10: Stereographic projection for 1-D.

Given a scalar x_e representing a point on the x-axis, we wish to find the point x_c lying on the circle that projects to it (see Figure 2.10). The equation of the line passing through the north pole and x_e is given by

$$f(x) = -\frac{1}{x_e}x + 1. \tag{2.76}$$

The equation of the circle is

$$x^2 + f(x)^2 = 1. \tag{2.77}$$

Substituting the equation of the line on the circle, we get

$$x^2 - 2xx_e + x^2x_e^2 = 0. \tag{2.78}$$

Which has the two solutions

$$x = 0, \quad x = 2\frac{x_e}{x_e^2 + 1}. \tag{2.79}$$

From which only the latter is meaningful. Substituting in the equation of the line we get

$$f(x) = \frac{x_e^2 - 1}{x_e^2 + 1}. \tag{2.80}$$

Hence, x_c has coordinates

$$x_c = \left(2\frac{x_e}{x_e^2 + 1}, \frac{x_e^2 - 1}{x_e^2 + 1}\right) \tag{2.81}$$

which can be represented in homogeneous coordinates as the vector

$$x_c = 2\frac{x_e}{x_e^2 + 1}e_1 + \frac{x_e^2 - 1}{x_e^2 + 1}e_+ + e_-. \tag{2.82}$$

From this equation we can infer the coordinates on the circle for the point at infinity as

$$x_\infty = \lim_{x_e \to \infty} \left(2\frac{x_e}{x_e^2+1}, \frac{x_e^2-1}{x_e^2+1} \right), \tag{2.83}$$

$$x_\infty = (0,1), \tag{2.84}$$

which are the coordinates of the north pole; or in homogeneous coordinates as

$$e_\infty = \lim_{x_e \to \infty} \left\{ 2\frac{x_e}{x_e^2+1}e_1 + \frac{x_e^2-1}{x_e^2+1}e_+ + e_- \right\}, \tag{2.85}$$

$$e_\infty = e_+ + e_-, \tag{2.86}$$

which matches our previous definition of e_∞. Similarly, evaluating at $x_e = 0$ gives

$$x_0 = (0,-1) \tag{2.87}$$

(which are the coordinates for the south pole), or in homogeneous coordinates

$$e_0 = -e_+ + e_-. \tag{2.88}$$

Note that this definition for e_0 differs from the original *by a scale factor* which is the normal equivalence under *homogeneous* coordinates. The geometric implications of this difference in scale will be discussed at the end of this section.

To prove that the stereographic projection is equivalent to a conformal mapping we note that Eq. 2.82 can be rewritten to

$$x_c = 2\frac{x_e}{x_e^2+1}e_1 + \frac{x_e^2-1}{x_e^2+1}e_+ + e_-, \tag{2.89}$$

$$= \frac{2}{x_e^2+1}\left(x_e e_1 + \frac{1}{2}(x_e^2-1)e_+ + \frac{1}{2}(x_e^2+1)e_- \right) \tag{2.90}$$

Dividing by the scale factor $\frac{2}{x_e^2+1}$ in order to achieve the constraint imposed by Eq. 2.66: $x_c \cdot e_\infty = -1$ we arrive at

$$\begin{aligned} x_c &= x_e e_1 + \tfrac{1}{2}(x_e^2-1)e_+ + \tfrac{1}{2}(x_e^2+1)e_-, \\ &= x_e e_1 + \tfrac{1}{2}x_e^2 e_\infty + e_0, \\ &= \mathbf{x}_e + \tfrac{1}{2}\mathbf{x}_e^2 e_\infty + e_0, \end{aligned} \tag{2.91}$$

where $\mathbf{x}_e = x_e e_1$, which is precisely Eq. 2.73. Hence, we have demonstrated that Conformal Geometric Algebra is projectively equivalent to a stereographic projection (i.e. *up to a scale factor*).

To conclude this section, we will elaborate on the geometrical meaning of the equivalence up to scale factor. Recall that from the definition of a stereographic projection we arrived at the values

$$e_\infty = e_+ + e_-, \quad \text{and} \tag{2.92}$$

$$e_0 = e_- - e_+, \tag{2.93}$$

for the point at infinity and the origin respectively. The value of e_0 in particular, differs from the original definition by a scale factor. This difference in scale factors has the effect of *displacing the hyperplane* $\mathbb{P}(e_\infty, e_0)$ as we will prove now. From the stereographic projection Eq. 2.82 it can be easily seen that all points mapped by this formula satisfy

$$e_- \cdot x_c = -1, \tag{2.94}$$

and more importantly,

$$e_- \ (x_c - e_0), \tag{2.95}$$

where $e_0 = e_- - e_+$ as previously stated. This means, that the hyperplane passes through e_0 and has e_- as normal. The resulting configuration of the null cone, hyperplane and horosphere can be seen in Figure 2.11.

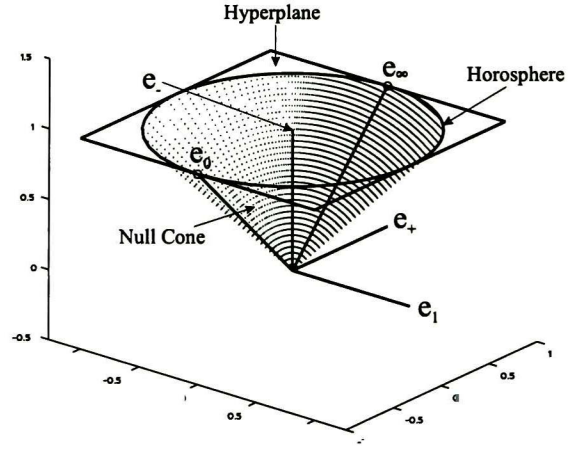


Figure 2.11: The null cone $\mathbb{N}^{n+1}$ hyperplane $\mathbb{P}(e_\infty, e_0)$ and horosphere $\mathbb{N}_e^n$ as generated by the equation $x_c = 2\frac{x_e}{x_e^2+1}e_1 + \frac{x_e^2-1}{x_e^2+1}e_+ + e_-$.

Therefore, the choice of normalization affects the orientation of the hyperplane, which in turn modifies the conformal mapping equation according to a scalar factor. The position of the hyperplane also determines the shape of the horosphere, but the properties of conformal geometry *remain invariant*. This fact will be useful later on, when we discuss the particular choice of hyperplane used in this work. For the remainder of this section, however, we will return to the original definition of the n-dimensional horosphere.

2.5.1 Spheres and planes

The equation of a sphere of radius ρ centered at point $\mathbf{p}_e \in \mathbb{R}^n$ can be written as

$$(\mathbf{x}_e - \mathbf{p}_e)^2 = \rho^2 \tag{2.96}$$

Since $x_c \cdot y_c = -\frac{1}{2}(\mathbf{x}_e - \mathbf{y}_e)^2$, we can rewrite the formula above in terms of homogeneous coordinates as

$$x_c \cdot p_c = -\tfrac{1}{2}\rho^2 \tag{2.97}$$

Since $x_c \cdot e_\infty = -1$ we can factor the expression above to

$$x_c \cdot (p_c - \tfrac{1}{2}\rho^2 e_\infty) = 0. \tag{2.98}$$

Which finally yields the simplified equation for the sphere as

$$x_c \;\; s = 0, \tag{2.99}$$

where

$$s = p_c - \frac{1}{2}\rho^2 e_\infty = \mathbf{p}_e + e_0 + \frac{\mathbf{p}_e^2 - \rho^2}{2} e_\infty \tag{2.100}$$

is the equation of the sphere (note from this equation that a point is just a sphere with zero radius). The vector s has the properties

$$s^2 = \rho^2 > 0, \tag{2.101}$$

$$e_\infty \;\; s = -1. \tag{2.102}$$

From these properties, we conclude that the sphere s is a point lying on the hyperplane, but *outside* the null cone. In particular, all points on the hyperplane outside the horosphere determine spheres with positive radius, points lying on the horosphere define spheres of zero radius (i.e. points), and points lying inside the horosphere have imaginary radius (see Figure 2.12). Finally, note that spheres of the same radius form a surface which is parallel to the horosphere.

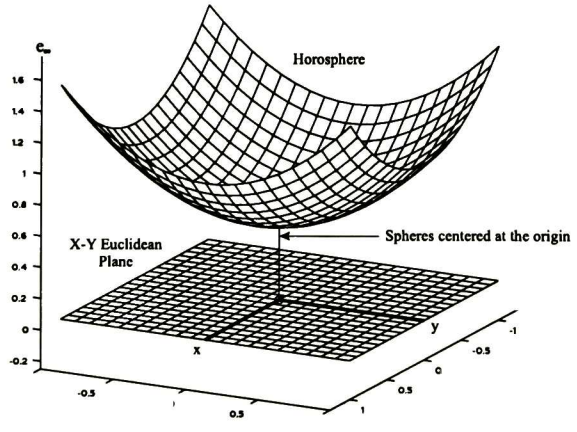


Figure 2.12: The horosphere for $\mathbb{R}^2$ The x and y axes are shown, and the z axis represents the e_∞ direction, e_0 is not shown. The vertical line falling down from the horosphere represents the spheres centered at the origin with varying radii.

The radius and center of a sphere can be recovered from s, using 2.101 and 2.102 as

$$\rho^2 = \frac{s^2}{(s \cdot e_\infty)^2}, \quad \text{and} \tag{2.103}$$

$$p_c = \frac{s}{-(s \cdot e_\infty)} + \frac{1}{2}\rho^2 e_\infty. \tag{2.104}$$

With the normalization $s \cdot e_\infty = -1$, each sphere is represented by a unique vector and the set $\{x_c \in \mathbb{R}^{n+1,1} | x_c \quad s > 0\}$ represents the interior of the sphere.

Alternatively, spheres can be dualized and represented as $(n+1)$-vectors $s^* = sI^{-1}$ Since

$$\widetilde{I} = (-1)^{\frac{1}{2}(n+2)(n+1)} I = -I^{-1}, \tag{2.105}$$

we can express constraints 2.101 and 2.102 as

$$\begin{aligned} s^2 &= -\widetilde{s^*} s^* = \rho^2, \\ e_\infty \cdot s &= e_\infty \cdot (s^* I) = (e_\infty \wedge s^*) I = -1. \end{aligned} \tag{2.106}$$

The equation for the sphere now becomes

$$x_c \wedge s^* = 0. \tag{2.107}$$

The advantage of the dual form is that the sphere can be directly computed from four points (in 3D) as

$$s^* = x_{c_1} \wedge x_{c_2} \wedge x_{c_3} \wedge x_{c_4}. \tag{2.108}$$

If we replace one of these points for the point at infinity we get

$$\pi^* = x_{c_1} \wedge x_{c_2} \wedge x_{c_3} \wedge e_\infty, \tag{2.109}$$

Developing the products, we get

$$\begin{aligned} x_{c_1} \wedge x_{c_2} &= \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} + \tfrac{1}{2}(\mathbf{x}_{e_2}^2 \mathbf{x}_{e_1} - \mathbf{x}_{e_1}^2 \mathbf{x}_{e_2}) \wedge e_\infty + \\ &\quad (\mathbf{x}_{e_1} - \mathbf{x}_{e_2}) \wedge e_0 + \tfrac{1}{2}(\mathbf{x}_{e_1}^2 - \mathbf{x}_{e_2}^2) E, \end{aligned} \tag{2.110}$$

$$x_{c_1} \wedge x_{c_2} \wedge e_\infty = \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge e_\infty - (\mathbf{x}_{e_1} - \mathbf{x}_{e_2}) \wedge E, \tag{2.111}$$

$$\begin{aligned} x_{c_3} \wedge x_{c_1} \wedge x_{c_2} \wedge e_\infty &= \mathbf{x}_{e_3} \wedge \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge e_\infty - \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge E + \\ &\quad \mathbf{x}_{e_3} \wedge (\mathbf{x}_{e_2} - \mathbf{x}_{e_1}) \wedge E. \end{aligned} \tag{2.112}$$

Since $\mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} = \mathbf{x}_{e_1} \wedge (\mathbf{x}_{e_2} - \mathbf{x}_{e_1})$, we get

$$\begin{aligned} x_{c_3} \wedge x_{c_1} \wedge x_{c_2} \wedge e_\infty &= \mathbf{x}_{e_3} \wedge \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge e_\infty + \\ &\quad ((\mathbf{x}_{e_3} - \mathbf{x}_{e_1}) \wedge (\mathbf{x}_{e_2} - \mathbf{x}_{e_1})) \wedge E. \end{aligned} \tag{2.113}$$

But since $\mathbf{x}_e \quad E = 0$, we can rewrite this as

$$x_{c_3} \wedge x_{c_1} \wedge x_{c_2} \wedge e_\infty = \mathbf{x}_{e_3} \wedge \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge e_\infty + ((\mathbf{x}_{e_3} - \mathbf{x}_{e_1}) \wedge (\mathbf{x}_{e_2} - \mathbf{x}_{e_1}))E, \tag{2.114}$$

So π^* becomes

$$\begin{aligned} \pi^* &= x_{c_1} \wedge x_{c_2} \wedge x_{c_3} \wedge e_\infty, \\ &= \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge \mathbf{x}_{e_3} \wedge e_\infty + ((\mathbf{x}_{e_1} - \mathbf{x}_{e_2}) \wedge (\mathbf{x}_{e_3} - \mathbf{x}_{e_2}))E, \end{aligned} \tag{2.115}$$

which is the equation of the plane passing through the points $\mathbf{x}_{e_1}$, $\mathbf{x}_{e_2}$ and $\mathbf{x}_{e_3}$. We can easily see that $\mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge \mathbf{x}_{e_3}$ is a scalar representing the volume of the parallelepiped with sides $\mathbf{x}_{e_1}$, $\mathbf{x}_{e_2}$ and $\mathbf{x}_{e_3}$. Also, since $(\mathbf{x}_{e_1} - \mathbf{x}_{e_2})$ and $(\mathbf{x}_{e_3} - \mathbf{x}_{e_2})$ are two vectors on the plane, the expression $((\mathbf{x}_{e_1} - \mathbf{x}_{e_2}) \wedge (\mathbf{x}_{e_3} - \mathbf{x}_{e_2}))$ is the normal to the plane. Therefore planes are spheres passing through the point at infinity.

2.5.2 Circles and lines

A circle z can be regarded as the intersection of two spheres s_1 and s_2. This means that for each point x_c on the circle z

$$x_c \in z \iff x_c \in s_1 \quad \text{and} \quad x_c \in s_2. \tag{2.116}$$

Assuming that s_1 and s_2 are linearly independent, we can write

$$\begin{aligned} x_c &\in z \\ \iff (x_c \cdot s_1)s_2 - (x_c \cdot s_2)s_1 &= 0, \\ \iff x_c \ (s_1 \wedge s_2) &= 0, \\ \iff x_c \ z &= 0, \end{aligned} \tag{2.117}$$

where $z = (s_1 \wedge s_1)$ is the intersection of the spheres. The intersection with a third sphere leads to a point pair.

The dual form of the circle (in 3D) can be expressed by three points lying on it as

$$z^* = x_{c_1} \wedge x_{c_2} \wedge x_{c_3}. \tag{2.118}$$

Similar to the case of planes, lines can be defined by circles passing through the point at infinity as

$$l^* = x_{c_1} \wedge x_{c_2} \wedge e_\infty. \tag{2.119}$$

This can be demonstrated by developing the wedge products as in the case of the planes to yield

$$x_{c_1} \wedge x_{c_2} \wedge e_\infty = \mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2} \wedge e_\infty + (\mathbf{x}_{e_2} - \mathbf{x}_{e_1}) \wedge E, \tag{2.120}$$

from where it is evident that the expression $\mathbf{x}_{e_1} \wedge \mathbf{x}_{e_2}$ is a bivector representing the plane where the line is contained and $(\mathbf{x}_{e_2} - \mathbf{x}_{e_1})$ is the direction of the line.

2.5.3 Conformal transformations

A transformation of geometric figures is said to be *conformal* if it preserves the *shape*; that is, if it preserves the angles and hence the shape of straight lines and circles. Any conformal transformation in $\mathbb{R}^n$ can be expressed as a composite of *inversions* in spheres and *reflections* in hyperplanes. In general, conformal transformations have the form

$$g(x_c) = Gx_c(G^*)^{-1} = \sigma x'_c, \tag{2.121}$$

where $x_c \in \mathbb{R}^{n+1,1}$, G is a versor and σ is a scalar. We will begin our discussion of conformal transformations with the inversion.

Inversions

The general form of a reflection about a vector is

$$s(x_c) = -sx_cs^{-1} = x_c - 2(s \cdot x_c)s^{-1} = \sigma x'_c, \tag{2.122}$$

where $sx + xs = 2(s \quad x)$ from the definition of the Clifford product between two vectors. We will analyze now what happens when s represents a sphere. Recall that the equation of a sphere of radius ρ centered at point c_c is the vector

$$s = c_c - \tfrac{1}{2}\rho^2 e_\infty. \tag{2.123}$$

If s represents the unit sphere centered at the origin, then s and s^{-1} reduce to $e_0 - \frac{1}{2}e_\infty$. Hence $-2(s \cdot x_c) = \mathbf{x}_e^2 - 1$ and Eq. 2.122 becomes

$$\sigma x'_c = (\mathbf{x}_e + \tfrac{1}{2}\mathbf{x}_e^2 + e_\infty) + (\mathbf{x}_e^2 - 1)(e_0 - \tfrac{1}{2}e_\infty) = \mathbf{x}_e^2(\mathbf{x}_e^{-1} + \tfrac{1}{2}\mathbf{x}_e^{-2}e_\infty + e_0), \tag{2.124}$$

which is the conformal mapping for $\mathbf{x}_e^{-1}$

To see how a general sphere inverts a point, we return to Eq. 2.123 to get

$$s \cdot x_c = c_c \quad x_c - \tfrac{1}{2}\rho^2 e_\infty \cdot x_c = -\tfrac{1}{2}[(\mathbf{x}_e - \mathbf{c}_e)^2 - \rho^2]. \tag{2.125}$$

Insertion into 2.122 and a little algebra gives

$$\sigma x'_c = \left(\frac{\mathbf{x}_e - \mathbf{c}_e}{\rho}\right)^2 [g(\mathbf{x_c}) + \frac{1}{2}g^2(\mathbf{x}_e)e_\infty + e_0], \tag{2.126}$$

where

$$g(\mathbf{x}_e) = \frac{\rho^2}{\mathbf{x}_e - \mathbf{c}_e} + \mathbf{c}_e = \frac{\rho^2(\mathbf{x}_e - \mathbf{c}_e)}{(\mathbf{x}_e - \mathbf{c}_e)^2} + \mathbf{c}_e \tag{2.127}$$

is the inversion in $\mathbb{R}^n$.

Reflections

A hyperplane with unit normal $\mathbf{n}$ and signed distance δ from the origin in $\mathbb{R}^n$ can be represented by the vector

$$s = \mathbf{n} + \delta e_\infty. \tag{2.128}$$

Inserting $s \cdot x_c = \mathbf{n} \cdot \mathbf{x}_e$ into 2.122, we find that

$$g(\mathbf{x}_e) = \mathbf{n}\mathbf{x}_e\hat{\mathbf{n}} + 2\delta\mathbf{n} = \mathbf{n}(\mathbf{x}_e - \delta\mathbf{n})\hat{\mathbf{n}} + \delta\mathbf{n}, \tag{2.129}$$

This expression is equivalent to a reflection $\mathbf{n}\mathbf{x}_e\mathbf{n}^\dagger$ at the origin, translated by δ along the direction of $\mathbf{n}$. A point $\mathbf{c}_e$ is on the hyperplane when $\delta = \mathbf{n} \cdot \mathbf{c}_e$, in which case 2.128 can be written

$$s = \mathbf{n} + e_\infty \mathbf{n} \cdot \mathbf{c}_e. \tag{2.130}$$

Via Eq. 2.129, this vector represents the reflection in a hyperplane through point $\mathbf{c}_e$.

Translations

Translations can be modeled as two reflections about two parallel hyperplanes (see Fig. 2.13). Without loss of generality, we can assume that both planes have been normalized so that the magnitude of their normals is 1 and one of them passes through the origin. Then, from Eq. 2.128 we can represent the operator for translation (called translator) as

$$\begin{aligned} T_{\mathbf{a}} = \pi_1\pi_2 &= (\mathbf{n} + \delta e_\infty)(\mathbf{n} + 0e_\infty), \\ &= 1 + \tfrac{1}{2}\mathbf{a}e_\infty, \end{aligned} \tag{2.131}$$

where $\mathbf{a} = 2\delta\mathbf{n}$ and $\|\mathbf{n}\| = 1$. The translation distance is twice the separation between the hyperplanes.

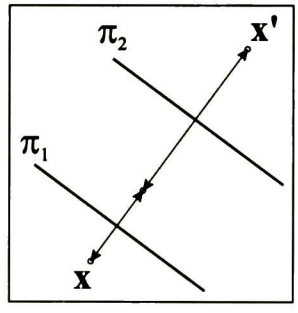


Figure 2.13: The translation as a reflection about two parallel planes.

Transversions

The transversion can be generated from two inversions and a translation. The transversor has the form

$$K_{\mathbf{a}} = e_+ T_{\mathbf{a}} e_+ = (\tfrac{1}{2}e_\infty - e_0)(1 + \tfrac{1}{2}\mathbf{a}e_\infty)(\tfrac{1}{2}e_\infty - e_0) = 1 + \mathbf{a}e_0. \tag{2.132}$$

The transversion generated by $K_{\mathbf{a}}$ can be put in various forms

$$g(\mathbf{x}_e) = \frac{\mathbf{x}_e - \mathbf{x}_e^2\mathbf{a}}{1 - 2\mathbf{a}\cdot\mathbf{x}_e + \mathbf{x}_e^2\mathbf{a}^2} = \mathbf{x}_e(1 - \mathbf{a}\mathbf{x}_e)^{-1} = (\mathbf{x}_e^{-1} - \mathbf{a})^{-1} \tag{2.133}$$

The last form can be written down directly as an inversion followed by a translation and another inversion.

Rotations

Rotations can be modeled by the composition of two reflections about two hyperplanes intersecting in a common point $\mathbf{c}_e$ (see Fig. 2.14) as

$$R = (\mathbf{a} + e_\infty \mathbf{a}\cdot\mathbf{c}_e)(\mathbf{b} + e_\infty \mathbf{b}\cdot\mathbf{c}) = \mathbf{a}\mathbf{b} + e_\infty \mathbf{c}_e\cdot(\mathbf{a}\wedge\mathbf{b}), \tag{2.134}$$

where $\mathbf{a}$ and $\mathbf{b}$ are unit normals. Rotations about the origin can also be written in exponential form as

$$R = e^{\frac{1}{2}\alpha B}, \tag{2.135}$$

where B is a unit bivector representing the axis of rotation and α is the magnitude of the angle of rotation.

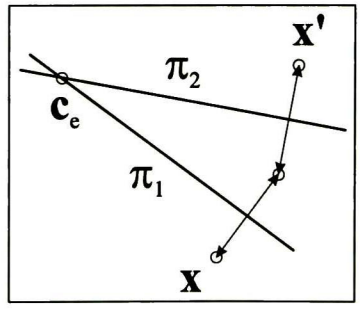


Figure 2.14: The rotation as a reflection about two intersecting planes.

Dilations

Dilations are the composite of two inversions centered at the origin. Using the unit sphere $s_1 = e_0 - \frac{1}{2}e_\infty$ and another sphere of arbitrary radius ρ, $s_2 = e_0 - \frac{1}{2}\rho^2 e_\infty$ as inversors, we get

$$(e_0 - \tfrac{1}{2}e_\infty)(e_0 - \tfrac{1}{2}\rho^2 e_\infty) = \tfrac{1}{2}(1 - E) + \tfrac{1}{2}(1 + E)\rho^2 \tag{2.136}$$

Normalizing to unity we have

$$D_\rho = \tfrac{1}{2}(1 + E)\rho + \tfrac{1}{2}(1 - E)\rho^{-1} = e^{E\phi}, \tag{2.137}$$

where $\phi = ln\rho$. To prove that this is indeed a dilation, we note that

$$D_\rho e_\infty D_\rho^{-1} = \rho^{-2} e, \quad \text{and similarly} \tag{2.138}$$
$$D_\rho e_0 D_\rho^{-1} = \rho^2 e_0. \tag{2.139}$$

Therefore,

$$D_\rho(\mathbf{x}_e + \tfrac{1}{2}\mathbf{x}_e^2 e_\infty + e_0)D_\rho^{-1} = \rho^2[\rho^{-2}\mathbf{x}_e + \tfrac{1}{2}(\rho^{-2}\mathbf{x}_e)^2 e_\infty + e_0], \tag{2.140}$$

which is the conformal mapping $g(\mathbf{x}_e) = \rho^{-2}\mathbf{x}_e$, producing the scaling as expected.

Involution

The bivector E (the Minkowski plane of $\mathbb{R}^{1,1}$) represents an operation which corresponds to the main involution whereas for an r-blade A_r, $\bar{A}_r = (-1)^r A_r$. In particular, for vectors $\bar{\mathbf{x}}_e = -\mathbf{x}_e$, which can be easily obtained by applying the versor E

$$E(\mathbf{x}_e + \tfrac{1}{2}\mathbf{x}_e^2 e_\infty + e_0)E = -(-\mathbf{x}_e + \tfrac{1}{2}\mathbf{x}_e^2 e_\infty + e_0). \tag{2.141}$$

This expression corresponds to the conformal mapping of $-\mathbf{x}_e$, thus confirming that the versor E represents the main involution for $\mathbb{R}^n$ This means that the main involution is a reflection via the Minkowski plane $\mathbb{R}^{1,1}$

2.6 The 3D Affine Plane

We have described the general conformal framework and its transformations. However, many of those operators were not employed in the present work. Indeed, since only rigid transformations were necessary to solve the problems we will present in the following chapter, we limited ourselves to the use of the *Affine Plane* which is a $n+1$ dimensional subspace of the Hyperplane of reference $\mathbb{P}(e_\infty, e_0)$.

We have chosen to work in the $\mathcal{G}_{4,1}$ algebra. We have shown that the particular choice of null vectors does not affect the properties of the conformal geometry. Thus, to further simplify calculations, we have chosen to define these vectors as

$$e = \tfrac{1}{2}(e_4 + e_5), \tag{2.142}$$
$$\bar{e} = e_4 - e_5. \tag{2.143}$$

With the following properties

$$e_i^2 = 1, \quad \text{for } i = 1,..,4, \tag{2.144}$$
$$e_5^2 = -1, \tag{2.145}$$
$$e^2 = \bar{e}^2 = 0. \tag{2.146}$$

Points in the affine plane $x \in \mathbb{R}^{4,1}$ are formed with

$$x_a = \mathbf{x}_e + e, \tag{2.147}$$

where $\mathbf{x}_e \in \mathbb{R}^3$. From this equation we note that e represents the origin (by setting $\mathbf{x}_e = 0$), similarly, $\bar{e}$ represents the point at infinity. The previous equation, allows the normalization equation 2.66 to be expressed as

$$\bar{e} \cdot x_a = 1. \tag{2.148}$$

In this framework, the conformal mapping equation is expressed with

$$x_c = \mathbf{x}_e - \tfrac{1}{2}\mathbf{x}_e^2\bar{e} + e = x_a - \tfrac{1}{2}\mathbf{x}_e^2\bar{e}. \tag{2.149}$$

However, most of the time we will be working on the affine plane exclusively. Therefore, we will be mainly concerned with a simplified version of the *rejection*. Noting that $E = e_\infty \wedge e_0 = \bar{e} \wedge e$, then Eq. 2.65 becomes

$$\begin{aligned} P_E^{\perp}(x_c) &= (x_c \wedge E)E = (x_c \wedge E) \quad E, \\ \mathbf{x}_e &= (x_c \wedge \bar{e}) \cdot e - e. \end{aligned} \tag{2.150}$$

Now, since the points in the affine plane have the form $x = \mathbf{x}_e + e$, we conclude that

$$x_a = (x_c \wedge \bar{e}) \cdot e, \tag{2.151}$$

is the mapping from the horosphere to the affine plane we needed.

2.6.1 Lines and Planes

We will not be using circles and spheres in this framework. The lines and planes are expressed in a similar fashion to their conformal counterparts as the *join* of 2 and 3 points, respectively

$$L = x_{a_1} \wedge x_{a_2}, \tag{2.152}$$

$$\Pi = x_{a_1} \wedge x_{a_2} \wedge x_{a_3}. \tag{2.153}$$

Note that unlike their conformal counterparts, the line is a *bivector* and the plane is a *trivector*. As seen earlier, these equations produce a moment-direction representation thus

$$L = e\mathbf{d} + B, \tag{2.154}$$

where $\mathbf{d}$ is a vector representing the direction of the line and B is a bivector representing the (orthogonal) moment of the line. Similarly we have that

$$\Pi = e\mathbf{n} + ke_{123}, \tag{2.155}$$

where $\mathbf{n}$ is the normal vector to the plane and k is a scalar representing the distance from the plane to the origin. Note that in any case, the direction and normal can be retrieved with $\mathbf{d} = \bar{e} \quad L$ and $\mathbf{n} = \bar{e} \cdot \Pi$, respectively.

In this framework, the intersection or *meet* has a simple expression too. Let $A = a_1 \wedge ... \wedge a_r$ and $B = b_1 \wedge ... \wedge b_s$, then the meet is defined as

$$A \cap B = A \quad (B \cdot \bar{I}_{A \cup B}), \tag{2.156}$$

where $\bar{I}_{A \cup B}$ is either $e_{12}\bar{e}$, $e_{23}\bar{e}$, $e_{31}\bar{e}$, or $e_{123}\bar{e}$, according to which basis vectors span the largest common space of A and B.

2.6.2 Rigid transformations in the 3D affine plane

Rotations and translations have the same form as previously stated. In our new definition, the rotor becomes

$$R = e^{\frac{\theta}{2}B} = \cos(\tfrac{\theta}{2}) + B\sin(\tfrac{\theta}{2}), \tag{2.157}$$

where θ is the angle of rotation and B is a unit bivector representing the axis of rotation.

The translator is defined by

$$T = e^{\frac{1}{2}\mathbf{t}\bar{e}} = 1 + \tfrac{1}{2}\mathbf{t}\bar{e}, \tag{2.158}$$

where $\mathbf{t}$ is a vector representing the translation. For the particular case of rotors and translators, the inverse is equal to the Clifford conjugate, which in turn is equal to the reversion. Thus, the transformation rule of a rotor and translator may be written as

$$X' = \widetilde{R}XR, \text{ and} \tag{2.159}$$
$$X' = \widetilde{T}XT. \tag{2.160}$$

Also, note that when a translator is applied to a geometric entity, the result will lie in the horosphere. Therefore, it is necessary to perform the *partial rejection* as defined by Eq. 2.151 followed by the normalization $\bar{e} \cdot x_a = 1$ if applicable.

Rotors and Translators can be combined multiplicatively to produce *Motors*. The motor M is defined, in general, as

$$M = TR, \tag{2.161}$$
$$= \left(1 + \tfrac{1}{2}\mathbf{t}\bar{e}\right)R, \tag{2.162}$$
$$= R + \tfrac{1}{2}\mathbf{t}\bar{e}R. \tag{2.163}$$

Therefore, it is rather simple to find the rotational part of a motor by simple inspection. The translational part can then be computed by right-multiplying the remainder of M with R^{-1}

Note in particular that the motor defined by

$$M = \widetilde{T}RT \tag{2.164}$$

represents a rotation about an arbitrary axis. There is a simple relationship between the axis of rotation and the form of a motor that produces the rotation about it. Let $L = m + e \wedge \mathbf{n}$ be a line such that $\|\mathbf{n}\| = 1$. Then, the motor M that rotates by α radians about the axis defined by L is given by

$$M = \cos(\tfrac{\alpha}{2}) + \sin(\tfrac{\alpha}{2})[\bar{e} \wedge (e_{123}m) - (e_{123}\mathbf{n})]. \tag{2.165}$$

The converse formula to extract the axis of rotation from a given motor is also simple. First, notice that from the previous equation, M can be written as

$$M = k_1 + k_2\bar{e} \wedge (e_{123}m) - k_2 e_{123}\mathbf{n}. \tag{2.166}$$

From this equation, it is easy to see that

$$m' = e_{123}(e \cdot (M - k_1)), \tag{2.167}$$
$$\mathbf{n}' = -e_{123} \cdot (M - k_1), \tag{2.168}$$

where m' and $\mathbf{n}'$ differ from the original m and $\mathbf{n}$ by a scale factor. Thus, the line representing the axis of rotation is given by $L = m' + e \wedge \mathbf{n}'$.

2.6.3 Directed distance

The directed distance is defined as the smallest vector joining two geometric objects (see Fig. 2.15). Due to this definition, this vector is always orthogonal to both entities. In general, the directed distance between two geometric objects $x = x_1 \wedge ... \wedge x_r$ and $y = y_1 \wedge ... \wedge y_s$ is defined by

$$\mathbf{d} = \frac{\bar{e} \cdot (x \wedge y)}{(\bar{e} \cdot x) \wedge (\bar{e} \cdot y)}. \tag{2.169}$$

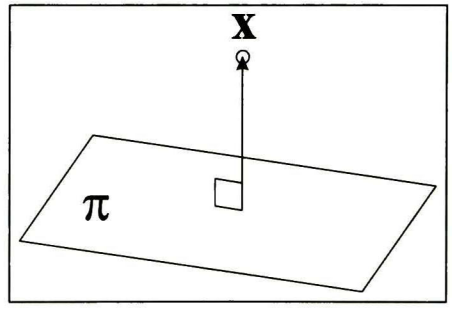


Figure 2.15: The directed distance between a plane and a point.

When x and y are parallel, the above formula fails since. To see why, note that $\bar{e}\ \ x$ is the direction of x and $\bar{e} \cdot y$ is the direction of y. When these vectors are parallel, their wedge product becomes zero. To solve this case, one merely needs to find one point in either x or y and compute the directed distance from this new point to the other object. A simple way to find a point on a geometric object A is with the formula

$$x = (\bar{e} \cdot A) \cdot A. \tag{2.170}$$

This equation returns the point on A which is closest to the origin. Note, however, that x will not be normalized, in general. Thus the constraint $\bar{e} \cdot x = 1$ must be enforced after x is computed.

Chapter 3

Reconstruction, Calibrated Case

We now proceed to describe a series of algorithms and techniques that enable any mobile robot to build a map for navigation purposes. In order to achieve this goal, several issues must be addressed. From these, we have chosen to tackle the problem of coordinate system calibration between multiple sensors. At the end of this chapter we present a reconstruction and a small navigation task, based on this algorithm. Throughout this chapter, we assume that the cameras have been calibrated and thus metric reconstruction can be easily achieved.

The problem of coordinate system calibration is basically a problem of motion estimation. We proceed now to describe two methods for motion estimation we have employed in this work.

3.1 Point-Based Rigid Motion Estimation

The most basic feature that can be detected is the point. The algorithm we describe here to perform rigid motion estimation based on point observations was originally presented in [38]. The algorithm proceeds as follows. Given the 3D coordinates of two sets of n corresponding points $\mathbf{X}_i$ and $\mathbf{X}'_i$ in $\mathcal{G}_3$, we wish to find the rotation R and translation $\mathbf{t}$ that minimizes

$$S = \sum_{i=1}^{n} \left[\mathbf{X}'_i - R(\mathbf{X}_i - \mathbf{t})\widetilde{R}\right]^2 \tag{3.1}$$

This therefore involves minimizing S with respect to R and $\mathbf{t}$. The differentiation with respect to $\mathbf{t}$ is straightforward

$$\partial_{\mathbf{t}} S = \sum_{i=1}^{n} \left[\mathbf{X}'_i - R(\mathbf{X}_i - \mathbf{t})\widetilde{R}\right] \partial_{\mathbf{t}}(R\mathbf{t}\widetilde{R}) = 0. \tag{3.2}$$

Which can be easily solved for $\mathbf{t}$ to give

$$\mathbf{t} = \frac{1}{n} \sum_{i=1}^{n} \left[\mathbf{X}_i - \widetilde{R}\mathbf{X}'_i R\right], \tag{3.3}$$

This can be rewritten as

$$\mathbf{t} = \overline{\mathbf{X}} - \widetilde{R}\overline{\mathbf{X}}' R, \tag{3.4}$$

where $\overline{\mathbf{X}}$ and $\overline{\mathbf{X}}'$ are the centroids of the data points in the two views. The differentiation with respect to the rotor R is much more involved, we direct the interested to [38] for details. The result of this differentiation produces

$$\sum_{i=1}^{n} \mathbf{X}'_i \wedge R(\mathbf{X}_i - \mathbf{t})\widetilde{R} = 0. \tag{3.5}$$

If we substitute the value of $\mathbf{t}$ in the previous equation we get

$$\sum_{i=1}^{n} \mathbf{X}'_i \wedge R(\mathbf{X}_i - \overline{\mathbf{X}} - \widetilde{R}\overline{\mathbf{X}}'R)\widetilde{R} = 0. \tag{3.6}$$

Noting that the term $\sum_{i=1}^{n} \mathbf{X}'_i \wedge \overline{\mathbf{X}}'$ vanishes, this reduces the previous equation to

$$\sum_{i=1}^{n} \mathbf{w}_i \wedge R\mathbf{u}_i\widetilde{R} = 0, \tag{3.7}$$

where

$$\begin{aligned} \mathbf{u}_i &= \mathbf{X}_i - \overline{\mathbf{X}}, \\ \mathbf{w}_i &= \mathbf{X}'_i. \end{aligned} \tag{3.8}$$

With these new values, we obtain a matrix $\mathtt{F}_{\alpha\beta}$ as

$$\mathtt{F}_{\alpha\beta} = \sum_{i=1}^{n} (e_\alpha \quad \mathbf{u}_i)(e_\beta \cdot \mathbf{w}_i). \tag{3.9}$$

Applying the SVD to matrix $\mathtt{F}_{\alpha\beta} = \mathtt{USV}^{\mathsf{T}}$ we can find the rotation matrix $\mathtt{R}$ as

$$\mathtt{R} = \mathtt{VU}^{\mathsf{T}} \tag{3.10}$$

Once the rotor is R is obtained from $\mathtt{R}$, the translator $\mathbf{t}$ can be computed from Eq. 3.4.

3.2 Line-Based Rigid Motion Estimation

Since lines are more robust to noise than points, it is important to describe how rigid motion estimation can be performed based on them. The algorithm we describe here was originally presented in [8] using motors or dual quaternions in the *Motor Algebra* $\mathcal{G}_{3,0,1}$, but we have extended it for the Affine Plane. In this framework, the motion of the line can be expressed as

$$L_B = ML_A\widetilde{M}, \tag{3.11}$$

where $\widetilde{X}$ is the Clifford conjugate of X and, in the case of motors, $M\widetilde{M} = 1$.

$$\begin{aligned} L_B &= b + eb', \\ L_A &= a + ea' = Rb\widetilde{R} + e(Rb\widetilde{R}' + Rb'\widetilde{R} + R'b\widetilde{R}). \end{aligned} \tag{3.12}$$

Separating the real and dual parts

$$a = Rb\widetilde{R}, \tag{3.13}$$
$$a' = Rb\widetilde{R}' + Rb'\widetilde{R} + R'b\widetilde{R}, \tag{3.14}$$

multiplying on the right by R and knowing that $\widetilde{R}R' + \widetilde{R}'R = 0$, we get

$$\begin{aligned} aR - Rb &= 0, \\ (a'R - Rb') + (aR' - R'b) &= 0. \end{aligned} \tag{3.15}$$

Which can be rewritten in matrix form as

$$\begin{bmatrix} a-b & [a+b]_\times & 0_{3\times1} & 0_{3\times3}, \\ a'-b' & [a'+b']_\times & a-b & [a+b]_\times \end{bmatrix} \begin{bmatrix} R \\ R' \end{bmatrix} \tag{3.16}$$

Taking $n \geq 2$ observations we stack the n line parameters in the left matrix, which is decomposed using SVD as $\mathtt{C} = \mathtt{U}\Sigma\mathtt{V}^\mathsf{T}$ Since the rank of the left matrix is 6 we use two vectors which span the null space for computing R and R'. For more details of the method refer to [8].

3.3 Body-Eye Calibration

Current approaches to compute the transformation between a coordinate system attached to a robotic hand and the camera on top of it work on the premise that the camera is always fixed at the same position with respect to the robotic arm. This is not the case for Pan-Tilt Units where the camera is constantly changing its position and orientation with respect to the robot's reference frame. To solve this problem, we proposed a novel algorithm which we describe next.

The robot-to-sensor relation can be seen as a series of joints $J_1, J_2, .., J_n$ where a rotation about joint J_i affects all joints $J_{i+1}, .., J_n$ and a measurement system U is rigidly attached to the last joint J_n. The problem can be stated as the computation of the transformations $M_1, M_2, .., M_{n-1}$ between the robot frame and the last joint and the transformation M_n between the last joint and the measurement device U, using only data gathered with U.

Note that this formulation is independent on the type of sensor U used; however, we will discuss how this calibration can be implemented to solve the robot-to-camera relationship and later on, how it was slightly modified to calibrate a laser sensor against the robot coordinate system.

Furthermore, we would like to solve this problem in a way that enables a real-time response when the spatial location of the joints varies. Therefore, we have divided our algorithm in two stages. The first stage computes the *screw axes* of the joints, and the second stage uses these axes to compute the final transformation between the coordinate systems.

3.3.1 Screw axes computation

To compute the axes of rotation, we use a motion estimator such as the one described in section 3.2. Each joint J_i is moved in turn while leaving the rest at their home position (see Figure 3.1). From the resulting motor M_i, the axis of rotation S_i can be extracted using Eqs. 2.167 and 2.168. For our particular robot, the sequence of motions is presented in Figure. The general procedure is presented in Algorithm 3.1.

1. For each joint J_i, $i = 1, ..n$ do:
2. Set all joints to their home position.
3. Rotate joint J_i by $-\alpha_1$ degrees.
4. Measure a set of 3D points X_j (or lines L_k) using the stereo camera.
5. Return joint J_i to its home position and rotate it by α_2 degrees.
6. Compute the corresponding set of points X'_j (or lines L'_j).
7. Compute the motor M_i such that $X'_j = M_i X_j$ (or $L'_j = M_i L_j$).
8. Compute the axis of rotation S_i using Eqs. 2.167 and 2.168 as

$$S_i = e_{123}(e \cdot (M_i - k_1)) + e \wedge [-e_{123} \quad (M_i - k_1)],$$

where k_1 is the scalar part of M_i.

Algorithm 3.1 *Computation of the axes of rotation.*

3.3.2 Calibration

Note that Algorithm 3.1 will produce a set of lines S_i in the camera's coordinate system. Once these axes are known, the transformation taking one point X_k measured in the camera's framework to the robot's coordinate system is easy to derive, provided that we know the angles α_i applied to each joint J_i. Basically, the algorithm undoes the implicit transformations applied on the camera's framework by first rotating about joint J_k and then translating the joint (and the framework, along with the rest of the joints) to the origin (see Figure 3.2). The full procedure is described in Algorithm 3.2.

Input: the number of joints n, a set of m points X_i,
... S_i and their angles of rotation α_i.

1. (Initialization)
   $k \leftarrow n$,
   $X'_i \leftarrow X_i$, for $i = 1, .., ...$
   $S'_i \leftarrow S_i$, for $i = 1, .., n$.
2. $P \leftarrow \texttt{nearest}(S'_k)$.
3. $T \leftarrow \texttt{makeTranslator}(-(P - e))$.
4. $M \leftarrow T\,\texttt{lineToMotor}(S'_k, -\alpha_k)$.
5. $X'_i \leftarrow M X'_i \widetilde{M}$, for $i = 1, .., m$.
6. $S'_i \leftarrow T S'_i \widetilde{T}$, for $i = 1, .., k - 1$.
7. $k \leftarrow k - 1$.
8. Repeat 2-7 until $k = 0$. The final corrected points are X'_i, $i = 1, .., m$.

Algorithm 3.2 *Computation of the transformation $X_i \mapsto X'_i$ for a system of n joints and m points where X_i is a point measured in the camera's framework and X'_i is the same point in the robot's coordinate system.*

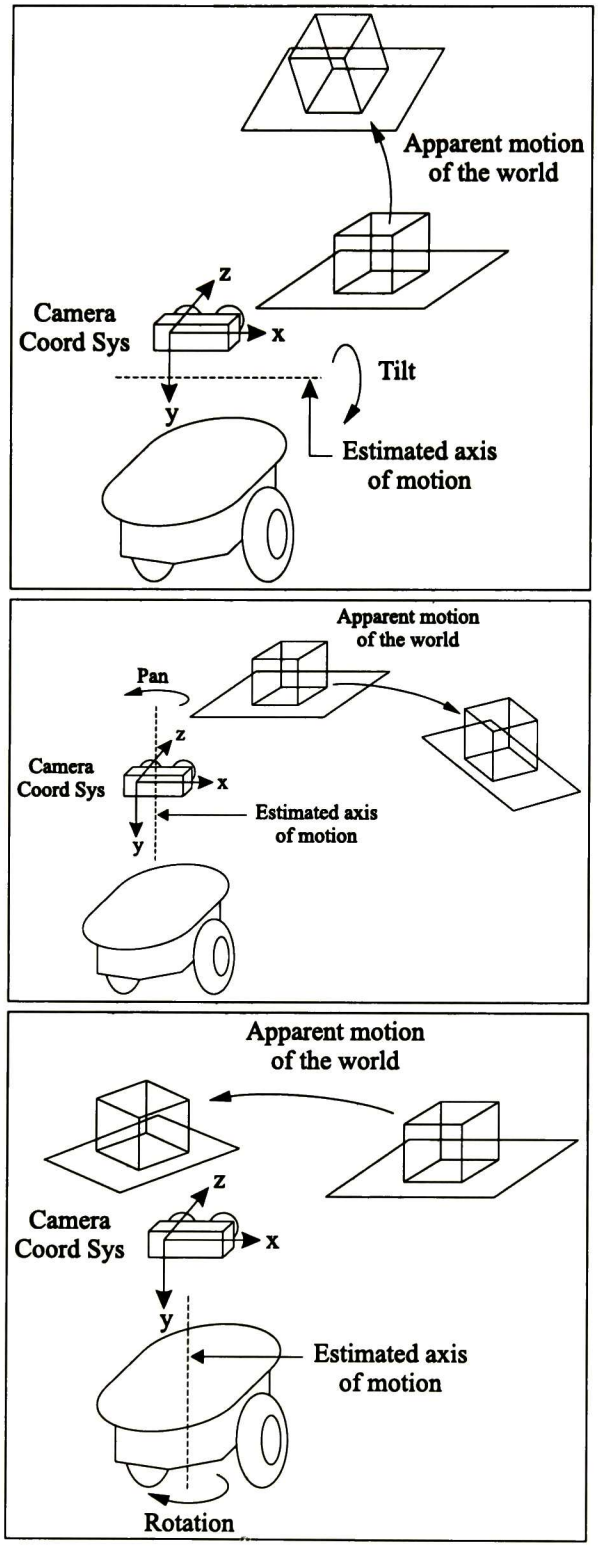


Figure 3.1: Estimation of the screw axes.

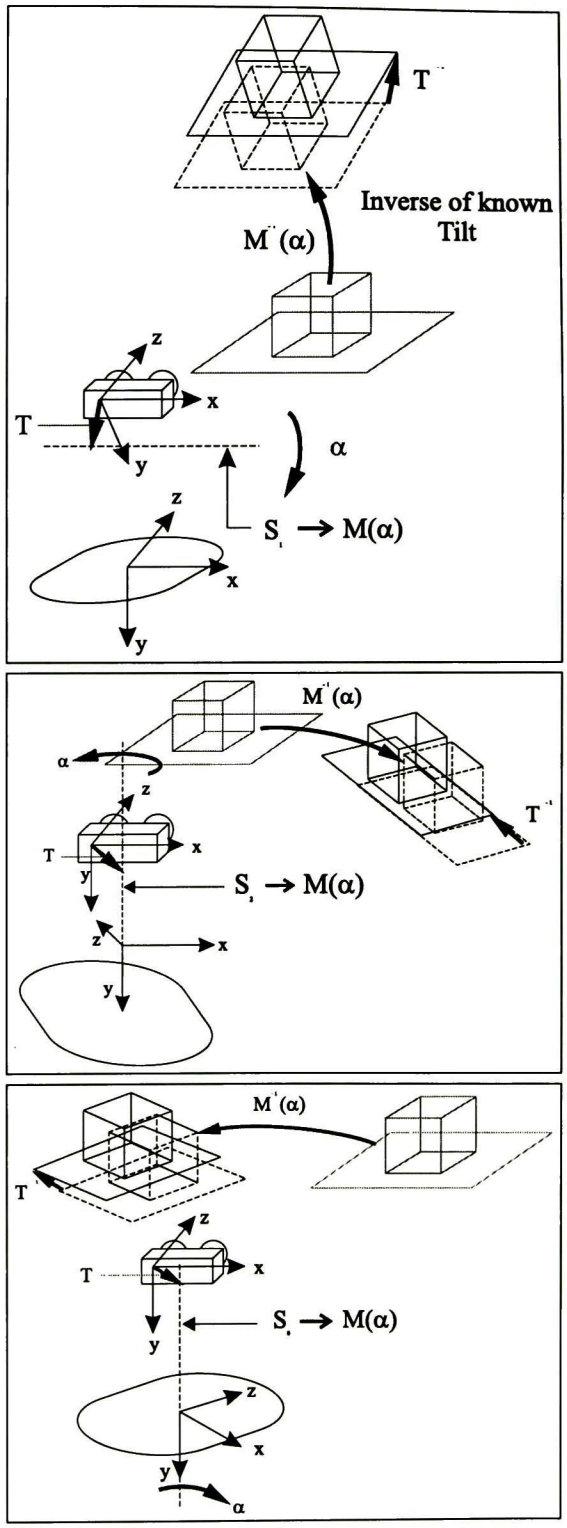


Figure 3.2: Correction of the rotation and relocation of the screw axes.

The functions used in Algorithm 3.2 are defined as follows

$$\texttt{nearest}(X) = \frac{(\bar{e} \cdot X) \cdot X}{\bar{e} \cdot [(\bar{e} \cdot X) \cdot X]}, \tag{3.17}$$

$$\texttt{makeTranslator}(\mathbf{t}) = 1 + \frac{\mathbf{t}}{2}\bar{e}, \tag{3.18}$$

$$\texttt{lineToMotor}(L, \alpha) = \cos(\tfrac{\alpha}{2}) + \sin(\tfrac{\alpha}{2})[\bar{e} \wedge (e_{123}m) - (e_{123}\mathbf{n})], \tag{3.19}$$

where $L = m + e \wedge \mathbf{n}$, and $\|\mathbf{n}\| = 1$. The function $\texttt{nearest}(X)$ returns the point on X which is nearest to the origin, $\texttt{makeTranslator}(\mathbf{t})$ returns a translator displacing by an amount $\mathbf{t}$, and $\texttt{lineToMotor}(L, \alpha)$ was previously explained in Eq. 2.165 and simply returns a motor that rotates α radians about the axis L.

3.4 Navigation

To test the calibration method, we conducted three experiments. For the first experiment, we applied a certain pan and tilt to our robot and reconstructed the scene (see Figure 3.3.a). The reconstruction is initially aligned with the camera coordinate system. Applying Algorithm 3.2, the reconstruction was transformed to match the robot's coordinate system (Figures 3.3.b-d). The comparison between the final reconstruction and the situation of the robot is presented in Figure 3.3.e.

The second test consisted in making a 3D map of the surroundings of the robot by superimposing together multiple reconstructions obtained at different pan and tilt angles. For each pan and tilt position, a reconstruction was obtained and transformed to the robot's reference coordinate system. The robot was left unmoved throughout the process. Some of the images used for the reconstruction can be seen in the top row of Figure 3.4, and the resulting reconstruction can be seen in the bottom row.

For the last experiment, a navigation and grasping task was tested. First, the laser system of our robot was calibrated according to Algorithm 3.1 (note that in this case, the only axis of rotation is the robot itself). Then, an object was placed on a box and the robot's pan-tilt unit was rotated to look at it. The object's coordinates were transformed into the robot's reference frame according to Algorithm 3.2 and then further correlated with the laser system. Once the coordinates were located in the laser's frame, the robot automatically navigated to place itself in position to grab the object. In order to guarantee a reliable navigation, a very simple algorithm for the control of the robot's position was implemented. This algorithm was based upon the line measurements obtained with the laser and the motion estimation algorithm of Section 3.2. Some of the images in the video sequence can be seen in Figure 3.5.

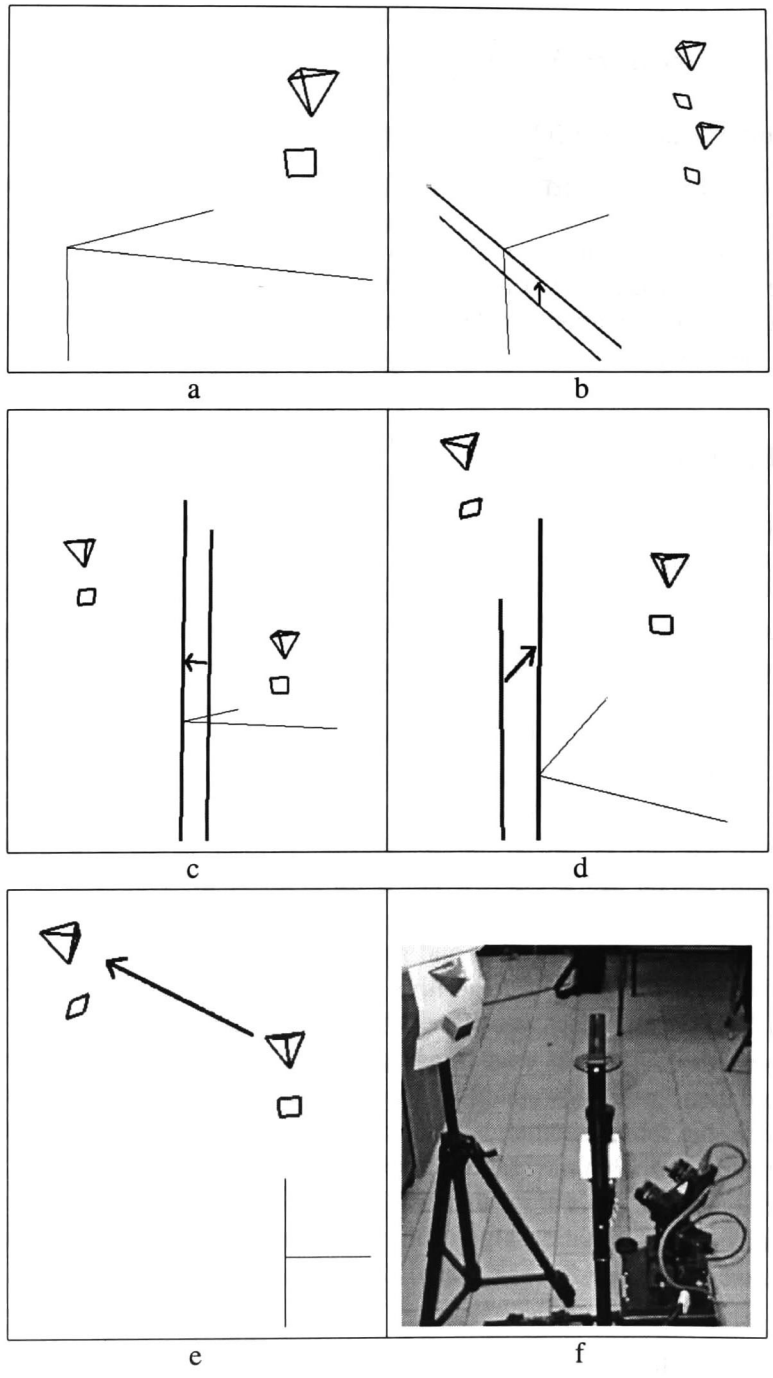


Figure 3.3: a) Reconstruction without calibration. b-d) Relocation of the screws according to Algorithm 3.2. e) Comparison of the final reconstruction with the real view

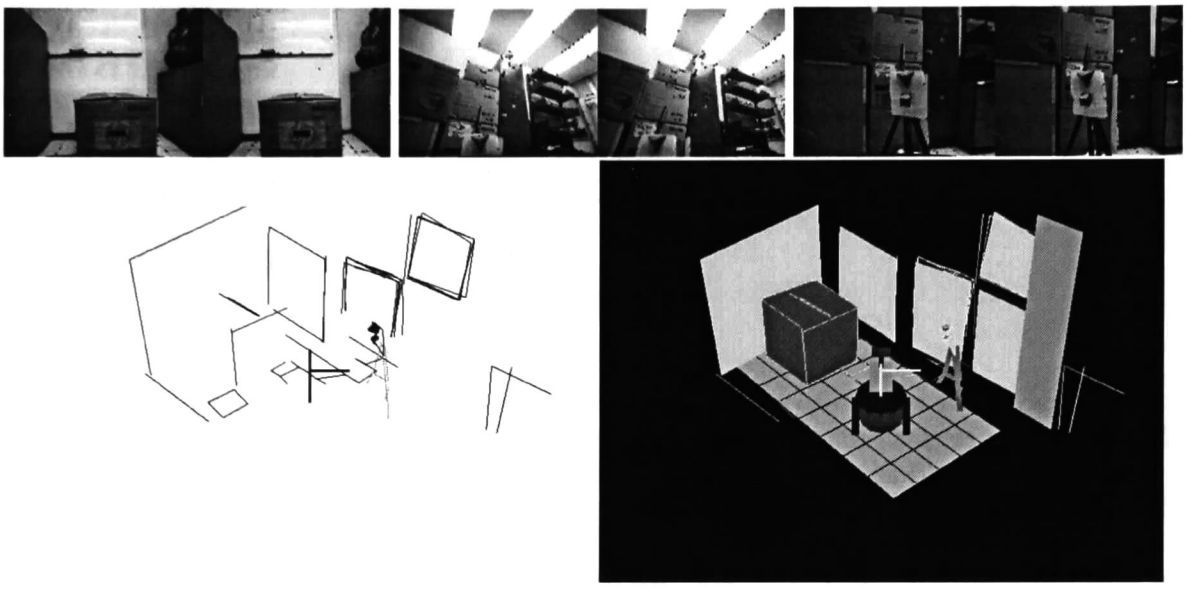


Figure 3.4: Top: some of the stereo pairs used in the reconstruction. Bottom: Extracted 3D data (left) and texture-mapped reconstruction (right).

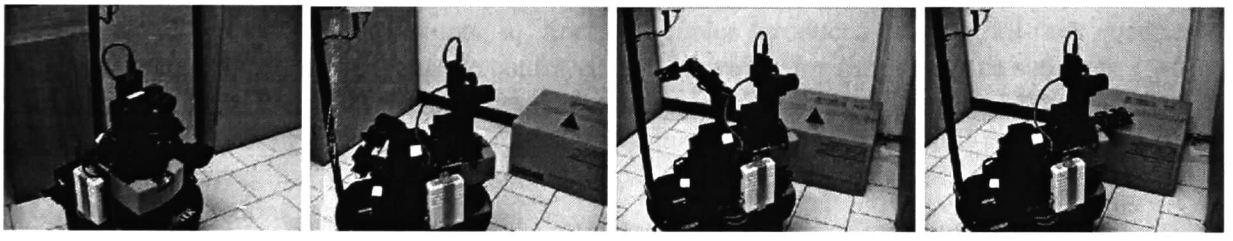


Figure 3.5: Full calibration: after the laser and the binocular systems are calibrated, the robot can navigate and reach objects in 3D accurately.

Chapter 4

Projective Reconstruction Using N Uncalibrated Views

In the third chapter, Geometric Algebra was used to solve sensor calibration and navigation tasks for a robot. However, all this work is based on the assumption that the cameras were calibrated, thus achieving a metric reconstruction of the space. When this condition is not met, it becomes increasingly difficult to employ Geometric Algebra since the camera model (in particular, the calibration matrix K) does not have a simple form in this framework. Therefore, in the case of uncalibrated views we resort to the use of classical Projective Geometry. This is because a treatment of this topic purely using the Clifford Geometric Algebra framework is a matter of future research. This will require designing a particular geometric algebra equipped with the mathematical tools to handle affine and projective transformations.

In this chapter, we present a Bundle Adjustment-based method to deal with the case of n uncalibrated views to produce a projective reconstruction of points, lines, quadrics, plane conics and degenerate quadrics in contrast to current research based solely on the reconstruction of points or liens. However, it is well known that this method requires a good initialization in order to converge. Here, we present a nice introduction to these initialization methods.

In the following sections, we will assume that $n_v \geq 3$ views are available. Each of these views may show projections of points, lines, quadrics, plane conics and degenerate quadrics. The correspondences between these entities are assumed to be known. No image calibration is needed, and, for our method, the features are not required to be visible in all views. Thus for each view $j = 1, ..., n_v$ we will have n_{x_j} points, n_{l_j} lines, n_{q_j} conics (produced by both full-rank quadrics and plane conics) and $n_{q_{d_j}}$ degenerate conics visible. However, for simplicity, the subindex j will be omitted. Note that full-rank quadrics and plane conics are counted together but degenerate quadrics are not. This is because the projection equations are the same for full-rank quadrics and plane conics but different for degenerate quadrics. This will be explained in full detail in the following subsections.

4.1 Initialization

4.1.1 Reconstruction of cameras and points

In our implementation the initialization is provided by first computing the fundamental matrix between views one and two. The labeling of the views is chosen so that the first two views have

the widest baseline possible to provide a more robust initialization. We use the normalized 8-point algorithm for the computation of F (see [27] or Section 2.2.1). Please note that this algorithm is not optimal, and in fact, it can be used as an initialization for the Gold Standard algorithm where a more meaningful geometric error is minimized and a better F is computed. However, this is only the initialization for a subsequent Bundle Adjustment stage where the refinement will be made. Thus we only need a "quick and rough" solution at this stage.

Computing F, P and P′ from point correspondences an initial reconstruction of all the points can be performed by triangulation (see [27] for details). From the projection equations

$$\lambda \mathbf{x} = \mathtt{P}\mathbf{X} \tag{4.1}$$

$$\lambda' \mathbf{x}' = \mathtt{P}'\mathbf{X}'. \tag{4.2}$$

In order to eliminate the scale factors λ and λ' we use the product $\mathbf{x} \times \mathbf{x} = \mathbf{x} \times (\mathtt{P}\mathbf{X}) = \mathbf{0}$, which can be written as

$$\begin{bmatrix} x(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{1\mathsf{T}}\mathbf{X}) \\ y(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{2\mathsf{T}}\mathbf{X}) \\ x(\mathbf{p}^{2\mathsf{T}}\mathbf{X}) - y(\mathbf{p}^{1\mathsf{T}}\mathbf{X}) \end{bmatrix} = \mathbf{0}, \tag{4.3}$$

where $\mathbf{p}^{i\mathsf{T}}$ is the i^{th} row of P. Combining the expression above with the corresponding equations for $\mathbf{x}' \times (\mathtt{P}'\mathbf{X}')$ yields

$$\begin{bmatrix} x\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{1\mathsf{T}} \\ y\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{2\mathsf{T}} \\ x'\mathbf{p}'^{3\mathsf{T}} - \mathbf{p}'^{1\mathsf{T}} \\ y'\mathbf{p}'^{3\mathsf{T}} - \mathbf{p}'^{2\mathsf{T}} \end{bmatrix} \mathbf{X} = \mathtt{Z}\mathbf{X} = \mathbf{0}. \tag{4.4}$$

Again, $\mathbf{X}$ can be obtained by extracting the null vector of Z via SVD.

After all the points have been reconstructed, we might further improve the first two cameras and compute *all the other cameras* by using Eq. 2.2 (notice that both $\mathbf{x}$ and $\mathbf{X}$ are known, thus in principle, P and the $\lambda's$ are the only unknowns and could therefore be estimated given enough points). However, that equation holds only *up to a scale factor*, thus we use the following formula instead

$$\mathbf{x}_{i,j} \times \mathtt{P}_j \mathbf{X}_i = \mathbf{0}, \tag{4.5}$$

thereby eliminating the scale factors and producing a strict equality. From this point on, we will use a more general notation where $\mathtt{P}_j$ represents the j^{th} camera, $\mathbf{X}_i$ represents the i^{th} 3D point and $\mathbf{x}_{i,j}$ is the image of the i^{th} point on the j^{th} camera. Thus, i ranges from $1 \ldots n_x$ and j ranges from $1 \ldots n_v$. A similar notation will be used with lines and quadrics in the following sections. Note that the preceding equation is linear in the entries of the $\mathtt{P}_j$'s. So we can factor out the camera coefficients to produce an equation of the form

$$\mathbf{x}_{i,j} \times \mathtt{P}_j \mathbf{X}_i = \mathtt{Z}_{p_{i,j}} \mathbf{p}_j = \mathbf{0}, \tag{4.6}$$

where

$$\mathtt{Z}_{p_{i,j}} = \begin{bmatrix} \mathbf{0}_{4\times1}^{\mathsf{T}} & -z_{i,j}\mathbf{X}_i^{\mathsf{T}} & y_{i,j}\mathbf{X}_i^{\mathsf{T}} \\ z_{i,j}\mathbf{X}_i^{\mathsf{T}} & \mathbf{0}_{4\times1}^{\mathsf{T}} & -x_{i,j}\mathbf{X}_i^{\mathsf{T}} \\ -y_{i,j}\mathbf{X}_i^{\mathsf{T}} & x_{i,j}\mathbf{X}_i^{\mathsf{T}} & \mathbf{0}_{4\times1}^{\mathsf{T}} \end{bmatrix} \tag{4.7}$$

$$\mathbf{p}_j = [p_{j,1} \dots p_{j,12}]^\mathsf{T}, \tag{4.8}$$

and

$$\mathrm{P}_j = \begin{bmatrix} p_{j,1} & p_{j,2} & p_{j,3} & p_{j,4} \\ p_{j,5} & p_{j,6} & p_{j,7} & p_{j,8} \\ p_{j,9} & p_{j,10} & p_{j,11} & p_{j,12} \end{bmatrix}, \tag{4.9}$$

being $\mathbf{x}_{i,j} = [x_{i,j}\ y_{i,j}\ z_{i,j}]^\mathsf{T}$ By stacking together all the $\mathrm{Z}_{p_{i,j}}$ for $i = 1 \dots n_x$ we can compute P_j by taking the null vector of the stacked matrices. Furthermore, if these matrices are stacked for all the views we can build a system of the form

$$\mathrm{Z}_p\mathbf{p} = \begin{bmatrix} \mathrm{Z}_{p_{1,1}} & \mathbf{0} & \dots & \dots & \mathbf{0} \\ \vdots & \vdots & & & \vdots \\ \mathrm{Z}_{p_{n_x,1}} & \mathbf{0} & \dots & \dots & \mathbf{0} \\ \mathbf{0} & \mathrm{Z}_{p_{1,2}} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & & & \vdots \\ \mathbf{0} & \mathrm{Z}_{p_{n_x,2}} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \dots & \dots & \mathbf{0} & \mathrm{Z}_{p_{1,n_v}} \\ \vdots & & & & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{0} & \mathrm{Z}_{p_{n_x,n_v}} \end{bmatrix} \mathbf{p} = \mathbf{0}, \tag{4.10}$$

where $\mathbf{p} = [\mathbf{p}_1^\mathsf{T} \dots \mathbf{p}_{n_v}^\mathsf{T}]^\mathsf{T}$ is a column vector with $12n_v$ entries containing all parameters for all the cameras, and Z_p is a $3n_xn_v \times 12n_v$ matrix (notice that this is just an extension for n-views of the DLT method presented in [27]).

Thus we can simultaneously estimate all cameras by extracting the null-space of Z_p and adding all the vectors that span this null-space. One way to do this is by taking the SVD of Z_p and taking the n_v singular vectors that correspond to the smallest singular values. Remarkably enough, in this way, we do not have to compute any n-view tensors and we do not have to rely on a priori knowledge like the projective depths or planarity in the scene.

Once all the cameras are computed, we back-project all the points *again* into 3-space and, according to Eq. 4.4, perform triangulation *considering all views simultaneously* (again, this is an extension to n-views of the DLT method, see [27]). In this case, Eq. 4.3 is stacked n_v times (once for each camera) to create a system of the form

$$\begin{bmatrix} x_i\mathbf{p}_1^{3\mathsf{T}} - \mathbf{p}_1^{1\mathsf{T}} \\ y_i\mathbf{p}_1^{3\mathsf{T}} - \mathbf{p}_1^{2\mathsf{T}} \\ \vdots \\ x_i\mathbf{p}_{n_v}^{3\mathsf{T}} - \mathbf{p}_{n_v}^{1\mathsf{T}} \\ y_i\mathbf{p}_{n_v}^{3\mathsf{T}} - \mathbf{p}_{n_v}^{2\mathsf{T}} \end{bmatrix} \mathbf{X}_i = \mathrm{Z}_X\mathbf{X}_i = \mathbf{0}. \tag{4.11}$$

We solve this problem using the SVD again, as described before. This step is not necessary but it improves slightly the reconstruction of the points $\mathbf{X}_i$. This procedure is easy to implement and it does not increase the computation time too much. To see an example of optimal n-view triangulation of points see [52].

4.1.2 Reconstruction of lines

Once the cameras and points are known, we proceed with the triangulation of lines. Once more, we estimate the lines by simultaneously using all cameras. Two methods for line reconstruction were tested. The first method is based on Eq. 2.9, by making the scalar factor explicit in

$$\lambda_{i,j}\mathbf{l}_{i,j} = \mathcal{P}_j\mathcal{L}_i, \tag{4.12}$$

which can be rewritten as

$$\mathcal{P}_j\mathcal{L}_i + \lambda_{i,j}\mathbf{l}_{i,j} = \mathbf{0}, \tag{4.13}$$

or in matrix form

$$\begin{bmatrix} \mathcal{P}_j & \mathbf{l}_{i,j} \end{bmatrix} \begin{bmatrix} \mathcal{L}_i \\ \lambda_{i,j} \end{bmatrix} = \mathbf{0}. \tag{4.14}$$

If the line is observed in n_v views, then a system of the form

$$\begin{bmatrix} \mathcal{P}_1 & \mathbf{l}_{i,1} & \mathbf{0} & \dots & \dots & \mathbf{0} \\ \mathcal{P}_2 & \mathbf{0} & \mathbf{l}_{i,2} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & & & & & \\ \mathcal{P}_{n_v} & \mathbf{0} & \dots & \dots & \mathbf{0} & \mathbf{l}_{i,n_v} \end{bmatrix} \begin{bmatrix} \mathcal{L}_i \\ \lambda_{i,1} \\ \lambda_{i,2} \\ \vdots \\ \lambda_{i,n_v} \end{bmatrix} = \mathtt{Z}_{\mathcal{P}\lambda}\mathbf{L} = \mathbf{0} \tag{4.15}$$

from where the line can be extracted by taking the first six entries of the null vector of $\mathtt{Z}_{\mathcal{P}\lambda}$.

The second method is described in [27], and consists of back-projecting the images of the line into planes according to the formula

$$\pi_{i,j} = \mathtt{P}_j^{\mathsf{T}}\mathbf{l}_{i,j}. \tag{4.16}$$

If we stack all the planes generated by the same line in all views we can form a $n_v \times 4$ matrix $\mathtt{Z}_L$

$$\mathtt{Z}_L = \begin{bmatrix} \pi_{i,1}^{\mathsf{T}} \\ \vdots \\ \pi_{i,n_v}^{\mathsf{T}} \end{bmatrix} \tag{4.17}$$

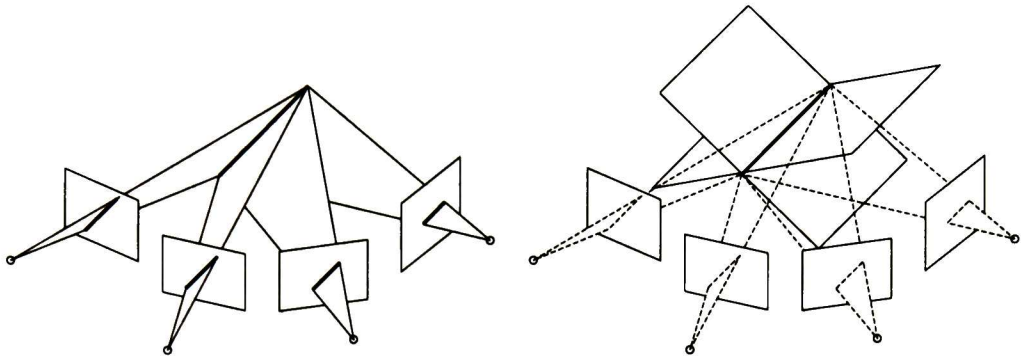
Now let $\mathtt{Z}_L = \mathtt{UDV}^{\mathsf{T}}$ be the singular value decomposition, and let $\mathbf{v}_1$, $\mathbf{v}_2$ be the two columns of $\mathtt{V}$ that correspond to the two largest singular values. $\mathbf{v}_1$, $\mathbf{v}_2$ span the best rank 2 approximation to $\mathtt{Z}_L$ and thus correspond to two planes which can be intersected (see Figure 4.1), according to Eq. 2.7, to find the dual line as

$$\mathbf{v}_1 \wedge \mathbf{v}_2 = \mathbf{v}_1\mathbf{v}_2^{\mathsf{T}} - \mathbf{v}_2\mathbf{v}_1^{\mathsf{T}} = \mathtt{L}^* \tag{4.18}$$

The actual line can be retrieved from the entries of $\mathtt{L}^*$ using Eq. 2.8:

$$[l_{12}\ l_{13}\ l_{14}\ l_{23}\ l_{42}\ l_{34}]^{\mathsf{T}} = [l_{34}^*\ l_{42}^*\ l_{23}^*\ l_{14}^*\ l_{13}^*\ l_{12}^*]^{\mathsf{T}} \tag{4.19}$$

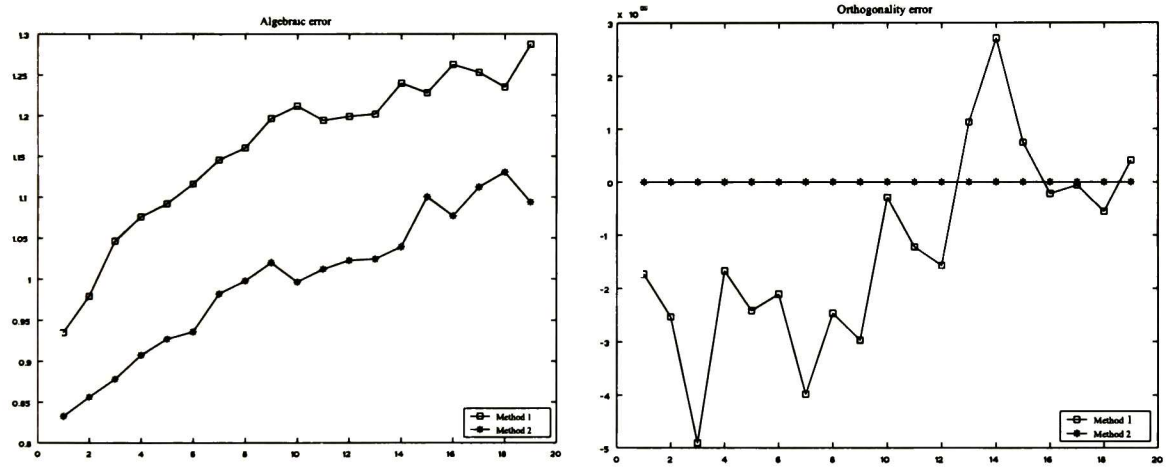
The test consisted in adding Gaussian, zero-mean, noise levels with a standard deviation ranging from 0.005 to 0.1 in steps of 0.005 (thus making 20 noise levels) to each component of the normalized images $\mathbf{l}_j$ (with $\|\mathbf{l}_j\| = 1$) of a randomly chosen line $\mathcal{L}$ (with $\|\mathcal{L}\| = 1$). For each noise



a b

Figure 4.1: a) Back-projection of a line from multiple views. b) Equivalent computation of the line as the intersection of two planes

level, the 3D line was reconstructed 100 times using both methods yielding two lines $\mathcal{L}_1$ and $\mathcal{L}_2$, with $\|\mathcal{L}_1\| = \|\mathcal{L}_2\| = 1$, and the errors were averaged. Two error measures were taken, the first being the algebraic error $\|\mathcal{L} - \mathcal{L}_1\|$ and $\|\mathcal{L} - \mathcal{L}_2\|$, and the second being the internal-consistency check of Eq. 2.5. This procedure was repeated for 100 random lines $\mathcal{L}$ and the results were averaged for each noise level across all lines. The results are shown in Figure 4.2.



a b

Figure 4.2: a) Algebraic error of both reconstruction algorithms at different noise levels. b) Internal orthogonality error

From Figure 4.2.a it can be seen that, apparently, the algebraic error is lower in the second method; and most importantly, the internal orthogonality constraint is better preserved too. Additionally, the implementation of the first method requires computing the SVD of a $3n_v \times 4 + n_v$ matrix, whereas the second method requires the computation of the SVD of a $n_v \times 4$ matrix, which is faster. Therefore, the latter was used for the initialization of our tests.

4.1.3 Reconstruction of quadrics

Whereas 3D points and lines are relatively easy to compute, the same does not hold for quadrics. To begin with, the triangulation of points and lines (using the second method described before) is invariant to scale factors. The scale factor was eliminated in the first case by taking the cross product (see Eq. 4.5), and by employing the *meet* operation (see Eq. 4.18) in the second. But there is no equivalent of the cross product or *meet* for conics. Hence, it becomes necessary to include an explicit scale factor in the triangulation of quadrics.

4.1.4 Quadric triangulation

Following the procedure suggested in [16], Eq. 2.16 becomes

$$\alpha \mathtt{C}^* = \mathtt{P}\mathtt{Q}^*\mathtt{P}^\mathsf{T} \tag{4.20}$$

Noting that this equation is linear in the entries of $\mathtt{Q}^*$ and $\mathtt{C}^*$, it can be rewritten as

$$\alpha \mathbf{c}^* = \mathtt{Z}_Q \mathbf{q}^* \tag{4.21}$$

with α being an arbitrary non-zero scale factor,

$$\mathbf{c}^* = [a\ b\ c\ d\ e\ f]^\mathsf{T}, \tag{4.22}$$

$$\mathbf{q}^* = [A\ B\ C\ D\ E\ F\ G\ H\ I\ J]^\mathsf{T} \tag{4.23}$$

being column vectors containing the 6 and 10 parameters of the dual conic and quadric respectively, and $\mathtt{Z}_Q =$

$$\begin{bmatrix}
p_1^2 & p_2^2 & p_3^2 & p_1p_2 & p_1p_3 & p_2p_3 & p_1p_4 & p_2p_4 & p_3p_4 & p_4^2 \\
2p_5p_1 & 2p_6p_2 & 2p_7p_3 & p_5p_2+p_6p_1 & p_5p_3+p_7p_1 & p_6p_3+p_7p_2 & p_5p_4+p_8p_1 & p_6p_4+p_8p_2 & p_7p_4+p_8p_3 & 2p_8p_4 \\
p_5^2 & p_6^2 & p_7^2 & p_5p_6 & p_5p_7 & p_6p_7 & p_5p_8 & p_6p_8 & p_7p_8 & p_8^2 \\
2p_9p_1 & 2p_{10}p_2 & 2p_{11}p_3 & p_{10}p_1+p_9p_2 & p_9p_3+p_{11}p_1 & p_{10}p_3+p_{11}p_2 & p_{12}p_1+p_9p_4 & p_{10}p_4+p_{12}p_2 & p_{11}p_4+p_{12}p_3 & 2p_{12}p_4 \\
2p_9p_5 & 2p_{10}p_6 & 2p_{11}p_7 & p_9p_6+p_{10}p_5 & p_9p_7+p_{11}p_5 & p_{11}p_6+p_{10}p_7 & p_9p_8+p_{12}p_5 & p_{10}p_8+p_{12}p_6 & p_{11}p_8+p_{12}p_7 & 2p_{12}p_8 \\
p_9^2 & p_{10}^2 & p_{11}^2 & p_9p_{10} & p_9p_{11} & p_{10}p_{11} & p_9p_{12} & p_{10}p_{12} & p_{11}p_{12} & p_{12}^2
\end{bmatrix} \tag{4.24}$$

where

$$\mathtt{P} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \tag{4.25}$$

Thus, Eq. 4.20 can be rewritten to

$$\mathtt{Z}_Q \mathbf{q}^* + \alpha \mathbf{c}^* = \mathbf{0}, \tag{4.26}$$

which finally enables the formulation of the matrix system

$$\begin{bmatrix} \mathtt{Z}_{Q1} & \mathbf{c}_1^* & \mathbf{0} & \dots & \mathbf{0} \\ \mathtt{Z}_{Q2} & \mathbf{0} & \mathbf{c}_2^* & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathtt{Z}_{Qn_v} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{c}_{n_v}^* \end{bmatrix} \begin{bmatrix} \mathbf{q}^* \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{n_v} \end{bmatrix} = \mathtt{Z}_Q \mathbf{v} = \mathbf{0}, \tag{4.27}$$

where $\mathtt{Z}_{Q_i}$ is formed by using the camera matrix $\mathtt{P}_i$ in Eq. 4.24, $\mathbf{c}_i^*$ is the dual conic in view i^{th}, $\alpha_1, ..., \alpha_{n_v}$ are independent arbitrary scale factors and $\mathbf{q}^*$ is the 10-vector that represents the quadric we are looking for. Once again, by getting the null vector of $\mathtt{Z}_Q$ via SVD, $\mathbf{q}^*$ (and the scale factors) can be obtained.

However, as shown in [16], it is necessary that the dual conics $\mathbf{c}_i^*$ are consistent with the epipolar geometry before Eq. 4.27 is used. Otherwise, the *topology* of the quadric may not be preserved. We have verified experimentally that even small differences of the order of one or two pixels may yield topologically inconsistent quadrics (e.g. a hyperboloid is reconstructed where a sphere was expected). Therefore, a preprocessing of the conic outlines is needed in order to make them epipolar-consistent, *before* Eq. 4.27 can be used. We will describe a method which guarantees the epipolar-consistency of conics in a subsequent section.

4.1.5 3D plane conic reconstruction

In contrast to the case of the full rank quadrics, a plane conic in space cannot be represented by a symmetric 4×4 matrix in the real space, but it can be represented in *dual space* as a dual cone $\mathtt{Q}_c^*$ (a degenerate quadric of rank 3). We will follow the reconstruction algorithm described in [15]. The interested reader will find the full details there.

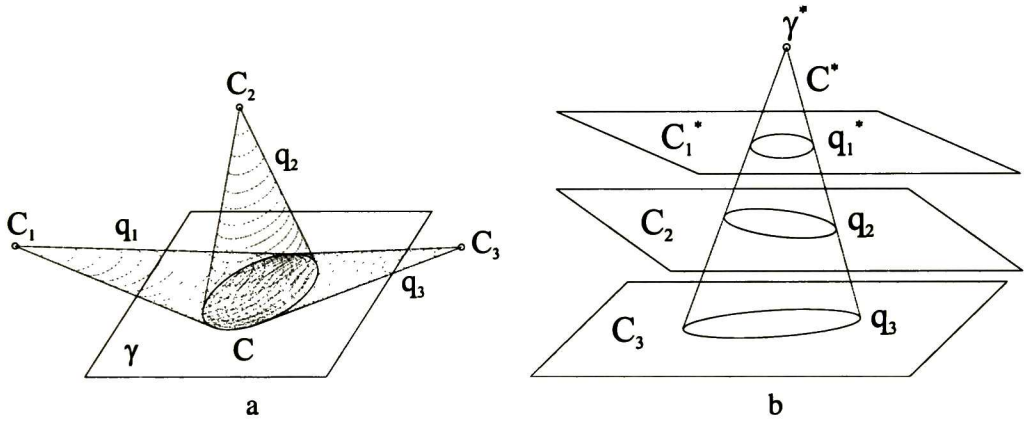


Figure 4.3: a) Real-space plane-conic and its dual representation b)

The first part of the reconstruction consists of computing $\mathtt{Q}_c^*$ by applying Eq. 4.27. We have found in practice that even in noise-free simulations, the resulting dual quadric may be of full rank. Therefore, after $\mathtt{Q}_c^*$ is found by triangulation, the rank-3 constraint is enforced by taking the SVD of $\mathtt{Q}_c^*$ and setting the smallest singular value to 0. That is, let $\mathtt{UDV}^\mathtt{T} = svd(\mathtt{Q}_c^*)$ with $\mathtt{D} = \mathrm{diag}(d_1, d_2, d_3, d_4)$ such that $d_i > d_{i+1}$. Let $\mathtt{D}' = \mathrm{diag}(d_1, d_2, d_3, 0)$, then the corrected dual cone is $\mathtt{Q}_c'^* = \mathtt{UD}'\mathtt{V}$.

The second part of the reconstruction consists of computing the real-space plane conic. Now, since $\mathtt{Q}_c^*$ is of rank-3, we cannot take the adjoint of $\mathtt{Q}_c^*$. Instead, we must find the plane where the conic lies and map the intersection of the quadric with this plane, to real-space. Following the development in [15], the plane γ where the conic lies is the dual to the apex of $\mathtt{Q}_c^*$ (see 4.3), thus

$$\gamma = \mathrm{null}(\mathtt{Q}_c^*). \tag{4.28}$$

Now, a coordinate system must be fixed to the plane γ. This is performed by finding a 4×3 transformation matrix $\mathtt{M}$ such that

$$\mathbf{X} = \mathtt{M}\mathbf{x} \tag{4.29}$$

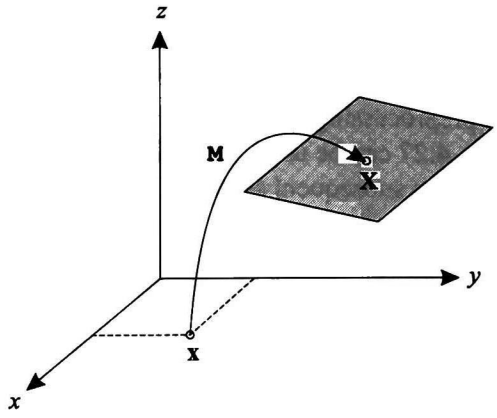


Figure 4.4: The mapping generated by the equation $\mathbf{X} = \mathtt{M}\mathbf{x}$.

where $\mathbf{X}$ is a 3D point and $\mathbf{x}$ is a 2D point lying on γ (see Figure 4.4). The computation of $\mathtt{M}$ has been discussed in [27]. In the coordinate system defined in γ, the conic is defined by $\mathbf{x}^\mathsf{T}\mathtt{C}\mathbf{x} = 0$, and its dual is defined by

$$\mathbf{l}^\mathsf{T}\mathtt{C}^*\mathbf{l} = 0. \tag{4.30}$$

The dual-space cone is defined by an envelope of tangent planes

$$\mathbf{\Pi}^\mathsf{T}\mathtt{Q}_c^*\mathbf{\Pi} = 0. \tag{4.31}$$

For each point $\mathbf{X}$ on $\mathbf{\Pi}$, $\mathbf{\Pi}^\mathsf{T}\mathbf{X} = 0$. The line of intersection $\mathbf{l}$, of γ and $\mathbf{\Pi}$ in the coordinate system defined by $\mathtt{M}$ follows by substitution (see Figure 4.5

$$\mathbf{\Pi}^\mathsf{T}\mathtt{M}\mathbf{x} = 0 \tag{4.32}$$

From $\mathbf{l}^\mathsf{T}\mathbf{x} = 0$, it is clear that $\mathbf{l} = \mathtt{M}^\mathsf{T}\mathbf{\Pi}$. Now since the conic envelope is defined both by Eq. 4.30 and Eq. 4.31, substituting for $\mathbf{l}$ in Eq. 4.30 yields

$$\mathbf{\Pi}^\mathsf{T}\mathtt{M}\mathtt{C}^*\mathtt{M}^\mathsf{T}\mathbf{\Pi} = 0 \tag{4.33}$$

and therefore

$$\mathtt{Q}_c^* = \mathtt{M}\mathtt{C}^*\mathtt{M}^\mathsf{T} \tag{4.34}$$

from where

$$\mathtt{C}^* = \mathtt{M}^{-1}\mathtt{Q}_c^*\mathtt{M}^{-\mathsf{T}} \tag{4.35}$$

defines the dual conic $\mathtt{C}^*$ in the coordinate system defined in γ. Since $\mathtt{M}$ is a 4×3 matrix we use the pseudo-inverse instead of $\mathtt{M}^{-1}$ in practice.

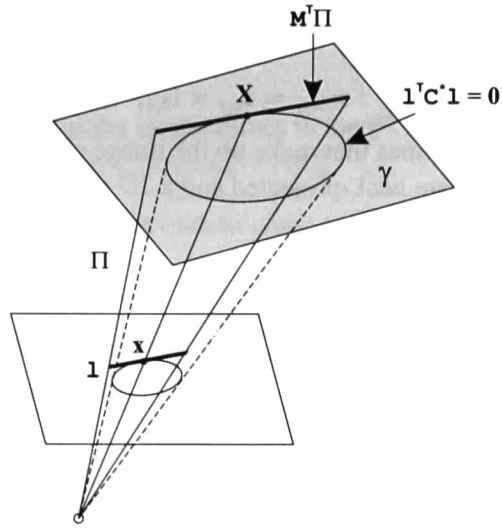


Figure 4.5: A plane conic and its projection on the image plane.

Once again, just like in the case of the full-rank quadrics, the outlines of the plane conic projected on all the images must be epipolar-consistent in order to preserve the topology of the conic. Fortunately, since the images of a plane conic are full-rank conics, the same method for outline correction can be used for full-rank quadrics and plane conics.

4.1.6 Degenerate quadric reconstruction

Degenerate quadrics cannot be triangulated using Eq. 4.27 since the adjoint of a rank-deficient matrix $\mathtt{Q}$ is undefined. Therefore a different method is used in this case (the following procedure is based on [15]).

The image of a degenerate (rank 2) quadric is a degenerate conic (two lines). The back-projection of these degenerate conics $\mathtt{C}_j$ produce pairs of planes $\mathbf{\Pi}_{1,j}$ and $\mathbf{\Pi}_{2,j}$ (see Figure 4.6). The back-projected planes are dual to points in the dual-space plane conic, and the support plane for this conic is the dual to the apex (singularity) of the degenerate quadric. The apex can be found by intersecting all the planes $\mathbf{\Pi}_{i,j}$. Therefore, the degenerate quadric is found by fitting a conic to the points dual to the back-projected planes and then transforming this plane conic into the real-space quadric. We give the details of these computations next.

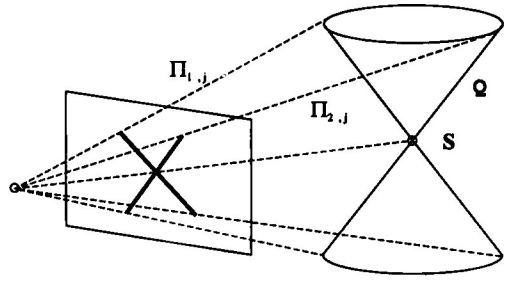


Figure 4.6: A degenerate quadric and its projection on the image plane.

First the singularity or apex of the degenerate quadric is computed by finding the intersections

$$\mathbf{x}_{int_j} = \mathbf{l}_{1,j} \times \mathbf{l}_{2,j}, \tag{4.36}$$

where $\mathbf{l}_{1,j}$ and $\mathbf{l}_{2,j}$ are the pair of lines that make up the image of the degenerate quadric in view j. The points of intersection $\mathbf{x}_{int_j}$ are back-projected and a 3D point $\mathbf{S}$ is triangulated using Eq. 4.11. Then, the transformation

$$\mathbf{X} = \mathtt{M}\mathbf{x} \tag{4.37}$$

is computed that maps 2D points $\mathbf{x}$ into 3D points $\mathbf{X}$ lying on the plane defined by $\mathbf{S}$, thus fixing a coordinate system. Then, the planes

$$\mathbf{\Pi}_{i,j} = \mathtt{P}_j^{\mathsf{T}} \mathbf{l}_{i,j} \tag{4.38}$$

are computed for $i = 1, 2$, and their equivalent dual-space points are computed with

$$\mathbf{x}_{dual_{i,j}} = \mathtt{M}^{-1} \mathbf{\Pi}_{i,j} \tag{4.39}$$

for $i = 1, 2$ (note that in practice, the pseudo-inverse is used instead of $\mathtt{M}^{-1}$). A conic $\mathtt{C}$ is fitted to the set of points $\mathbf{x}_{dual_{i,j}}$ and finally, the degenerate quadric is found by

$$\mathtt{Q} = \mathtt{M}\mathtt{C}\mathtt{M}^{\mathsf{T}} \tag{4.40}$$

Just as in the case of general quadrics and plane conics, this procedure is only valid for epipolar-consistent degenerate quadrics. Therefore it is necessary to correct the outlines of the quadric before attempting the reconstruction. We will proceed now to describe how outlines are corrected for the case of full-rank quadrics and plane conics, then the description of the algorithm for the outline correction in degenerate quadrics is presented.

4.1.7 Geometric conic outline correction

Before we describe our method, we shall make a review of the existing work on conic outline correction. In [16] Cross and Zisserman work with error-free camera matrices. In this paper, the outline of the conic $\mathtt{C}'$ on the second image is adjusted to be consistent with $\mathtt{C}$ the image of the conic in the first image. This correction is performed by geometric means. Then using $\mathtt{C}$ and $\mathtt{C}'$ as a starting point, a minimization is performed over the parameters of the reconstructed quadric $\mathtt{Q}$. The cost function being minimized is the sum of squared perpendicular distances from the measured points to the conic outlines. In this paper the authors do not discuss in detail how the initial correction is performed for 3 views and assumes cameras are known without error.

In [15], Cross describes a similar method for outline correction. Here, the minimization equation is

$$\min_{\mathbf{v}} \sum_{i=0}^{m} \sum_{j} D(\mathtt{C}_i'(\mathbf{v}, \mathbf{s}_{i,j}), \mathbf{s}_{i,j}), \tag{4.41}$$

where $\mathbf{v}$ represents the parameters (the angles) of the epipolar tangent lines, $D(\mathtt{C}, \mathbf{s})$ is the orthogonal distance between conic $\mathtt{C}$ and point $\mathbf{s}$. The feature $\mathbf{s}_{i,j}$ is the j^{th} point measured on the outline of image i and $\mathtt{C}_i'(\mathbf{v}, \mathbf{s}_{i,j})$ is a conic fitted first to the epipolar tangent lines $\mathbf{v}$ and then to the image

points $\mathbf{s}_{i,j}$. The computation of $\mathbf{C}_i'$ is given for the case of two and three views. For the case of three views, the following procedure is followed:

Given v and hence four epipolar tangent lines in the i^{th} image, fit a dual conic to these lines, giving a pencil of dual conics: $\mathbf{C}_i^* = \alpha_1 \mathbf{C}_1^* + \alpha_2 \mathbf{C}_2^*$

- Solve for α_1 and α_2 by finding the best-fit conic to the image data, $\mathbf{s}_{i,j}$:

$$\min_{\alpha_1,\alpha_2} \sum_j D(\mathbf{C}_i, \mathbf{s}_{i,j}) \tag{4.42}$$

(Levenberg-Marquardt is used)

Compute $\mathbf{C}_i = (\mathbf{C}_i^*)^*$.

No details are given in [15] for 4 or more views. For the case of two or three views, the problem of fitting a dual conic to 2 and 4 lines respectively is under-determined, which means that a family of dual conics is found that *exactly* passes through the required lines. Note, however that for 4 or more views the problem is *over-determined* (with 4 views, each conic must pass through 6 epipolar lines). This means that only one dual conic will be found which, in general, *will not pass exactly through any given line*, but rather will approach all of them (see Figure 4.7 for an example). This is a major flaw in the algorithm of Cross [15], since any following minimization will not ensure, in general, that the dual conic will pass *exactly* through all the lines. Note also, that with 4 or more views the parameterization used for the minimization in step 2 of the algorithm cannot be used anymore. The thesis does not mention how to cope with these problems. In addition to this problem, the camera matrices are, again, assumed to be known in advance without error.

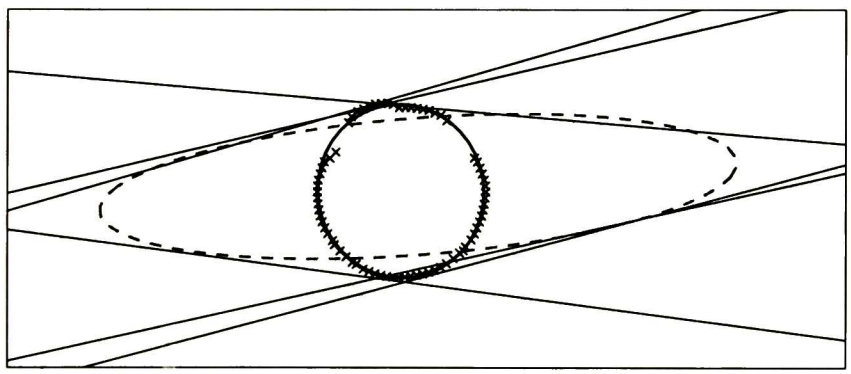


Figure 4.7: Conic fitted to six epipolar lines (dashed line) using the method described in [15] (before any minimization). This example comes from a real experiment with 4 cameras. The noisy measured points (crosses) are also shown with a conic fitted to them for reference (solid outline). Note the gross error between the points and the fitted conic (dashed line) and notice also that the conic approaches the lines but does not pass exactly through all of them.

We have taken the ideas just outlined and improved them to create a novel method for the correction of outlines in the presence of noisy camera matrices for 4 or more views. Our method requires the sets of measured points $\mathcal{M}_{c_i}, i = 1, .., n_v$ on the conic outlines, and at least 3 views. If

only 2 views are present, then the algorithm in [16] may be used instead and then the noisy camera matrices may be adjusted with Algorithm 4.2 (explained later). An initial, noisy approximation of the camera matrices $\mathtt{P}_i, i = 1, .., n_v$ is assumed to be known.

Before proceeding, we must note that we will be using a notation of the form $X_{i,j,k}$. This does not mean we are using tensors though. The first subindex stands for the view where things are projected to, the second indicates the view of origin where objects are projected from, and the last index merely ranges through all the objects of the same type.

The method presented here has two stages. In the first stage, the outlines in three views are corrected by fitting dual conics to five epipolar lines. Four lines are obtained in view i by mapping the two tangents from view j to view i:

$$\mathbf{l}_{mapped_{i,j,k}} = ([\mathbf{e}_{i,j}]_\times \mathtt{F}_{i,j}^{\mathsf{T}})^{\mathsf{T}} \mathbf{l}_{j,i,k}, \tag{4.43}$$

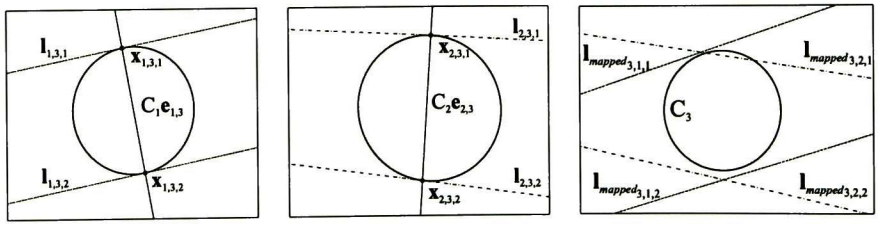
where $\mathbf{l}_{mapped_{i,j,k}}$ is the k^{th} tangent line mapped from view j to view i ($k = 1, 2$), $\mathbf{e}_{i,j}$ is the epipole produced by projecting the center of the j^{th} camera on the i^{th} view, $\mathtt{F}_{i,j}$ is the fundamental matrix between views i and j satisfying $\mathbf{x}_i^{\mathsf{T}} \mathtt{F}_{i,j} \mathbf{x}_j = 0$, and $\mathbf{l}_{j,i,k}$ is the k^{th} tangent line in view j obtained with

$$\mathbf{l}_{j,i,k} = \mathtt{C}_j \mathbf{x}_{j,i,k} \tag{4.44}$$

$$\mathbf{x}_{j,i,k} = (\mathtt{C}_j \mathbf{e}_{j,i}) \wedge \mathtt{C}_j, \tag{4.45}$$

$$\tag{4.46}$$

where $\mathtt{C}_j$ is the matrix of the conic outline in view j and $\mathbf{x}_{j,i,k}$ is the k^{th} intersection or *meet* of the polar $\mathtt{C}_j \mathbf{e}_{j,i}$ with $\mathtt{C}_j$ (see Figure 4.8). The intersection between a conic and a line is easy to derive and implies the solution of the quadratic equation that results from substituting the line equation into the conic equation and solving for x or y, hence $k = 1, 2$. $\mathtt{C}_j$ can be computed by fitting a conic to each set of points $\mathcal{M}_{c_j}$ (an easy algorithm to do this is presented later in this section, for more sophisticated algorithms see [62]).



a b c

Figure 4.8: a) and b) views 1 and 2, respectively with local tangent lines. c) Third view with mapped tangent lines

If the conic outlines and the camera matrices were error-free, then the following would hold:

$$\mathbf{l}_{mapped_{i,j,k}} \times \mathbf{l}_{tangent_{i,j,k}} = \mathbf{0}, \quad k = 1, 2, \tag{4.47}$$

with $\mathbf{l}_{tangent_{i,j,k}} = \mathbf{e}_{i,j} \times \mathbf{x}_{i,j,k}$ and $\mathbf{x}_{i,j,k}$ as defined by Eq. 4.45. Eq. 4.47 simply states that the local and the mapped tangents to the conic must coincide. This, however does not hold in the presence of noise. Note that this situation is analogous to the epipolar mismatch between two corresponding points in the presence of noise (see section 11.5, pg. 301 of [27]). In that case, the optimal solution

is found by varying the epipolar lines in such a way as to minimize the orthogonal distance to the measured points. In a similar fashion, we produce new epipolar lines lying halfway between the local and mapped tangent lines (though, no minimization is performed in this stage). Let

$$\mathbf{l}_{mid_{i,j,k}} = \mathbf{e}_{i,j} \times (\text{dir}(\mathbf{l}_{mapped_{i,j,k}}) + \text{dir}(\mathbf{l}_{tangent_{i,j,k}})), \tag{4.48}$$

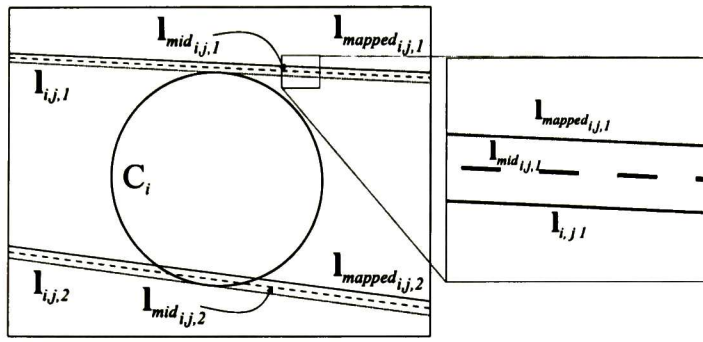


Figure 4.9: Relationship between $\mathbf{l}_{mid_{i,j,k}}$, $\mathbf{l}_{mapped_{i,j,k}}$ and $\mathbf{l}_{i,j,k}$

where dir(l) returns the normalized direction of the line l in a vector of the form $\mathbf{v} = [d_x \; d_y \; 0]^\mathsf{T}$ such that $\|\mathbf{v}\| = 1$, and $\text{dir}(\mathbf{l}_{mapped_{i,j,k}}) \;\; \text{dir}(\mathbf{l}_{tangent_{i,j,k}}) \geq 0$ (both vectors have the same sense). Then, the following equation holds (up to scale):

$$\mathbf{l}_{mid_{i,j,k}} = ([\mathbf{e}_{i,j}]_\times \mathbf{F}_{i,j}^\mathsf{T})^\mathsf{T} \mathbf{l}_{mid_{j,i,k}}. \tag{4.49}$$

Which means that the middle line between the mapped tangent and the local tangent in view j maps to the corresponding middle line in view i (see Figure 4.9). Hence the conics that have these lines as tangents are epipolar-consistent. Since each $\mathbf{l}_{mid_{i,j,k}}$ imposes one constraint in the equation

$$\mathbf{l}_{mid_{i,j,k}}^\mathsf{T} \mathbf{C}^* \mathbf{l}_{mid_{i,j,k}} = 0 \tag{4.50}$$

we need at least 5 such lines to compute $\mathbf{C}^*$. However, three cameras provide only four epipolar lines for each view. To find the fifth line we need to resort to the set of measured points $\mathcal{M}_{c_i}$ on the outline of the conic. The fifth line is found by using each measured point, getting its tangent line, fitting a conic to the set of five lines and choosing the point that yields the smallest geometric error between the fitted conic and the set of points (see Figure 4.10). The full procedure is summarized in Algorithm 4.1. In that procedure, the geometric distance between a point and a conic is used. The reader may consult an algorithm to compute such distance in [62]; however in that report, there are some slight errors in the equations, which were corrected. The revised version appears in Appendix B.

Before proceeding, we present a simple linear algorithm to fit a (dual) conic to a set of (lines) points. This algorithm is similar to the one presented in [27], but instead of assuming that the points have the normalized form $(x, y, 1)$ we derive our own algorithm from the homogeneous equation of the conic:

$$\mathbf{x}^\mathsf{T} \mathbf{C} \mathbf{x} = 0. \tag{4.51}$$

Which can be rewritten as

1. Fit a conic $\mathtt{C}_i$ to the set of measured points $\mathcal{M}_{c_i}$ for each view $i = 1, .., 3$.
2. Find the set of middle lines $\mathbf{l}_{mid_{i,j,k}}$.
3. For each view $i = 1, .., 3$ do:
4. Initialize $CurrentDistance \leftarrow \infty$, $\mathtt{C}_{i_{best}} \leftarrow null$.
5. Choose a point $\mathbf{x} \in \mathcal{M}_{c_i}$ not previously chosen.
6. Find the point $\mathbf{x}_c$ on $\mathtt{C}_i$ that is closest to $\mathbf{x}$.
7. Get the tangent at this point by $\mathbf{l}_t = \mathtt{C}_i\mathbf{x}_c$.
8. Using $\mathbf{l}_{mid_{i,j,k}}$ for all $j \neq i$, $k = 1, 2$ and $\mathbf{l}_t$, fit a dual conic $\mathtt{D}_i^*$ that passes through these five lines.
9. If $\sum d((\mathtt{D}_i^*)^*, \mathcal{M}_{c_i}) < CurrentDistance$ then
   - $CurrentDistance \leftarrow \sum d((\mathtt{D}_i^*)^*, \mathcal{M}_{c_i})$
   - $\mathtt{C}_{i_{best}} \leftarrow (\mathtt{D}_i^*)^*$
10. Repeat steps 5-9 until all points in $\mathcal{M}_{c_i}$ are chosen.
11. The new corrected conics are $\mathtt{C}_{i_{best}}$, $i = 1, .., 3$.

Algorithm 4.1 *Conic outline correction in 3 views.* $\sum d((\mathtt{D}_i^*)^*, \mathcal{M}_{c_i})$ *denotes the sum of orthogonal distances from the points* $\mathbf{x}_i \in \mathcal{M}_{c_i}$ *to the conic* $(\mathtt{D}_i^*)^*$

$$\begin{bmatrix} x^2 & xy & y^2 & xz & yz & z^2 \end{bmatrix} \mathbf{c} = 0, \tag{4.52}$$

where $\mathbf{c} = [a\ b\ c\ d\ e\ f]^\mathsf{T}$ is a vector containing the entries of the conic. Stacking n such equations we get

$$\begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1z_1 & y_1z_1 & z_1^2 \\ \vdots & & & & & \vdots \\ x_n^2 & x_ny_n & y_n^2 & x_nz_n & y_nz_n & z_n^2 \end{bmatrix} \mathbf{c} = \mathtt{Z}_c\mathbf{c} = \mathbf{0}. \tag{4.53}$$

We can obtain $\mathbf{c}$ by finding the null vector of $\mathtt{Z}_c$ via the SVD. Finally, to find the matrix $\mathtt{C}$ representing the conic we use

$$\mathtt{C} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \tag{4.54}$$

The advantage of this method is that the input points have the general form (x, y, z). This is specially useful since we can use points at infinity too (i.e. $z = 0$). The dual of a point at infinity is a line that passes through the origin. Therefore we are able to use lines passing through the origin in our formulation, whereas in [27], the algorithm assumed that points were normalized so that $z = 1$, thus precluding the use of these lines.

The second stage of the algorithm uses $\mathtt{C}_{best_i}$, $i = 1, .., 3$ to produce an initial reconstruction of the quadric $\mathtt{Q}_{est}$ using Eq. 4.27. Then $\mathtt{Q}_{est}$ is projected on the remaining $\mathtt{P}_i$, $i = 4, .., n_v$ to generate epipolar-consistent outlines in views 4 through n_v (see Figure 4.11), and a minimization is performed on $\mathtt{Q}_{est}$ such that the geometric distance from the outlines to the sets $\mathcal{M}_{c_i}$ is minimized. The procedure is described in detail in Algorithm 4.2.

The set of conic outlines $\mathtt{C}^*_{final_i}$ are epipolar-consistent. However, the precision of the reconstruction depends on the degree of noise present in the camera matrices $\mathtt{P}_i$. Thus, in order to im-

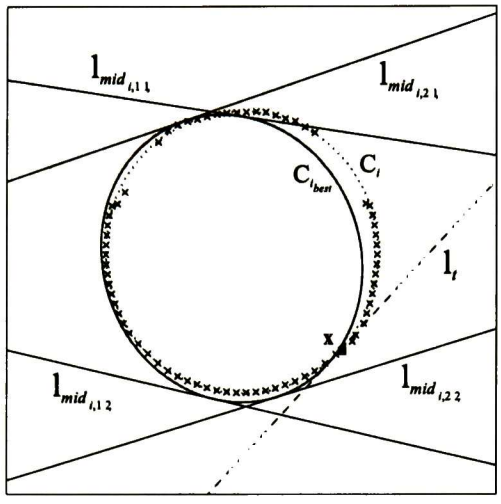


Figure 4.10: Illustration of Algorithm 4.1. The middle tangent lines $l_{mid_{i,j,k}}$ are shown in solid lines. The set $\mathcal{M}_{c_i}$ is shown with crosses. The conic C_i fitted to $\mathcal{M}_{c_i}$ is shown in a dotted line. The point $\mathbf{x} \in \mathcal{M}_{c_i}$ that yields the smallest error is shown in a square. The tangent at $\mathbf{x}$ is shown in a dashed line. The final epipolar-correct conic is shown in a solid line.

1. Compute an initial reconstruction of the quadric Q^*_{est} by Eq. 4.27, using the conics $C^*_{best_i}$, $i = 1, .., 3$ computed by Algorithm 4.1.
2. Compute $C^*_{est_i} = P_i Q^*_{est} P_i^T$ for $i = 1, .., n_v$. This step generates epipolar-consistent outlines in all views.
3. Perform a non-linear minimization of the expression

$$Q^*_{final} = \min_{Q^*_{est}} \sum_{i=1}^{n_v} d(C_{best^*_i}, \mathcal{M}_{c_i})$$

4. Compute the final conics with $C^*_{final_i} = P_i Q^*_{final} P_i^T$ for $i = 1, .., n_v$

Algorithm 4.2 *Quadric reconstruction and conic outline correction in n views.*

prove the reconstruction it is necessary to simultaneously refine the cameras and the quadrics. We will go one step further and show how simultaneous refinement of cameras, points, lines, quadrics, plane conics and degenerate quadrics is possible via Bundle Adjustment.

4.1.8 Degenerate quadric outline correction

Finally, before discussing Bundle Adjustment, we show how we perform outline correction in degenerate quadrics. To the best of our knowledge, no one else has mentioned a similar outline correction algorithm in the literature. The epipolar constraint for the case of degenerate quadrics can be formulated as

$$l^T_{epipolar_{j,k}} \mathbf{x}_{int_j} = 0, \tag{4.55}$$

where $l_{epipolar_{j,k}}$ is the epipolar line generated by $\mathbf{x}_{int_k}$ in view j with

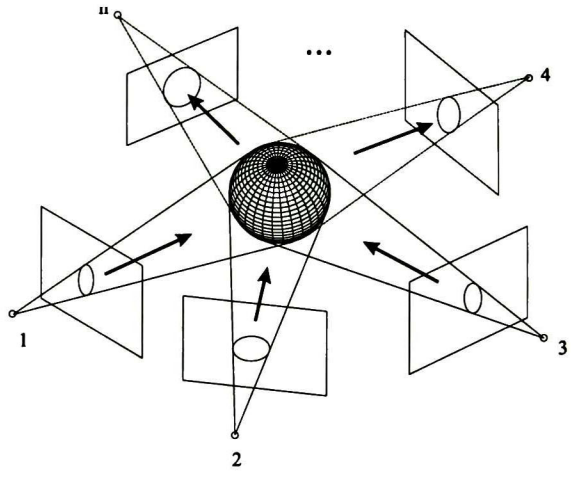


Figure 4.11: Illustration of Algorithm 4.2. Using the epipolar-consistent outlines in the first 3 views, the quadric is reconstructed. This quadric is topologically correct and thus it can be projected to the remaining $4, .., n_v$ views to generate epipolar-consistent conic outlines. Then, the parameters of this quadric are modified in order to minimize the error between the measured outline and the projected conics.

$$\mathbf{l}_{epipolar_{j,k}} = \mathrm{F}_{j,k}\mathbf{x}_{int_k}. \tag{4.56}$$

and $\mathrm{F}_{j,k}$ is the fundamental matrix between views j and k satisfying $\mathbf{x}_j^{\mathsf{T}}\mathrm{F}_{j,k}\mathbf{x}_k = 0$.

The epipolar constraint for degenerate quadrics states simply that the singularity or apex of the quadric must be epipolar-consistent. One way to achieve this is to triangulate the singularity S (see Figure 4.6) using Eq. 4.11 and then re-projecting S into all the views generating new points of intersection $\mathbf{x}'_{int_j}$

$$\mathbf{x}'_{int_j} = \mathrm{P}_j\mathbf{S}. \tag{4.57}$$

the existing lines $\mathbf{l}_{i,j}$, $i = 1, 2$ making up the images of the degenerate quadric in all views will not pass through the new intersection points $\mathbf{x}'_{int_j}$ if the images are not epipolar-consistent. But the lines are corrected easily according to

$$\mathbf{l}'_{i,j} = \mathrm{dir}(\mathbf{l}_{i,j}) \times \mathbf{x}'_{int_j}, \tag{4.58}$$

where $\mathrm{dir}(\mathbf{l}_{i,j})$ returns the direction of the line $\mathbf{l}_{i,j}$ in a vector of the form $\mathbf{v} = [d_x\ d_y\ 0]^{\mathsf{T}}$ The procedure is summarized in Algorithm 4.3.

Before proceeding, note that in all the methods mentioned above, there is not a requirement for all features (points, lines, and conics generated by quadrics, plane conics and degenerate quadrics) to be present in all views. If any given feature is absent, its equation is simply left out. This will be explained in detail in the following section.

4.2 Refinement of the Reconstruction

In the previous subsection we described how we make our initial reconstruction. This reconstruction is used as a seed for a Bundle Adjustment algorithm that refines the solution. Briefly, the

1. Find the points of intersection $\mathbf{x}_{int_j} = \mathbf{l}_{1,j} \times \mathbf{l}_{2,j}$ for $j = 1 \ldots n_v$, where $\mathbf{l}_{1,j}$ and $\mathbf{l}_{2,j}$ are the two lines making up the degenerate conic $\mathtt{C}_j$ in view j.
2. Using Eq. 4.11, triangulate the singularity point $\mathbf{S}$.
3. Project $\mathbf{S}$ on all the views, generating the set of points $\mathbf{x}'_{int_j} = \mathtt{P}_j\mathbf{S}$.
4. Compute $\mathbf{l}'_{i,j} = \mathrm{dir}(\mathbf{l}_{i,j}) \times \mathbf{x}'_{int_j}$ for $i = 1, 2$ and $j = 1 \ldots n_v$.
4. The new, epipolar-consistent, degenerate conics $\mathtt{C}_j$ are made up with the lines $\mathbf{l}'_{i,j}$, $i = 1, 2$.

Algorithm 4.3 *Degenerate conic outline correction in n views.*

classical algorithm consists in minimizing

$$\min_{\hat{\mathtt{P}}_i, \hat{\mathbf{X}}_j} \sum_{i,j} d(\hat{\mathtt{P}}_i \hat{\mathbf{X}}_j, \mathbf{x}_{i,j})^2, \tag{4.59}$$

where $\hat{\mathtt{P}}_i$ is set of estimated cameras, $\hat{\mathbf{X}}_j$ is the set of estimated 3D points, $\mathbf{x}_{i,j}$ is the set of measured 2D features and $d(\mathbf{x}, \mathbf{y})$ is the geometric distance between points $\mathbf{x}$ and $\mathbf{y}$. Usually, this minimization is performed with the Levenberg-Marquardt algorithm which we briefly describe now.

4.3 Levenberg-Marquardt

We follow the derivation given in [27] where a functional relation of the form

$$\mathbf{V} = f(\mathbf{P}), \tag{4.60}$$

is given where $\mathbf{V}$ is a *measurement vector* and $\mathbf{P}$ is a *parameter vector* in $\mathbb{R}^N$ and $\mathbb{R}^M$ respectively. An initial estimate of $\mathbf{P}$ is given and the goal is to find the vector $\hat{\mathbf{P}}$ satisfying $\mathbf{V} = f(\hat{\mathbf{P}}) + \epsilon$ for which $\|\epsilon\|$ is minimized. At each iteration, a new parameter vector is found by

$$\mathbf{P}_{i+1} = \mathbf{P}_i + \mathbf{\Delta}_i, \tag{4.61}$$

where $\mathbf{\Delta}_i$ is found by solving the augmented normal equations:

$$\mathrm{diag}(1 + \lambda)(\mathtt{J}_i^\mathsf{T} \mathtt{J}_i)\mathbf{\Delta}_i = \mathtt{J}_i^\mathsf{T} \epsilon_i, \tag{4.62}$$

where $\mathtt{J}_i$ is the Jacobian of f evaluated at $\hat{\mathbf{P}}_i$, ϵ_i is the error between $\mathbf{x}_{ij}$ and $f(\hat{\mathbf{P}}_i)$ and $\mathrm{diag}(k) = kI$ is a square matrix with k in its diagonals. The value of λ is typically initialized at $\lambda = 10^{-3}$. If the value of $\mathbf{\Delta}_i$ obtained in this way leads to a reduction in ϵ_i, then $\hat{\mathbf{P}}_{i+1}$ is accepted and $\lambda = \lambda/10$. Otherwise, if $\mathbf{\Delta}_i$ leads to an increment in the error, then $\lambda = 10\lambda$ and the augmented normal equations are solved again, this process is repeated until a $\mathbf{\Delta}_i$ is found that decreases error. This process constitutes an *iteration* of the Levenberg-Marquardt algorithm.

4.3.1 Refining the reconstruction of points, lines, quadrics, conics and degenerate quadrics

The classical Bundle Adjustment algorithm, as described in [27] was modified by stacking together the parameters of cameras ($\mathbf{P}_v$), points ($\mathbf{P}_x$), lines ($\mathbf{P}_l$), dual quadrics and plane conics ($\mathbf{P}_{q^*}$, note

that plane conics are parameterized as dual quadrics too and thus can be included in the same parameter vector) and degenerate quadrics ($\mathbf{P}_{q_d}$, note we are parameterizing degenerate quadrics in real-space, in contrast to full-rank quadrics and plane conics which are parameterized in dual-space) in a *parameter vector* $\mathbf{P}_{vxlq^*q_d} = \left[\mathbf{P}_v^\mathsf{T}\, \mathbf{P}_x^\mathsf{T}\, \mathbf{P}_l^\mathsf{T}\, \mathbf{P}_{q^*}^\mathsf{T}\, \mathbf{P}_{q_d}^\mathsf{T}\right]^\mathsf{T}$ This vector has $12n_v+4n_x+6n_l+10n_q+10n_{q_d}$ entries. Similarly, the measurement vector $\mathbf{m}_{xlcc_d}$ has $3n_vn_x+3n_vn_l+6n_vn_q+6n_vn_{q_d}$ entries. Using the projection equations 2.2, 2.9, 2.16 and 2.20, the Jacobian takes the form

$$\mathrm{J} = \begin{bmatrix}
\frac{\partial \mathbf{x}_{1,1}}{\partial \mathbf{P}_v} & \frac{\partial \mathbf{x}_{1,1}}{\partial \mathbf{P}_x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\vdots & & & & \vdots \\
\hline
\triangleright\ \frac{\partial \mathbf{x}_{i,j}}{\partial \mathbf{P}_v} & \frac{\partial \mathbf{x}_{i,j}}{\partial \mathbf{P}_x} & \mathbf{0} & \mathbf{0} & \mathbf{0}\ \triangleleft \\
\hline
\vdots & & & & \vdots \\
\frac{\partial \mathbf{x}_{n_x,n_v}}{\partial \mathbf{P}_v} & \frac{\partial \mathbf{x}_{n_x,n_v}}{\partial \mathbf{P}_x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\frac{\partial \mathbf{l}_{1,1}}{\partial \mathbf{P}_v} & \mathbf{0} & \frac{\partial \mathbf{l}_{1,1}}{\partial \mathbf{P}_l} & \mathbf{0} & \mathbf{0} \\
\vdots & & & & \vdots \\
\frac{\partial \mathbf{l}_{n_l,n_v}}{\partial \mathbf{P}_v} & \mathbf{0} & \frac{\partial \mathbf{l}_{n_l,n_v}}{\partial \mathbf{P}_l} & \mathbf{0} & \mathbf{0} \\
\frac{\partial \mathbf{c}^*_{1,1}}{\partial \mathbf{P}_v} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{c}^*_{1,1}}{\partial \mathbf{P}_{q^*}} & \mathbf{0} \\
\vdots & & & & \vdots \\
\frac{\partial \mathbf{c}^*_{n_q,n_v}}{\partial \mathbf{P}_v} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{c}^*_{n_q,n_v}}{\partial \mathbf{P}_{q^*}} & \mathbf{0} \\
\frac{\partial \mathbf{c}_{d_{1,1}}}{\partial \mathbf{P}_v} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{c}_{d_{1,1}}}{\partial \mathbf{P}_{q_d}} \\
\vdots & & & & \vdots \\
\frac{\partial \mathbf{c}_{d_{n_{q_d},n_v}}}{\partial \mathbf{P}_v} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{c}_{d_{n_{q_d},n_v}}}{\partial \mathbf{P}_{q_d}}
\end{bmatrix} \tag{4.63}$$

where $\mathbf{x}_{i,j}$, $\mathbf{l}_{i,j}$, $\mathbf{c}^*_{i,j}$ and $\mathbf{c}_{d_{i,j}}$ are the i^{th} point, line, dual conic and degenerate conic respectively as measured on the j^{th} view. Each element in Eq. 4.63 is a block matrix of the form

$$\frac{\partial \mathbf{x}_{i,j}}{\partial \mathbf{P}_v} = \begin{bmatrix}
\frac{\partial x_{i,j}}{\partial P_{v_{1,1}}} & \cdots & \frac{\partial x_{i,j}}{\partial P_{v_{12,1}}} & \frac{\partial x_{i,j}}{\partial P_{v_{1,2}}} & \cdots & \frac{\partial x_{i,j}}{\partial P_{v_{12,2}}} & \cdots & \frac{\partial x_{i,j}}{\partial P_{v_{12,n_v}}} \\
\frac{\partial y_{i,j}}{\partial P_{v_{1,1}}} & \cdots & \frac{\partial y_{i,j}}{\partial P_{v_{12,1}}} & \frac{\partial y_{i,j}}{\partial P_{v_{1,2}}} & \cdots & \frac{\partial y_{i,j}}{\partial P_{v_{12,2}}} & \cdots & \frac{\partial y_{i,j}}{\partial P_{v_{12,n_v}}} \\
\frac{\partial z_{i,j}}{\partial P_{v_{1,1}}} & \cdots & \frac{\partial z_{i,j}}{\partial P_{v_{12,1}}} & \frac{\partial z_{i,j}}{\partial P_{v_{1,2}}} & \cdots & \frac{\partial z_{i,j}}{\partial P_{v_{12,2}}} & \cdots & \frac{\partial z_{i,j}}{\partial P_{v_{12,n_v}}}
\end{bmatrix} \tag{4.64}$$

where $\mathbf{x}_{i,j} = [x_{i,j}\, y_{i,j}\, z_{i,j}]^\mathsf{T}$ and $P_{v_{i,j}}$ denotes the i^{th} parameter of the j^{th} camera in vector $\mathbf{P}_v$. Note that the 3×4 matrix P is decomposed as a 12-vector (see Eq. 4.25). Similar block-matrices are denoted by $\frac{\partial \mathbf{l}_{i,j}}{\partial \mathbf{P}_v}$, $\frac{\partial \mathbf{c}^*_{i,j}}{\partial \mathbf{P}_v}$, $\frac{\partial \mathbf{c}_{d_{i,j}}}{\partial \mathbf{P}_v}$, $\frac{\partial \mathbf{x}_{i,j}}{\partial \mathbf{P}_x}$, $\frac{\partial \mathbf{l}_{i,j}}{\partial \mathbf{P}_l}$, $\frac{\partial \mathbf{c}^*_{i,j}}{\partial \mathbf{P}_{q^*}}$, and $\frac{\partial \mathbf{c}_{d_{i,j}}}{\partial \mathbf{P}_{q_d}}$.

If a point $\mathbf{x}_{i,j}$ is missing, then the corresponding 3 rows of the expressions $\frac{\partial \mathbf{x}_{i,j}}{\partial \mathbf{P}_v}$ and $\frac{\partial \mathbf{x}_{i,j}}{\partial \mathbf{P}_x}$ are omitted from the Jacobian (this is shown in the rows marked between the symbols $\triangleright$ and $\triangleleft$ in Eq. 4.63). The same holds for missing lines, dual conics and degenerate conics.

A simple cost function to minimize might be

$$\|\mathbf{m}_{xlcc_d} - f(\mathbf{P}_{vxlq^*q_d})\|, \tag{4.65}$$

where $f(\mathbf{P}_{vxlq^*q_d})$ is the projection of the parameter vector as defined by equations 2.2, 2.9, 2.16 and 2.20.

This initial, simple formulation has several disadvantages. First, Eq. 4.65 is the algebraic error between the projected parameters and the measured data. This makes the problem scale-dependent which in turn forces equations 2.2, 2.9, 2.16 and 2.20 to be satisfied up to a strict equality. Second, as we have seen earlier, the outlines of the quadrics must satisfy the epipolar geometry which is not explicitly taken into account in this formulation. This will result in topologically erroneous reconstructions of the quadrics. Finally, the 3D Plücker lines must satisfy an internal orthogonality constraint (see Eq. 2.5), which is not considered in the formulation either.

The first problem is solved by simply using the geometric distance instead of the algebraic distance

$$\begin{aligned}\epsilon_g = & \sum_{i,\,j}^{n_x,n_v} d(\mathbf{x}_{i,j}, f(\mathbf{P}_{v_j}, \mathbf{P}_{x_i})) + \sum_{i,\,j,\,\mathbf{x}_l \in \mathcal{M}_{l_{i,j}}}^{n_l,n_v} d(\mathbf{x}_l, f(\mathbf{P}_{v_j}, \mathbf{P}_{l_i})) + \\ & \sum_{i,\,j,\,\mathbf{x}_c \in \mathcal{M}_{c_{i,j}}}^{n_q,n_v} d(\mathbf{x}_c, f(\mathbf{P}_{v_j}, \mathbf{P}_{q_i^*})) + \sum_{i,\,j,\,\mathbf{x}_{c_d} \in \mathcal{M}_{c_{d_{i,j}}}}^{n_{q_d},n_v} d(\mathbf{x}_{c_d}, f(\mathbf{P}_{v_j}, \mathbf{P}_{q_{d_i}})),\end{aligned} \tag{4.66}$$

where $\mathcal{M}_{l_{i,j}}$, $\mathcal{M}_{c_{i,j}}$ and $\mathcal{M}_{c_{d_{i,j}}}$ are the sets of measured points on the line, conic and degenerate conic i from view j, respectively, and $d(\mathbf{X}, \mathbf{Y})$ is the geometric distance between objects $\mathbf{X}$ and $\mathbf{Y}$ (note that this distance is defined differently depending on the types of the objects).

To cope with the second problem (the epipolar consistency between conic outlines), we use

$$\epsilon_e = \sum_{i,\,j,\,k}^{n_q,n_v,2} d(\mathbf{l}_{mapped_{i,j,k}}, f(\mathbf{P}_{v_j}, \mathbf{P}_{q_i^*})) + \sum_{j,\,k\ (j \neq k)}^{n_v,n_v} d(\mathbf{l}_{epipolar_{j,k}}, \mathbf{x}_{int_j}), \tag{4.67}$$

where $\mathbf{l}_{mapped_{i,j,k}}$ are the epipolar lines, for full-rank quadrics and plane conics, mapped from view j to view i as defined in Eq. 4.43, $\mathbf{l}_{epipolar_{j,k}}$ are the epipolar lines, for degenerate quadrics, mapped from view k to view j as defined in Eq. 4.56 and $\mathbf{x}_{int_j}$ are the intersections of the lines forming the degenerate conic in view j. It is assumed that the initial dual quadric and degenerate quadric parameters are epipolar-consistent and thus ϵ_e is zero when Bundle Adjustment begins. The final cost function is the sum of the above expressions

$$\epsilon = \epsilon_g + \epsilon_e. \tag{4.68}$$

During the iterations. Eq. 4.67 serves as a *geometric constraint* in order to keep the epipolar-consistency of the quadrics. However, unless this constraint is achieved *exactly* at each iteration (which is not guaranteed), the quadrics may lose their topology. Therefore, a final epipolar-consistency correction as described in Algorithms 4.2 and 4.3 is necessary after Bundle Adjustment finishes. Nevertheless, it is worth emphasizing that Eq. 4.67 effectively prevents the parameters of the quadrics to evolve freely. In practice, this means that the quadrics will be nearly epipolar-consistent throughout Bundle Adjustment at the expense of probably stopping short of the global minimum.

It is worth noting that the algorithm described thus far produces a projective reconstruction. In a projective framework, topology preservation makes little sense since, for example, a hyperboloid

is projectively equivalent to a sphere. However, when the reconstruction is rectified to a metric framework, topology becomes important. The purpose of the conic outline correction algorithms described in this paper is to preserve the topology once the metric rectification is performed.

Finally, in order to achieve the internal orthogonality constraint for lines, two strategies can be adopted:

1. Start with a set of internally-orthogonal lines via plane intersections as described in Section 4.1.2 and let the parameters evolve freely during Bundle Adjustment. When the iterations finish, the lines are orthogonalized again.

2. Add an orthogonalization step at each iteration of the Bundle Adjustment.

In practice, we have found both options produce nearly the same results. The only practical difference being the extra time needed for each iteration if the second strategy is adopted. The orthogonalization of two 3D vectors is an easy problem. In our case, the direction of the line remained fixed and only the momentum of the line was modified.

4.4 Experimental Analysis

We conducted four experiments where some objects displaying points, lines, quadrics, plane conics and degenerate quadrics were reconstructed. In the first experiment, a few objects were placed in a box and some images were taken with the stereo rig of our robot, but only the left image was used (see Figures 4.13.a-c). In the second and third experiments, various objects were also placed in boxes but we used a commercial digital camera (see Figures 4.14.a-c and 4.15.a-c). For the last experiment, a fountain scene displaying quadrics, plane conics and degenerate quadrics was reconstructed (see Figure 4.16.a-c). The same camera was used in the last three experiments.

The Canny edge detector was used to find edgels and corners to sub-pixel precision. All sets of points for lines and conics were chosen by hand. Lines were found by doing a linear regression. Conics are fitted to the sets of points measured on the conics and degenerate conics ($\mathcal{M}_{c_{ij}}$ and $\mathcal{M}_{c_{d_{ij}}}$ respectively) but degenerate conics receive special attention: the sets making up both lines are manually identified and their intersection (the image of the singularity) is also computed.

The full procedure can be visualized in Figure 4.12. We proceed to describe the steps in detail next.

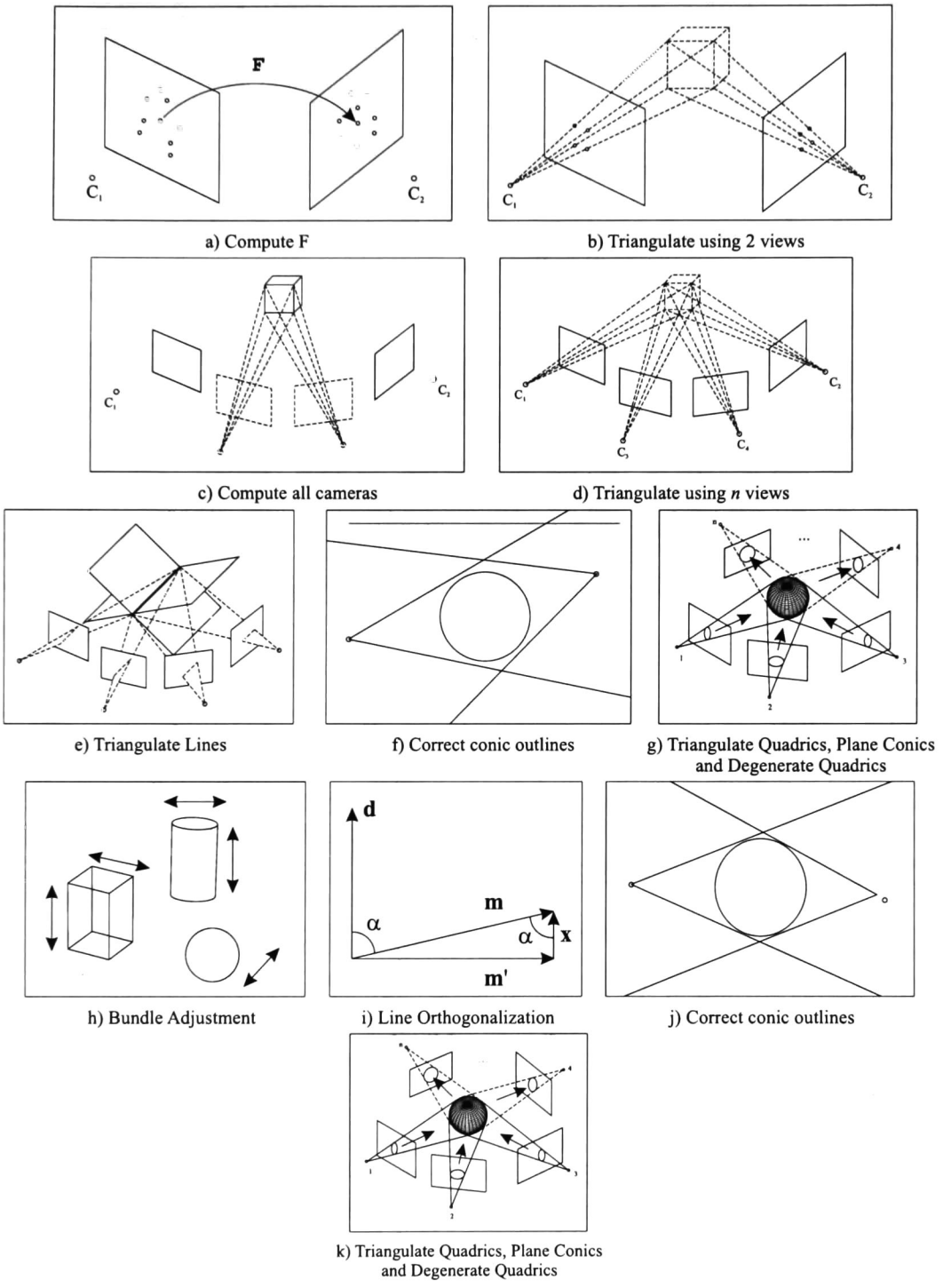


Figure 4.12: Steps in the Bundle Adjustment algorithm.

4.4.1 Initialization of points and cameras

The initialization is produced as follows. A set of $n \geq 8$ point correspondences (manually chosen) is normalized with homographies $\mathtt{H}$ and $\mathtt{H}'$ so that their centroid lies at the origin and their mean distance to it is $\sqrt{2}$. Using these points the Fundamental matrix is computed with Eq. 2.25 and later modified to enforce the rank-2 constraint. Finally, the Fundamental matrix is de-normalized according to $\mathtt{F}' = \mathtt{H}'\mathtt{F}\mathtt{H}$ (Figure 4.12.a).

Using the Fundamental matrix, the first two cameras are computed using equations Eq. 2.26 and Eq. 2.27 (Figure 4.12.a). Then all the points are triangulated according to Eq. 4.4 using only the first two cameras (Figure 4.12.b). With all the points available, all the cameras are computed using Eq. 4.10 (Figure 4.12.c). Finally, all the points are triangulated *again* taking all cameras into account, according to Eq. 4.11 (see Figure 4.12.d).

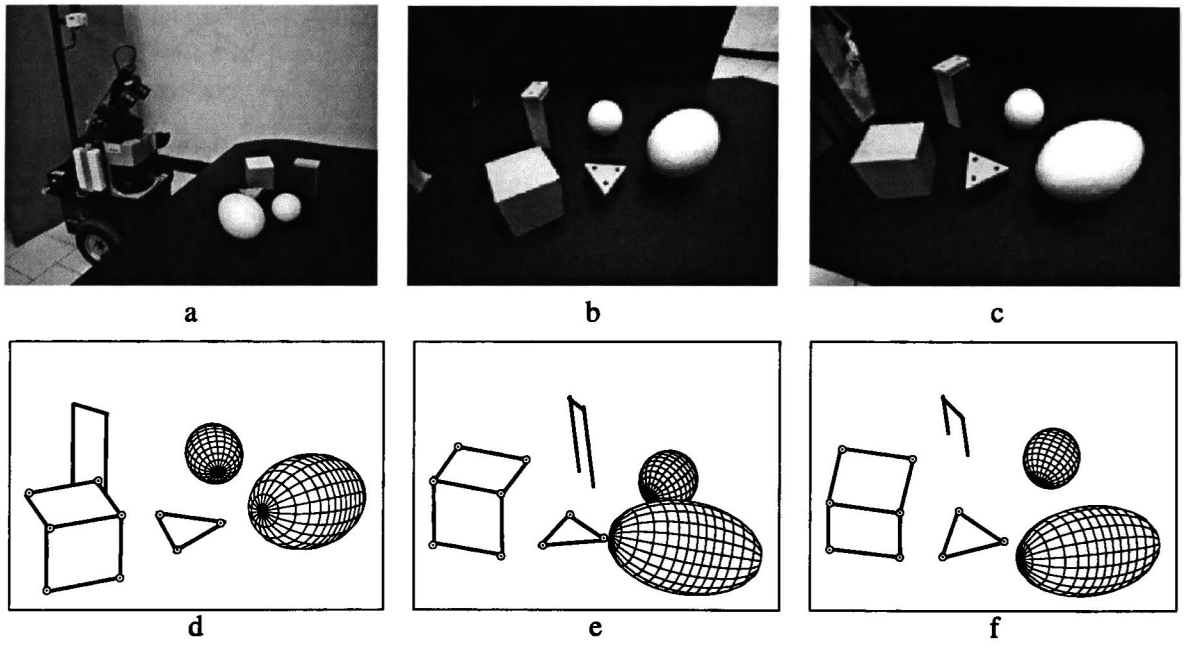


Figure 4.13: a) Setup of the scene. b) and c) A couple of images in the sequence (top). d), e) and f) Resulting reconstruction from different view points (bottom).

4.4.2 Initialization of lines

With all camera matrices available, the lines are reconstructed by back-projecting the images of the lines to 3D planes according to Eq. 4.16 and stacking the planes produced by all the views as shown in Eq. 4.17. The SVD is then applied to this matrix to find the best two planes that produce the line of intersection (Figure 4.12.e). This procedure is repeated for all the lines.

4.4.3 Initialization of quadrics

Finally, the initial set of quadrics is produced. First, conics $\mathtt{C}_i$ are fitted to the sets $\mathcal{M}_{c_{ij}}$ and $\mathcal{M}_{c_{d_{ij}}}$ of measured points for conics and degenerate conics, respectively, in every view. Three

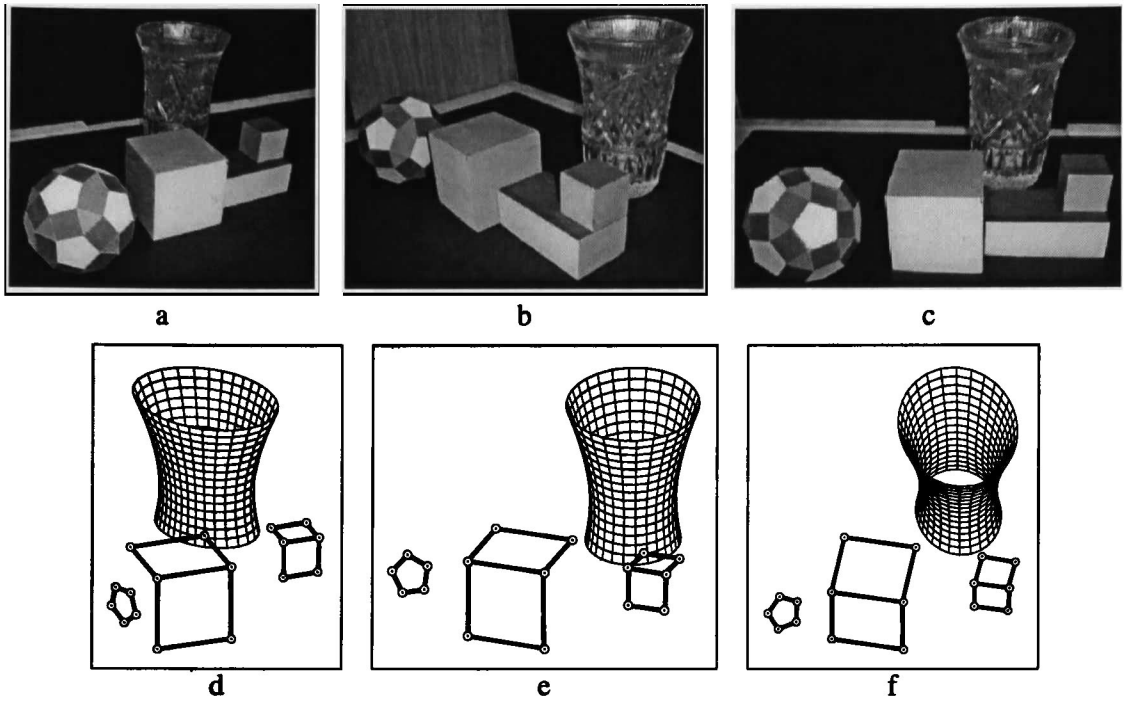


Figure 4.14: a), b) and c) Three of the images in the sequence (top). d), e) and f) Resulting reconstruction from different view points (bottom).

views are chosen (the ones that have the widest baselines between them) and the conic outlines are corrected in these views using Algorithm 4.1 or Algorithm 4.3 if the conics are degenerate (Figure 4.12.f). Then, 3D quadrics and plane conics are obtained by the solving the projection equation $\alpha C_i^* = P_j Q_j^* P_j^T$ (recall that the scalar factor is important in this case) as described in Eq. 4.27 using the corrected outlines. In the case of degenerate quadrics, the reconstruction is slightly different, see Section 4.1.6 for details. The quadrics thus obtained are used in Algorithm 4.2 to obtain the corrected outlines in the rest of the views (Figure 4.12.g).

We must note, however, that a different procedure was used in the last experiment. Since the contours of the fountain are highly irregular (compare the contour seen in Figures 4.16.a-b), a consistent conic could not be extracted from image features at all. Instead, two hyperbolas and a degenerate conic were placed near the features by hand (see Figure 4.16.c). Nevertheless, besides this, the algorithm was implemented as described.

4.4.4 Bundle adjustment

Once the cameras, points, lines, quadrics, plane conics and degenerate quadrics are initially reconstructed, they are used as seed for the Bundle Adjustment stage (Figure 4.12.h). For this case the Jacobian used in the Levenberg-Marquardt iteration was approximated by finite differences. During Bundle Adjustment, lines and quadrics receive special treatment. Recall that lines must satisfy an internal-orthogonality constraint. Both line-orthogonalization strategies described in Section 4.3.1 were tested, the only noticeable difference being the longer time needed for Bundle Adjustment to converge for the second strategy. Therefore, we recommend the use of the first strategy

where line parameters evolve freely and orthogonalization is performed only once after Bundle Adjustment has finished (Figure 4.12.i).

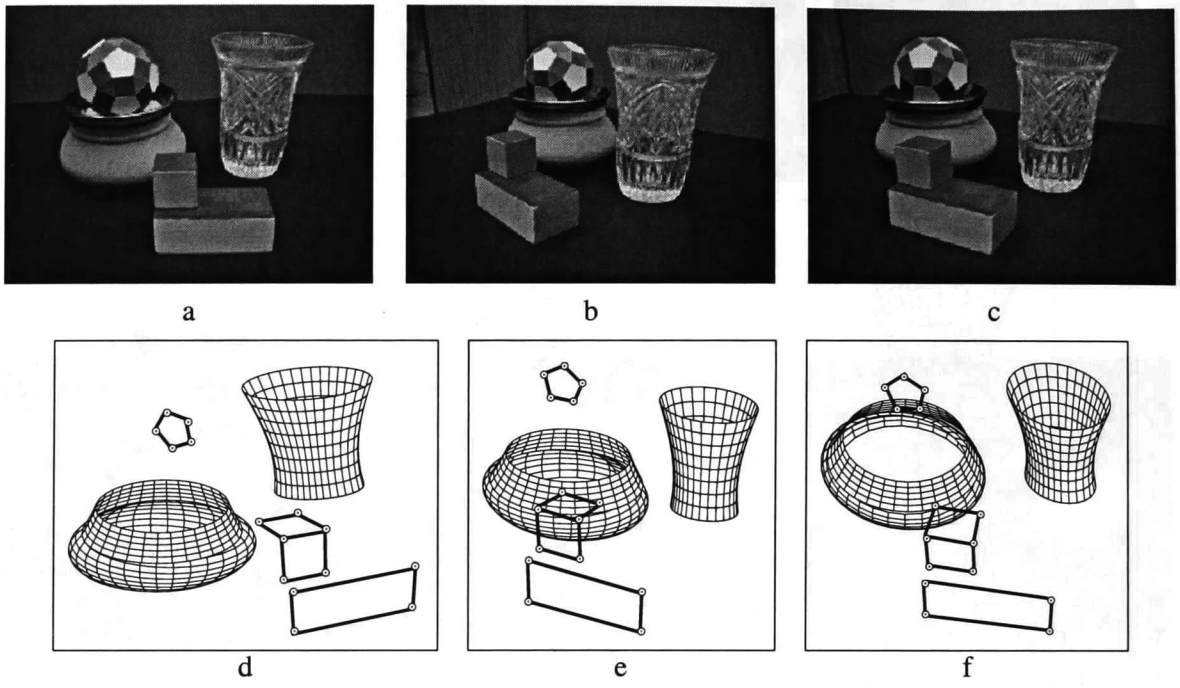


Figure 4.15: a), b) and c) Three of the images in the sequence (top). d), e) and f) Resulting reconstruction from different view points (bottom)

Quadrics, on the other hand, must satisfy the epipolar consistency constraints, namely, that the tangent lines in each view are mapped to tangent lines in the rest of the views via the fundamental matrices. The initialization stage produces consistent quadrics, but it would be extremely expensive to use Algorithms 4.1, 4.2 and 4.3 during the iterations of Bundle Adjustment. However, an extra measurement is added to the cost function being minimized (see Eq. 4.67) which measures the geometric distance between mapped epipolar tangent lines and the conics. This distance should be kept near zero. Unfortunately, this goal is seldom achieved exactly which means that quadrics may indeed lose topology during Bundle Adjustment. Nevertheless, this constraint *effectively prevents the quadric parameters from drifting aimlessly*. In fact, this check may prevent the system from reaching the global minimum while keeping the quadric parameters near a topologically-correct state. Finally, when Bundle Adjustment finishes it is necessary to apply Algorithms 4.1 and 4.2 again to ensure that the topology of the quadrics is preserved (Figure 4.12.j and 4.12.k).

The projective reconstruction is rectified to a metric framework by computing a homography that takes a few control points on the projective reconstruction (at least 5 points to account for the 15 dof of a general 3D homography) to their ideal position (see Fig. 4.17). The average geometric error in millimeters between the actual position of the control points after rectification and the ideal position is presented in Table 4.1. For proper reference, the approximate minimum bounding box for each scenario is also presented. Note that the error is not related to the size of the bounding box.

For all experiments, tests were conducted using 4 and 5 views (though, only the results for 4 views are presented in Table 4.1). Both line orthogonalization strategies were tested. Some of the

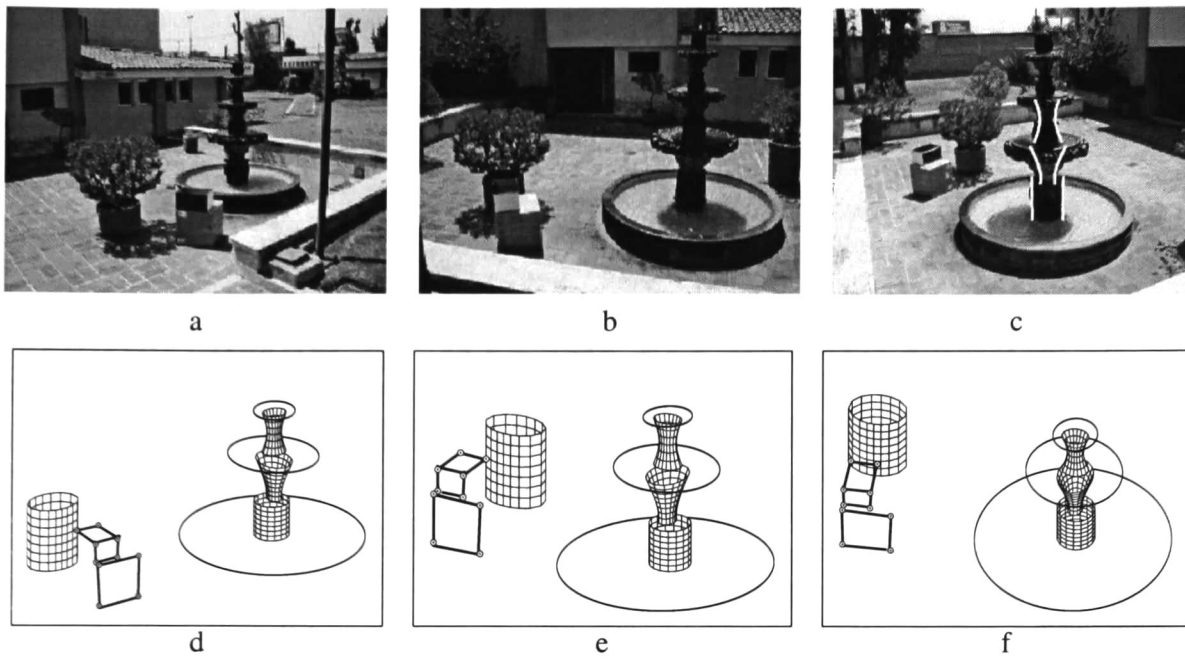


Figure 4.16: a), b) and c) Three of the images in the sequence (top). d), e) and f) Resulting reconstruction from different view points (bottom).

Experiment	Average error in mm	Control points	Bounding box size
1	0.80mm	6	450mm × 350mm × 300mm
2	1.14mm	12	300mm × 250mm × 300mm
3	3.18mm	10	270mm × 200mm × 220mm
4	9.52mm	10	370cm × 250cm × 270cm

Table 4.1: Average error in millimeters measured at the control points of the scenes reconstructed.

images used and the resulting reconstructions are shown in Figures 4.13, 4.14, 4.15 and 4.16.

4.4.5 Initializing with the trifocal tensor

For comparison purposes, we also tested using the Trifocal Tensor instead of the Fundamental Matrix in the initialization of points and cameras. The Trifocal Tensor was computed using the Gold Standard Algorithm as described in [27] (pages 383-386). Both point-point-point and combined point-point-point and line-line-line correspondences were used to compute the tensor. The same views used to compute F were used for $\mathcal{T}$ plus another one lying approximately halfway between the first two views. Similarly the same points used to compute the Fundamental Matrix were used for the tensor. The first 3 cameras were computed directly from $\mathcal{T}$, and the points were triangulated using these three views using Eq. 4.11. The rest of the cameras were computed using the triangulation method of Eq. 4.10. Again, we measured the average geometric error between the control points after metric rectification of the projective reconstruction produced using Bundle Adjustment and their ideal positions. Only the second experiment was tested (Figure 4.14).

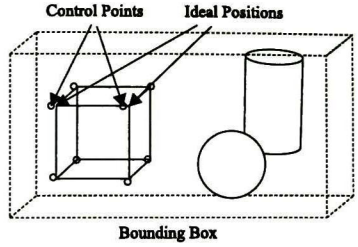


Figure 4.17: Situation of the control points and the bounding box.

The results are presented in Table 4.2. From this table it can be seen that using the trifocal tensor instead of the fundamental matrix does not produce significantly better results.

Error using F	$\mathcal{T}$, 3-point corresp.	$\mathcal{T}$, 3-point and 3-line corresp.
1.14mm	1.37mm	1.34mm

Table 4.2: Comparison of average error in millimeters measured at the control points of the second experiment.

Chapter 5

Registration of 2D and 3D Points Using Geometric Algebra and Tensor Voting

5.1 Introduction

We have already stated, in the beginning of this thesis, that the problem of registering data sets is common in the computer vision literature. Applications range from the alignment of range measurements for the automatic re-construction of maps for robotic navigation [31], [42]; registration of CT and MR images for medical purposes [24], [51], [20], [37], [22]; computer graphics and CAD modeling [58], [19] and recognition of objects [11], [59].

The algorithm proposed herein is based on re-casting the correspondences problem as a problem of finding a couple of lines in 2D or a plane in 3D using Tensor Voting. Therefore, before proceeding we shall present a short introduction to this methodology.

5.2 Tensor Voting

Tensor voting is a methodology for the extraction of dense or sparse features from nD data. Some of the features that can be detected with this methodology include lines, curves, points of junction and surfaces.

The Tensor Voting methodology is grounded in two elements: *tensor calculus* for data representation and *tensor voting* for data communication. Each input site propagates its information in a neighborhood (the information itself is encoded as a tensor and is defined by a predefined voting field). Each site collects the information cast there by its neighbors and analyzes it, building a saliency map for each feature type. Salient features are located at local extrema of these saliency maps, which can be extracted by non-maximal suppression.

For the present work, we found that sparse tensor voting, along with Geometric Algebra, was enough to solve the problem. Also, we only needed the detection of lines (or curves) in 2D, and the detection of planes (or surfaces) in 3D. Therefore we will confine our explanation of tensor voting to these particular processes. The interested reader may find further information in [47].

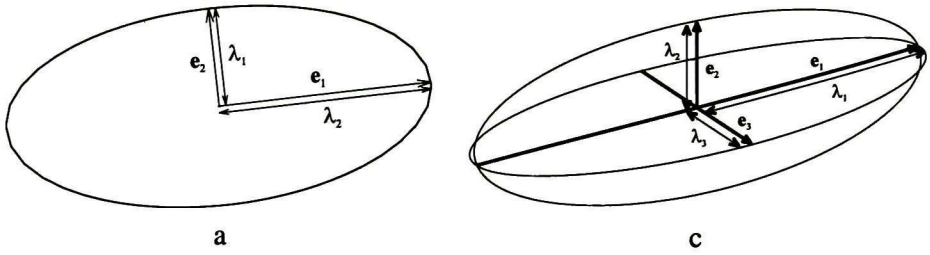


Figure 5.1: a) Graphic representation of a second order 2D symmetric tensor in 2D, and 3D b).

5.2.1 Tensor representation

To make this presentation easier, we will be mainly describing the 3D Tensor Voting formalism, because this is a generalization of the 2D case. However, we will still make the differences with the 2D case whenever necessary.

In tensor voting, all points are represented as tensors. To express a second order symmetric tensor S graphically depicted as an ellipse (2D) or an ellipsoid (3D), we choose to take the associated quadratic form, and to diagonalize it, leading to a representation based on the eigenvalues λ_1, λ_2 and λ_3 and the eigenvectors $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$. Therefore, we can write the tensor S as

$$S = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \end{bmatrix} \tag{5.1}$$

The eigenvectors correspond to the principal directions of the ellipse (2D) or the ellipsoid (3D) and form an orthonormal basis, while the eigenvalues encode the size and shape of the ellipse (see Fig. 5.1).

For the rest of this paper, we will use the convention that the eigenvectors have been arranged so that $\lambda_1 > \lambda_2 > \lambda_3$. In this scheme, 3D points are encoded as *ball* tensors (i.e. tensors with eigenvalues $\lambda_1 = \lambda_2 = \lambda_3 = 1$); *curvels* as *plate* tensors (i.e. tensors with $\lambda_1 = \lambda_2 = 1$, and $\lambda_3 = 0$, tangent direction given by $\mathbf{e}_3$); and *surfels* as *stick* tensors (i.e. $\lambda_1 = 1$, $\lambda_2 = \lambda_3 = 0$, normal direction given by $\mathbf{e}_1$).

A ball tensor encodes complete uncertainty of orientation, a plate tensor encodes uncertainty of orientation in two axis, but complete certainty in the other one, and a stick tensor encodes absolute certainty of orientation. Tensors that lie between these three extremes encode differing degrees of orientation certainty. The point-ness of any given tensor is represented by λ_3, the curve-ness is represented by $\lambda_2 - \lambda_3$ and the surface-ness by $\lambda_1 - \lambda_2$. Also, note that a second order tensor only encodes direction, but not *sense*.

For the 2D case, only two types of tensors exist: the ball tensor and the stick tensor. The ball tensor represents complete uncertainty about the orientation of the point, whereas the stick tensor represents absolute certainty of the *normal direction* of the curve that passes through the current point.

5.2.2 Voting Fields

We have just seen how the various types of input data are encoded in tensor voting, now we will describe how these tensors communicate between them. The input usually consists of a set

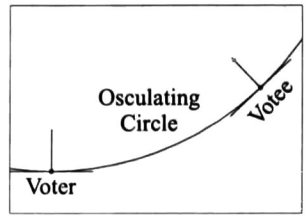
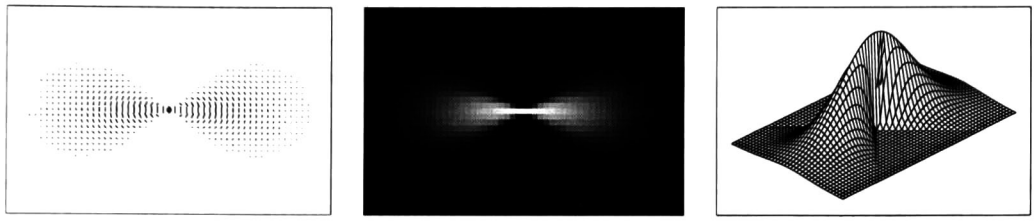


Figure 5.2: The osculating circle and the corresponding normals of the voter and votee.



a b c

Figure 5.3: The fundamental voting field. a) and b) orientation and strength fields. c) 3D display of the strength field.

of sparse points. These points are encoded as ball tensors if no information is available about their direction. If only a tangent is available, the points are encoded as plate tensors. Finally, if information about the normal of the point is given, it is encoded as a stick tensor. Then, each encoded point, or *token* communicates with its neighbors using either a ball voting field (if no orientation is present), a plate voting field (if local tangents are available), or a stick voting field (when the normal is available).

These *voting fields* have been derived from a 2D fundamental voting field that encodes the constrains of surface continuity and smoothness, among others. To see how the fundamental voting field was derived, suppose that we have a voter p with an associated normal $\mathbf{n}_p$. At each votee site x surrounding the voter p, the direction of the fundamental field $\mathbf{n}_x$ is determined by the normal of the osculating circle at x that passes through p and x and has normal $\mathbf{n}_p$ at p, see Fig. 5.2.

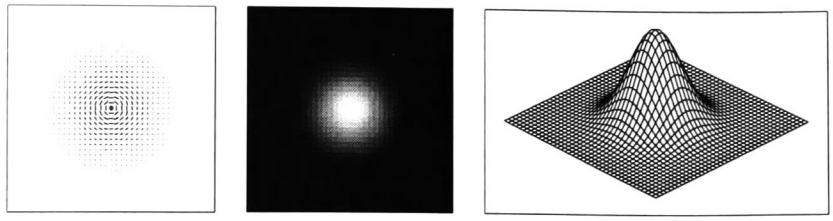
The strength of the fundamental field $s(\mathbf{d}, \rho, \sigma)$ at each point depends on the distance $\mathbf{d}$ and curvature ρ between p and x and is given by the following Gaussian function

$$s(\mathbf{d}, \rho, \sigma) = e^{-\left(\frac{|\mathbf{d}^2|+\rho^2}{\sigma^2}\right)}, \tag{5.2}$$

where σ is a scale factor that determines the overall rate of attenuation. The shape of the fundamental field can be seen in Fig. 5.3.

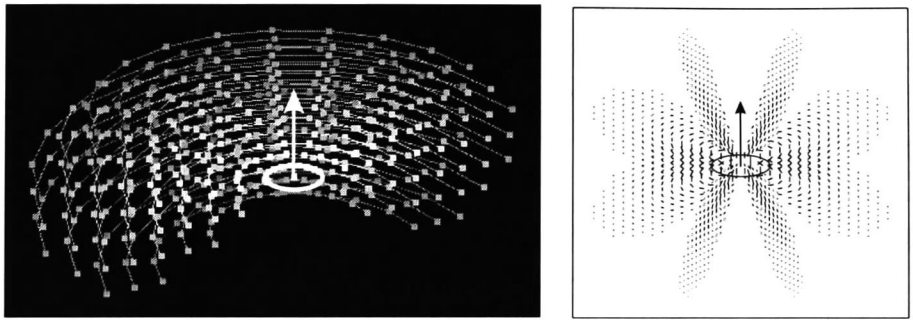
Finally, both orientation and strength are encoded as a stick vote. Hence, each voting site is also encoded as a tensor, and communication is performed by the addition of the tensor present at the votee and the tensor produced by the field at that site.

The fundamental voting field, in the 2D case, is also the stick voting field. The stick voting field can be used when there is information about the local normal of the token. However, when no information is available, we must use the ball voting field. This field is produced by rotating the fundamental field about the z axis and integrating the contributions at each site surrounding the voter. The resulting field is shown in Fig. 5.4.



a b c

Figure 5.4: The ball voting field. a) and b) orientation and strength fields. c) 3D display of the strength field.



a b

Figure 5.5: The stick voting field in 3D. a) Strength field cut in half showing the intensity at several slices. b) Some slices of the orientation field showing only the $\mathbf{e}_1$ eigenvalues. The normal voting direction is shown in both cases.

The 3D voting fields can be generated by rotating the fundamental voting field about the x, y and z axes, depending on the type of field we must generate. In the present work, however, we only needed the stick voting field and the ball voting field. The stick voting field in 3D is generated by rotating the fundamental voting field about the y axis, as shown in Fig. 5.5. The 3D ball voting field forms an isotropic sphere and looks virtually equal to its 2D counterpart.

5.2.3 Sparse Tensor Voting

Now that we have defined the tensors used to encode the information and the voting fields, we can describe the sparse voting process.

If no information is available about the normals at each token, we initialize them all as ball tensors. Then, in order to acquire the preferred orientations at each token, we place a ball voting field at each voter and cast votes to all the neighbors. The process is repeated until all tokens have cast votes to all their neighbors. Once this step is finished, we can extract the preferred orientations of each token by analyzing the eigensystem encoded in its tensor. The preferred normal direction is given by the eigenvector $\mathbf{e}_1$, and the saliency of this orientation is given by $\lambda_1 - \lambda_2$.

To enforce the orientation of the tokens, or if we have the normals available from the beginning, a sparse stick voting is performed. In this case, a stick voting field is placed on each token and is rotated until it becomes aligned with the local normal. Then, the resulting field is used to cast votes to all the neighbors and reinforce the local normals.

After the voting stage is finished. The tensors at each site are decomposed into their eigensystem. The resulting eigenvalue λ_1 encodes the absolute saliency of the token. This value roughly represents the number of neighbors that have lent support to this token. Tokens that are isolated will have a low absolute value and can be discarded as noise. From the remaining tokens, we can identify which ones belong to a surface, a curve, or a junction as follows. If $\lambda_1 \gg \lambda_2$, then there is a high confidence about the normal orientation of the token, and it belongs to a surface. If $\lambda_1 \approx \lambda_2$ and $\lambda_2 \gg \lambda_3$ then the token belongs to a curve, with tangent direction $\mathbf{e}_3$. Finally, if $\lambda_1 \approx \lambda_2 \approx \lambda_3$, then the current point belongs to a junction.

In this work, we have also used another surface-ness measure that is independent of the absolute saliency of the token. We simply take the ratio λ_2/λ_1. If the token belongs to a surface, the ratio will be close to 0. On the other hand, if the token is not oriented, the ratio becomes closer to 1. This is useful to find tokens with a low saliency but with a high orientation.

In the general formalism, after the preferred directions have been obtained, a *dense tensor voting* is performed. In this stage, each voting field casts votes all around the space surrounding the voter (not only to neighboring tokens), and features are extracted by performing an analysis that detects maxima from the dense junction, curvature and surface saliency maps. However, since we did not need that part of the engine in our present work, we refer the interested reader to [47] for more details.

5.3 Formulation of the problem in 2D

Using the Geometric Algebra of 2D, $\mathcal{G}_{2,0,0}$, the rigid motion model of a point $\mathbf{x} = x\sigma_1 + y\sigma_2$ can be expressed as

$$\mathbf{x}' = \widetilde{R}\mathbf{x}R + \mathbf{t}, \tag{5.3}$$

where $R = e^{\frac{\theta}{2}\sigma_{12}} = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})\sigma_{12}$ is a rotor and $\mathbf{t} = t_x\sigma_1 + t_y\sigma_2$ is a translation vector. From the previous equation, by left-multiplication with R we get

$$\begin{aligned} R\mathbf{x}' &= \mathbf{x}R + R\mathbf{t}, && (5.4)\\ (\cos(\tfrac{\theta}{2}) + \sin(\tfrac{\theta}{2})\sigma_{12})(x'\sigma_1 + y'\sigma_2) &= (x\sigma_1 + y\sigma_2)(\cos(\tfrac{\theta}{2}) + \sin(\tfrac{\theta}{2})\sigma_{12}) + && (5.5)\\ &\quad (\cos(\tfrac{\theta}{2}) + \sin(\tfrac{\theta}{2})\sigma_{12})(t_x\sigma_1 + t_y\sigma_2). && (5.6) \end{aligned}$$

Developing products and separating the resulting expression into its basis vectors we get the following equations

$$\begin{aligned} x'\cos(\tfrac{\theta}{2}) + y'\sin(\tfrac{\theta}{2}) &= x\cos(\tfrac{\theta}{2}) - y\sin(\tfrac{\theta}{2}) + \cos(\tfrac{\theta}{2})t_x + \sin(\tfrac{\theta}{2})t_y, && (5.7)\\ y'\cos(\tfrac{\theta}{2}) - x'\sin(\tfrac{\theta}{2}) &= x\sin(\tfrac{\theta}{2}) + y\cos(\tfrac{\theta}{2}) - \sin(\tfrac{\theta}{2})t_x + \cos(\tfrac{\theta}{2})t_y. && (5.8) \end{aligned}$$

Factoring together the cosines and sines we get

$$\begin{aligned} \cos(\tfrac{\theta}{2})(x' - x) + \sin(\tfrac{\theta}{2})(y' + y) - (\cos(\tfrac{\theta}{2})t_x + \sin(\tfrac{\theta}{2})t_y) &= 0, && (5.9)\\ -\sin(\tfrac{\theta}{2})(x' + x) + \cos(\tfrac{\theta}{2})(y' - y) + (\sin(\tfrac{\theta}{2})t_x - \cos(\tfrac{\theta}{2})t_y) &= 0. && (5.10) \end{aligned}$$

Which are clearly the equations of two lines in the joint spaces $(y'+y)$, $(x'-x)$ and $(x'+x)$, $(y'-y)$:

$$\begin{aligned} \begin{bmatrix} \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) & -(\cos(\frac{\theta}{2})t_x + \sin(\frac{\theta}{2})t_y) \end{bmatrix} \begin{bmatrix} (y' + y) & (x' - x) & 1 \end{bmatrix}^{\mathsf{T}} &= 0, && (5.11)\\ \begin{bmatrix} -\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) & (\sin(\frac{\theta}{2})t_x - \cos(\frac{\theta}{2})t_y) \end{bmatrix} \begin{bmatrix} (x' + x) & (y' - y) & 1 \end{bmatrix}^{\mathsf{T}} &= 0. && (5.12) \end{aligned}$$

Finally, it can be easily seen by inspection that the normals of these two lines are not independent. They are related by a reflection about the y-axis and obey the following constraint

$$ab' + ba' = 0, \tag{5.13}$$

where (a, b) and (a', b') are the normals of the first and second line respectively. This equation will prove highly useful in the following sections and we will refer to it as *The Reflection Constraint.*

Equations 5.12 and 5.11 tell us that if we have two sets of points, we can detect which ones are in correspondence, via a rigid transformation, by checking for two lines in the joint spaces $(x' + x), (y' - y)$ and $(y' + y), (x' - x)$. Previous work has been done for the case of 2D affine transformations [35], which in the end was formulated as a problem of finding two independent 3D planes. However, by restricting the problem to a rigid motion model and by using Geometric Algebra, we have now simplified the problem to finding two non-independent lines in 2D. We will now show how this problem can be solved in an efficient way by applying tensor voting to find these lines.

5.4 Estimation of correspondences in 2D

Given two sets of 2D points $\mathtt{X}_1$ and $\mathtt{X}_2$, we are expected to find the correspondences between these two sets assuming a rigid transformation has taken place. The points that are actually in correspondence will be referred to as *inliers* and the rest of the points as *outliers*, and we have an unspecified number of them. No other information is given.

In the absence of better information, we populate the joint spaces as modeled by Eqs. 5.12 and 5.11 by matching all points from the first set with all the points from the second set. Let n_1 and n_2 be the number of points in the sets $\mathtt{X}_1$ and $\mathtt{X}_2$ respectively, and let m be the number of inliers in these sets. In the best possible case (no outliers), we have to find m points out of a set of m^2 points in each joint space. The percentage of noise in the best case is thus $100m\%$. To give us an idea of the extent of the problem, suppose that we have no outliers and $m = 50$. In this case, we are trying to find a line composed of 50 points out of a set of 2500 points (5000% noise, see Fig. 5.6) in each joint space. If we add outliers, the noise ratio in the joint spaces increases even more.

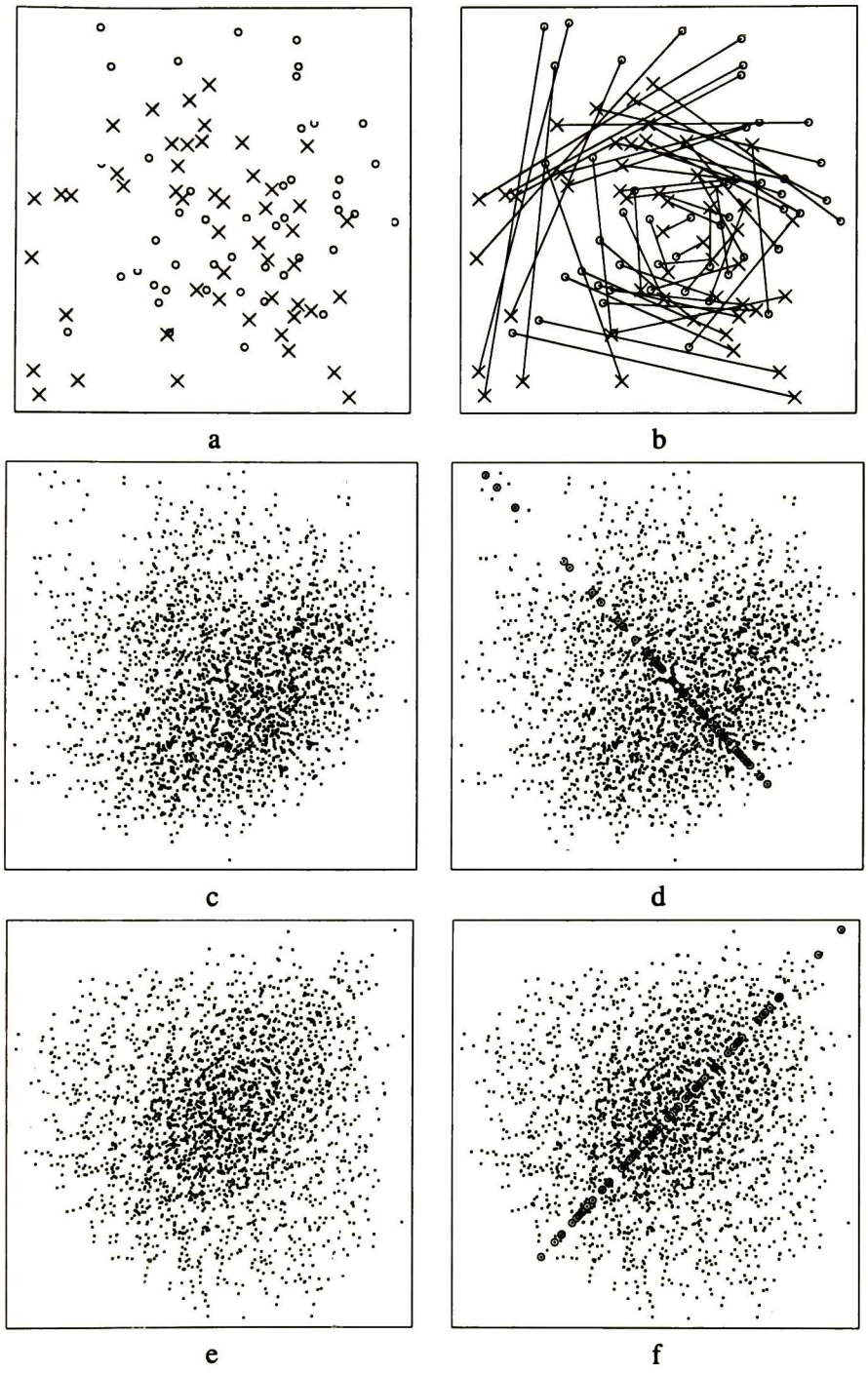


Figure 5.6: a) An example of 50 points with no outliers obeying a rigid motion transformation (95.5 deg, $T_x = 1.6$, $T_y = -1.5$), the first set is marked with crosses, the transformed set, with circles. c) and e) the joint spaces for this example $(x' + x)$, $(y' - y)$ (first row, center) and $(y' + y)(x' - x)$ (first row, right). Note the huge amounts of noise that must be filtered in these spaces, even with no outliers. Nevertheless, our method correctly identifies 48 of the 50 points b), d) and f).

To alleviate this problem, we have adopted a divide-and-conquer strategy. First, we populate the joint spaces by taking all possible matches between the sets $\mathbf{X}_1$ and $\mathbf{X}_2$ as candidate matches. Then, we translate the centroid of these spaces to the origin and perform a skew so as to separate the outliers from the real line, but taking care to preserve the constraint expressed by Eq. 5.13. To ease the exposition of our algorithm, we will refer to the points in the joint spaces as *tokens* to distinguish them from the *points* in the original 2D spaces.

In practice we do not know the normal of the line we are seeking. Therefore we do a series of hypothesize-and-test runs, varying the axis of skew at discrete intervals, until we find the desired lines that satisfy Eq. 5.13 or the full range (360 degrees) is covered. The skew is implemented as a rotation about the origin followed by a ten-fold scale in the y-axis (the extent of skew was chosen arbitrarily). The sense of rotation must be opposite across the different joint spaces so as to preserve Eq. 5.13 (see Fig. 5.7).

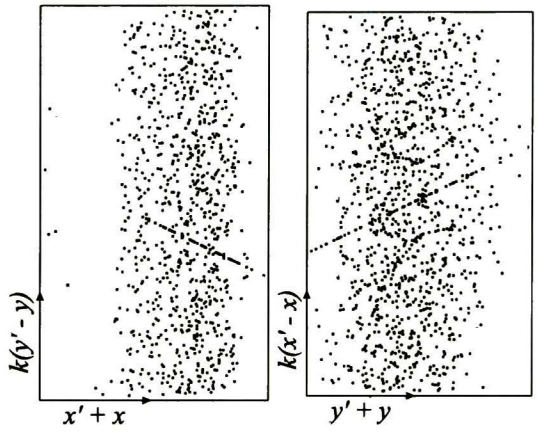


Figure 5.7: Section of the voting spaces showing the effect of the skew (the skew factor is k in this case). Note how the lines are much more salient now

The problem can be simplified if a search window is used to produce the candidate matches (instead of matching all points against themselves), but this limits the range of possible transformations that can be detected (Note however, that this is common practice and is in fact the solution used by the ICP-based algorithms [10], [61]). However, for the rest of this discussion we will assume the worst case where we match all points from $\mathbf{X}_1$ with all the points from $\mathbf{X}_2$ unless otherwise stated.

Once the joint spaces have been normalized and skewed with the hypothesized angle, we proceed as follows. First, we need to find the preferred local orientations of all the points. We initialize all the points as ball tensors and perform a sparse ball voting using a small ball voting field. After this voting is finished, we get the preferred orientations from the eigenvectors e_1. If the space has been skewed, we discard the points with a preferred normal direction $\mathbf{n} = [n_x, n_y]$ that is "too horizontal" (i.e. all those points for which $||n_x|| > 0.8$), since we expect to find lines with normals close to the vertical. Then, we reject all those points with low curve saliency: $(\lambda_1 - \lambda_2)/\lambda_1 > 0.75$.

The next step is to reinforce the preferred orientations by means of a sparse stick voting with a slightly higher reach. Only the points that passed the tests mentioned before emit a vote, but *all* points in the space receive votes. This is done so that points that might have been erroneously rejected in the previous stage may have a chance to be re-activated in further stages. After the sparse stick voting is done, we test *all* the points and reject (or re-activate) the points with $||n_x|| >$

1. Populate the voting spaces with tokens generated from the candidate correspondences. Initialize the skew angle $\alpha = 0°$.
2. Skew the voting space according to α.
3. Initialize all tokens to ball tensors.
4. Perform sparse ball voting and extract the preferred normals.
5. Perform sparse stick voting using the preferred normals. Optionally, repeat this step to eliminate outliers.
6. Obtain the equations of the lines described by the tokens with highest saliency in each voting space. Perform sparse stick voting between the tokens that lie in these lines. Enforce the Reflection Constraint of Eq. 5.13. Iterate until a stable line is found or a set number of iterations is hit.
7. If a satisfactory line is found, output the correspondences. Otherwise, increment α and repeat steps 2-6 until $\alpha = 360°$.

Algorithm 5.1 *General algorithm for the detection of correspondences between two 2D point sets under a single rigid transformation.*

0.8. From the remaining set of active points we reject those with low curve saliency ratio as mentioned before.

The sparse voting step just described is repeated several times, but using a smaller angle for the stick vote and wider reach each time to refine the sets of orientations and produce a highly salient line (if present). Typically, it is enough to increment the reach of the stick votes σ at discrete intervals until it covers at least half the total range of the voting space in the x-direction (the axis that is not skewed). We also reject all points with an absolute saliency λ_1 smaller than the average, at each step, to speed up the process of outlier rejection. When this is done, we should have a highly salient line (if one is present). The process is done simultaneously in both joint spaces and all the points must pass all tests in both spaces in order to cast votes on the next stage. Note also that depending on the amount of outliers present, this repetition can be omitted.

Once we have found the salient lines, we enter the final stage of the process where we refine the initial line estimates as follows. First, we save the preferred orientations of the active points and then reset all the votes at each site to the original ball tensors. Then, we perform sparse stick voting with a high σ so that the stick votes reach the whole space in the x-direction, using the stored preferred normals. Again, we cast votes only from active sites but receive votes everywhere.

We repeat the same tests of rejection by n_x and plane ratio; however, we add another test that takes into account the constraint of Eq. 5.13. Namely, we take the preferred orientation of the most highly salient token in each joint space and compute the equation of the line for each one. If the normals of the lines satisfy Eq. 5.13, then we only keep the points that simultaneously line in both lines and reject the rest. We also overwrite the preferred directions of all the tokens lying on a line to be consistent with the direction of the line. On the other hand, if the normals of the line do not satisfy Eq. 5.13, then we keep the tokens that lie on either line instead and reject the rest.

Since we populated the space by taking all possible matches, for each point in the first set there are multiple candidate matches on the second. The test of forcing each token to lie simultaneously in both lines in the joint spaces in order to remain active also has the effect of rejecting these multiple matches from the same point. Hence by using this simple rule we are also assuring that the final set will be at most a one-to-one mapping. We will prove why this happens in the following

sub-section

This process is repeated (including resetting the votes to ball tensors each time) until we find the lines that satisfy the reflection constraint and the amount of inliers detected is stable, or we hit a certain predefined number of iterations. If the lines found are indeed the correct ones but the reflection constraint is not satisfied well enough, then the subsequent iterations will have the effect of slightly modifying the direction of the line by mutual reinforcement of the votes cast by the active tokens. Since the large majority of the active tokens will be inliers (because the lines we found are the right ones), then the line that is computed in the next iteration will be closer to satisfying the orthogonality constraint. However, if the lines are wrong, this reinforcement will not occur and the lines will vary randomly. There is a chance that wrong lines will randomly satisfy this constraint, but these lines are rejected based on the amount of inliers detected. The number of iterations is typically low, only 4 or 5 are needed in most cases, but we use 10 iterations as a top limit.

If the number of iterations is hit and no stable pair of lines are detected that satisfy Eq. 5.13, then we know we have not found the desired correspondence and continue to test the next axis of skew until all possibilities are exhausted. Typically using the described normalization, lines with tangent directions ranging from -15 to 15 degrees are detected with confidence. Hence, the axis of skew is incremented at about 10 degrees each time. An outline of the process is presented in Algorithm 5.1.

5.4.1 Uniqueness constraint enforcement in 2D

We have mentioned previously that requiring that both tokens generated by the same correspondence lie on the lines detected using Tensor Voting is enough to guarantee a one-to-one mapping. We will now prove why this is correct. Let (x, y) be a point that is in correspondence with a point (x', y') via a known rigid transformation. Then, these points satisfy the equations of the lines 5.9 and 5.10

$$\cos(\tfrac{\theta}{2})(x' - x) + \sin(\tfrac{\theta}{2})(y' + y) - (\cos(\tfrac{\theta}{2})t_x + \sin(\tfrac{\theta}{2})t_y) = 0, \quad (5.14)$$

$$-\sin(\tfrac{\theta}{2})(x' + x) + \cos(\tfrac{\theta}{2})(y' - y) + (\sin(\tfrac{\theta}{2})t_x - \cos(\tfrac{\theta}{2})t_y) = 0. \quad (5.15)$$

Suppose that there is another point $(x'', y'') \neq (x', y')$ that is also in correspondence with (x, y) by the same transformation. Then (x'', y'') must satisfy the equations 5.9 and 5.10 too

$$\cos(\tfrac{\theta}{2})(x'' - x) + \sin(\tfrac{\theta}{2})(y'' + y) - (\cos(\tfrac{\theta}{2})t_x + \sin(\tfrac{\theta}{2})t_y) = 0 \quad (5.16)$$

$$-\sin(\tfrac{\theta}{2})(x'' + x) + \cos(\tfrac{\theta}{2})(y'' - y) + (\sin(\tfrac{\theta}{2})t_x - \cos(\tfrac{\theta}{2})t_y) = 0. \quad (5.17)$$

If we substract Eqs. 5.16 and 5.17 from Eqs. 5.14 and 5.15, respectively we get

$$\cos(\tfrac{\theta}{2})(x' - x'') + \sin(\tfrac{\theta}{2})(y' - y'') = 0 \quad (5.18)$$

$$-\sin(\tfrac{\theta}{2})(x' - x'') + \cos(\tfrac{\theta}{2})(y' - y'') = 0 \quad (5.19)$$

It can be easily demonstrated that the only values for $(x' - x'')$ and $(y' - y'')$ that simultaneously satisfy Eqs. 5.18 and 5.19 are 0, which in turn implies that $(x', y') = (x'', y'')$. This means that if we have a multiple match between (x, y) and two other points (x', y') and (x'', y'') under the same rigid transformation, then either $(x', y') = (x'', y'')$ or only one of these points will simultaneously satisfy the equations of the lines 5.9 and 5.10, and we can safely reject the other as a false match.

In practice this is implemented by checking against the lines obtained through Tensor Voting: if these lines satisfy the reflection constrain, then we can reject multiple matches by requiring that the points lie simultaneously on both lines.

5.5 Formulation in 3D

Using the Geometric Algebra of 3D space $\mathcal{G}_{3,0,0}$ the rigid motion of a 3D point $\mathbf{x} = x\sigma_1 + y\sigma_2 + z\sigma_3$ can be formulated as

$$\mathbf{x}' = \widetilde{R}\mathbf{x}R + \mathbf{t}, \tag{5.20}$$

where $R = e^{\frac{\theta}{2}A} = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})A$, $\mathbf{t} = t_x\sigma_1 + t_y\sigma_2 + t_z\sigma_3$ and $A = a_x\sigma_{23} + a_y\sigma_{31} + a_z\sigma_{12}$ is a bivector of magnitude 1 that specifies the axis of rotation. For simplicity, we will represent R as rotor of the form

$$R = q_0 + q_x\sigma_{23} + q_y\sigma_{31} + q_z\sigma_{12}. \tag{5.21}$$

By left-multiplication with R, the equation of rigid motion becomes

$$R\mathbf{x}' = \mathbf{x}R + R\mathbf{t}, \tag{5.22}$$

$$(q_0+q_x\sigma_{23}+q_y\sigma_{31}+q_z\sigma_{12})(x'\sigma_1+y'\sigma_2+z'\sigma_3) = (x\sigma_1+y\sigma_2+z\sigma_3)(q_0+q_x\sigma_{23}+q_y\sigma_{31}+q_z\sigma_{12})+ \tag{5.23}$$

$$(q_0+q_x\sigma_{23}+q_y\sigma_{31}+q_z\sigma_{12})(t_x\sigma_1+t_y\sigma_2+t_z\sigma_3). \tag{5.24}$$

Developing products, we get

$$\begin{aligned} &q_0x'\sigma_1+q_xx'\sigma_{231}+q_yx'\sigma_3-q_zx'\sigma_2+q_0y'\sigma_2-q_xy'\sigma_3+q_yy'\sigma_{312}+q_zy'\sigma_1+q_0z'\sigma_3+q_xz'\sigma_2-q_yz'\sigma_1+q_zz'\sigma_{123} = \\ &xq_0\sigma_1+yq_0\sigma_2+zq_0\sigma_3+xq_x\sigma_{123}+yq_x\sigma_3-zq_x\sigma_2-xq_y\sigma_3+yq_y\sigma_{231}+zq_y\sigma_1+xq_z\sigma_2-yq_z\sigma_1+zq_z\sigma_{312} + \\ &q_0t_x\sigma_1+q_xt_x\sigma_{231}+q_yt_x\sigma_3-q_zt_x\sigma_2+q_0t_y\sigma_2-q_xt_y\sigma_3+q_yt_y\sigma_{312}+q_zt_y\sigma_1+q_0t_z\sigma_3+q_xt_z\sigma_2-q_yt_z\sigma_1+q_zt_z\sigma_{123}. \end{aligned} \tag{5.25}$$

Re-arranging terms according to their multivector basis (see Eq. 2.41) we obtain the following four equations

$$\sigma_1 : \; q_0x' + q_zy' - q_yz' = q_0x + q_yz - q_zy + q_0t_x + q_zt_y - q_yt_z, \tag{5.26}$$

$$\sigma_2 : \; q_0y' + q_xz' - q_zx' = q_0y + q_zx - q_xz + q_0t_y + q_xt_z - q_zt_x, \tag{5.27}$$

$$\sigma_3 : \; q_0z' + q_yx' - q_xy' = q_0z + q_xy - q_yx + q_0t_z + q_yt_x - q_xt_y, \tag{5.28}$$

$$\sigma_{123} : \; q_xx' + q_yy' + q_zz' = q_xx + q_yy + q_zz + q_xt_x + q_yt_y + q_zt_z. \tag{5.29}$$

These equations can be re-arranged to express linear relationships in the joint difference and sum spaces

$$\sigma_1 : \; q_0(x'-x) - q_y(z+z') + q_z(y+y') + (q_yt_z - q_0t_x - q_zt_y) = 0, \tag{5.30}$$

$$\sigma_2 : \; q_0(y'-y) + q_x(z+z') - q_z(x+x') + (q_zt_x - q_0t_y - q_xt_z) = 0, \tag{5.31}$$

$$\sigma_3 : \; q_0(z'-z) - q_x(y+y') + q_y(x+x') + (q_xt_y - q_0t_z - q_yt_x) = 0, \tag{5.32}$$

$$\sigma_{123} : \; q_x(x'-x) + q_y(y'-y) + q_z(z'-z) - (q_xt_x + q_yt_y + q_zt_z) = 0. \tag{5.33}$$

These equations clearly represent four 3D planes. Thus, in order to estimate the correspondences due to a rigid transformation, it suffices with finding these planes in the joint spaces $\{(x'-x), (z+z'), (y+y')\}$, $\{(y'-y), (z+z'), (x+x')\}$, $\{(z'-z), (y+y'), (x+x')\}$ and $\{(x'-x), (y'-y), (z'-z)\}$.

However, as will be shown in the following sections, it will be enough to simply search for the plane described by Eq. 5.33 and verify it using constraints derived from the other three planes. In a sense, we will seek only a plane but require the points on this plane to simultaneously lie on the other three planes. We will now show how this is done and how this check helps in eliminating multiple matches.

5.6 Discarding multiple matches in 3D

First, we will derive the set of constraints that will be used to verify that the plane we have obtained through tensor voting is indeed the plane we are looking for. Let $(\mathbf{x}_i, \mathbf{x}'_i)$ and $(\mathbf{x}_j, \mathbf{x}'_j)$ be two correspondences. Then, these points satisfy Eq. 5.30

$$\begin{aligned} q_0 d_x - q_y s_z + q_z s_y + q_k &= 0, \quad (5.34) \\ q_0 d'_x - q_y s'_z + q_z s'_y + q_k &= 0, \quad (5.35) \end{aligned}$$

where $d_x = x'_i - x_i$, $s_z = z'_i + z_i$, $s_y = y'_i + y_i$; $d'_x = x'_j - x_j$, $s'_z = z'_j + z_j$, $s'_y = y'_j + y_j$; and $q_k = q_y t_z - q_0 t_x - q_z t_y$. If we substract these equations we get

$$q_0 v_x - q_y v s_z + q_z v s_y = 0, \quad (5.36)$$

where $v_x = d_x - d'_x$, $v s_z = s_z - s'_z$ and $v s_y = s_y - s'_y$. Using the definition of R, this equation can be rewritten as

$$k v_x - a_y v s_z + a_z v s_y = 0, \quad (5.37)$$

where $k = \cos(\frac{\theta}{2}) / \sin(\frac{\theta}{2})$. Using a similar procedure, for the Eqs. 5.31 and 5.32 we end up with the following system of equations

$$\begin{aligned} k v_x + a_z v s_y - a_y v s_z &= 0, \quad (5.38) \\ k v_y + a_x v s_z - a_z v s_x &= 0, \quad (5.39) \\ k v_z + a_y v s_x - a_x v s_y &= 0. \quad (5.40) \end{aligned}$$

Where v_y, v_z and $v s_x$ are defined accordingly. Note that we now have a system of equations depending on the unitary axis of rotation $[a_x, a_y, a_z]$. Since we can obtain the axis of rotation as the normal of the plane described by Eq. 5.33, then we only have one unknown: k. These equations can be mixed to yield the following three constraints

$$\begin{aligned} v_y(a_y v s_z - a_z v s_y) - v_x(a_z v s_x - a_x v s_z) &= 0, \quad (5.41) \\ v_z(a_y v s_z - a_z v s_y) - v_x(a_x v s_y - a_y v s_x) &= 0, \quad (5.42) \\ v_z(a_z v s_x - a_x v s_z) - v_y(a_x v s_y - a_y v s_x) &= 0. \quad (5.43) \end{aligned}$$

These equations only depend on the points themselves and on the plane spanned by them. Thus, we can use these constraints to tell whether the plane produced by Tensor Voting actually corresponds to a rigid motion model. Also note that these equations never become undefined since the factor k was removed. Furthermore, since these constraints have been derived from the original plane equations 5.30, 5.31 and 5.32 they are, in a sense, expressing the requirement that these points lie simultaneously on all three planes.

On the other hand, Eqs. 5.41, 5.42 and 5.43 have an interesting geometric interpretation. They are in fact expressing a double cross product that is only satisfied for true correspondences. To see this, note that if $A = [a_x, a_y, a_z]^\mathsf{T}$ $V = [v_x, v_y, v_z]^\mathsf{T}$ and $Vs = [vs_x, vs_y, vs_z]^\mathsf{T}$ then Eqs. 5.41, Eqs. 5.42 and Eqs. 5.43 can be rewritten as a vector equation of the form

$$V \times (A \times Vs) = 0. \tag{5.44}$$

This equation only holds due to an inherent symmetry that is only present for true correspondences; in other words, these equations can be used to reject false matches too. To prove this, first remember the well-known fact that any rigid motion in 3D is equivalent to a screw motion (rotation about the screw axis followed by a translation along it). Hence, without loss of generality, we can consider the case where the screw axis is aligned with the z axis. In this case, the screw motion consists of a rotation about z followed by a translation t_z along it. Therefore,

$$v_z = d_z - d'_z = z'_i - z_i - z'_j + z_j = (z_i + t_z) - z_i - (z_j + t_z) + z_j = 0. \tag{5.45}$$

Also, note that since $A = [001]^\mathsf{T}$, then the first cross product of Eq. 5.44, $A \times Vs = [-vs_y, vs_x, 0]^\mathsf{T}$ hence the vs_z component of Vs is irrelevant in this case and can be safely disregarded. Thus, we can analyze this problem in 2D by looking only at the x and y components of V and Vs. Accordingly, the difference and sum vectors will only have two components $d_i = [d_x, d_y]^\mathsf{T}$, $d_j = [d'_x, d'_y]^\mathsf{T}$, $s_i = [s_x, s_y]^\mathsf{T}$, and $s_j = [s'_x, s'_y]^\mathsf{T}$ The situation is illustrated in Figs. 5.8.a and d.

Since the angle between x_i and x'_i is the same as the angle between x_j and x'_j, then the rhombi spanned by the sum and difference of these points (the dashed lines in Fig. 5.8.d) are equal up to a scale factor. That is, there is a scale factor k such that $k||s_i|| = ||s_j||$ and $k||d_i|| = ||d_j||$. From whence $||s_i||/||d_i|| = ||s_j||/||d_j||$. In turn, this means that the triangle formed by the vectors d_i, d_j and V is proportional to the triangle s_i, s_j, Vs. And since, by construction, $s_i \perp d_i$ and $s_j \perp d_j$, then $V \perp Vs$.

Now, let us return to Eq. 5.44. The cross product $A \times Vs$ has the effect of rotating Vs by 90 degrees since $A = [0, 0, 1]^\mathsf{T}$ in this case. But since $Vs \perp V$, then the vector $A \times Vs$ will be parallel to V and hence, their cross product will always be 0, which is consistent with the analytic derivation.

This symmetry is broken if we have points that belong to different transformations, as shown in Figs. 5.8.b and e (we assume the worst case in which points x_i and x_j were applied the same translation but different rotation angle ϕ. If a different translation is present, the planes of motion will be different for both points, breaking the symmetry). Note how the angle between V and Vs is not orthogonal (Fig. 5.8.e).

In a similar way, when we have multiple correspondences, i.e., x_i matches both x'_i and x'_j, $i \neq j$ (Fig. 5.8.c), the symmetry is also broken and V is not orthogonal to Vs (see Fig. 5.8.f). Hence, the constraint expressed by Eq. 5.44 can be used to reject multiple matches too.

Having explained how we can identify that the correct plane has been detected, we will now show how we used Tensor Voting to find this plane.

5.7 Estimation of 3D correspondences

Given two sets of 3D points $\mathtt{X}_1$ and $\mathtt{X}_2$, we are expected to find the correspondences between these two sets assuming a rigid transformation has taken place, and we have an unspecified number of outliers in each set. No other information is given.

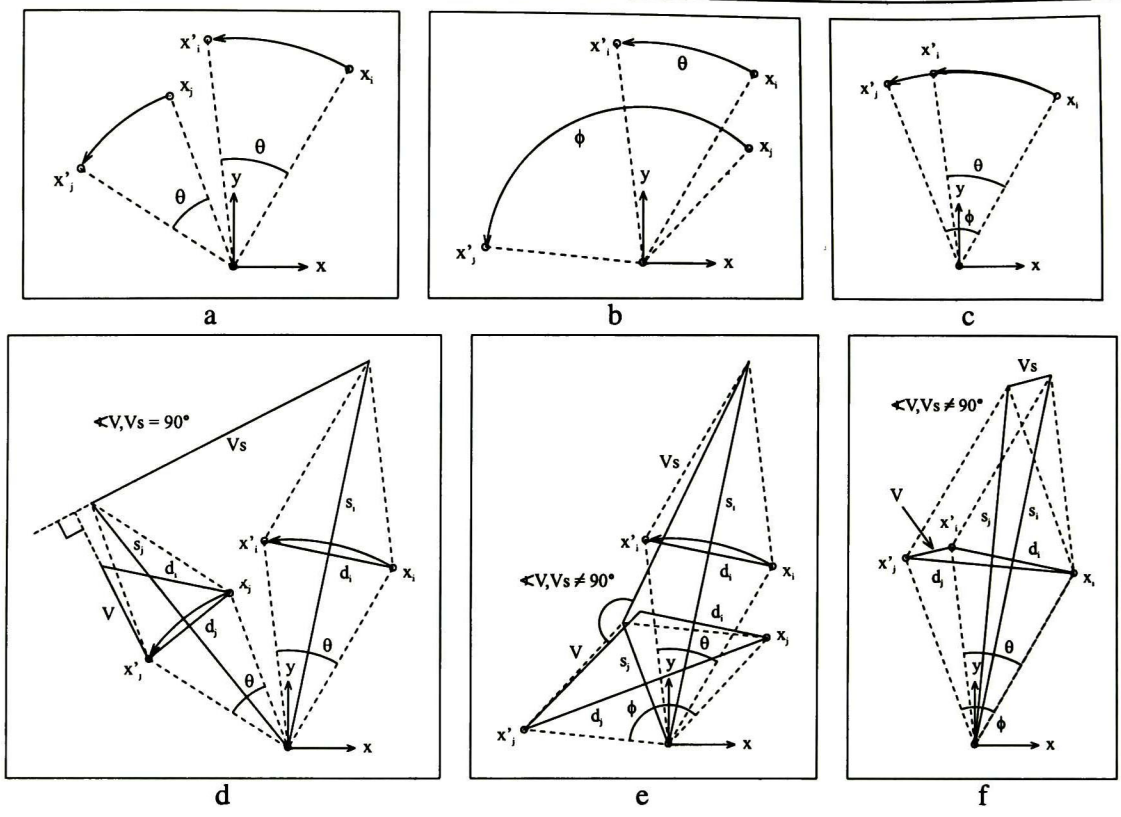


Figure 5.8: a) and d) Two correspondences belonging to the same transformation and the geometry of the plane of rotation. b) and e) Two correspondences belonging to different transformations (different angle) and their corresponding geometry. c) and f) Multiple correspondence case and its geometry.

In the absence of better information, we populate the joint space $(x' - x), (y' - y), (z' - z)$ by matching all points from the first set with all the points from the second set. Thus, the amount of outliers present in the joint space is multiplied 100-fold by this mapping.

This mapping enables us to detect the transformation regardless of the magnitude of the transformation. However, the drawback is that the density of the plane we are seeking decreases as the angle of rotation increases. To solve this problem, we have adopted a hypothesize-and-test scheme were a series of rotation axes are tested at discrete intervals and angles until the desired plane is identified. In order to identify the plane we use the constraint of Eq. 5.44, as explained in the previous section. This constraint requires the specification of two different points. In practice we use the point on the plane with the highest saliency and test it against the rest of the candidate points. If any of these points does not satisfy the constraint, we remove it from the plane. If not enough points remain after this pruning is completed, we reject the plane.

The plane is detected using the following procedure. First, the set of points $\mathbf{X}_1$ is rotated according to the current hypothesized axis and angle, and the joint space specified by Eq. 5.33 is populated with a dense correspondence scheme (all points from $\mathbf{X}_i$ matched against all the points from $\mathbf{X}_2$). These tokens are initialized as ball tensors and a sparse ball voting is done with a small σ to find the preferred normals.

As can be easily noted, Eq. 5.33 collapses for the case of pure translation. However, in this case, all points that belong to the motion model will cluster together in a *single token* in the joint space. This token is easy to detect because the first voting stage will produce a large absolute saliency value for this point. If a token with these characteristics is found, we stop the algorithm and produce the correspondences based on the tokens that were found clustered together.

In the general case, however, the next step is to reject all those tokens with a low plane saliency ratio $\lambda_2/\lambda_1 > 0.9$. Then, we perform a series of sparse plane voting iterations. At each iteration, we reject tokens with low plane ratio, and absolute saliency value below the mean. Only the tokens that remain active from the previous iteration cast votes in the current iteration, but all sites receive votes, in order to enable tokens that may have been erroneously rejected in a previous cycle. The σ value is increased slightly at each iteration so as to make the plane we are looking for "grow" gradually.

When the σ value has reached a predetermined value, we switch to the last stage of the algorithm where we enforce the constrain of Eq. 5.44. The value of σ is left fixed, but all the tensors at the tokens are reset to ball tensors. Then, a sparse plane voting is performed using the stored preferred normals at each active token. Now, instead of rejecting tokens with low plane saliency, we reject all those tokens that do not lie close to the plane defined by the token with the maximum absolute saliency. Then, we reject tokens with an absolute saliency value below the mean and finally, we enforce the constraint expressed by Eq. 5.44.

This last stage is repeated until the number of active tokens becomes stable (meaning we have actually found a plane representing a rigid motion model) or a pre-set number of iterations is hit. It is worth noting that, in general, it is not enough with finding a plane in the joint space $(x' - x), (y' - y), (z' - z)$ only. The checks against the other three joint spaces are needed to determine a correct rigid motion has been detected. Recall that this is precisely the purpose of Eq. 5.44, plus the added benefit of rejecting multiple matches.

If a plane that satisfies Eq. 5.44 is not found, or it has not enough support, then we try again the whole procedure using a different angle. In practice we have used increments of 5 degrees to have a greater confidence in the detection of the plane. If all angles have been tested and no plane was detected, we try with a different axis of rotation. Again, the axes are tested in a systematic fashion at small increments until the full hemisphere of possible orientations is tested. The whole algorithm is sketched in Algorithm 5.2

Finally, we recognize that the scheme proposed here is far from perfect. The exhaustive search of all angles and rotation axes can be quite time-consuming, and appears to be a little simplistic. Unfortunately, the density of the plane we are seeking varies with the angle of the rotation applied to the set of points. That is, the density of this plane is minimum (the plane spans the full voting space) when the rotation is 180°, and it becomes infinite when we have pure translation (all the points of the plane cluster in a single location in space). Hence there does not seem to be some type of heuristic or constraint we can apply to prune the search. An alternative to this is to use the other three search spaces as described in Eqs. 5.30, 5.31, and 5.32 and perform Tensor Voting to detect these planes to help improve the search method. This is a matter of future research.

However, this disadvantage is only apparent if the magnitude of the transformation is unbounded. Algorithms like the ICP require that the transformation be relatively small. If we use the same limitation in our method, we do not need this exhaustive search, and our method works without an iterative scheme. In this case, our algorithm has a complexity of $O(n^2)$ in the worst case (this will be proven in a subsequent section), where n is the number of tokens in the voting space.

1. Initialize the rotation angle $\alpha = 0°$, and axis $A = [0, 0, 1]^T$
2. Rotate the set X_1 according to α and A.
   Populate the voting space with tokens generated from the candidate correspondences.
3. Initialize all tokens to ball tensors.
4. Perform sparse ball voting and extract the preferred normals.
5. Check for the presence of a set of tokens clustered about a single point in space.
   If this cluster is found, finish and output the corresponding translation detected.
6. Perform sparse plate voting using the preferred normals. Optionally, repeat this step to eliminate outliers. After each iteration, increase the reach of the votes slightly, so as to make the plane grow.
7. Obtain the equation of the plane described by the tokens with highest saliency.
   Perform sparse plate voting between the tokens that lie in this plane.
   Enforce the constraint of Eq. 5.44. Iterate until a stable plane
   is found or a set number of iterations is hit.
8. If a satisfactory plane is found, output the correspondences. Otherwise, increment α and A, and repeat steps 2-7 until all angles and axes of rotation have been tested.

Algorithm 5.2 *General algorithm for the detection of correspondences between two 3D point sets under a single rigid transformation.*

5.8 Tests with synthetic data in 2D

The method just described was validated by performing a large set of experiments. We also compared our method against the Iterated Closest Points algorithm (ICP). The classic ICP was described in [10], but we actually used Fitzgibbon's implementation as described in [21]. Briefly, Fitzgibbon's ICP (F-ICP) is based on an iterative scheme where an error measure is minimized to find the correspondences.

The experimental setup to test these methods is as follows. We generated 50 points uniformly distributed in a 10×10 square centered at the origin. Then, we applied a known rigid motion to this set of points. After this transformation was applied, an equal number of random outliers distributed uniformly throughout the space spanned by both sets, was added to both the model and the data sets. It is well-known that the reliability of F-ICP depends largely on the similarity between the two input sets: the closer the transformation is to the identity, the better it performs. Thus, to try to keep things fair, we chose to constrain the range of rotations between $-5°$ and $5°$ and the translation range to ± 1.5 in both axes.

The results of these tests can be seen in Table 5.1. The success rate is defined by computing the percentage of correct matches produced by the algorithm. If the percentage is at least equal to 50%, we deem the experiment a success. The output of F-ICP is not a set of correspondences, but we computed these simply by selecting the closest points to each other across both sets. It must be noted that using this procedure, we could find a correspondence for *all* the outliers. However, we only counted the correspondences produced for the actual inliers in Table 5.1. Another approach would have been to use a threshold window to reject the outliers, but we did not want to risk rejecting a true inlier with a poorly chosen window in some experiment.

However, given that the core of the problem, as we have formulated it, is finding a line from a set of points, we also compared Tensor Voting against the simpler Hough Transform. It is interest-

Angle	T_x, T_y	Outliers	Success rate	Number of trials
−5°..5°	-1.5..1.5	100 %	75 %	20
−5°..5°	-1.5..1.5	200 %	30 %	20
−5°..5°	-1.5..1.5	300 %	10 %	20
−5°..5°	-1.5..1.5	400 %	10 %	20
−5°..5°	-1.5..1.5	500 %	5 %	20

Table 5.1: Success rates for the F-ICP algorithm.

ing to note that the Hough Transform performs slightly better than Tensor Voting in many cases. However, the disadvantage of the Hough Transform is that its range is limited. Hence, the input space must be normalized so that it always lies within the working range of the Hough Transform, whereas Tensor Voting works regardless of the size of the input space. This normalization has adverse effects for the Hough Transform in many cases. Another advantage of Tensor Voting over the Hough Transform is its ability to detect curves, we will return to this point later.

To keep things fair, we performed a set of tests where the parameters of the transformation where well within working range of the Hough Transform at varying levels of noise. The setup of the experiment was similar to the previous case. We chose the same working space of a 10×10 square centered on the origin. However, in this case, the full range of angles was used and the translation was constrained so that the sets remained in the general vicinity of this square, so as not to disqualify the Hough Transform by applying an out-of-range transformation. In practice, this means that we set the maximum possible translation of ± 10 units in each axis. Then, we added random outliers uniformly distributed throughout the space spanned by both sets, to each set. The results of this experiment at varying degrees of noise are presented in Table 5.2. From this table it is clear that in these cases, the Hough Transform performs better than Tensor Voting (Tensor Voting stops working reliably before the 300% outlier mark, whereas the Hough Transform still works in this situation).

Angle	T_x, T_y	Density	Outliers	HT success	TV success	Trials
0..360°	-10..10	1.0	100 %	100 %	100 %	20
0..360°	-10..10	2.0	200 %	100 %	100 %	20
0..360°	-10..10	2.6	260 %	100 %	80 %	20
0..360°	-10..10	2.8	280 %	100 %	60 %	20
0..360°	-10..10	3.0	300 %	100 %	60 %	20
0..360°	-10..10	4.0	400 %	100 %	15 %	20

Table 5.2: Success rates for the Hough Transform (HT) versus Tensor Voting (TV).

Note that unlike F-ICP, our algorithm is insensitive to the magnitude of the transformation. Therefore we have allowed the full range of rotations and a larger range of translations in our experiments. Also, our algorithm is still stable at 200% outliers, whereas the performance of F-ICP was reduced to approximately 30% by this mark.

We also performed a series of tests where the parameters of the transformation where not so strictly bounded. For this case, we generated points in the same fashion as previously described, but the transformations were now much larger (the translation ranged between 50 and 100 units).

Then, we added random noise evenly distributed throughout the area covered by both sets. Since we had optimized the Hough Transform to work in the original 10×10 square area, a normalization of the data was always needed (or else, the Hough Transform would not detect any lines at all). However, no normalization of the data was needed for the Tensor Voting algorithm. The results of these tests are shown in Table 5.3.

Angle	T_x, T_y	Density	Outliers	HT success	TV success	Trials
0..360°	-100..-50,50..100	0.03	300 %	100 %	100 %	20
0..360°	-100..-50,50..100	0.04	400 %	100 %	100 %	20
0..360°	-100..-50,50..100	0.06	600 %	50 %	100 %	20
0..360°	-100..-50,50..100	0.08	800 %	10 %	100 %	20
0..360°	-100..-50,50..100	0.10	1000 %	0 %	100 %	20
0..360°	-100..-50,50..100	0.12	1200 %	0 %	100 %	20
0..360°	-100..-50,50..100	0.14	1400 %	0 %	100 %	20

Table 5.3: Success rates for the Hough Transform (HT) versus Tensor Voting (TV).

Note how the results of the Hough Transform start deteriorating because of its intrinsic range limits whereas Tensor Voting remains stable. For instance, starting at 600% outliers, the Hough Transform starts detecting false or multiple matches whereas the Tensor Voting remains stable.

This is because of the outlier density to inlier density ratio (shown in both tables under the column labeled "Density"). In the first batch of experiments, the density of the inliers is 0.5 points per square area unit. The density of the outliers at the 100% case is the same. Hence, by adding more outliers, the ratio of the density of outliers versus the density of inliers increases quickly.

On the other hand, in the second batch of experiments, the density of the inliers is *still* 0.5 points per square area unit, but the density of the outliers is reduced to a mere 0.005 points per square area unit since we are adding them over the full space spanned by the transformation. Remember that the Hough Transform works by counting the number of votes for each line, whereas Tensor Voting works by using the local density of the tokens. This explains why in the first set of experiments, the Hough Transform performs better since there are less overall points with a higher density. Accordingly, in the second set of experiments, there are much more points but their density is much lower, hence Tensor Voting works well but the Hough Transform fails.

From these experiments we conclude that our approach seems to be less outlier-sensitive than F-ICP, in general. We believe that F-ICP *might* still work under outlier levels of 1000% in a few cases, but the *reliability* of that algorithm would be rather low. However, our algorithm managed to remain stable throughout all these experiments.

On the other hand, we can also see that the Hough Transform may be more reliable than Tensor Voting in a limited range of transformations; however, Tensor Voting is by far more stable in the more general sense where the range of the transformations is not bounded by any limits. Furthermore, as will be shown later in this thesis, Tensor Voting can also be used to detect general curves in the voting spaces which correspond to certain types of non-rigid transformations. In such cases, the use of the Hough Transform is completely out of the question.

5.9 Tests with synthetic data in 3D

Again, we validated our method for 3D correspondences by performing several experiments with synthetic data. Since the core of the problem is the detection of a 3D plane among a set of points, we considered using the Hough Transform for this. However, unlike the 2D case, the Hough Transform did not prove to be useful in a large portion of the cases. This is because the voting space has 100 times more points due to false matches than the actual inliers (this is, of course, the best case: assuming that no outliers are present in the original data sets) and the actual plane only covers a small region of the space; whereas for the 2D case, the lines we were seeking almost always covered the whole space. Hence, the global maximum in the Hough space almost always corresponds to a random plane generated by the vast majority of the tokens present. Whereas tensor voting is based on the *density* of these tokens which is not only a function of the number of points present, but also on their relative positions. An illustration of this problem appears in Fig. 5.9. In this figure, the left image shows the joint space. The plane detected by Hough is highlighted in Fig. 5.9.b, whereas the actual plane is shown in Fig. 5.9.c.

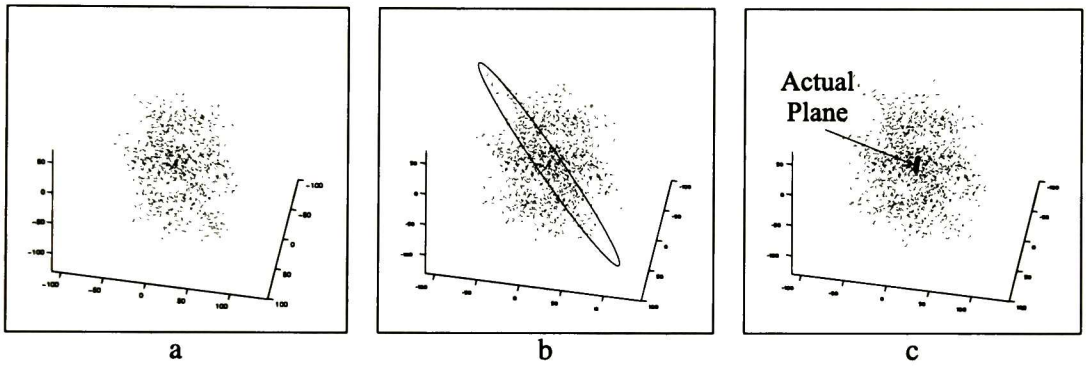


Figure 5.9: a) The joint space $(x' - x), (y' - y), (z' - z)$. b) The incorrect plane detected by the 3D Hough Transform. c) The actual plane we are seeking (correctly detected with our algorithm).

Of course, it can be argued that increasing the resolution of the Hough space will help in the detection of the correct plane, however, due to the large amounts of points that will be tested, the computation time becomes prohibitive. On the other hand, Tensor Voting can be optimized to run with the use of Oct-trees so that only a tiny amount of tokens receive votes at each stage. Hence, we only compared our method against the Iterated Closest Points algorithm (ICP). The classic ICP can be found in [10], though we actually used Fitzgibbon's implementation (F-ICP) as described in [21].

For the case of F-ICP, the tests were conducted in the following fashion. A set of points was generated inside a $10 \times 10 \times 10$ cube centered on the origin. A known rigid transformation was then applied to this set. The bounds of these points are computed and the same amount of random outliers is added to both sets. The outliers are distributed uniformly throughout the bounds set by the original sets.

It is well-known that the reliability of F-ICP depends largely on the similarity between the two input sets: the closer the transformation is to the identity, the better it performs. Since our algorithm effectively rotates one set of points until it finds a solution, it can be considered that we too, are dependent upon the quality of the initialization. Therefore, to try to keep things fair, we chose to constrain the range of rotations between $-5°$ and $5°$ and the translation range to ± 1.5

in all axes while testing the F-ICP algorithm. The angles, axis of rotation and translations were generated randomly following an uniform distribution. The range of the axis of rotation was largely immaterial and therefore covered the whole hemisphere of possible orientations. An experiment is deemed a success if at least 50% of the correspondences are correctly identified. The results of these tests, are presented in Table 5.4.

Angle	T_x, T_y, T_z	Outliers	Success rate	Trials
$-5..5°$	-1.5..1.5	100 %	90 %	20
$-5..5°$	-1.5..1.5	200 %	80 %	20
$-5..5°$	-1.5..1.5	300 %	60 %	20
$-5..5°$	-1.5..1.5	400 %	67 %	20
$-5..5°$	-1.5..1.5	500 %	50 %	20

Table 5.4: Success rates for the F-ICP algorithm.

Our algorithm was tested in a similar fashion. However, in this case, the full range of possible rotation angles was employed. The translation was also increased to lie between -10 and 10 units. The results of these experiments can be seen in Table 5.5. It will be noted that we did not include the density ratio column in this table, as we did for the 2D case. This is because the actual density of the plane varies with the rotation between both sets of points. The density of the plane reaches a minimum of 0.05 points per cubic volume unit at 180° and a maximum of ∞ at 0° (all the tokens become clustered in the same point).

Angle	T_x, T_y, T_z	Outliers	TV success	Trials
0..360°	-10..10	300 %	100 %	20
0..360°	-10..10	400 %	90 %	20
0..360°	-10..10	600 %	85 %	20
0..360°	-10..10	800 %	85 %	20
0..360°	-10..10	1000 %	80 %	20
0..360°	-10..10	1200 %	70 %	20

Table 5.5: Success rates for the estimation of correspondences for 3D rigid motion using Tensor Voting.

As can be appreciated from these tables, our algorithm still performs *reliably* at 300% outliers, whereas F-ICP only succeeds about 60% of the times at this mark. Also, the performance of our algorithm deteriorates at much slower pace than F-ICP in general. This shows that, just as in the 2D case, F-ICP does not withstand as much outliers as our algorithm.

5.10 Analysis of the algorithm

The complexity analysis is rather straightforward. Let us consider the first voting step of the 2D case where a sparse ball voting is performed. In each voting space there will be n tokens. Suppose that in average, these tokens reach m neighbors with their voting fields. Then, the average number of votes cast is nm. Therefore, the complexity of this step is $\bar{O}(nm)$ where n is the

number of tokens in each voting space and m is the number of tokens within reach. Note that all subsequent voting is also of the same order. Hence, the overall complexity of our algorithm is $O(mn)$, in average. In the worst case, where every single token casts a vote to every other token, the complexity increases to $O(n^2)$ (note that in practice this will seldom happen).

For the 3D case, the analysis is quite similar. On average, each voting stage will be of order $O(mn)$ where n is the number of tokens in the voting space and m is average the number of tokens within reach. This becomes $O(n^2)$ in the worst case, where every single token receives a vote from every other token.

The complexity of the algorithms, in their most general implementation (voting space skewing for the 2D case, test-and-hypothesize runs, iterations to make the lines or planes grow slowly) is $O(kmn)$ where k is the number of hypothesize-and-test runs. This also makes these algorithms run rather slow. It may take from several minutes to hours to compute a single 2D or 3D motion with 1000 % outliers. However, bear in mind that this is an excessive scenario. Most applications rely on the assumption that the motion between both data sets is small and, more importantly, *that there are no outliers*. If we apply these same constrains to our algorithm, we can produce an optimized version that produces the desired results in a fast way. This is possible mainly because in these cases no iterations are needed at all, and the problem is solved in two or three tensor voting passes.

5.11 Experimental results: 2D case

5.11.1 Range readings

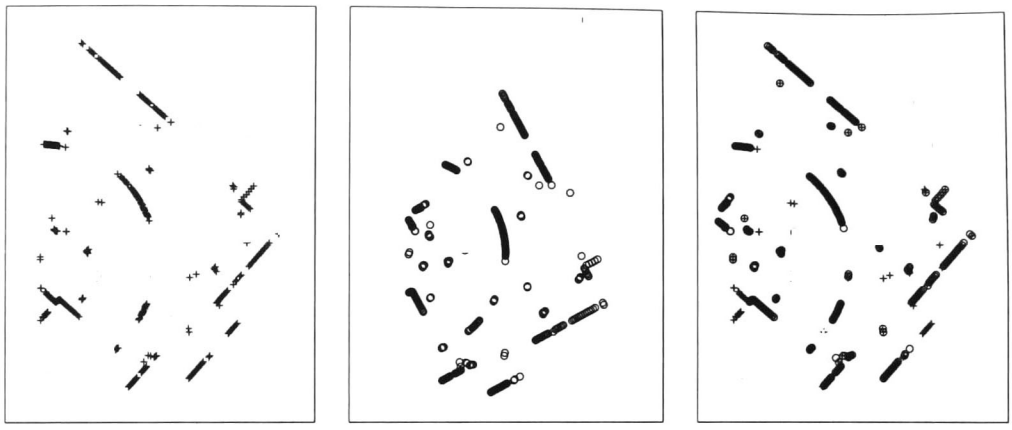
We tested our 2D engine with various experiments. For the first experiment, we aligned to sets of points obtained with a laser range scanner mounted on a mobile robot In the 2D case, we tested our method with some range readings taken with a laser system mounted on a mobile robot [1] We took two non-consecutive readings after the robot had performed both an unknown rotation and translation. The matches were computed using the procedure described in previous sections with the difference that instead of populating the voting spaces with a full matching scheme, a search window was used instead. With the correspondences thus obtained, the motion between the frames was computed using the standard Direct Linear Transform (DLT) algorithm. The original data sets and the aligned sets are shown in Fig. 5.10.

5.11.2 Experiments with images

In another experiment, we used our algorithm to register a couple of images of a plane with a random pattern performing an unknown rigid transformation (Figures 5.11.a and b). The images were first processed to compensate for any projective distortion produced by the camera and then, the Harris corner detector was used to find features of interest. Note that the features extracted need not be the same number in both images and no initialization of correspondences is needed. The resulting matches produced by our algorithm were used to compute a rigid transformation between the two images and register them (Fig. 5.11.c).

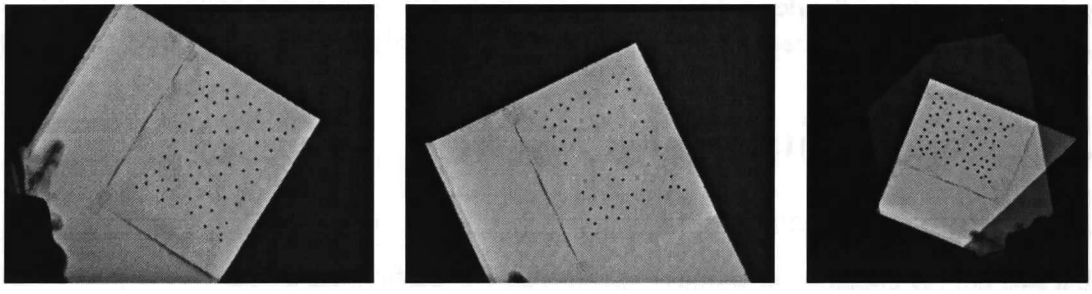
It is worth noting that in this case, due to the noise in the detection of the features, our algorithm had to be adapted to account for this. Thus, instead of looking for a perfect line in the joint space, we looked for a small band of points lying nearly on a line. The search criteria were

[1] Thanks to Denis Wolf of the Robotics Embedded Systems Laboratory of the USC for providing the data.



a b c

Figure 5.10: a) and b) Two laser range readings. c) The alignment produced by our algorithm.



a b c

Figure 5.11: a) and b) Two images of a plane performing an unknown rigid motion. c) The alignment produced by our algorithm (The features have been enhanced for clarity).

accordingly relaxed. This, however, allowed for a few multiple matches to "pass through" the original algorithm. To correct this situation, an additional final step was employed to reject the multiplicity of correspondences based on the closeness to the lines found in the joint spaces. Figure 5.12 illustrates this situation. In that figure, we have two tokens x_i and x_j which violate the uniqueness constraint. In order to reject one or the other, we fit a line to all the inliers and choose the token with the smallest distance to this line. In our particular example, x_j would be chosen.

5.11.3 Multiple overlapping motions

Another advantage of our algorithm is that it can easily be extended to account for multiple overlapping motions. In order to detect multiple motions, we apply the algorithm as described in Algorithm 5.1; but in the last stage, instead of just taking the token with the highest saliency and assume that it belongs to the only line present, we first group all existing lines and then perform the final stage for each line separately.

The grouping of lines proceeds as follows. First, we create two lists of line equations (one for each joint voting space). Then, for each active token, we compute the equation of the line passing through this token based on its position and its preferred orientation. If the equation of the line

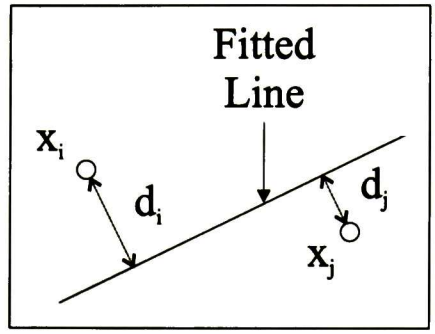


Figure 5.12: If two tokens x_i and x_j violate the uniqueness constraint, only the point lying closer to the fitted line is chosen.

is similar to one of the equations that has already been stored in the list, we update the equation on the list by averaging both equations and we add the point to the set of points belonging to this equation. If, on the contrary, no line equation is found that is similar enough with the current line, we simply add a new line to the list, create a new set of points and add the current token to this set. We repeat this algorithm until all tokens have been assigned to a line equation. Then, we find corresponding line pairs across both voting spaces (i.e., the lines that roughly satisfy Eq. 5.13 and discard the rest. Finally, for each pair of lines we run the last stage of the algorithm to discard possible outliers and enforce the uniqueness constraint.

We present a synthetic example with three overlapping motions in a 10 × 10 square in Fig. 5.13.a. The resulting correspondence sets as detected by our algorithm are shown in Figures 5.13.b-d. Note that the traditional approaches like ICP and Chamfer matching cannot deal with multiple overlapping motions.

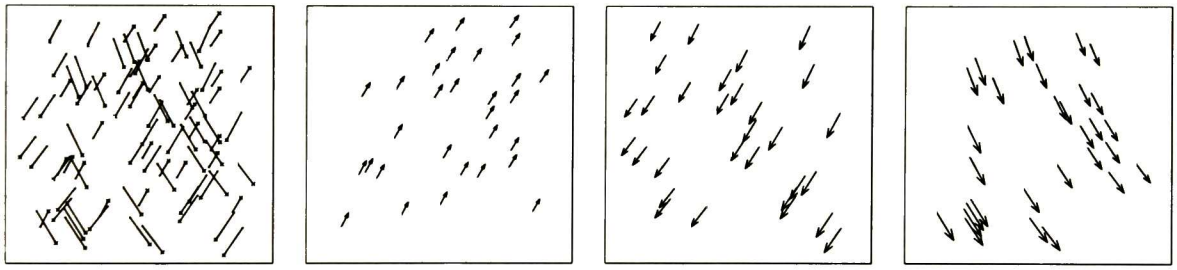


Figure 5.13: a) Three overlapping motions (dots map to crosses). b),c) and d) The resulting correspondence sets as detected by our algorithm.

We also performed an experiment in the field of object recognition. In this case, we took a picture of some pliers (see Fig. 5.14.a), and computed its outline using the Canny edge detector. Then, we placed the pliers and *opened them* (see Fig. 5.14.b), taking another picture and again detecting the edges in the same fashion. However, since the model and the target both had a structure, we used this to our advantage. Instead of blindly matching all points of the model with all the points in the target, we first computed the local tangents of the edges using sparse ball voting.

We used these tangents to guide the matchings (only points with similar tangents were matched) and prune the voting space by limiting the number of outliers. Other than this, we applied our

algorithm as usual. Several solutions were provided by our algorithm at different angles, however, we chose the two solutions that yielded the highest number of matches. The model and the target object can be seen in Fig. 5.14.c. The first solution found with our algorithm can be seen in Fig. 5.14.d, and the second solution in Fig. 5.14.e. The multiplicity of solutions is due to the symmetry of the pair of pliers themselves.

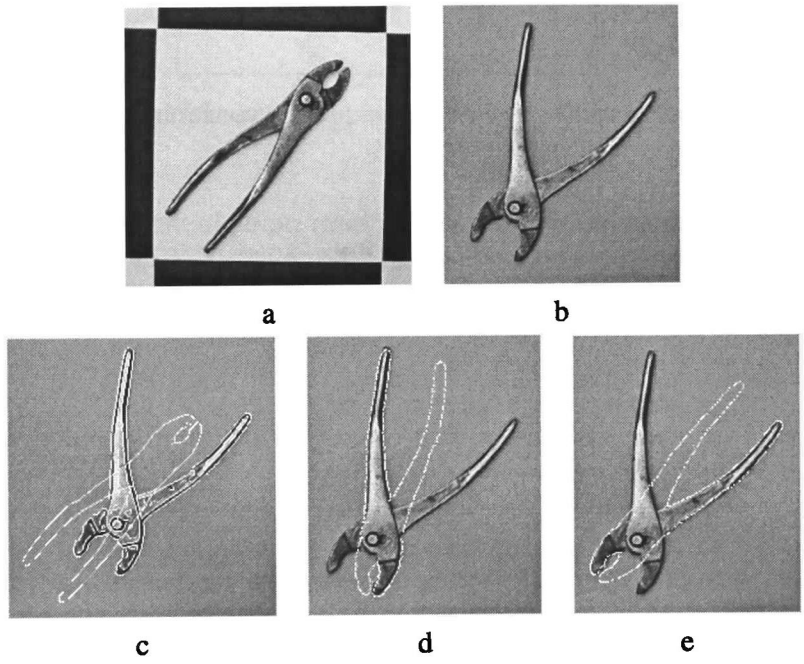


Figure 5.14: a) Image of the pliers used as model. b) Target object. c) The model and the target edges overlapped. d) and e) The two solutions found by our algorithm.

5.11.4 Extension to 2D non-rigid motion

Our algorithm can also detect some instances of non-rigid motion. Namely, those that still produce detectable curves in the voting spaces. We have found that point sets that display smoothly-varying rigid transformations produce smooth curves in the voting spaces. For example Fig. 5.15.a displays one such set. We generated this set by slowly varying the rotation angle over the length of the string of points. The voting spaces display two clearly salient curves, as seen in Fig. 5.15.b.

In order to detect these curves, we employ the Algorithm 5.1. However, in the last stage, instead of globally enforcing the reflection constraint of Eq. 5.13, we only require that each token is locally consistent with this constraint across both voting spaces, and reject the rest. This is illustrated in Fig. 5.15.b. Here we show two tokens along with their normals, the arrows across the voting spaces show which tokens are used to enforce the Reflection Constraint locally. The resulting correspondences in this synthetic test are shown in Fig. 5.15.c.

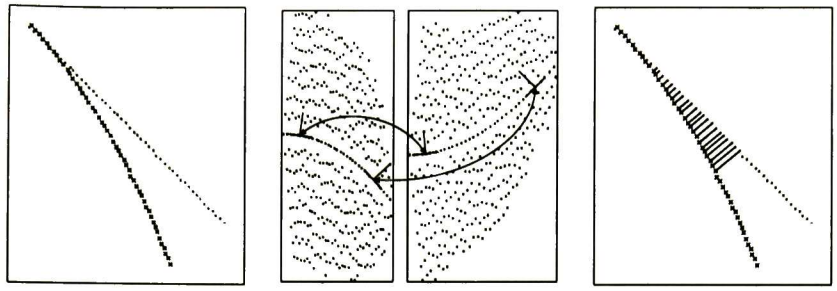


Figure 5.15: a) The set of points that were applied a non-rigid transformation (dots transform to crosses). b) The curves generated in the voting spaces. The arrows across the voting spaces show the local enforcement of the Reflection Constraint. c) The correspondences found with our algorithm.

5.12 Experimental results: 3D case

In the 3D case, we performed the following experiments. First, we followed the position of a robotic arm in 3D in a sequence of stereo pairs (Fig. 5.16 shows only the left images of the sequence). The model of the object was picked by hand from the first reconstruction and then we computed the position of the arm in the subsequent reconstructions using an optimized version of our algorithm (namely, it only consisted of two stages: sparse ball voting and sparse stick voting, no iterations were used). Note that the sequence of images do not form a video, hence, the features cannot be tracked between successive frames due to the relatively large differences between the snapshots. After the motion was computed, the position of the arm in 3D was reprojected on the images as shown in Fig. 5.16.

In a second experiment we made a reconstruction of a Styrofoam model of a head using a stereo camera. The two reconstructions are shown in Fig. 5.17.a-c. The aligned sets can be seen in Fig. 5.17.d-f. In this case, however, another optimization was used. Since the sets are close to each other, and the points provide enough structure, we used tensor voting to compute the preferred normals at each site.

The computation of the normals proceeds as in standard sparse tensor voting. First, we initialized each point to a ball tensor. Then, we placed a normal ball voting field on each point and cast votes to all the neighbors. Then, the preferred normal at each site is obtained by computing the eigensystem at each point and selecting the eigenvector with the greatest eigenvalue. A close-up of the surface of the model and some of the normals found by this method is shown in Fig. 5.17.g. We used this information to prune the candidate matches to those that shared a relatively similar orientation only.

Also, note that in this case, there are non-rigid differences between both sets, due to the noise in the reconstruction process. This can be noted in places like the shin, where the alignment could not be made simply because the size of this section of the reconstruction differs slightly between both sets. However, our algorithm still manages to align the sets as much as possible using a rigid transformation. Thus, it can be noted that the forehead and upper part of the nose were aligned successfully.

Finally, in our last experiment, we aligned a model of a Toyota car taken with a laser range scanner and aligned it with a noisy reconstruction performed with a stereo camera. The noisy

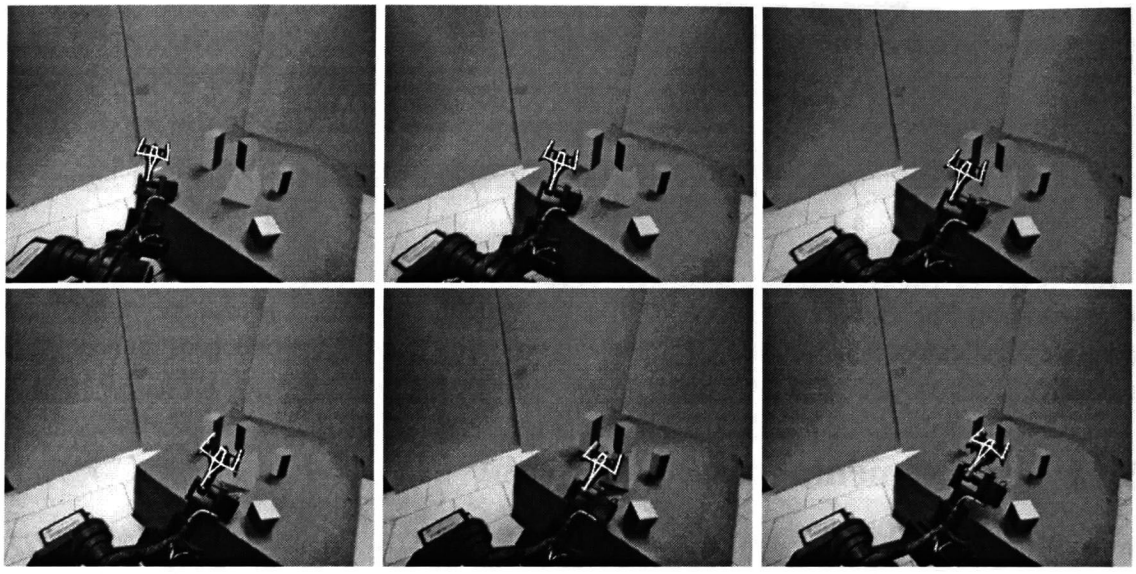


Figure 5.16: Sequence of (left) images from a stereo camera showing the position of the reprojected arm. This is not a video.

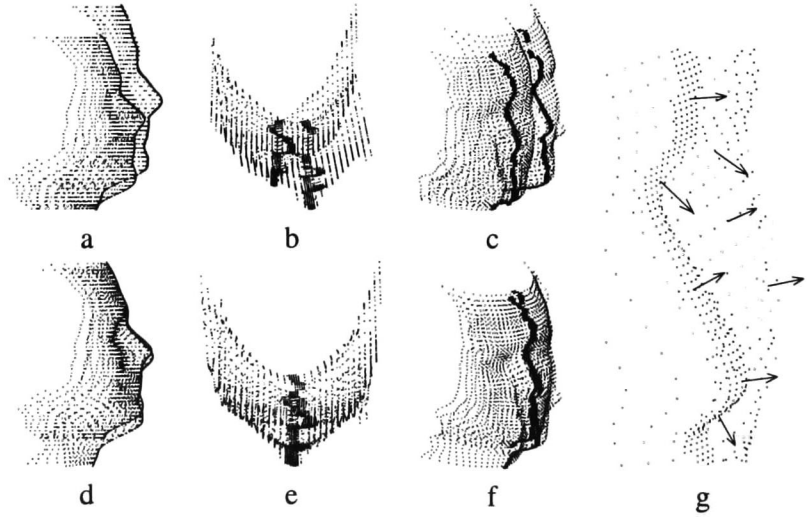


Figure 5.17: a)-c) Sets to be aligned. d)-f) Sets after alignment. g) Close-up of the surface of the model showing some of the normals computed using tensor voting.

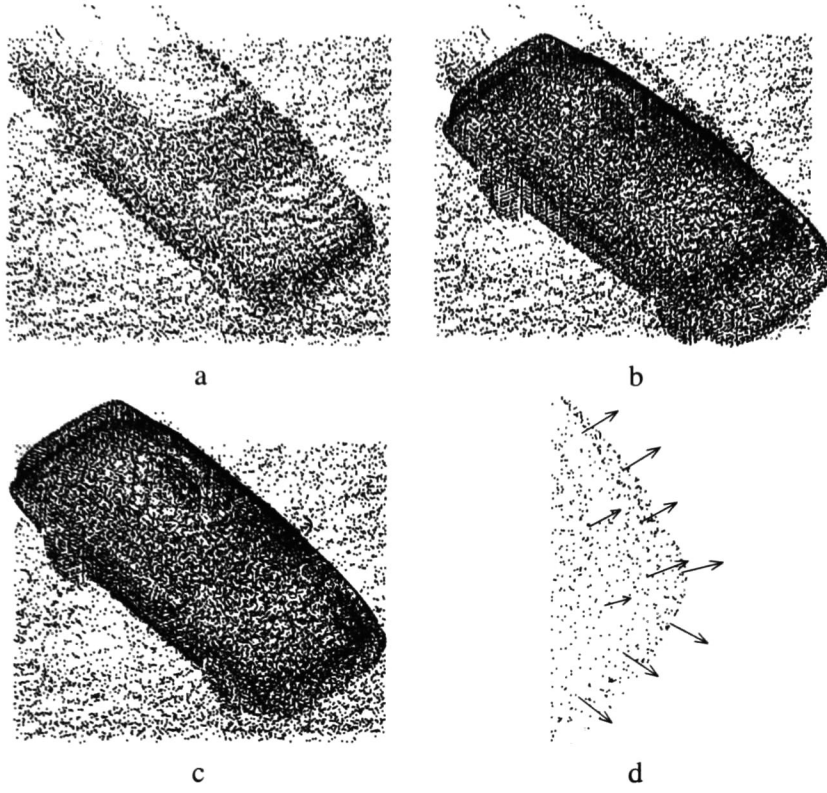


Figure 5.18: a) Target for alignment, note the noisy surface. b) Model displayed over the target. c) Model and data after alignment. d) Close-up of the surface of the model showing some of the normals computed with Tensor Voting.

target is shown in Fig. 5.18.a, the model and the target are shown in Fig. 5.18.b, and the final alignment in Fig. 5.18.c. The procedure is the same as in the previous case. Again, since the data sets provided structure, we used it to our advantage by computing the preferred normals using tensor voting and pruning the candidate matches as described previously (Fig. 5.18.d).

5.13 Multiple overlapping motions

As noted previously, another advantage our method has over ICP and similar methods is the ability to simultaneously detect multiple overlapping motions. This is also true for the 3D case. In this case, each different motion simply produces another plane in the voting space. There are limitations to the motions that can be differentiated, though. A quick analysis of Eq. 5.33 reveals that if two different motions share the same axis of rotation and same overall translation, then they will span the same 3D plane in the voting space. However, in these circumstances, it suffices with analyzing the other three voting spaces (Eqs. 5.30, 5.31 and 5.32) to disambiguate this case.

To illustrate this, we present a synthetic example where three overlapping motions with different axes of rotation, angles and translations were generated in a $10 \times 10 \times 10$ cube centered at the origin (see Fig. 5.19). Our algorithm is applied as described in Algorithm 5.2. However, after the first plane was detected, we removed its tokens from the voting space and the process

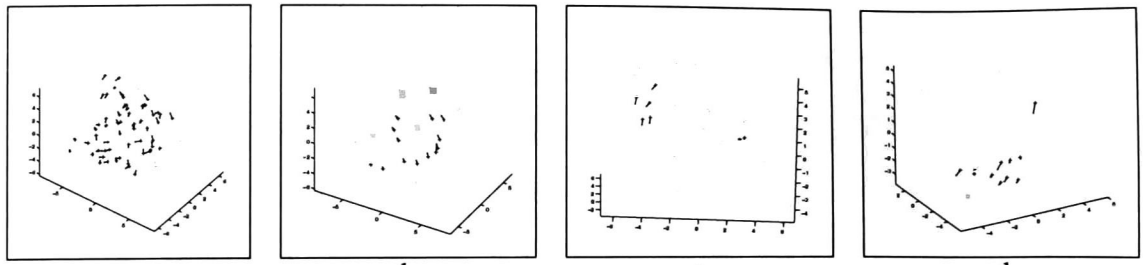


Figure 5.19: a) Three overlapping rigid motions in 3D. b)-d) The different motions as detected by our algorithm.

was repeated until no more planes were found. This is, of course, the naive implementation of the solution. However, the algorithm can be modified to account for the presence of multiple planes. In that case, only the final stage, where the constraint from Eq. 5.44 is enforced, would be executed separately for each set of points.

5.14 Extension to 3D non-rigid motion

While it can still be argued that, with some work, the Hough Transform might also be used to detect the same plane we obtain through Tensor Voting, there is another advantage to using the latter over the former: Tensor Voting enables us to find general surfaces. This means that we can also detect certain non-rigid motions that produce non-planar surfaces in the voting spaces.

To illustrate this, we generated a synthetic plane and then applied a twist transformation to it (see Fig. 5.20.a). This transformation produces a curved surface in the voting space (clearly visible in the center of Figs. 5.20.b-c, a close-up of the surface is also presented Fig. 5.21). The surface is easily detected using Tensor Voting and the resulting correspondences, from two different viewpoints, can be seen in Figs. 5.20.d-e.

In order to detect this surface, we had to modify our algorithm as follows. The first two stages (sparse ball voting and sparse stick voting) are performed as usual. However, in the last stage, Eq. 5.44 was not enforced globally, but only locally around each active token. In other words, we enforced presence of rigid transformations only on a local level. It must be remembered that Eq. 5.44 depends on two points. Therefore, for each token that was verified, we used the closest active neighbor. We illustrate this in Fig. 5.21. In that figure, the token x_i is being verified using its closest neighbor, x_j. The normals of the tokens are also shown.

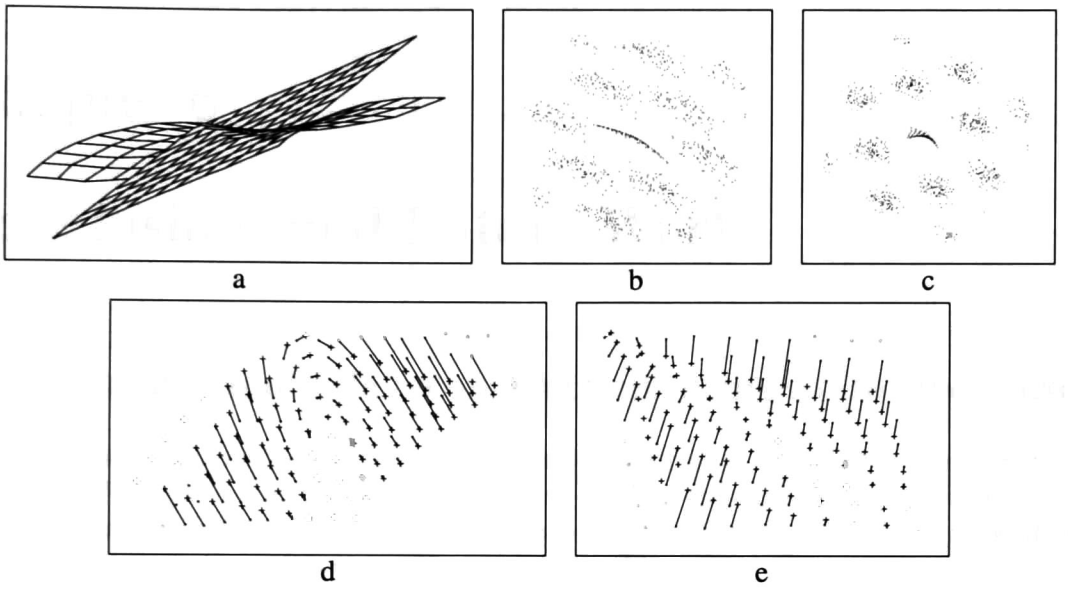


Figure 5.20: a) Non-rigid motion applied to a 3D plane. b) and c) The curved surface that was generated in the voting space from two different view points. d) and e) The resulting correspondences found with our algorithm seen from two different viewpoints.

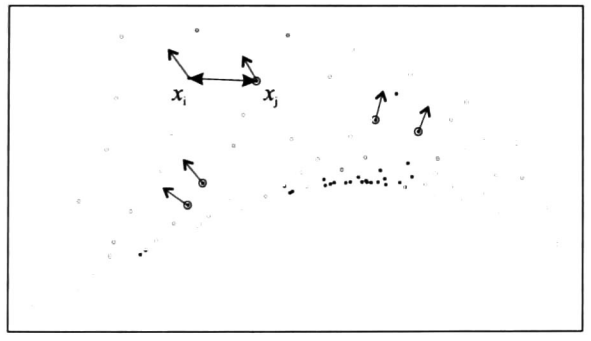


Figure 5.21: A close-up of the surface corresponding to an elastic motion. The constrains of Eq. 5.44 are only verified locally between the closest point pairs. In this figure, token x_i is verified with its closest neighbor, x_j. Other pairs to be verified are also highlighted in the figure.

Chapter 6

Conclusions and Future Work

6.1 Fusion of multiple reconstructions using calibrated sensors

In the calibrated case, we have demonstrated a simple but effective algorithm to compute the axes of rotation of the pan-tilt unit. We have used these axes to compute the transformation between the camera and the robot without the need for re-calibration, thus enabling the superposition of reconstructions taken from different positions into a single map.

The algorithm presented works with any sensor. We have proven this by incorporating information gathered from the laser system into the maps reconstructed with the stereo system. In particular, we have used motion estimation techniques on line measurements obtained with the laser system to guide our robot with enough precision to perform simple grasping and reaching tasks.

6.2 Uncalibrated reconstruction

In the uncalibrated reconstruction part, we have presented a method that simultaneously estimates points, lines, quadrics, plane conics, degenerate quadrics and cameras. The advantage of this method is that it does not impose any kind of constraints on the scene, e.g. there's no need for planarity and the features are not required to be visible in all views. The only requirement is that there is enough information in a couple of images to compute the Fundamental matrix. However, we have also tested initializing with the Trifocal tensor. Even though the initial reconstruction is better using the Trifocal tensor, the final scene after Bundle Adjustment is nearly the same in both cases. The gain in precision is only meager. We expect similar results with the Quadrifocal tensor.

The problem of line orthogonalization was properly addressed and we found out that orthogonalization is not needed at each iteration of the Bundle Adjustment. A final correction step suffices.

We have also presented an algorithm for the correction of conic outlines in n-views, and a geometric constraint (Eq. 4.67) that can be implemented inside the Bundle Adjustment iterations. This constraint does not preserve the topology of the quadrics by itself. However, it must be stressed that this constraint prevents the parameters of the quadrics to evolve freely and keeps them close to an epipolar-consistent state. A secondary effect of this constraint — albeit, not less important —, is that it effectively prevents the system from reaching a global minimum when this implies a great increase in the error in the epipolar-consistency check. This means that reaching the global minimum is *not always desirable*, for it may produce quadrics that are *topologically*

incorrect. Thus, we have designed a *geometric constraint* that helps in determining when Bundle Adjustment should stop.

6.3 Estimation of correspondences in 2D and 3D

We have also presented a novel non-iterative algorithm that combines the power of expression of Geometric Algebra with the efficiency of Tensor Voting to find the correspondences between two sets of 2D or 3D points with an underlying rigid transformation. This algorithm was also shown to work with excessive amounts of outliers in both sets.

We have also used Geometric Algebra to derive two sets of constraints (Eq. 5.13, for the 2D case and Eq. 5.44 for the 3D case) that serve a double purpose: on one hand, it lets us decide whether or not the current plane (or lines) corresponds to a rigid motion; and on the other hand, it allows us to reject multiple matches and enforce the uniqueness constraint.

Our algorithm does not require an initialization (though it can benefit from one). Works equally well for large and small motions. And can be easily extended to account for multiple overlapping motions and even certain non-rigid transformations.

It must be noted that our algorithms can detect multiple overlapping motions, whereas the current solutions only work for a single global motion. We have also shown that our algorithm can work with data sets that present small non-rigid deformations. During our synthetic experiments in the 3D case we also noticed that some types of non-rigid motion produce *3D curves* instead of surfaces in the voting space. The next logical step is to explore what other types of motion can be detected, and to extend this to multiple non-rigid motions.

In the unconstrained case, with a large amount of outliers (500-1000%), our algorithm can take several minutes to finish. However, in most real-life applications, these extreme circumstances are not found, and a good initialization can be computed. When these conditions are met, our algorithm can be rather fast.

It is worth noting the effect of the "sampling rate" of the input points in our algorithm. Our method is based on an equation for point motion and assumes a direct point-to-point correspondence can be established. If the input data sets have been sampled at different rates, this condition will not be met and the performance of our algorithm will degrade, maybe even stop working altogether. We are currently working on ways to overcome this problem.

Another aspect of our 3D algorithm is that we must rotate one of the sets of points in order to densify the plane we are looking for. We are currently exploring other ways to make this more efficient. One possible solution might come from considering all four voting spaces simultaneously. Up until now, only the space produced by the trivector part was considered and the other spaces were only used in an implicit fashion to provide extra constraints to identify valid rigid motions. Maybe these other spaces can be used to direct the search of the transformation we are seeking, but this is a subject for future research.

6.4 Final thoughts

It has come to our attention one interesting fact. The general correspondences problem might be considered as an NP problem, since it consists of two parts: the non-deterministic generation of a set of correspondences, and the testing of the correctness of the set. In our particular case, the complexity of the problem is reduced significantly by using the extra information that a rigid

transformation has taken place. This alone serves to diminish the complexity of the problem from NP to P time.

Let us focus this discussion on the 2D case first. Assume we have two sets of points $\mathtt{X}$ and $\mathtt{Y}$. For each $\mathbf{x}_i \in \mathtt{X}$ there is one, and only one $\mathbf{y}_j \in \mathtt{Y}$ such that $\mathbf{y}_i = \mathtt{H}\mathbf{x}_j$, where $\mathtt{H}$ is 2D rigid transformation consisting of an arbitrarily large rotation and translation. We are expected to find the correspondences mapping $\mathtt{X}$ to $\mathtt{Y}$ via $\mathtt{H}$.

The naive approach to solve this problem would be as follows: generate a tentative correspondences set between $\mathtt{X}$ and $\mathtt{Y}$, compute the $\mathtt{H}$ generated by this set and test if the points $\mathtt{X}$ are actually mapped to the points $\mathtt{Y}$. The non-deterministic part of the algorithm is the generation of the correspondences set. However, this can be easily formulated in a systematic fashion and, since we are dealing with a 2D rigid transformation, the set does not need to comprise all the points in $\mathtt{X}$, but only enough to account for the three degrees of freedom of $\mathtt{H}$ (two points).

The algorithm to choose the correspondences may proceed as follows. Select $\mathbf{x}_i \in \mathtt{X}$ for each $i = 1, .., n$ where n is the number of points in $\mathtt{X}$. For each $\mathbf{x}_i$ chosen, select a tentative corresponding point $\mathbf{y}_j \in \mathtt{Y}$ for each $j = 1, .., n$, where again, n is the number of points in $\mathtt{Y}$ (see Fig. 6.1.a). This can be easily coded with two nested iterations, one to iterate over i and the other to iterate over j. The total number of possible selections is therefore $O(n^2)$. Now, if we consider that we need two such correspondences, it is evident that we must nest four iterations (two iterations for each pair) and therefore the algorithm is $O(n^4)$ in the worst case. Since the computation of $\mathtt{H}$ given a set of tentative correspondences is of $O(1)$, this does not increase the complexity of the problem.

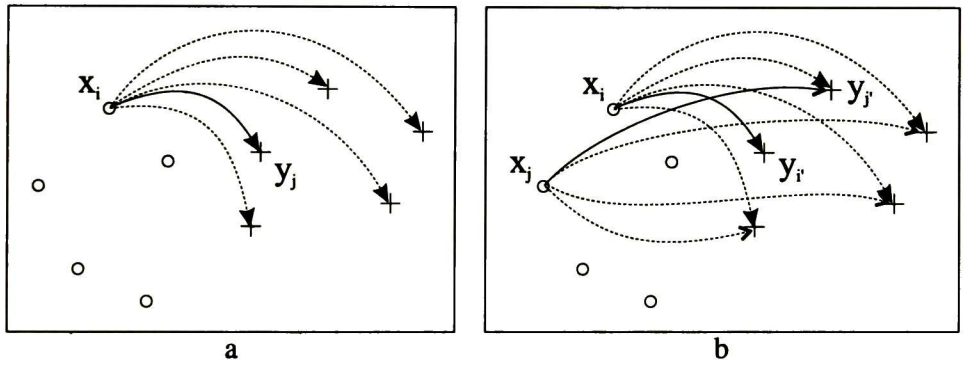


Figure 6.1: a) A set of points $\mathtt{X}$ (circles) and the same set after a rigid transformation has been applied $\mathtt{Y}$ (crosses). A tentative set of correspondences between one point $\mathbf{x}_i$ and the set $\mathtt{Y}$ is shown. b) The same set, in this case, two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are chosen and left fixed and all the possible correspondences are then considered (shown).

This simple scheme can be further optimized. We can choose two points $\mathbf{x}_i$ and $\mathbf{x}_j$ from $\mathtt{X}$ and leave them fixed. Then we can iterate over the possible selections of the corresponding $\mathbf{y}_{i'}$ and $\mathbf{y}_{j'}$ for these points (see Fig. 6.1.b). This operation is of $O(n^2)$ since only two cycles are nested, one to iterate over i' and another to iterate over j'.

In the 3D case a similar analysis can be performed. In this case we have six degrees of freedom to account for. But since each point $\mathbf{x}_i$ introduces three constraints, we only need two points to compute $\mathbf{H}$. Therefore the complexity is the same: $O(n^2)$. Note that this naive algorithm will be capable of finding the correct answer regardless of the magnitude of the transformation applied.

On the other hand, in current approaches like ICP, the generation of correspondences is much more efficient but is based on the assumption that corresponding points are close to each other. Therefore, algorithms like the ICP will not work *even in the absence of outliers* if the transformation is large. Another problem of the ICP is that the solution produced is highly dependent on the quality of the initialization. This algorithm might become trapped in a "local minimum" and thus might not produce the desired answer.

We have already shown that our algorithm is of $O(nm) < O(n^2)$ where $m < n$ is the number of neighbors that receive votes from the current token. In the 2D case the reader might remember that we also proposed a hypothesize-and-test scheme where different skewing angles were tested in a systematic fashion. However, we must stress that when no outliers are present, we can detect the correct transformation without skewing the space *regardless of the magnitude of the transformation* (see Fig. 5.6 for an example of this). Hence, no extra iterations are needed, and our algorithm does provide a real decrease in computational costs. This has been possible because we have re-cast the correspondences problem into another space where a particular constraint of the problem was more readily detected and exploited. It is interesting to note that this space was found by using the Geometric Algebra.

Finally, we have discussed how a general NP problem could be solved using a simple algorithm that runs in P time thanks to the use of extra knowledge that constrains the problem. Namely, that a rigid transformation has taken place between the sets we must match. This same constraint is not properly exploited by other algorithms that also solve this problem. In the case of ICP, for example, the problem is further limited by bounding the magnitude of the transformation to a relatively small range. In our case, we have applied a transformation to the problem, taking it into another space where the aforementioned constraint is readily detectable, without introducing extra limitations to the problem (like bounding the magnitude of the transformation). Thus producing a more efficient algorithm by exploiting this powerful constraint.

The author is tempted to think about possible generalizations of this particular result. Maybe similar approaches could be used in other problems. It is granted that we are employing a very specific constraint to this case; but perhaps the solution to the NP-completeness problem lies in casting the problems into different search spaces that require simpler, but more powerful detection techniques?

Appendix A

Numerical Algorithms

In this appendix, we briefly describe two algorithms that were mentioned in various parts of this document. These algorithms and several others can be found in [27]

A.1 Least-Squares Solution of Homogeneous Equations

Throughout this document, several problems were solved by finding the solution to a set of equations of the form $\mathtt{A}\mathbf{x} = \mathbf{0}$. In particular, we consider the case when there are more equations than unknowns (the system is over-determined). The trivial solution $\mathbf{x} = \mathbf{0}$ is not of interest. Observe that if $\mathbf{x}$ is a solution to the set of equation, so is $k\mathbf{x}$ for any scalar k. Therefore, we need to constraint the norm of $\mathbf{x}$ when seeking a solution. A reasonable choice is to fix $\|\mathbf{x}\| = 1$. In general, such a set of equations will not have an exact solution. Suppose $\mathtt{A}$ has dimension $m \times n$, then there is an exact solution if and only if $rank(\mathtt{A}) < n$. In the absence of an exact solution, we will normally seek a least-squares solution. Therefore problem may be stated as

- *Find the* $\mathbf{x}$ *that minimizes* $\|\mathtt{A}\mathbf{x}\|$ *subject to* $\|\mathbf{x}\| = 1$

This problem can be solved as follows. Let $\mathtt{A} = \mathtt{U}\mathtt{D}\mathtt{V}^T$ The problem then requires us to minimize $\|\mathtt{U}\mathtt{D}\mathtt{V}^T\mathbf{x}\|$. However, $\|\mathtt{U}\mathtt{D}\mathtt{V}^T\mathbf{x}\| = \|\mathtt{D}\mathtt{V}^T\mathbf{x}\|$ and $\|\mathbf{x}\| = \|\mathtt{V}^T\mathbf{x}\|$. Thus we need to minimize $\|\mathtt{D}\mathtt{V}^T\mathbf{x}\|$ subject to the condition $\|\mathtt{V}^T\mathbf{x}\| = 1$. We write $\mathbf{y} = \mathtt{V}^T\mathbf{x}$, and the problem is: minimize $\|\mathtt{D}\mathbf{y}\|$ subject to $\|\mathbf{y}\| = 1$. Now, $\mathtt{D}$ is a diagonal matrix with its diagonal entries in descending order. It follows that the solution to this problem is $\mathbf{y} = [0, 0, .., 0, 1]^T$ having one non-zero entry, 1 in the last position. Finally, $\mathbf{x} = \mathtt{V}\mathbf{y}$ is simply the last column of $\mathtt{V}$. The procedure is summarized in Algorithm A.1.

Alternatively, the last column of $\mathtt{V}$ may be described as the eigenvector of $\mathtt{A}^T\mathtt{A}$ corresponding to the smallest eigenvalue.

Problem: Find the vector $\mathbf{x}$ that minimizes $\|\mathtt{A}\mathbf{x}\|$ subject to $\|\mathbf{x}\| = 1$.

Solution:

1. $\mathbf{x}$ is the last column of $\mathtt{V}$, where $\mathtt{A} = \mathtt{U}\mathtt{D}\mathtt{V}^T$ is the SVD of $\mathtt{A}$.

Algorithm A.1 *The solution to the equation system* $\mathtt{A}\mathbf{x} = \mathbf{0}$.

A.2 Constrained Minimization

Sometimes the minimization is subject to constraints that can be expressed as a matrix multiplication. The algorithm discussed here to solve this problem was presented in [26]. The problem can be stated as follows: given a matrix $\mathtt{A}$, and a constraint matrix $\mathtt{C}$, we wish to find the unit norm vector $\mathbf{q}$ that minimizes $\|\mathtt{Aq}\|$ subject to $\mathbf{q} = \mathtt{Ca}$ for some vector $\mathbf{a}$. This is equivalent to minimizing $\|\mathtt{ACa}\|$ subject to $\|\mathtt{Ca}\| = 1$, then set $\mathbf{q} = \mathtt{Ca}$. The procedure to solve this problem is presented in Algorithm A.2.

Problem: minimize $\|\mathtt{ACa}\|$ subject to $\|\mathtt{Ca}\| = 1$, where $\mathbf{q} = \mathtt{Ca}$.

Solution:

1. Compute the SVD $\mathtt{C} = \mathtt{UDV}^{\mathsf{T}}$ such that the non-zero values of $\mathtt{D}$ appear first down the diagonal.
2. Let $\mathtt{U}'$ be the matrix comprising the first r columns of $\mathtt{U}$, where r is the rank of $\mathtt{C}$. Further, let $\mathtt{V}'$ consist of the first r columns of $\mathtt{V}$ and $\mathtt{D}'$ consist of the r first rows and columns of $\mathtt{D}$.
3. Find the unit vector $\mathbf{q}'$ that minimizes $\|\mathtt{AU}'\mathbf{q}'\|$. This is the singular vector corresponding to the smallest singular value of $\mathtt{AU}'$
4. The required vector $\mathbf{q}$ is given by $\mathbf{q} = \mathtt{U}'\mathbf{q}'$. A vector $\mathbf{a}$ such that $\mathbf{q} = \mathtt{Ca}$ is given by $\mathbf{a} = \mathtt{V}'\mathtt{D}'^{-1}\mathbf{q}'$.

Algorithm A.2 *The solution to the constrained minimization problem.*

Appendix B

Computation of the orthogonal distance between a point and a conic

This algorithm has been previously discussed in [62]. However in that report, there was a minor error in the equations. We now present the corrected version of that algorithm.

Recall that the equation of a conic:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0, \tag{B.1}$$

can be expressed in homogeneous form as

$$\mathbf{x}^T C \mathbf{x} = 0. \tag{B.2}$$

With

$$C = \begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix} \tag{B.3}$$

and $\mathbf{x}$ being a 3-vector representing a 2D point in homogeneous coordinates. In [62], however, a different formulation is used to express a conic

$$C(x, y) = A(x - x_0)^2 + 2B(x - x_0)(y - y_0) + C(y - y_0)^2 - 1 = 0, \tag{B.4}$$

where (x_0, y_0) is the center of the conic. To convert a conic $c = [A\,B\,C\,D\,E\,F]^T$ satisfying Eq. B.1 to the Conic-Center format of Eq. B.4, Algorithm B.1 can be used.

1. $y_0 \leftarrow \frac{E-BD/(2A)}{B^2/(2A)-2C}$
2. $x_0 \leftarrow \frac{-(D+By_0)}{2A}$
3. $s \leftarrow Cy_0^2 + Bx_0y_0 + Ax_0^2 - F$
4. $A' \leftarrow A/s$
5. $B' \leftarrow B/(2s)$
6. $C' \leftarrow C/s$
7. The conic parameters are: $[A', B', C', x_0, y_0]$

Algorithm B.1 *Conversion from the inhomogeneous conic equation format to the Conic-Center format.*

The conversion from the Conic-Center format to the inhomogeneous conic format is straightforward, with a little algebra it can be seen that:

$$\begin{aligned} A &\leftarrow A' \\ B &\leftarrow 2B' \\ C &\leftarrow C' \\ D &\leftarrow -2B'y_0 - 2A'x_0 \\ E &\leftarrow -2C'y_0 - 2B'x_0 \\ F &\leftarrow C'y_0^2 + 2B'x_0y_0 + A'x_0^2 - 1 \end{aligned} \tag{B.5}$$

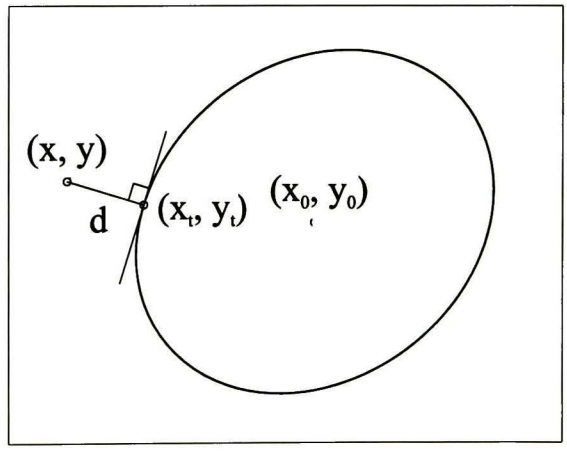


Figure B.1: Orthogonal distance d from point (x, y) to a conic. The point (x_t, y_t) marks the point on the conic which is closest to (x, y).

The problem is to compute the orthogonal distance d from point (x, y) to the conic $C(x, y)$ (see Figure B.1). We will follow closely the development in [62]. In that report, the objective is to find the point $\mathbf{x_t} = (x_t, y_t)$ on the conic which is closest to (x, y). Thus, point $\mathbf{x_t}$ must satisfy the equation of the conic (Eq. (B.4)):

$$A(x_t - x_0)^2 + 2B(x_t - x_0)(y_t - y_0) + C(y_t - y_0)^2 - 1 = 0 \tag{B.6}$$

and must pass by the line which is orthogonal to the tangent at $\mathbf{x_t}$:

$$(y - yt)\frac{\partial C(x_t, y_t)}{\partial x} = (x - x_t)\frac{\partial C(x_t, y_t)}{\partial y} \tag{B.7}$$

Now let $\Delta_x = x_t - x_0$ and $\Delta_y = y_t - y_0$, from B.6,

$$\Delta_y = \frac{-B\Delta_x \pm \xi}{C}, \tag{B.8}$$

where $\xi^2 = B^2\Delta_x^2 - C(A\Delta_x^2 - 1) = (B^2 - AC)\Delta_x^2 + C$. From B.7,

$$(A\Delta_x + B\Delta_y)(y - y0 - \Delta_y) = (C\Delta_y + B\Delta_x)(x - x_0 - \Delta_x). \tag{B.9}$$

Substituting the value of Δ_y in the above equation leads to

$$e_1\Delta_x^2 + e_2\Delta_x + e_3 = (e_4\Delta_x + e_5), \xi \tag{B.10}$$

where

$$\begin{aligned} e_0 &= B^2 - AC \\ e_1 &= 2Be_0 \\ e_2 &= Ce_0(y - y_0) \\ e_3 &= BC \\ e_4 &= B^2 + C^2 + e_0 \\ e_5 &= C\left[B(y - y_0) - C(x - x_0)\right]. \end{aligned} \tag{B.11}$$

Squaring the above equation, we have

$$\left(e_1\Delta_x^2 + e_2\Delta_x + e_3\right)^2 = (e_4\Delta_x + e_5)^2\left(e_0\Delta_x^2 + 4C\right). \tag{B.12}$$

Rearranging terms, we obtain an equation of degree four in Δ_x:

$$f_4\Delta_x^4 + f_3\Delta_x^3 + f_2\Delta_x^2 + f_1\Delta_x^1 + f_0 = 0, \tag{B.13}$$

where

$$f_4 = e_1^2 - e_0e_4^2 \tag{B.14}$$
$$f_3 = 2e_1e_2 - 2e_0e_4e_5 \tag{B.15}$$
$$f_2 = e_2^2 + 2e_1e_3 - e_0e_5^2 - e_4^2C \tag{B.16}$$
$$f_1 = 2e_2e_3 - 2e_4e_5C \tag{B.17}$$
$$f_0 = e_3^2 - e_5^2C. \tag{B.18}$$

Note that in the original formulation [62], the expressions for f_2, f_1 and f_0 were wrong. The two or four real roots of Eq. B.13 can be found in closed form. For one solution Δ_x, we can obtain ξ from Eq. B.10, as follows

$$\xi = \left(e_1\Delta_x^2 + e_2\Delta_x + e_3\right) / \left(e_4\Delta_x + e_5\right). \tag{B.19}$$

Thus, Δ_y is computed from Eq. B.8. Eventually, the orthogonal distance d can be computed by

$$d = \sqrt{(x - x_0 - \Delta_x)^2 + (y - y_0 - \Delta_y)^2} \tag{B.20}$$

Note that there might be up to four solutions, but only the one with the smallest distance is the one we need.

Bibliography

[1] N. Andreff. Towards the embedding of on-line hand-eye calibration into visual servoing. In *10th Scandinavian Conference on Image Analysis*, pages 455–461, June 1997.

[2] J. Angeles, G. Soucy, and F. P. Ferrie. The online solution of the hand-eye problem. *IEEE Transactions on Robotics and Automation*, 16(6):720–731, December 2000.

[3] K. Åström. Using combinations of points, lines and conics to estimate structure and motion. In *SSAB Symposium on Image Analysis*, March 1998.

[4] A. Bartoli and P. Sturm. Three new algorithms for projective bundle adjustment with minimum parameters. Technical Report 4236, INRIA, August 2001.

[5] E. Bayro and V. Banarer. A geometric approach for the theory and applications of 3d projective invariants. *Journal of Mathematical Imaging and Vision*, 16:131–154, 2002.

[6] E. Bayro and G. Sobczyk. *Geometric Algebra with Applications in Science and Engineering*. Birkhäuser, 2001.

[7] E. Bayro-Corrochano. *Geometric Computing for Perception Action Systems*. Springer-Verlag, 2001.

[8] E. Bayro-Corrochano, K. Daniilidis, and G. Sommer. Motor algebra for 3d kinematics. the case of the hand–eye calibration. *International Journal of Mathematical Imaging and Vision*, 13(2):79–99, October 2000.

[9] R. Berthilsson, K. Åström, and A. Heyden. Reconstruction of curves in R^3, using factorization and bundle adjustment. In *7th International Conference on Computer Vision*, volume 1, pages 674–679, September 1999.

[10] P. J. Besl and N. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[11] G. Borgefors. Hierarchical chamfer matching: a parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865, 1988.

[12] S. Carlsson and D. Weinshall. Dual computation of shape and camera positions from multiple images. *International Journal of Computer Vision*, 27(3):1–16, 1998.

[13] G. Champleboux, S. Lavallée, R. Szeliski, and L. Brunnie. From accurate range imaging sensor calibration to accurate model-based 3d subject localization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 83–89, 1992.

[14] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 44–51, 2000.

[15] G. Cross. *Surface Reconstruction from Image Sequences, Texture and Apparent Contour Constraints*. PhD thesis, University of Oxford, 2000.

[16] G. Cross and A. Zisserman. Quadric reconstruction from dual-space geometry. In *6th International Conference on Computer Vision*, pages 25–31, 1998.

[17] S. J. Cunnington and A. J. Stoddart. N-view point set registration: a comparison. In *British Machine Vision Conference*, pages 234–244, 1999.

[18] F. Dornaika and R. Horaud. Simultaneous robot-world and hand-eye calibration. *IEEE Transactions on Robotics and Automation*, 14(4):617–622, August 1998.

[19] D. Eggert, A. Fitzgibbon, and R. Fisher. Simultaneous registration of multiple range views satisfying global consistency constraints for use in reverse engineering. In *Computer Vision and Image Understanding*, volume 69, pages 253–272, 1998.

[20] J. Feldmar, G. Malandain, J. Declerck, and N. Ayache. Extension of the icp algorithm to non-rigid intensity-based registration of 3d volumes. In *Workshop on Mathematical Methods in Biomedical Image Analysis*, pages 84–93, June 1996.

[21] A. W. Fitzgibbon. Robust registration of 2d and 3d point sets. *Image Vision Computing*, 21:1145–1153, 2003.

[22] C. Fookes, J. Williams, and M. Bennamoun. Global 3d rigid registration of medical images. In *International Conference on Image Processing*, volume 2, pages 447–450, September 2000.

[23] G. H. G. Wei, K. Arbter. Active self-calibration of robotic eyes and hand-eye relationships with model identification. *IEEE Transactions of Robotics and Automation*, 14(1):158–166, February 1998.

[24] W. Grimson, T. Lozano-Pérez, W. Wells, G. Ettinger, S. White, and R. Kikinis. An automatic registration method for frameless stereotaxy, image-guided surgery, and enhanced reality visualization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 430–436, 1994.

[25] E. Guest, E. Berry, R. Baldock, M. Fidrich, and M. Smith. Robust point correspondence applied to two- and three-dimensional image registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):165–179, February 2001.

[26] R. Hartley. Computation of the quadrifocal tensor. In *5th European Conference on Computer Vision*, volume 1, pages 20–35, June 1998.

[27] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[28] D. Hestenes. *Space-Time Algebra*. Gordon and Breach, 1966.

[29] D. Hestenes and G. Sobczyk. *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Dordrecht, 1984.

[30] G. Hu. 3-d object matching in the hough space. In *IEEE International Conference on Systems, Man and Cybernetics 'Intelligent Systems for the 21st Century'*, volume 3, pages 2718–2723, October 1995.

[31] D. Ionescu, S. Abdelsayed, and D. Goodenough. A registration and matching method for remote sensing images. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 710–712, September 1993.

[32] A. Johnson and S. Kange. Registration and integration of textured 3d data. In *3DIM'97*, pages 234–241, 1997.

[33] F. Kahl and A. Heyden. Affine structure and motion from points, lines and conics. *International Journal of Computer Vision*, 33(3):163–180, 1999.

[34] H. Kalviainen, E. Oja, and L. Xu. Randomized hough transform applied to translational and rotational motion analysis. In *11th IAPR International Conference on Pattern Recognition, Conference A: Computer Vision and Applications*, volume 1, pages 672–675, August 1992.

[35] E. Y. Kang, I. Cohen, and G. Medioni. Robust affine motion estimation in joint image space using tensor voting. In *16th International Conference on Pattern Recognition*, volume 4, pages 256– 259, August 2002.

[36] R. Kaucic, R. Hartley, and N. Dano. Plane-based projective reconstruction. In *8th International Conference on Computer Vision*, pages 420–427, July 2001.

[37] J. Kybic and M. Unser. Fast parametric elastic image registration. *IEEE Transactions on Image Processing*, 12(11):1427–1442, November 2003.

[38] J. Lasenby. *Geometric Algebra: Applications in Engineering*, pages 423–440. Birkäuser, 1996.

[39] P. Lounesto. Clical, a calculator type computer program for vectors, complex numbers, quaternions, bivectors, spinors, and multivectors in clifford algebras.

[40] P. Lounesto. *Clifford Algebras and Spinors*. Cambridge University Press, 1997.

[41] P. Lounesto, R. Mikkola, and V. Vierros. Clical user manual. Technical Report A248, Institute of Mathematics, 1987.

[42] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. In *Conference on Computer Vision and Pattern Recognition*, pages 935–938, 1994.

[43] B. Luo and E. Hancock. Matching point-sets using procrustes alignment and the em algorithm. In *10th British Machine Vision Conference*, pages 43–52, 1999.

[44] E. Malis and A. Bartoli. Euclidean bundle adjustment independent on camera intrinsic parameters. Technical Report 4377, INRIA, December 2001.

[45] H. Malm and A. Heyden. A new approach to hand-eye calibration. In *15th International Conference on Pattern Recognition*, volume 1, pages 525–529, September 2000.

[46] P. F. McLauchlan. Gauge invariance in projective 3d reconstruction. In *Workshop on Multi-View Modeling & Analysis of Visual Scenes*, pages 183–199, June 1999.

[47] G. Medioni, M. Lee, and C. Tang. *A Computational Framework for Segmentation and Grouping*. Elsevier Science, 2000.

[48] W. H. Press, S. A. Teukolsky, W. T. Vettering, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, 1988.

[49] C. Rother and S. Carlsson. Linear multi view reconstruction and camera recovery using a reference plane. *International Journal of Computer Vision (Special Issue on Multi-View Modeling and Rendering of Visual Scenes)*, 49(2/3):117–141, 2002.

[50] Y. Shan, Z. Liu, and Z. Zhang. Model-based bundle adjustment with application to face modeling. In *8th International Conference on Computer Vision*, volume 2, pages 644–651, July 2001.

[51] D. Simon, M. Herbert, and T. Kanade. Techniques for fast and accurate intra-surgical registration. *Journal of Image Guided Surgery*, 1(1):17–29, 1995.

[52] G. Slabaugh, R. Schafer, and M. Livingston. Optimal ray intersection for computing 3d points from n-view correspondences. Technical report, Georgia Tech, October 2001.

[53] G. Sommer, editor. *Geometric Computing with Clifford Algebras. Theoretical Foundations and Applications in Computer Vision and Robotics*. Springer-Verlag, 2001.

[54] A. Stoddart and A. Hilton. Registration of multiple point sets. In *In Proceedings of the International Conference on Pattern Recognition*, pages 40–44, 1996.

[55] P. Sturm and B. Triggs. A factorization based algorithm for multi-image projective structure and motion. In *4th European Conference on Computer Vision*, volume 2, pages 709–720, 1996.

[56] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment — a modern synthesis. In *Vision Algorithms Workshop: Theory & Practice*, pages 298–372, September 1999.

[57] R. Y. Tsai and R. K. Lenz. A new technique for fully autonomous and efficient 3-d robotics hand/eye calibration. *IEEE Transactions of Robotics and Automation*, 5:345–358, June 1989.

[58] G. Turk and M. Levoy. Zippered polygons meshes from range images. In *ACM SIGGRAPH Conference on Computer Graphics*, pages 311–318, 1994.

[59] W. Wells. Statistical approaches to feature-based object recognition. *International Journal of Computer Vision*, 21:63–98, 1997.

[60] J. You, W. Zhu, E. Pissaloux, and H. Cohen. Hierarchical image matching: A chamfer matching algorithm using interesting points. In *Third Australian and New Zealand Conference on Intelligent Information Systems*, pages 70–75, November 1995.

[61] Z. Zhang. Iterative point matching for registration of free-form curves. Technical Report 1658, INRIA, 1992.

[62] Z. Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. Technical Report 2676, INRIA, October 1995.

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis:

Métodos Geométricos para Visión Robótica, Geometric Methods for Robot Vision

del (la) C.

Leo Hendrik REYES LOZANO

el día 23 de Julio de 2004.

Dr. José Luis Alejandro NAREDO VILLAGRÁN
Investigador Cinvestav 3C
CINVESTAV GDL
Jalisco

Dr. Eduardo Jose BAYRO CORROCHANO
Investigador Cinvestav 3C
CINVESTAV GDL
Jalisco

Dr. Juan Manuel RAMÍREZ ARREDONDO
Investigador Cinvestav 3B
CINVESTAV GDL
Jalisco

Dr. Gerard GUY MEDIONI
Professor
Universidad del Sur de California
California, USA

Dr. Mariano José Juan RIVERA MERAZ
Investigador Titular B
Centro de Investigación en Matemáticas
Guanajuato