OPTICAL NAVIGATION USING QUADRIC PROJECTIONS AND INDIRECT INDEXING & SEARCH METHODS

Devin T. Renshaw

Submitted in Partial Fullfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

Approved by:
John A. Christian, III, Chair
Kurt S. Anderson
John T. Wen
Christopher N. D'Souza

Department of Mechanical, Aerospace, and Nuclear Engineering
Rensselaer Polytechnic Institute
Troy, New York

[May 2022]
Submitted April 2022

© Copyright 2022
by
Devin T. Renshaw
All Rights Reserved

CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

ACKNOWLEDGMENT xiii

ABSTRACT xvi

1. INTRODUCTION 1

1.1 A matter of perspective 1

1.2 Contributions 5

1.2.1 Sub-pixel boundary localization 7

1.2.2 Indirect P^2-invariant indexing and query 7

1.2.3 Quadric-based pose estimation 7

2. BACKGROUND 9

2.1 Image processing and computer vision 9

2.1.1 Camera model 9

2.1.2 Coordinate frames and conventions 12

2.1.3 Real cameras 13

2.1.4 Streaks and edges 14

2.1.5 Optics and Zernike polynomials 17

2.2 Geometry and invariants 18

2.2.1 Geometric hierarchy 18

2.2.2 Projective space and homogeneous coordinates 19

2.3 Conics, quadrics, and projections 21

2.3.1 Shortcomings of existing techniques 22

2.3.2 Conics 22

2.3.3 Quadric surfaces 25

2.3.4 Invariant descriptors, permutations, and search 26

2.3.5 There and back again: quadrics to conics, and vice versa 27

3. SUBPIXEL STREAK LOCALIZATION 29

3.1 Zernike polynomials 31

3.1.1 Rotational properties of Zernike moments 33

3.2 Discretized Zernike moments for digital images 34

3.2.1 Localization update 35

3.2.2 Computing incidence angle . . . 35
3.2.3 Computing ℓ . . . 37
3.2.4 Equation parameter fitting . . . 41
3.2.5 Steps to refine streak locations . . . 43
3.2.6 Limitations . . . 43
3.3 Numerical validation of estimation functions on synthetic data . . . 44
3.3.1 Comparing alternative methods . . . 45
3.4 Validation on real images . . . 47
4. INDIRECT P^2-INVARIANT INDEXING & SEARCH . . . 51
4.1 What are P^2-invariants? . . . 52
4.2 Databases and search . . . 52
4.2.1 Database Management Systems (DBMS) . . . 53
4.3 R-trees . . . 54
4.3.1 R-tree alternatives . . . 55
4.3.2 R-tree algorithm . . . 56
4.4 Reforming the R-tree for invariants . . . 57
4.4.1 Symmetric and antisymmetric functions . . . 58
4.4.2 A new paradigm for database storage and query . . . 59
4.4.2.1 Indirect search and store . . . 61
4.5 Numerical results . . . 66
5. QUADRIC POSE ESTIMATION . . . 72
5.1 Circular craters and elliptical projections . . . 75
5.1.1 Quadrics and the intersections of circles . . . 76
5.1.2 Canonical position and orientation . . . 77
5.2 Solution using homography . . . 89
5.2.1 Alternative solution to the homography . . . 96
5.3 Numerical Results . . . 97
5.3.1 Ellipse parameter study . . . 99
6. CONCLUSION . . . 104
6.1 Putting it all together . . . 104
6.2 Summary and future work . . . 105
6.3 Future work . . . 106
6.4 Final note . . . 107

REFERENCES 108

APPENDICES 118

A. CAMERA MODELING 118

A.1 Pinhole camera model 118

A.2 A distorted view of the world 119

B. PROJECTIVE GEOMETRY AND INVARIANTS 123

B.1 Commutative algebra 124

B.1.1 Fields, rings, and ideals 124

B.2 The pose problem 126

B.3 Gröbner bases 127

B.4 Applying Gröbner basis 129

LIST OF TABLES

4.1 Python tree creation time in seconds. Both flavors of the *kd*-tree outperformed the both R-tree variants (though only one is shown here). Creation times for both R-tree versions were sufficiently similar to include only one. 68

4.2 Standard indexing R-tree vs *kd*-tree. Time (in milliseconds) to complete ten thousand random searches at varying levels of noise. Prior to each search, noise is added to each of the invariants of the descriptor by a Gaussian distribution with the standard deviation of σ and zero mean. The total number of descriptors in the catalog was 106615. No permutations occurred on the query data. 69

4.3 Indirect indexing R-tree vs *kd*-tree. Time (in milliseconds) to complete ten thousand random searches at varying levels of noise. The same zero-mean Gaussian noise was added to each query as in the above table. Random cyclic permutations shifted the data which meant that both versions of the *kd*-tree and the vanilla R-tree all required three searches each. 70

5.1 Quadric localization error table for varying levels of noise on the ellipse axis parameters since the axes shows the greatest sensitivity to noise. RMS error and standard deviation are given in kilometers, while percentage error shows the relationship between location error and closest distance to the lunar surface. . . 101

LIST OF FIGURES

1.1 Leonardo da Vinci's *Last Supper* (*Cenacolo Vinciano*) with perspective lines overlaid on the original artwork. Leonardo's use of perspective urges the viewer to the focus of the painting. Image in the public domain: housed at the Museo del Cenacolo Vinciano of the Santa Maria delle Grazie in Milan, Italy. 2

1.2 Illustration of cross ratio: four lines coincident at a single point. 3

1.3 High-level flow diagram starting from image capture through to pose estimation. Image processing identifies ellipses in the image, computes the invariant associated with a particular triad of craters, searches the catalog for a suitable set of probable matches, then determines the pose. This particular image is near the lunar south pole. The camera position is shown in the bottom left figure with the camera field of view shown as the orange frustum. 6

2.1 Cassini's view of Saturn. Image ID: $N1460342538_1$. Courtesy NASA/JPL-Caltech. 10

2.2 Ideal pinhole camera model. Note that the image on the focal plane is a scaled, mirrored image of the original object. The focal plane represents the point at which the image is in perfect focus and is located at a distance f (focal length) from the pinhole location, and it represents a physical plane in space. The image plane, conversely, represents a mathematical simplification of the focal plane. Using the image plane avoids working with mirror images as well as the requisite mathematical bookkeeping; this plane is parallel to the focal plane and lies in front of the camera's center of projection (i.e., along the principal axis). The points indicated in red on the object and its projected image corresponds to the projection of a specific point onto both the image and focal planes. 12

2.3 Point spread function of a camera. Collimated light enters the camera lens and diffracts; all light from a given direction focuses onto one point on the focal plane. Any displacement δ of the photodetector away from the focal point produces defocus distortion, shown as rays spreading out beyond the focal point, which is shown in green. 14

2.4 Edge in an image. On left, the underlying signal of the edge overlaid with the photodetector array grid, with the top right showing a magnified portion of the signal and the array. Bottom right shows how discretization fails to produce a crisp boundary even in the absence of noise and the point spread function. . . 15

2.5 Streak (line) in an image. On left, the underlying signal of the edge overlaid with the photodetector array grid, with the top right showing a magnified portion of the signal and the array. Bottom right shows how discretization fails to produce a crisp boundary even in the absence of noise and the line spread function. . . 16

2.6 The projective geometry hierarchy. Moving down from Euclidean to projective geometry, we reduce the set of invariants available in each group. 19

2.7 The projective geometry transformations. Illustrates the groups of transformations allowed in each group. Every group inherits allowable transformations from those above it in the hierarchy; that is, the similarity group has all of the invariants of the affine and projective groups, but not those of the Euclidean. . 20

2.8 The ambiguity of choosing points on the projected ellipse and the original ellipse (here, a circle). It is to illustrate that no point on the projected ellipse can be identified uniquely given no information about the camera pose. 23

2.9 Ellipse. 24

2.10 Model of the quadric defined as the intersection of a quadric cone, $Q_{co,i}$, and the plane π_i defined by the ellipse formed by the crater rim, A_I. The Moon-centered frame has the origin $V = [0, 0, 0, 1]^T$. 26

2.11 Quadric projection onto a plane. Each disk quadric projects to a corresponding ellipse on the image plane. Only the ellipse data is known, and the purpose here is to determine the pose (i.e., the position and attitude) of the camera given crater images. 27

3.1 Example of streaks in an image. Bottom left image is area of original showing several streaks, and the bottom right shows the pixel intensity level for a selected patch. Note the gradual dip in pixel values where the fence overlays the grass background. 30

3.2 Selected set of Zernike polynomials in 2D. The row of the triangle represents the order n, and the column is the repetition or angular dependence (m). The ordering shown is one particular way to uniquely number each moment. This figure was produced by the author using Jacopo Antonello's Zernike polynomial function. 32

3.3 Illustration of the original frame and the prime frame, which is oriented such that the prime axes align with the direction of the streak and its normal. . . . 36

3.4 Moment masks for M_{11} (real and imaginary parts) and M_{20} (bottom). These are the masks to apply to the region centered on each pixel-level guess. It is the ratio of the imaginary to real that computes the angle and the ratio of the magnitude of M_{11} to M_{20} that gives the offset ℓ according to the equations in the next section. The vertical axis represents the magnitude, and the two horizontal axes (u and v), represent the local pixel coordinates. 37

3.5 Wedge model of streak. 39

3.6 Components of wedge model. The gray section is the uniform background, dark blue the ascending ramp up to the maximum $k + h$ over a distance w, and light blue the descending ramp. 39

3.7 Assessment of streak localization RMS error using Zernike moments over a range of blur sizes and SNR. Note that the error never reaches 0.1 pixels, even in extreme cases where the noise approaches parity with signal level (i.e., high noise, low SNR). 45

3.8 Example of a curved synthetic streak with varying levels of noise, with decreasing SNR as we move down. The red $\times$'s denote the pixel-level guesses, and the green circles the refined streak locations. 48

3.9 Illustration of a two comparison runs between the normalized sum of gradients approach (in blue) and the Zernike moments discussed here (in green). The left compares the NSG approach with a 3×3 window and the right with a 5×5 window. Both methods used the same starting pixel values for coarse estimates. The RMS error for Zernike moments was around 0.11 pixels and for NSG it was slightly less than 0.29 and 0.23, respectively. 49

3.10 Inertially pointed star field image captured with with the Omnidirectional Space Situational Awareness (OmniSSA) system. This example image has a 10 second exposure time and contains a satellite that appears as a streak within the image. The original OmniSSA image is courtesy of Dr. Marcus Holzinger of University of Colorado Boulder. 49

3.11 Image of Kuiper belt object Arrokoth (formerly called Ultima Thule) collected by the New Horizon spacecraft's Long Range Reconnaissance Imager (LORRI) during a flyby in early 2019 (credit for raw image: NASA/Johns Hopkins University Applied Physics Laboratory/Southwest Research Institute). 50

3.12 Image of a retinal scan for a healthy eye, where we seek to localize blood vessels. The original image is im00032 from the STARE database. 50

4.1 R-tree visualization given a set of 2D points shown with dots. This shows three levels of nodes (shown in descending order: purple, green, orange) when the number of child nodes is four. The image shows that overlapping regions are possible unless additional structuring takes place, which reduces redundancy within the tree. The nodes in the upper right of the image have color-coded values to illustrate which point belongs with which minimum bounding rectangle (MBR). Note that here a point only shows one color, but depending on implementation several MRBs may contain the same point. 57

4.2 Plots of the original set of over 400,000 invariants from the crater triads (craters ranging from 20 to 500 kilometers in radius). The top shows the original distribution of points, and the bottom shows the scaled values to ensure an integer representation for a smaller file size. The colors merely intend to differentiate points at different heights in the image. J_{ab}, J_{bc}, J_{ca} are the invariants associated with a crater triad ordering of A, B, and C. 67

5.1 The disk quadric as the flat circular plate whose rim is the intersection of a sphere and a circular quadric right cone $Q_{co,i}$ or whose rim is also the intersection of the same quadric cone and the plane π. 75

5.2 Example of conic backprojection given an ellipse on the image plane and the circular object in space. An elliptical cone (the "enveloping cone") has its vertex as the focal point of the camera and expands in the direction of the circular object; the conic it forms by intersecting the image plane *is* the projection of the disk quadric onto the image plane. 78

5.3 Illustration of the transformation of an ellipse (black) into the "canonical pose" (canonical position and orientation) (red) through an intermediate translation step (grey). This translation of the ellipse center to the coordinate frame origin followed by a rotation to align the ellipse axes with that of the coordinate frame. 79

5.4 Example of where the centers of the circles and normals are calculated. Notice that the second ellipse is a mirror image of the first, and the angles and distances are the same, save the negative or positive angle. 82

5.5 Real image with several circular objects with known pose; image personally taken with camera used at Intuitive Machines. The algorithm produced an accuracy of 15cm and within two degrees of the known camera pose. OpenCV's contour function computed ellipses after finding the contours in the image. 83

5.6 Top shows a custom Python visualization (using PyVista) of the camera position and its view frustum in the previous image. The two possible normals are shown at the bottom image, and we can see that the two geometrically indistinguishable options face in varying directions. That is, if we were to place a disk at either orientation for all of the disks in the image, it would be impossible to extract the correct disk without more information. 84

5.7 Illustration of the distance between the camera vertex and the circle center (rather than the ellipse center). 88

5.8 Visualization of the two possible surface normals given the camera view of each circular feature. The top view shows the location of the camera and the craters on the body surface, and the bottom shows the view from the camera. The true normals are shown in black and the counterfeit normals in red. 90

5.9 Spherical body showing the normals of the circular surface features on top and the corresponding dual normals (in red) given the camera view. Note that the restriction for circular features to lie on a non-degenerate surface is not a necessary condition for finding the pose. 91

5.10 Given two disk quadrics, we define the angle ϕ as that between the two planes at the intersecting line (shown in red). Using the idea of homogeneous space, we can say that all planes in 3D intersect in a line, even parallel planes, where the line of intersection is the ideal line and the angle is 0 or 180 degrees. 92

5.11 Graphic of geometric algebra solution to the position problem. Three spheres intersect at precisely two points: one of which is the correct solution. This approach is highly intuitive, but seemingly not familiar in engineering at large. Note that the intersection of two spheres results in a circle (in geometric algebra, two spheres have an intersection even if they do not meet, similar to how two non-intersecting conics intersect), and the intersection of a circle and another sphere has two solutions: here, the value in green. 98

5.12 This figure illustrates the change to a conic when the ellipse size increases by 10%. While these changes seem relatively insignificant, they can have drastic effects on the position estimation: the resulting position error from this 10% increase was nearly 600 kilometers! 100

5.13 Sensitivity plot of localization on noise on different ellipse parameters on both linear (top) and logarithmic (bottom) y-values. Both plots reflect the same Monte Carlo simulation, with 100 samples per data point. Note the heavy dependence on the ellipse axes but little sensitivity to the other ellipse parameters. 100

5.14 Blender render of the Moon using NASA surface images and displacement maps. The main crater is found in the bottom left, and is known as the Copernicus Impact Crater, a well-studied lunar crater. The craters are outlined in the next figure, but this example and others like it were used to determine the precision of results in realistic situations. The camera here is approximately 5000 kilometers from the lunar surface. 102

5.15 Blender render of the Moon with conics highlighting the crater rims. It is from this image that we may apply the full optical navigation approach to first detect then identify craters and work our way to determining the camera pose. A first pass edge detection algorithm can be used for crater detection and refined using subpixel edge localization for the edges of each crater. An ellipse fitting algorithm must then be applied to those points to estimate the ellipse parameters. Using invariant theory or other method, we can identify each of the chosen craters. Once we know the crater locations in the Moon frame using the catalog, we can estimate both the attitude and the position using either of the approaches discussed above. 103

6.1 Reiteration of flowchart starting from image capture through to pose estimation. Ellipse fitting leads to computing the associated projective invariants, which we search the database for matches. We then have identified the individual craters and can reproject other ellipses in the image for verification. Any of the matched ellipses are then used to compute both attitude and position using either approach in the last chapter. The lunar images in this flowchart were all produced in Blender, where the images were taken from the position shown in the lower left, whose position and attitude we attempt to compute. 105

A.1 Pinhole camera model. All light crossing the tiny aperture passes undistorted on to the detector array on the left. 119

A.2 Ideal camera lens. All light from a given direction (assumed as collimated light) focuses at a particular point on the focal plane at a distance f from the lens, as shown by the orange and purple lines and the lit portion of the focal plane array. 120

A.3 Camera detector misalignment. All light previously focused on a single pixel now spread to a local neighborhood, thereby diffusing the intensity of the single pixel to adjacent cells. 121

A.4 Example raw star image taken in upstate New York on a Pixelink camera. The single points of light are hot pixels, or pixels whose values are stuck at a certain value (usually 255 for 8-bit images). The center blob is a real star; note the gradual intensity change, which is the defocus effect spreading the intensity of this single star to a few dozen nearby pixels. 122

B.1 The projective geometry transformations. Illustrates the groups of transformations allowed in each group. Every group inherits allowable transformations from those above it in the hierarchy. 124

ACKNOWLEDGMENT

Despite a rocky start in my college career and many bumps along the way, the path has finally broadened and given way to new adventures beyond my imagination thanks to this journey. This future would have been impossible without extensive support, encouragement, and direction along the way. While even just listing the number of people involved would be an impossible task, I want to mention and thank those most pivotal in this achievement.

I want to thank my advisor and mentor, Dr. John Christian, who took me under his wing and welcomed me into his lab despite lacking any experience in aerospace and navigation. Your tutelage was unparalleled, walking the fine line of providing challenging opportunities (and allowing me to fail at times!) and giving support and direction. Even while I meandered from topic to topic and project to project, your unwavering faith and patience gave me the confidence to explore new ideas and concepts that ultimately led to this work. After meeting your colleagues in industry and academia, it is apparent that I am not alone in this sentiment, and I doubt I could have found a better mentor. Thank you for taking a chance on me.

To my committee: Dr. Kurt Anderson, Dr. John Wen, and Dr. Chris D'Souza. Thank you for asking questions to challenge and critique me; your questions after the proposal made me rethink and reformulate my ideas and helped me challenge my own assumptions about the problems I was facing. Your expertise and experience were critical, and not just during my presentations. Several articles from each of you caught my attention in my pursuit of knowledge, and I was impressed with the breadth and depth of your knowledge. I have to say that I'm rather pleased to have you along for the ride.

It seems like the ragtag group of the Sensing, Estimation, and Automation Laboratory (SEAL) at Rensselaer Polytechnic Institute was the ideal group to join. Not only was it led by Dr. John Christian, it had some of the finest colleagues I could ask for. I didn't even know how good I had it until I looked at other labs where the camaraderie and interactions were noticeably absent; that our group was the exception rather than the rule. Our weekly paper-reading sessions, the happy hours, and the late-night outings (i.e., really anything past 7 PM) took away the stress of the week and made the journey much more memorable. I'm glad to have known all of you in SEALab: Dr. Kevin Kobylka, Paul McKee, Chris Ertl, Jacob Puritz, and Jacob Kowalski, with emeritus member Courtney Hollenberg. You brightened my day, and your ideas and quals prep really made this possible. I look forward to collaborating

with all of you in the future. Good luck!

While I finished this work, Intuitive Machines in Houston, Texas, provided me with challenging work and a rewarding and exciting career path. Thank you for taking a chance on me and allowing me to work part-time and focus on this work, which was instrumental in honing my ideas as well as being a part of a once-in-a-lifetime chance to land on the Moon. People like Tim Crain and Shaun Stewart who mentored me; you are both giants in our field. Special thanks to the team on hazard detection and avoidance with Joel Getchius, Michael Hansen, Daniel Posada, Dr. Troy Henderson, and Giovanni Molina for supporting me and making me look good. To you and everyone else at IM—Ad astra!

I also want to thank Dr. Pierre Gaillardon of the University of Utah, my master's advisor in electrical and computer engineering. You always were ready to help the students learn, and I appreciated your humor and dedication in your work. Your insights into engineering were invaluable and made me a better engineer. I have carried much of what you taught me into what I do today, and your class in embedded systems remains one of my favorite classes.

For the unsung heroes supporting students, from Jon Davies at the University of Utah to Sue Miller at RPI, I want to thank you for helping not just me through my programs, but for showing a genuine interest and concern for students and learning—in and out of school. Thank you for fighting for the students. Your efforts do not go unnoticed.

To my family: parents, siblings, in-laws (and out-laws), and our "couple parents". Your love and support pushed me through all of this schooling that was the furthest thing from my mind in high school. My love of learning has blossomed into something that I will hold dear the rest of my life.

To my parents, thank you for being patient and pushing me to do and be better. You probably enjoyed a few laughs as I tried to sort out what I was doing along the way, and you probably still do. I can't imagine doing this without your help and support.

To my siblings who kept asking me the question, "Are you done with school *yet*?" Thank you for letting me be who I am around you and for laughing along with me throughout this lifelong journey. Thank you for the examples that you and your families give to me every day. Thanks for being nerds.

Thank you to my in-laws, who have been nothing but supportive, and I'm proud to have been a part of such a wonderful family. You are every bit a part of my life as the family that grew up with me. To the out-laws, thanks for being a bit of respite when we need a

break. You understand me and my struggle with being a better spouse to my wife.

To our "couple parents", David and Barbara Acheson. You were there when things seemed the hardest, and you have remained dear to my heart throughout these years. We will always cherish your kindness. Thank you for looking past my flaws and foibles to see the best in me.

To my two children, Rylie and Silas. Rylie, your smile brightens my day even when you're supposed to be in bed. Your energy and curiosity are unrivaled, and I am proud to be your father. Looking up at the night sky has always filled me with awe and brings me closer to the infinite, and I'm glad that I get to share that with you. Never lose your wonder of the world; never stop asking questions. Silas, even with the short time so far with you, it seems you have an insatiable curiosity about the world around you and how things work. You seem undaunted with challenges that you face and I admire your bravery and tenacity in achieving your goals. I love nothing more than to sit with you as you lean against me with something interesting in your hands.

Finally, this whole journey would not have started without my wonderful wife, Rachel. You have stuck by my side as we explored life together, and you were the one to support me throughout the long path of graduate school, even when I myself doubted that I wanted it. You were there when it felt like no one else was. I wouldn't be where I am today or who I am today without your unwavering support. Your passion, bravery, and dedication are all virtues I cherish in you, and you enthrall me with your stories. You have sacrificed so much to marry me and to stay by my side throughout all of life's trials. It is to you that I dedicate this work, for without you this would all have been a pipe dream. Thank you.

ABSTRACT

Autonomy plays a crucial role in navigation to other celestial bodies and around our own planet. With steadily increasing demand for radiometric data on an already strained network, autonomous decision-making is becoming evermore central to spacecraft navigation. Autonomy relies on external sensory data, including digital imagery. While images certainly contain invaluable information, they possess inherent limitations—including scale ambiguities and perspective distortions. Leveraging projective geometry and invariant theory—both vital foundations to computer vision—this work resolves particular questions in navigation by exploiting geometric properties using invariant theory, namely: boundary localization, invariant data storage and interrogation, and pose estimation.

CHAPTER 1
INTRODUCTION

> Only art and science make us suspect
> the existence of life to a higher level,
> and maybe also instill hope thereof.
>
> Ludwig van Beethoven

1.1 A matter of perspective

Art and science are intimitely intertwined. From ancient engineers and architects to the Renaissance painters and sculptors, art guides science, and science informs art. Famed artists dabbled in biology, technology, and mathematics; and renowned scientists used painting and sculpting techniques to promote and advance science. Indeed, some blurred the line between the two, such as Leonardo da Vinci, whose paintings are among the most revered in the world, while simultaneously designing aqueducts, castle defenses, and military contraptions that were centuries ahead of their time.

Of particular interest here is the discovery of perspective and its role in how we interpret the world. Although ancient geometers experimented with perspective [1], [2], little remains of this work with the exception of only a handful of theorems (e.g., Pappus' theorem) and references to lost works [3]. Rigorous sustained inquiry into perspective began with the architects Brunelleschi and later Alberti [4]. Perspective theory shows that what we see depends on our location. It is this perspective that moved us from artwork that felt flat and lifeless to scenes filled with depth and vitality. The great Leonardo da Vinci himself pioneered new aspects of perspective and perception, employing techniques that added richness and focus in his work [5]. Among his masterpieces, *The Last Supper* [6] entices the viewer to focus on the central figure of the painting—simply using the vanishing point. This insight (and many others) led to more realistic depictions of landscapes, buildings, and people. Thanks to their contributions, our understanding of perspective took root.

Like masterpieces of art, space exploration has similarly captivated mankind. Views of Saturn's rings [7] or Pluto's icy surface display the grandeur of the universe around us, captured in the perspective of the camera. Our understanding of perspective (which we call projective transformation) has helped interpret information about what we see—from the

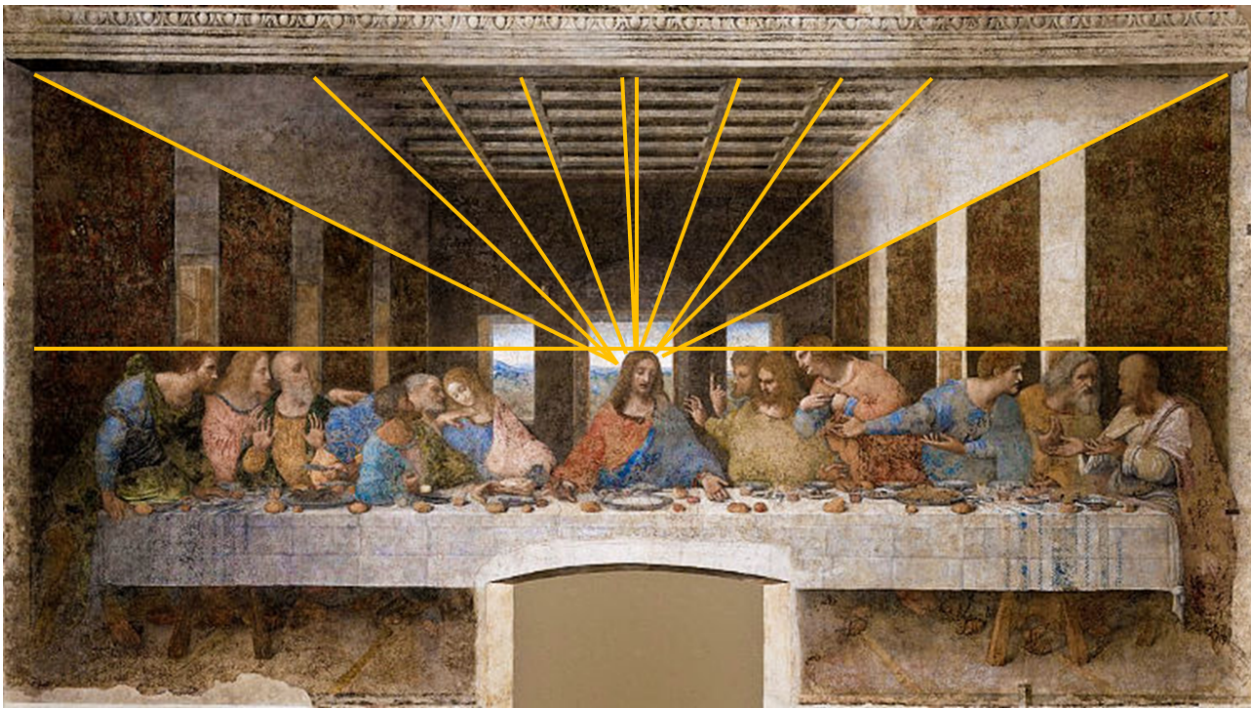

Figure 1.1: Leonardo da Vinci's *Last Supper* (*Cenacolo Vinciano*) with perspective lines overlaid on the original artwork. Leonardo's use of perspective urges the viewer to the focus of the painting. Image in the public domain: housed at the Museo del Cenacolo Vinciano of the Santa Maria delle Grazie in Milan, Italy.

depth and size of craters on the Moon and Mars to using stars and other bodies to determine attitude and position.

While perspective lends realism to art, it introduces some ambiguity. Take, for example, two shapes: a circle and a square. If we are directly overhead, the circle remains a circle and the square a square. We can move closer or farther away to change the scale, but the shape remains the same. As we move about in other directions, however, the shape of this disk changes from circular to elliptical and the square to a general quadrilateral. The shapes appear to stretch and shrink, and the formerly perfectly proportioned shapes become distorted versions. Parallel lines are no longer parallel, and apparent distances change as well. This suggests that seemingly inviolate properties in our world—angles, distances, areas, even shape!—are subject to change under a perspectivity.

This work delves into what properties and relationships (e.g., shape, angle) change under such conditions and which remain the same, or *invariant*. It is not hard, therefore, to understand why certain applications—particularly in navigation—tend to use invariants

instead of relying on naïve metrics that change when position changes. These *geometric invariants* (which hereon I refer to simply as *invariants*) form the backbone of this work. Invariants derive from relationships, and most relationships do not have invariant characteristics, so we must determine which relationships form invariant properties, and how many exist.

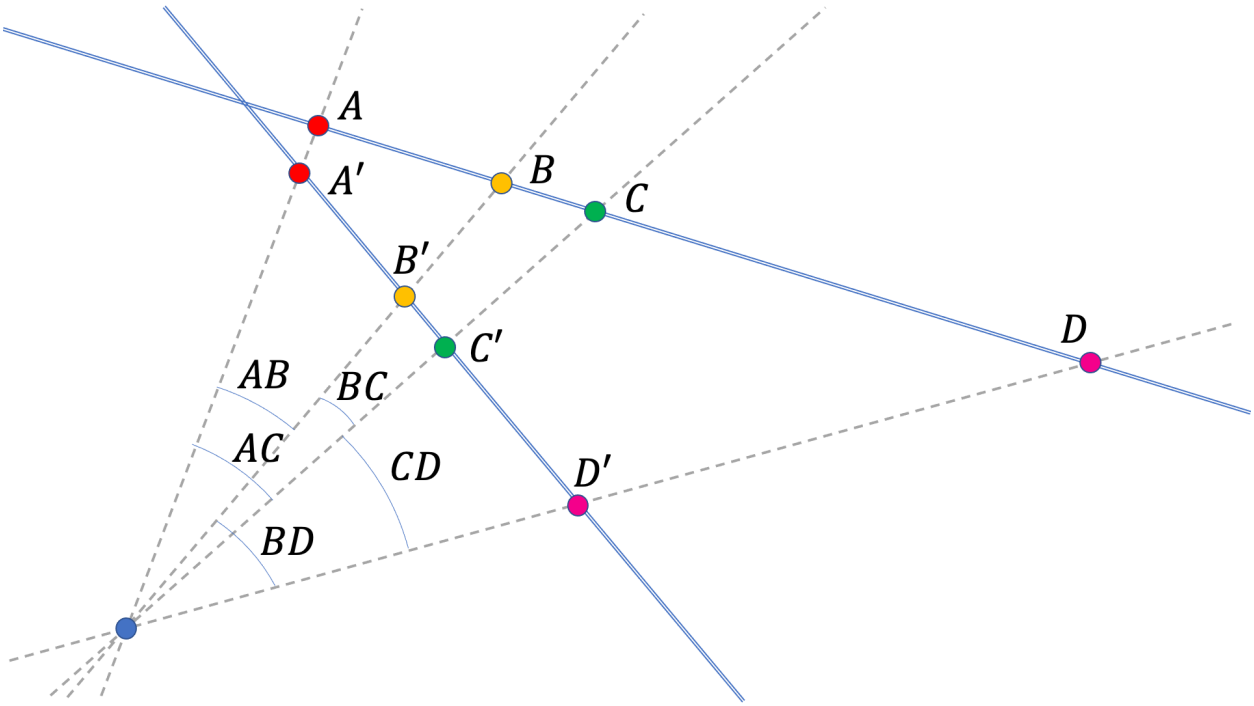

Figure 1.2: Illustration of cross ratio: four lines coincident at a single point.

One particular invariant, the cross ratio, is widely used in various industries and applications. The cross ratio is a particular relationship that four collinear points maintain after a perspectivity (i.e., PGL(n) transformations); this invariance, however, depends on the order of the points. Figure 1.2 shows the two sets of collinear points related by a perspectivity. The cross ratio of points A, B, C, and D is

$$Cr(A, B; C, D) = \frac{(AC)(BD)}{(BC)(AD)} \tag{1.1}$$

It is important to know how many independent invariants are possible for a given transformation group. This is crucial for eliminating redundant information while still retaining all independent information. Choosing too few invariants means that we ignore information about the system; too many means wasted cycles and computational storage.

That is, using more or fewer invariants can be either wasteful or negligent. We shall discuss how to choose the correct number of d-tuples for the situation.

This may seem to be a bit of useless—albeit interesting—information, useful only for shapes with perfectly straight edges. That lines and points are duals of each other in $\mathbb{P}^2$ expands the use of the cross ratio considerably for arbitrary shapes. Therefore, if we consider a pencil of four lines that intersect at one point, the relationship holds—though we consider not the length of AB and BC but the angle formed from their respective intersections (see Figure1.2). These angles convey the same information as the set of points. We are able to do this using Chasles' theorem [8], [9].

An important caveat is that each of the orderings relates to the others in the form of the following values of a reference cross ratio τ

$$\tau, \tau^{-1}, 1-\tau, (1-\tau)^{-1}, \tau(\tau-1)^{-1}, \tau^{-1}(\tau-1) \tag{1.2}$$

Leveraging this knowledge, one well-accepted technique to reduce the 120 different combinations from five particular points forming a cross ratio is the J-invariant. The result is that for each reference point of choice, the J-invariant remains unchanged despite the ordering of the remaining four points, and the function is given

$$J(\tau) = \frac{(\tau^2 - \tau + 1)^3}{\tau^2(\tau-1)^2} \tag{1.3}$$

for which it is easy to show that for any value of the cross ratio τ in Eq. 1.2, the J-invariant is identical.

This eliminates the remaining points once the first point is chosen as the reference, but this leaves five options that depend on the choice of reference star. We could, as many geometers do, simply run all five of those reference stars and perform a search for each choice, then choose among the $5k$ (k nearest neighbor) results, but this is hardly an optimized choice considering the significant costs to searching a massive database. So we must find some way to reduce our queries to the catalog. Furthermore, in reducing the cost of a very expensive operation (i.e., search), we have a large benchmark impact according to Amdahl's Law of computing performance [10], [11].

A final word on invariants: it can be easy to misapply invariant theory, and many applications supply too many or too few of these relationships, and some may use properties

that actually *do* change under perspective transformations, rendering the final result suspect at best; this situation is present when using angles for projective transformations, which are clearly not invariant in this group [12]. For example, too many invariants for a given problem contains redundant information (linearly dependent), which can slow down computation, whereas too few leaves out available information (underdetermined). Examples in navigation include crater identification, with some applications using only two invariants for a pair of craters [13]–[15], to some works using as many as thirty for a triad of craters [16]. Though not the focus of this work, I will provide the total number of independent invariants for any scenario I present, and any additional invariants are merely linear combinations of that independent set. Each set depends on the scenario, but several works delve into how to find a minimal set of independent invariants [17]–[24].

1.2 Contributions

Finally, these subsequent contributions might seem quite varied in scope and discipline, yet they share an original underlying purpose. While this is not the unique purpose for any of the below ideas, the purpose was the guiding hand that directed me to explore these ideas fully.

Let me illustrate this with a scenario with a series of needs to explain the genesis of each idea. Imagine a spacecraft approaching a celestial body (e.g., the Moon) riddled with craters, as in Figure 1.3. The onboard camera takes a series of digital photographs to study the surface; some images go back to Earth for processing, and some remain on the spacecraft for internal use. The flight computer needs those images to detect several craters on the surface. With this information it searches a pre-computed database of crater information to uniquely identify each crater, but this only works if the crater fitting algorithm has an accurate set of crater rim points. Then, if the computer can determine the relationship between at least three craters on the surface, the onboard database search finds a unique match to those relationships—uniquely identifying each of the craters. Finally, the spacecraft must determine where it is relative to the surface if it has a chance of landing successfully. Computing the vehicle pose (i.e., its position and attitude) is this final step before descending towards the surface.

I hope that this sample of uses motivates each of the contributions below.

Based on the above discussion, I present three novel applications. These applications

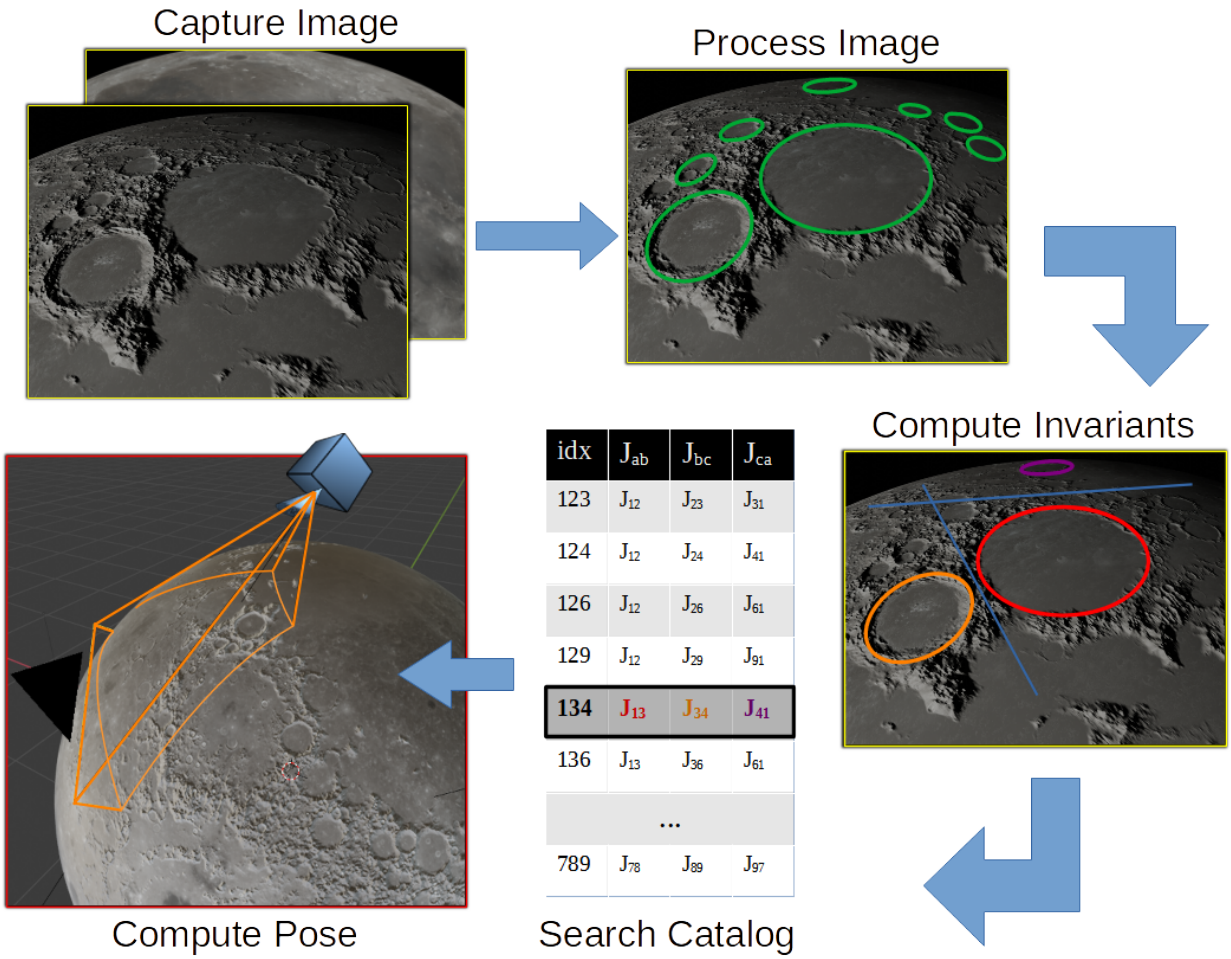

Figure 1.3: High-level flow diagram starting from image capture through to pose estimation. Image processing identifies ellipses in the image, computes the invariant associated with a particular triad of craters, searches the catalog for a suitable set of probable matches, then determines the pose. This particular image is near the lunar south pole. The camera position is shown in the bottom left figure with the camera field of view shown as the orange frustum.

range from image processing to database structures. The purpose here is to apply these general contributions (for which spacecraft navigation is just one application) to existing problems in science and technology. I aim to enhance the ability of platforms to extract and process more information with fewer computations, which can be critical when bits and flops (floating point operations—effectively, computation cycles) are a scarce commodity, such as onboard computers, without sacrificing accuracy. These contributions are listed below:

1.2.1 Sub-pixel boundary localization

Object detection and classification is an essential part of computer vision; detection relies on separating foreground objects and background, which are delineated by boundaries. Here, the focus is line localization, which remains relatively unexplored compared to its boundary cousin: the edge. To differentiate the concept of lines in image processing and that in geometry, I designate image processing lines as *streaks*. While many algorithms extract streak locations at pixel-level accuracy, precise applications rely on more fine-tuned refinement than pixel-level techniques can provide. Refined boundaries better inform the navigator with a better picture of scene layout, and provide a smoother contour over the object. Using a moment-based technique, I develop a streak localization method that uses a simple, non-iterative method that is resilient to noise. This analytic technique allows a simple update step to refine a pixel-level estimation. A comparison of existing techniques suggests a competitive advantage over other applications in accuracy, speed, and simplicity.

1.2.2 Indirect P^2-invariant indexing and query

The solutions for addressing permutation and projective (P^2) invariants (such as in star identification with an uncalibrated camera or for a triad of craters) consist largely of "search all the permutations in succession", which is a highly unsatisfying answer. Using a variant of a popular database system (albeit largely unknown and unused in spacecraft systems), I advance the idea of an indirect "search and store" technique for a set of P^2 invariants into a consistent pattern that should be searched but once, and storing and returning a result with unambiguous ordering. This idea is a concept that applied not only to the specific database implementation, but to spatial database algorithms more generally, meaning that this *indirect search and store* holds promise for fewer cycles on searching enormous databases.

1.2.3 Quadric-based pose estimation

Many natural and artificial objects resemble 3-dimensional extensions of conic shapes, known as quadratic surfaces, or quadrics. Further, planar projections of these quadric objects are always conics (a line is technically a degenerate conic), albeit changed under projective transforms. The relationship between the objects and their projection is a function of location and orientation: exactly what we expect from projective theory. This relationship, unfortunately, is quadratic—that is, nonlinear. While linear relationships are well-behaved

and well-understood, nonlinear systems of equations are comparatively poorly understood. Whether a solution even exists is subject to areas of active research, though for polynomials we do have some relatively well-understood approaches [25], [26]. Exploiting the particular relationship between the quadric and its projection, this section shows how to determine analytically both attitude and position of a spacecraft relative to three circular features with known size and position. If the pose of each quadric is also known, then we can also determine the pose in the world frame.

CHAPTER 2
BACKGROUND

We often ask ourselves en route to a destination: where am I, where am I going, and how to I get there? While these could be metaphorical or existential questions, let us instead address them in the sense of wayfinding. Knowing where we are currently (i.e., our state) tells us the potential paths we must take to arrive at our destination. Additionally—as the adage goes—plans oft go astray, and we periodically reassess our location to detect deviations and refine our plan to correct our wanderings. These updates and assessments are increasingly crucial as mission costs—and that of their failure—skyrocket. *Guidance, navigation, and control* (GNC) answers the questions: *Where am I?* (navigation), *Where am I going?* (guidance), and *How do I get there?* (control).

Navigation is the initial step: it marks the current point of our journey and provides periodic updates to our location. Navigation is the process that determines the vehicle's state (position, attitude, velocity, etc.) [21]. Attitude control, for example, is a critical component of any vehicle and relies heavily on attitude information. The aerospace GNC community employs an array of sensors and techniques to estimate these parameters—LiDAR, GNSS receivers, IMUs, and cameras. Of particular interest to the space imaging community is the digital camera; while configurations may vary (e.g., monocular or stereoscopic, monochromatic or color, visible light or multi-spectral), a camera's purpose is gathering information from the electromagnetic spectrum (i.e., light) that conveys important information. Understanding and interpreting a scene is the domain of computer or machine vision.

2.1 Image processing and computer vision

2.1.1 Camera model

Computer vision—like human vision—seeks to interpret and analyze information in a scene. Its purpose might be to identify faces, detect surface features or landmarks, or find particular objects. Facial recognition uses human facial patterns to identify individuals, while terrain-relative navigation (TRN) [27] uses feature detection to navigate relative to those landmarks, and hazard detection and avoidance (HDA) flags objects which may hinder operations—the latter two of these (TRN and HDA) are of particular interest to space navigators. While digital cameras remain the primary device to capture images, this approach

is fundamentally the same for film cameras and other imaging systems.

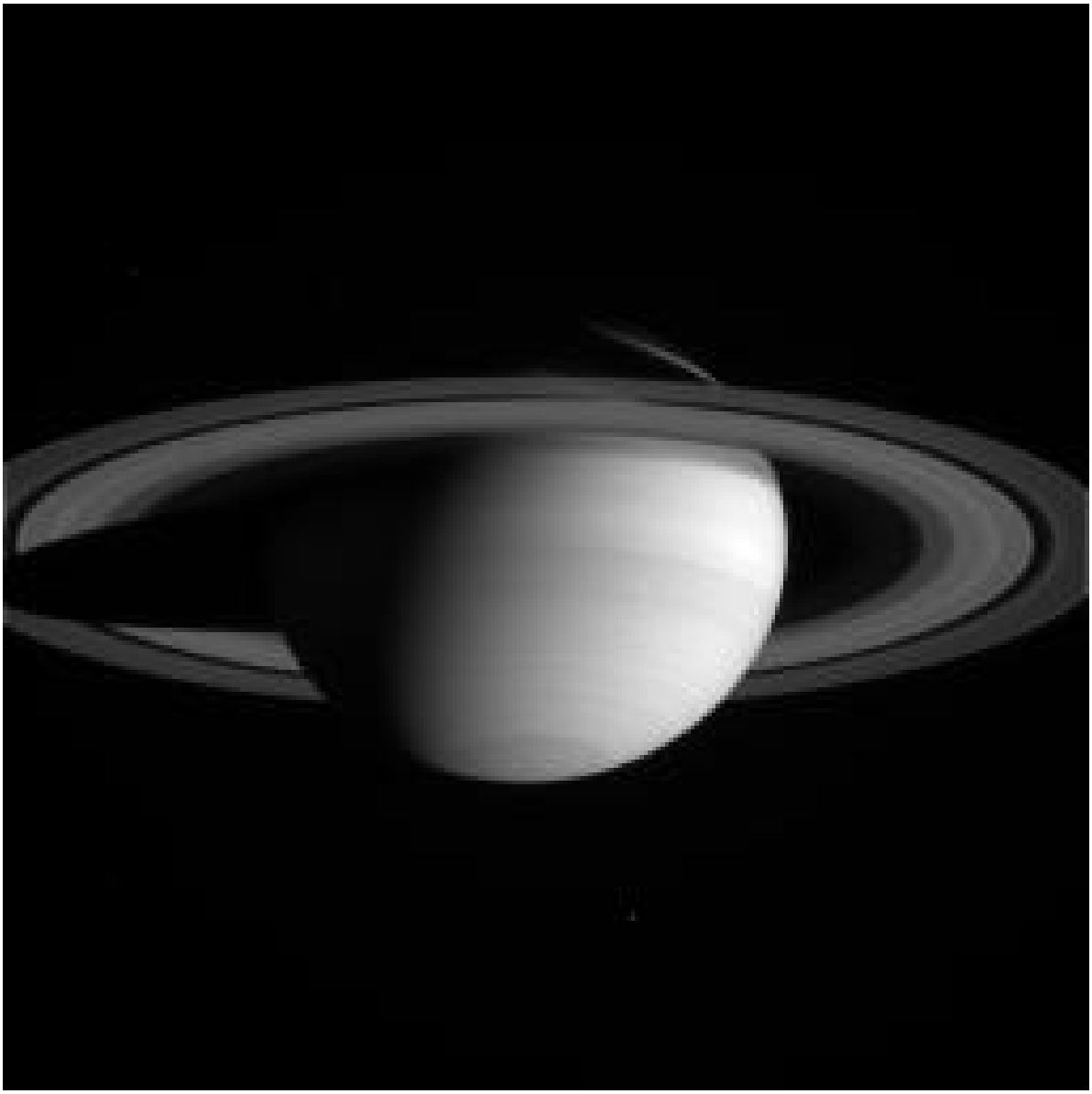

Figure 2.1: Cassini's view of Saturn. Image ID: $N1460342538_1$**. Courtesy NASA/JPL-Caltech.**

Cameras capture rays of light that emanate from light sources like the Sun. This light can be direct or can find its way to the camera after reflecting or scattering off of other objects. The resulting image is a net result of all of these interactions. For example, Figure 2.1 shows Saturn as captured by Cassini [7]; different areas absorb more light than others, and others disperse light at different rates, giving sometimes subtle and sometimes bold differences. The culmination of these reflections, absorptions, and dispersions gives us a breathtaking view of

Saturn, even in monochrome.

This interaction between the outside world and a camera depends heavily on the optics, which itself is its own scientific discipline. We will not discuss most of the physics here, but will consider a simplified representation. Let us consider a ray of light say, from the sun, that reaches an ideal camera—that is, a camera devoid of distortions, aberrations, and misalignments. The angle it makes with respect to the camera determines where it impinges the focal detector, usually a CCD (charge-coupled device) or CMOS (complementary metal oxide semiconductor) device. This detector acts much like a film in older cameras, but converts photons into electrons. Figure 2.2 illustrates how collimated (parallel) light from a given direction focuses onto a single point on the photodetector. This is known as the ideal pinhole camera model.

This work uses the pinhole camera model, of which a simple explanation is provided here. More information about this model and its simplifications and limitations is available in Appendix A, along with discussion about modelling lenses, misalignment, and distortions. Furthermore, this topic also requires some knowledge about projective geometry, which is explained in Section 2.2, and treated in more depth in Appendix B.

The image plane shown in Figure 2.2 contains a virtual image; that is, this plane—unlike the focal plane which has a physical existence—is purely a construct to avoid worrying about inverted images. The pinhole model permits a special relationship between points in 3D and projected points onto the 2D focal plane and image plane, given by [28],

$$\begin{bmatrix} x_f \\ y_f \\ -f \end{bmatrix} \propto \begin{bmatrix} \bar{x}_i \\ \bar{y}_i \\ 1 \end{bmatrix} \propto \begin{bmatrix} x_i \\ y_i \\ f \end{bmatrix} \propto \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{2.1}$$

where $[X_c, Y_c, Z_c]^T$ denotes the 3D object coordinates relative to the camera's frame, $[x_f, y_f]^T$ the 2D coordinates of that same point projected onto the focal plane, and $[x_i, y_i]^T$ the 2D coordinates of the point projected onto the image plane ($[\bar{x}_i, \bar{y}_i]^T$ are the homogeneous pixel coordinates) [28]. This means that points lying on a ray emanating from the focal point are equivalent. Therefore, all points expressed in Eq. 2.1 describe the same point in $\mathbb{P}^2$, where $\mathbb{P}^n$ is n-dimensional projective space.

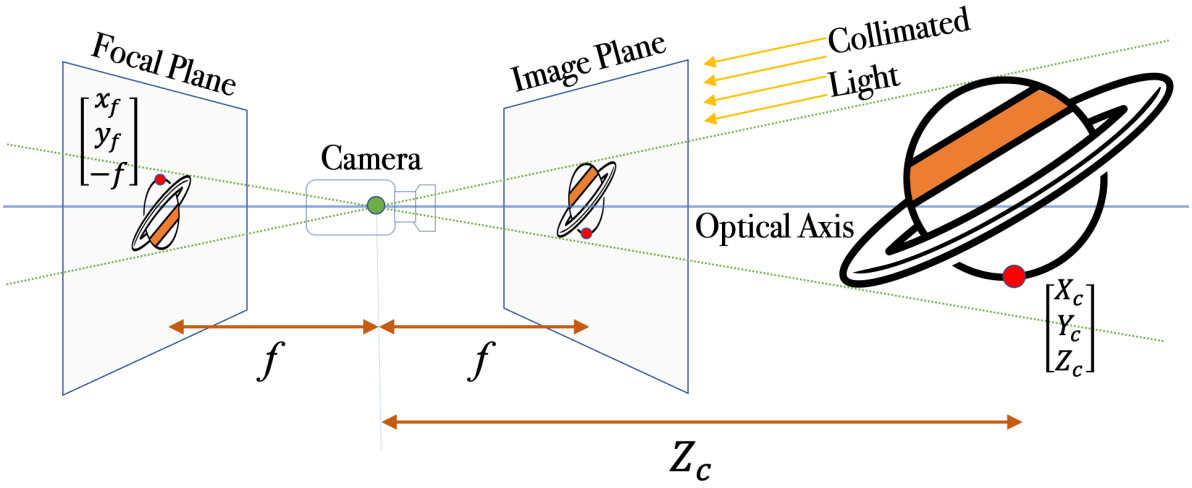

Figure 2.2: Ideal pinhole camera model. Note that the image on the focal plane is a scaled, mirrored image of the original object. The focal plane represents the point at which the image is in perfect focus and is located at a distance f (focal length) from the pinhole location, and it represents a physical plane in space. The image plane, conversely, represents a mathematical simplification of the focal plane. Using the image plane avoids working with mirror images as well as the requisite mathematical bookkeeping; this plane is parallel to the focal plane and lies in front of the camera's center of projection (i.e., along the principal axis). The points indicated in red on the object and its projected image corresponds to the projection of a specific point onto both the image and focal planes.

2.1.2 Coordinate frames and conventions

Given a single-channel image (like a monochrome image as opposed to a multi-channel RGB color image) with n rows and m columns, we can describe any point in the image using row-column indexing in pixel space with the origin at the upper left corner of the image, and the center of each pixel denotes a non-negative integer value of u or v. Conversely, we can describe the same point in the image plane using xy coordinates, where the principal axis of the camera denotes the origin (often the center of the image). Each pixel, denoted (u_i, v_i), stores a single intensity value, with the range depending on the number of storage bits per pixel—usually 8 bits (0-255) or 12 bits (0-4095). The x- and u-axes are parallel and

point right in the image plane; similarly, the y- and v-axes are likewise aligned and point downward in the image plane.

Expressing a set of k points in an image, we could write the set

$$\{\tilde{u}_i, \tilde{v}_i\}_{i=0}^{k} \subset \mathbb{Z}^{*2} \tag{2.2}$$

where $\mathbb{Z}^*$ is the set of non-negative integers, and the tilde denotes a selected point in the image.

2.1.3 Real cameras

Ideal pinhole cameras are physically unrealizable since the perfect alignment and structure of an ideal camera is not possible, even with the most sophisticated manufacturing techniques. In addition to optical assemblies exhibiting imperfections and misalignment, the finite size of the aperture (rather than being infinitely small) requires systems of lenses to focus the light, causing aberrations and distortions. Designing cameras therefore becomes a balancing act of designing which distortions to minimize at the cost of increasing others. Aside from lens distortions (e.g., spherical, radial, and curvature of field [28]), we note distortions deriving from the physical sensor itself: defocus and discretization.

Improper alignment and placement of the focal plane array induces defocus distortion. Ideally, the focal plane array (comprised of either CCD or CMOS technology) lies precisely at the focal length. Any deviation, shown in Figure 2.3, produces defocus distortion: an aberration that blurs objects. Sometimes, this defocus is intentional, as in star trackers [29], [30] that better pinpoint a star's centroid due to the spread of that same ray across a neighborhood of pixels rather than a single pixel.

Discretization occurs from discrete photon collectors in pixel arrays, with each collector taking up physical space. The detector absorbs light from rays that enter the camera from potentially slightly different source directions; see Figure 2.3. This is similar to a binning procedure. These aberrations, along with other distortions, mean that points in space are imperfectly focused and are therefore projected distortions of physical objects. This particular fact is useful when considering object boundaries in the image.

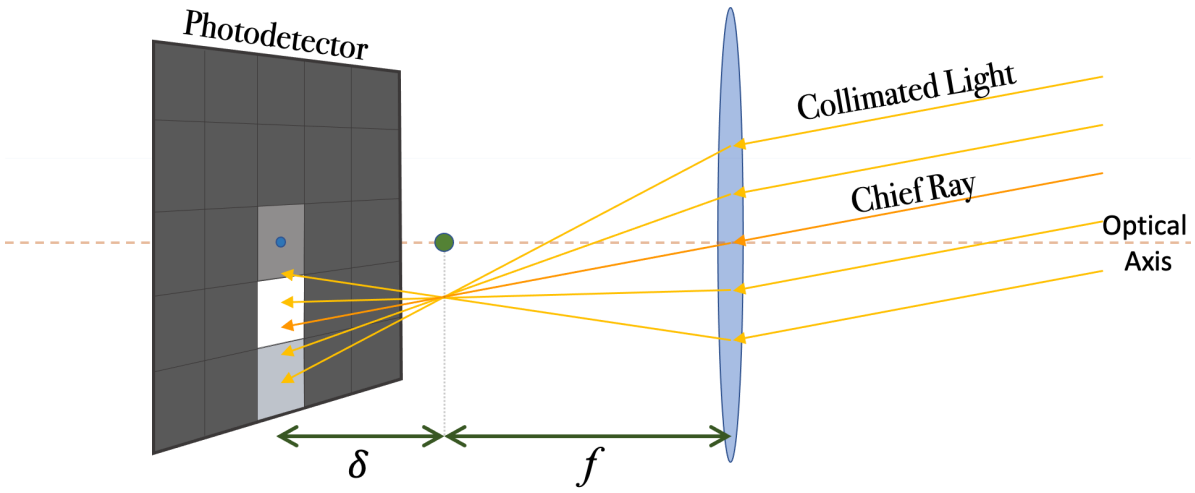

Figure 2.3: Point spread function of a camera. Collimated light enters the camera lens and diffracts; all light from a given direction focuses onto one point on the focal plane. Any displacement δ of the photodetector away from the focal point produces defocus distortion, shown as rays spreading out beyond the focal point, which is shown in green.

2.1.4 Streaks and edges

Object identification is a complex domain. Even in simple images—maybe of only one or two objects—the information contained in an image can be hard to decompose into objects. The desire is to simplify the task down to a number of primitives that define an object; these primitives can then help determine locations and orientations, which are useful in object recognition and navigation purposes. For example, object detection relies on defining borders between foreground (objects of interest) and background. These boundaries are of natural interest to the computer vision specialist, as these define the shape and features of objects in the image. Let us differentiate two basic primitives that can define simple boundary: streaks and edges. We therefore have two questions about streaks and edges: how do we define these primitives and how do we detect them for object detection?

This task is easy for humans: our brains naturally process boundaries. The task becomes more nuanced and complicated as we consider how to mathematically define a boundary. Note the use of the term "streak" instead of "line"—the common term in image

processing literature; this is to disambiguate our later discussion in geometry where *lines* are fundamental constructs and differ substantially in meaning. Despite both terms—streaks and edges—being common, everyday terms, image processing experts have long disagreed on a concrete definition of both. Here I provide a working definition for both in boundary detection for digital images to differentiate the two.

Imagine two regions: one dark and the other light, as seen as the gray and white areas in Figure 2.4. An edge is a discontinuity—a jump or a drop—in pixel values in one direction across a boundary with little change perpendicular to the boundary. An edge is a change, therefore, from one uniform region to another uniform region and can be straight or curved. Figure 2.4 illustrates the general appearance of an edge.

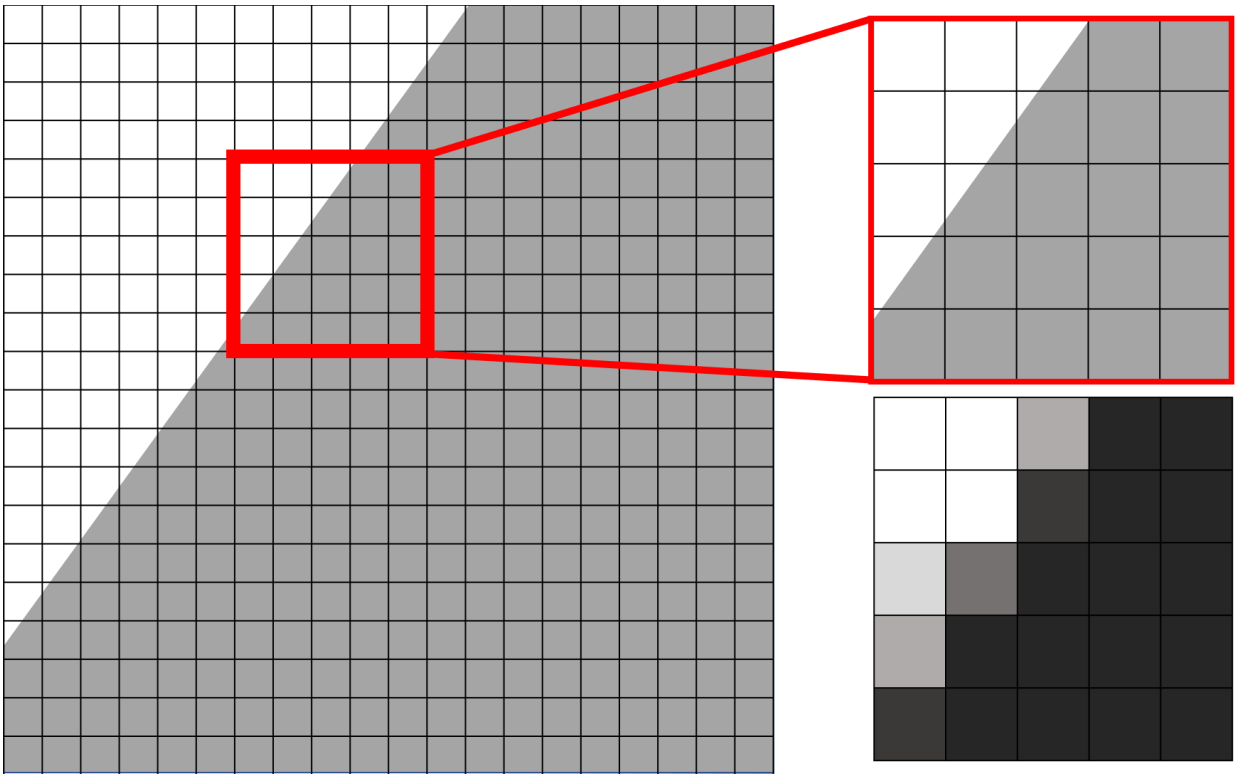

Figure 2.4: Edge in an image. On left, the underlying signal of the edge overlaid with the photodetector array grid, with the top right showing a magnified portion of the signal and the array. Bottom right shows how discretization fails to produce a crisp boundary even in the absence of noise and the point spread function.

Conversely, imagine a portion of the image with a uniform color with a single thin strand of contrasting pixels passing through this area, as in Figure 2.5. This is an example of a streak. A streak, in contrast to an edge, is a sudden change in pixel intensity to a higher or

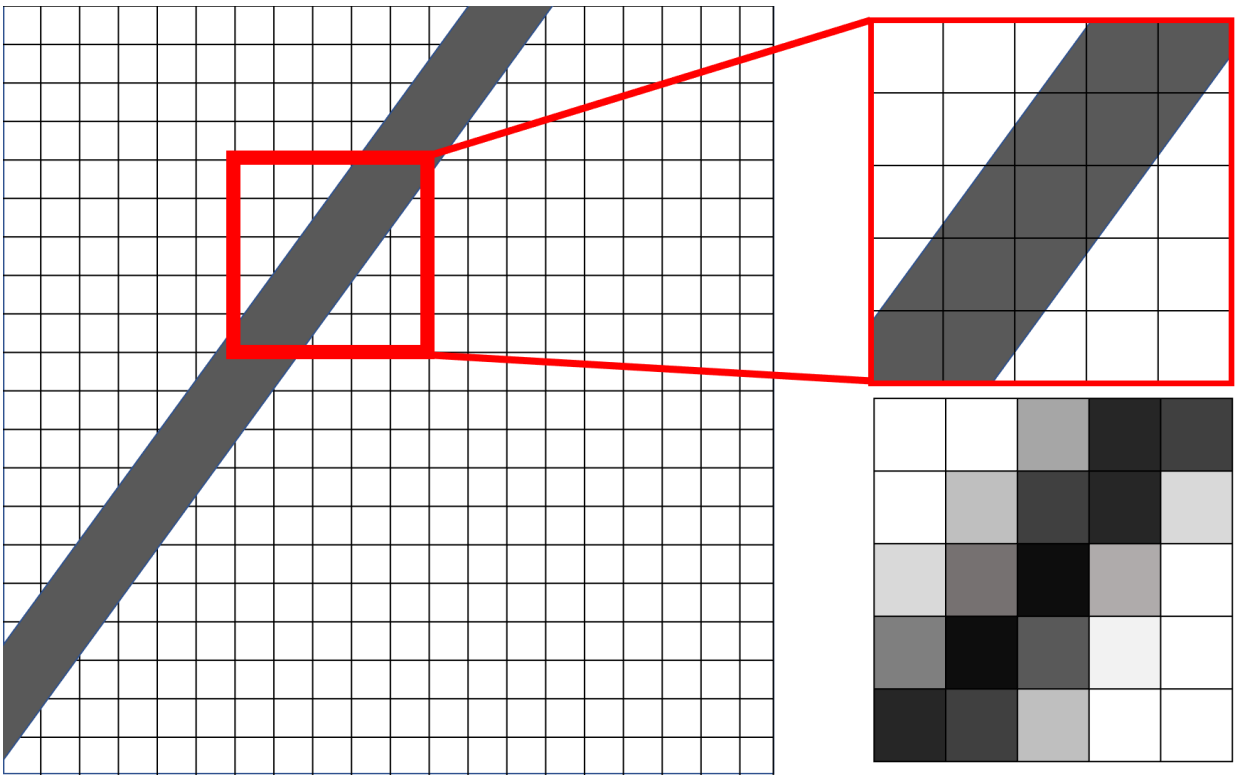

Figure 2.5: Streak (line) in an image. On left, the underlying signal of the edge overlaid with the photodetector array grid, with the top right showing a magnified portion of the signal and the array. Bottom right shows how discretization fails to produce a crisp boundary even in the absence of noise and the line spread function.

lower value in one direction, then a reversal back to the original value. Again, the change occurs in one direction and little change in the normal direction. While Figure 2.5 shows a straight region of dark pixels, a streak may also be curved. Some authors describe streaks as ridges or valleys [31], but we have no need of this differentiation due to the agnostic view that this contribution takes.

Defocus and discretization affect the way in which the detector registers edges and streaks, making blurred boundaries from crisp, distinct delineations. It is possible to estimate both the distortions from quantization and the point spread function (PSF) of a given camera.

Several algorithms address pixel-level edge detection; among them are the well-known Sobel [32], Prewitt [33], Marr-Hildreth [34], Canny [35], among others. Additionally, recent advances in deep learning have spurred machine learning algorithms for edge detection [36], with prominent contemporary examples including DeepEdge [37], DeepContour [38], Holistic Edge Detector (HED) [39], and Crisp Edge Detection (CED) [40]. Streak detection often uses

the brushfire algorithm (also known as the medial axis algorithm), likened to how a brushfire will burn inwards until the fronts meet at the middle point [41]. Each method determines the set of pixels associated with a discontinuity.

2.1.5 Optics and Zernike polynomials

The question is: Are pixel-level estimates "close enough"? The answer depends on the application. Some might find quick and dirty estimates sufficient; other uses, however, might demand more refinement at the cost of more computation. For example, accurate ellipse fitting in images for planetary optical navigation [42] trades higher computing requirements for better refinement; without refined edge locations, the ellipse fit could suffer and mistakenly produce invalid or—worse—incorrect outputs. Refining the rough pixel-level estimates to finer granularity is known as *sub-pixel localization*.

These techniques fall into four general approaches: moment-based [43], [44], interpolation [45], least-squares fitting [46], and partial area effect [47]—currently considered the state of the art for edge localization. The method described in this section falls under moment-based methods. It uses the an orthogonal polynomial set used in optics known as Zernike polynomials [48]. Friedrich Zernike, winner of the Nobel Prize in Physics, contributed the orthogonal circle polynomials used in the algorithms. Originally used to describe optical aberrations in lenses, these polynomials are popular among optics and image processing experts and have found uses in many applications [42], [44], [49]–[53].

Some works have already presented Zernike polynomials and Zernike moments in sub-pixel edge refinement; this previous work [43], however, uses an approximation with several disadvantages. Recent work with Zernike moments [42], [54] better approximate the transition between regions, with the work of [54] on streaks acting as the foundation of this section. The advantage of using these methods over previous attempts at streak localization will become readily apparent, especially considering the dearth of alternative applications with similar analytic approaches. Chapter 3 derives the functions and further develops this idea; these results will then be analyzed quantitatively on synthetic images and qualitatively on real images.

2.2 Geometry and invariants

Geometric properties are fundamental concepts in mathematics. Certain properties formed by sets of lines—distance and angles in Euclidean geometry—are preserved under particular transformations like translation and rotation, meaning that the lengths and angles are *invariant* properties of Euclidean transformations. While Euclidean geometry and its unchanging lengths and angles might be the most familiar form, it exists as part of a geometrical hierarchy, with the familiar Euclidean being the most restrictive. I shall only mention the fundamentals of this geometry hierarchy (see Figure 2.6) and projective geometry to exploit associated invariant properties for image analysis, with further explanation in Appendix B.

Geometric invariants are underlying properties that remain unchanged for certain geometric constructs (known as *varieties*) under the action of particular groups. Groups and varieties are defined and described in greater detail in Appendix B.1. Note in Figure 2.6) that each level's invariant set includes all invariants listed in lower layers—that is, invariants propagate up the hierarchy, but not down. For example, in Euclidean geometry, distance and area are invariants, meaning that rotations and translations don't change these two properties. Conversely, we find that the notions of distance and area invariance in projective space are nonexistent, which partly explains the ambiguity and complexities of working with images. This ambiguity is often problematic for image processing, but filmmakers can use this to great effect to represent huge objects by using detailed miniatures, and even have done this for people as in the Lord of the Rings trilogy [55] to simulate size differences between actors.

2.2.1 Geometric hierarchy

We start at the most restrictive level of the geometry hierarchy: Euclidean geometry. Known for millennia simply as geometry (they never imagined needing anything else!), basic axioms established the behavior of points, lines, and angles in 2D and 3D. Euclid's *Elements* [56] is a foundational work in mathematics: a compilation of existing knowledge into a strong framework used for millennia afterwards. Certain additions followed that aided in analytic treatment within the space, such as the coordinate system often attributed to Descartes (though coordinate systems predated him by millenia) [57], [58], allowing algebraic treatment of geometric properties. Certain transformations, known as groups, preserve particular properties of objects despite these objects undergoing a transformation. These constant

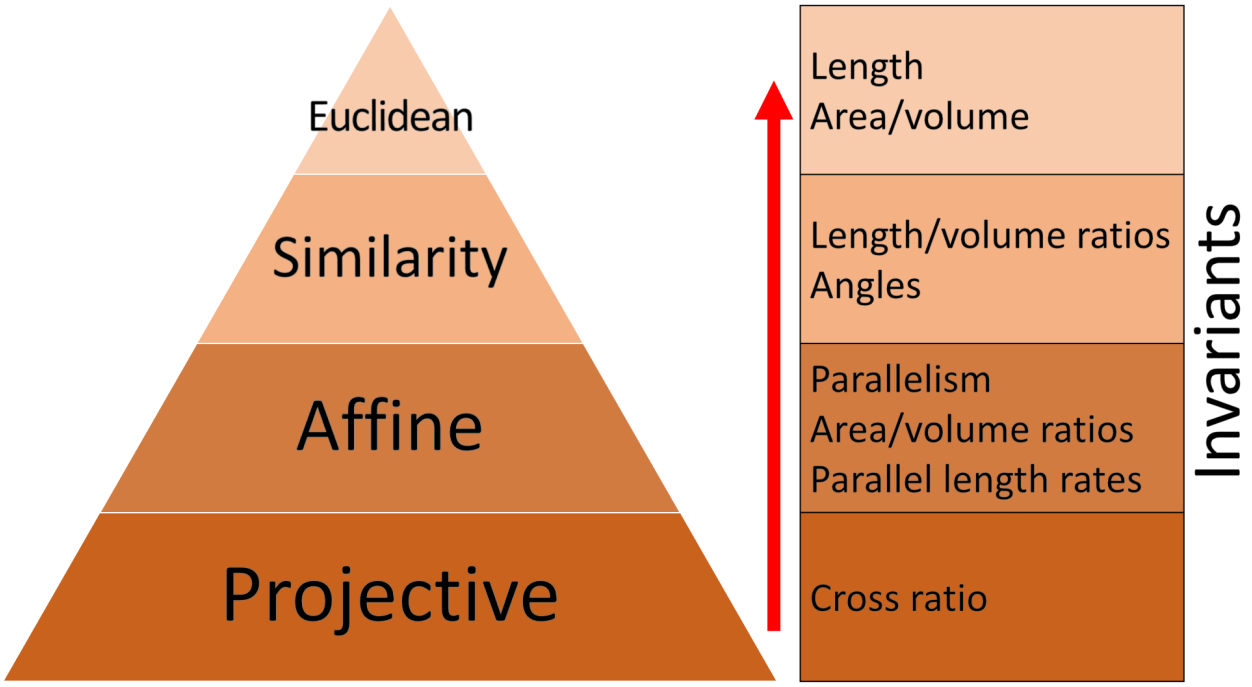

Figure 2.6: The projective geometry hierarchy. Moving down from Euclidean to projective geometry, we reduce the set of invariants available in each group.

properties, known as invariants, provide valuable, unchanging information about the object. Translation and rotation comprise Euclidean transformation group, and the invariants are lengths and areas [21]. We can see that despite undergoing both transformations, the polygons' sides in the upper left of Figure 2.7 sides remain the same lengths, and the enclosed area remains exactly the same [9]. Geometers often label this group $E(n)$, where n denotes the number of dimensions of the space.

Skipping to the base of the hierarchy, we arrive at the least restricted transformation: the projective transformation group, denoted $PGL(n+1)$ (projective general linear group). Here, we lose all notion of parallel lines and constant areas and volumes, as seen in the bottom right of Figure 2.7. At this point the question arises: What is preserved under such a transformation?

2.2.2 Projective space and homogeneous coordinates

Before answering that question, let us introduce the concept of projective space—specifically the real projective space, though it is also more generally defined over the complex

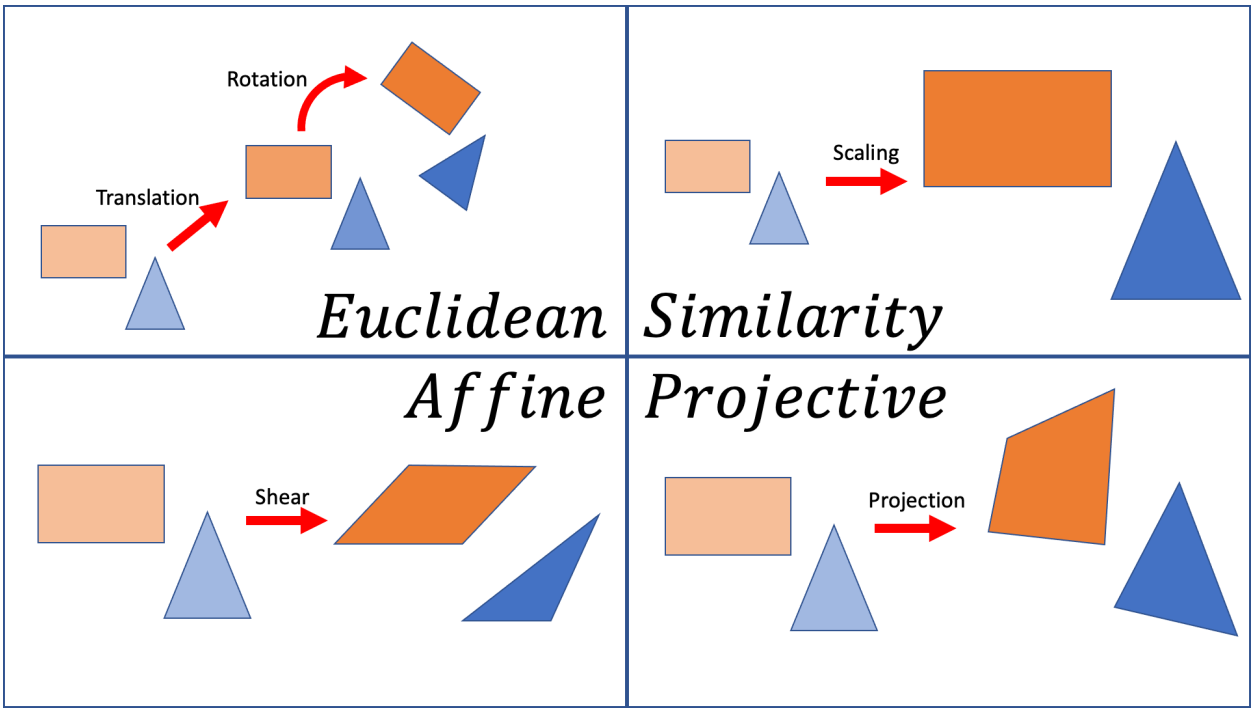

Figure 2.7: The projective geometry transformations. Illustrates the groups of transformations allowed in each group. Every group inherits allowable transformations from those above it in the hierarchy; that is, the similarity group has all of the invariants of the affine and projective groups, but not those of the Euclidean.

numbers [9], [21], [59]. We use the notation $\mathbb{P}^n$ to mean n-D projective space, where each vector representing a fundamental object contains $(n+1)$ components. In addition, we familiarize the reader with homogeneous coordinates—a popular representation of coordinates in use from geometry to computer graphics to spacecraft navigation. We introduce an additional parameter to define a point in n-dimensional space. While this parameter seems extraneous it provides several useful properties. We actually caught the first glimpse of homogeneous coordinates in Section 2.1.1 with its equivalence relationship between a point in 3D space and its projection to both the image plane and focal plane, shown in Equation 2.1. While the points are not *equal*, we refer to the relationship as *equivalent* in 2D space. We can see this by scaling the components of the vector on the right—X_c, Y_c, and Z_c—by Z_c, which gives,

$$x = \frac{X_c}{Z_c} \tag{2.3}$$

$$y = \frac{Y_c}{Z_c} \tag{2.4}$$

$$z = 1 \tag{2.5}$$

which produces exactly the second vector of Eq. 2.1. The same can be said of the left vector if we know that the relationship between the x-components and y-components is a negative sign for both after we normalize by the focal length.

Another trait of homogeneous coordinates is we can express a pure direction, or the concept of a point at infinity in 2D or a plane at infinity in 3D projective space—known as ideal points and ideal planes, respectively. Ideal points or plains are expressed when the additional component is zero. One stipulation is that not all values in homogeneous coordinates may be simultaneously zero, which we express as $\mathbb{R}^n/\{0\}$. The projective space and homogeneous coordinates have a special relationship [59], [60]. Furthermore, homogeneous coordinates give rise to another important concept in projective geometry: duality.

Duality in projective space implies that there is a one-to-one reciprocal relationship [17], [19], [59]. Any theorem in projective geometry that applies to one side of the relationship, applies also to its dual [61]. In $\mathbb{P}^2$, the two dual objects are points and lines; in $\mathbb{P}^3$, points and planes.

In later sections, these dualities and invariants provide the mathematical structure to exploit geometric relationships in determining attitude estimation of both calibrated and uncalibrated cameras.

2.3 Conics, quadrics, and projections

Normally, computer vision experts and practitioners employ several techniques to determine pose; chief among them is the *perspective-n-point*, or PnP, method [62]–[64]. This approach uses the point correspondences that exists between the 3D locations of n points and their projections onto the image plane. The PnP method assumes a calibrated camera (which we do as well), but methods such as UPnP [65] and the Direct Linear Transform [66] relax this assumption.

The PnP method is a well-tested and well-used tool in the computer vision quiver; aside from a non-collinearity constraint and some reasonable assumptions, this method is simple, direct, and non-iterative. So why is this powerful and popular technique inappropriate here?

2.3.1 Shortcomings of existing techniques

While the PnP has serious power to compute the camera pose given a set of point correspondences, therein lies the inability for PnP (or any of its various forms or alternatives): **points**. While the conic locus *is* the set of points making up the conic surface, there are no unique points that we can match one-to-one to image points. To illustrate this shortcoming, observe that we cannot just select a point on the image conic in Figure 2.8 and trace it to its original shape without knowing the projection; that is, we cannot solve the PnP problem because we cannot uniquely identify points. We must, therefore, seek an alternative approach, leveraging what we know about conic sections.

In addition to PnP and its variants, there are several existing techniques using conics and conic correspondences [67]–[70]; these techniques generally use planar conics, which is explicitly not the case here, where each of the conics may exist in different planes.

2.3.2 Conics

Elementary geometry defines conic sections as the set of shapes formed by the plane intersecting a two-sided 3D cone. These shapes include circles, ellipses, parabolas, and hyperbolas. We focus here on the ellipse and the circle (a special case of an ellipse). Ellipses are of particular interest to space navigators since can approximate the general shape of planets and other large, self-gravitating bodies in images [28], [71], [72]; furthermore, we may express many craters on the surface of the Moon or other similarly cratered surfaces as ellipses. The set of points forming a conic section satisfy the equation of an ellipse centered at the origin

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{2.6}$$

where a and b denote the major and minor axes, respectively. If, however, the ellipse does not lie at the center or the ellipse is tilted at an angle φ as in Figure 2.9

$$Ax^2 + Bxy + Cy^2 + Dx + Fy + G = 0 \tag{2.7}$$

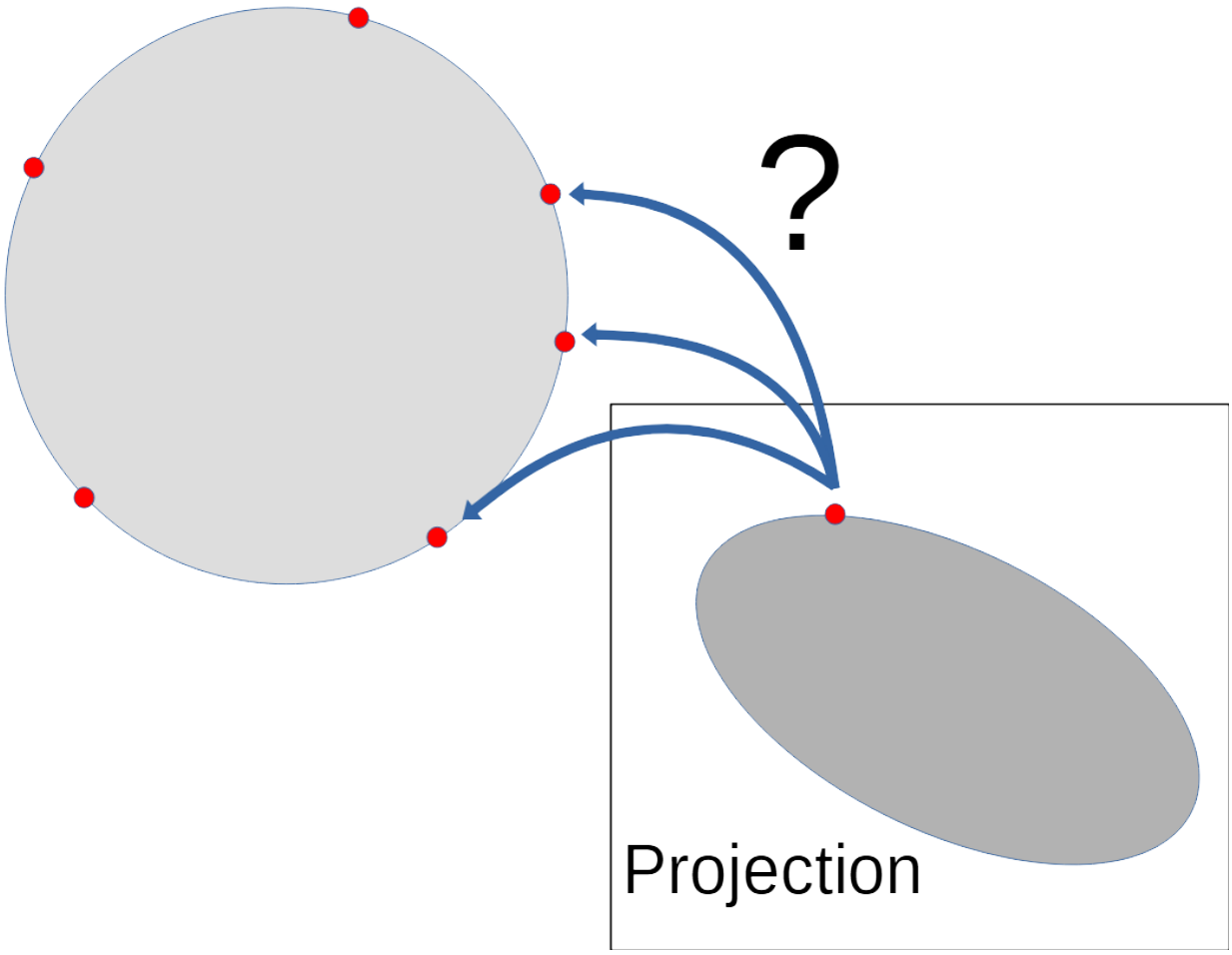

Figure 2.8: The ambiguity of choosing points on the projected ellipse and the original ellipse (here, a circle). It is to illustrate that no point on the projected ellipse can be identified uniquely given no information about the camera pose.

and forms an ellipse when the coefficients satisfy

$$B^2 - AC < 0 \tag{2.8}$$

We can also express the same idea using homogeneous coordinates, where the points must satisfy the following equation

$$\bar{x}^T C \bar{x} = 0 \tag{2.9}$$

where C denotes the symmetric matrix of full rank

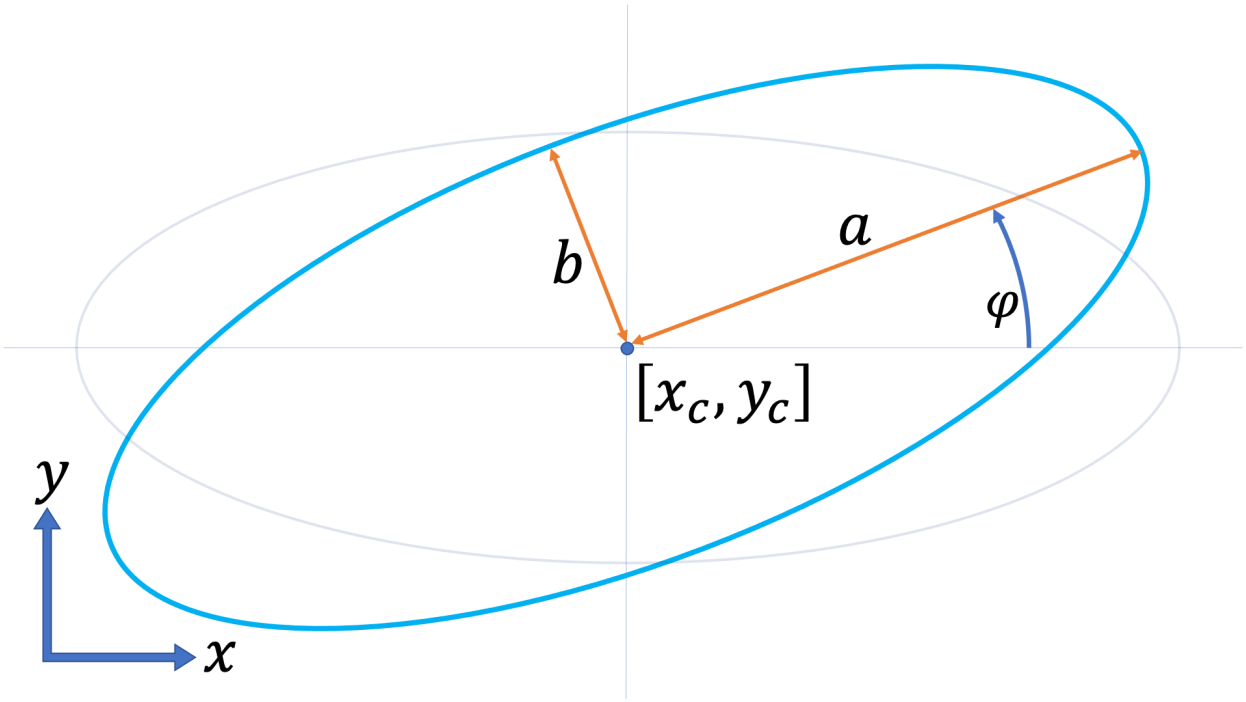

Figure 2.9: Ellipse.

$$C = \begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & F/2 \\ D/2 & F/2 & G \end{bmatrix} \tag{2.10}$$

which represents the conic in matrix form. Each point satisfying Eq. 2.9 belongs to a set known as the *conic locus*. The matrix C describes a conic of unknown scale, meaning that the matrices C and sC denote the same conic, with $s \neq 0$. The conic locus, consisting of points, has a dual in $\mathbb{P}^2$: the *conic envelope*. The conic envelope describes the set of lines tangent to the surface of the conic (remember that points and lines are dual objects in $\mathbb{P}^2$).

To obtain the dual of the conic locus, we note that a point $\bar{p}$ in $\mathbb{P}^2$ lies on a line ℓ when $\bar{p}^T\ell = 0$, and that a line is tangent to a conic surface if $\ell = C\bar{p}$ [66]. Therefore, using $C^{-1}\ell = \bar{p}$ [61]

$$(C^{-1}\ell)^T C(C^{-1}\ell) = 0 \Rightarrow \ell^T C^{-1} C C^{-1}\ell = 0 \Rightarrow \ell^T C^{-1}\ell = 0 \tag{2.11}$$

We can further simplify computation using the adjugate matrix of C, denoted by C^*, since $C \propto C^*$ and the scale of the conic is ambiguous. The adjugate (or classical adjoint) is

similar to a matrix inverse, but does not involve a scaling by the matrix determinant, so the adjugate always exists for a matrix. It can be computed by finding the cofactor matrix and taking its transpose. This dual (the adjugate matrix C^*) represents the set of lines tangent to the ellipse. We will use the conic envelope later rather than the conic locus based on the recommendations of [66].

2.3.3 Quadric surfaces

Conics reside in 2-dimensional space, yet the objects we hope to model lie in 3D. Let's extend the idea of conic sections to $\mathbb{R}^3$ and $\mathbb{P}^3$ using the notion of quadratic surfaces, often called quadrics.

Similar to conics, we can represent quadric surfaces with a matrix, this time a 4×4 symmetric matrix, where each point satisfying the equation

$$\bar{X}^T Q \bar{X} = 0 \tag{2.12}$$

lies on the surface and comprises the *quadric locus*, where $\bar{X}$ represents a homogeneous point in $\mathbb{P}^3$. As in conics, the quadric locus has a dual: the set of planes tangent to the quadric surface. If the matrix Q is full rank, it is a non-degenerate quadric surface. Examples of non-degenerate surfaces are spheres, ellipsoids, and paraboloids; degenerate quadrics include cones and various cylinders.

In many cases, the shape of large celestial bodies (e.g., the planets) are well-modeled as quadric surfaces such as ellipsoids; some bodies, like the Moon, are very nearly spherical. The Earth itself is often represented as an oblate spheroid—a somewhat flattened sphere. Many space missions have used this representation to identify their position relative to a celestial body using optical navigation techniques [28], [42].

Invariants in $\mathbb{P}^3$ include the cross ratio, discussed earlier. Christian et al [20] further develop this idea to represent craters on the surface of the Moon as quadrics, modeling each large crater on the lunar surface as the intersection of a quadric cone $Q_{co,i}$ (a degenerate surface) and the plane π_i formed by the conic A_i representing the crater in a local frame in $\mathbb{P}^2$, as seen in Figure 2.10.

Additionally, they use the dual of the intersection (for purposes described in [20]), which they refer to as the *disk quadric*, denoted Q^*, which is defined by the planes satisfying

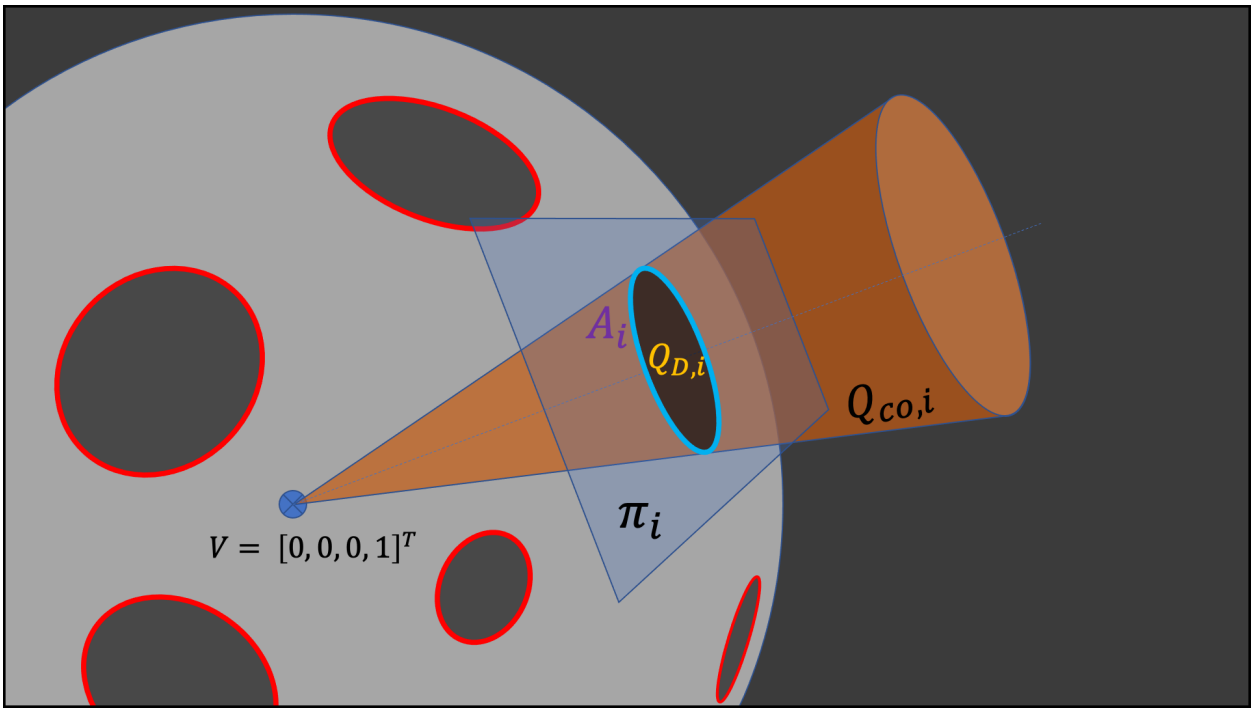

Figure 2.10: Model of the quadric defined as the intersection of a quadric cone, $Q_{co,i}$, and the plane π_i defined by the ellipse formed by the crater rim, A_I. The Moon-centered frame has the origin $V = [0,0,0,1]^T$.

$$\pi^T Q^* \pi = 0 \tag{2.13}$$

where π represents a plane in $\mathbb{P}^3$.

2.3.4 Invariant descriptors, permutations, and search

One of the foundations of [20] is that for a given set of non-coplanar craters (modeled as conics), there exists a set of invariants from the intersection of the respective crater quadrics (if they lie on a non-degenerate quadric surface). The order of the quadrics matters, meaning that $ABC \neq BCA \neq CBA$ or any other combination. Therefore, the six different combinations of the three craters produces six different invariant descriptors. This implies that we must search the catalog six times to guarantee that we could possibly find the correct combination. That is, unless we can reduce this search, which is resolved in a Chapter 4.

2.3.5 There and back again: quadrics to conics, and vice versa

Cameras gather 3D scene information and store it in a noisy 2D array of integer pixel values. Aside from the problems posed in image processing such as boundary localization, the camera effectively performs a projection of 3D objects onto a 2D plane, as shown in Figure 2.11.

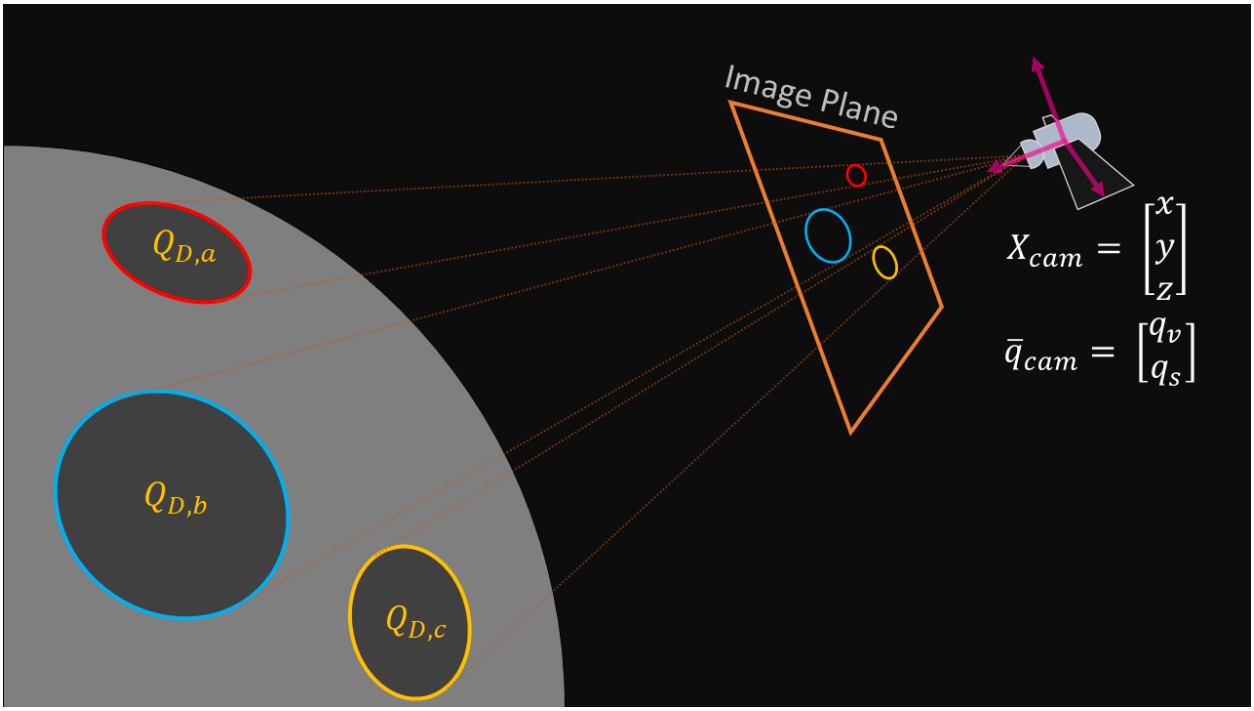

Figure 2.11: Quadric projection onto a plane. Each disk quadric projects to a corresponding ellipse on the image plane. Only the ellipse data is known, and the purpose here is to determine the pose (i.e., the position and attitude) of the camera given crater images.

The next step shows how quadrics project onto a plane, resulting in a conic. Zisserman and Hartley describe this relationship [66] in their seminal work as follows

$$C^* \propto PQ^*P^T \tag{2.14}$$

where C^* is the dual of the image conic (conic envelope), P is the camera projection matrix, and Q^* is the quadric envelope; Chapter 8 of *Multiple View Geometry* [66] discusses the choice of the envelope over its dual—the locus—for both the quadric and the conic. Chapter 5 describes how to use this relationship to calculate camera pose. Also note that the relationship between the two is proportional since the scalar multiple of both the conic or the quadric

equation (or matrix) describes the same object.

CHAPTER 3
SUBPIXEL STREAK LOCALIZATION

Streaks and edges, as mentioned in Section 2.1.4, constitute two basic primitives in image processing. Both primitives can represent straight or curved boundaries, and are fundamental in object recognition and machine vision. While this work describes streak localization, a similar approach is also possible to refine edges.

Streaks, like edges, separate background and foreground objects; unlike edges, however, they separate two regions of the same general intensity rather than remaining at the new intensity level, similar to a thin strand of material superimposed on a background, as seen in Figure 3.1. Note the rise and fall of the intensity across the image. As mentioned previously, some works describe streaks as "ridges" or "valleys", discerning between a rise and a fall, respectively, of intensity across a 1D profile; this works makes no attempt to distinguish the two since they are handled in exactly the same way with no particular treatment between rising or falling intensities.

Instead of sharp, crisp discontinuities, Figure 3.1 shows the border more gradual, a combination of discretization, defocus, and other effects—intentional or otherwise.

These combinations of distortions combine into a function akin to the *point spread function*. The PSF is the response of a camera or other image system to a single point, as demonstrated in Figure 2.3 where a single point source spreads to neighboring pixels rather than a single pixel in a perfectly aligned system. This function for streaks, which we call a *line spread function* (LSF) [54], describes the effect of the imaging system on a single streak. This LSF causes once crisp dilineations in scenes to bleed information into a local neighborhood of pixels.

The goal in streak localization, therefore, is to determine precisely where the crisp boundary would be despite the effects of the camera. One method to find these is image moments.

Image moments are a foundational primitive of boundary detection and image segmentation, as well as image processing at large. Moments compactly describe a 2D intensity pattern of a local neighborhood of pixels with a minimal set of parameters. Describing a local neighborhood is simply applying a kernel of weights to the pixels, with the weights corresponding to a particular basis function. Applying a basis function P_{nm} to a given 2D

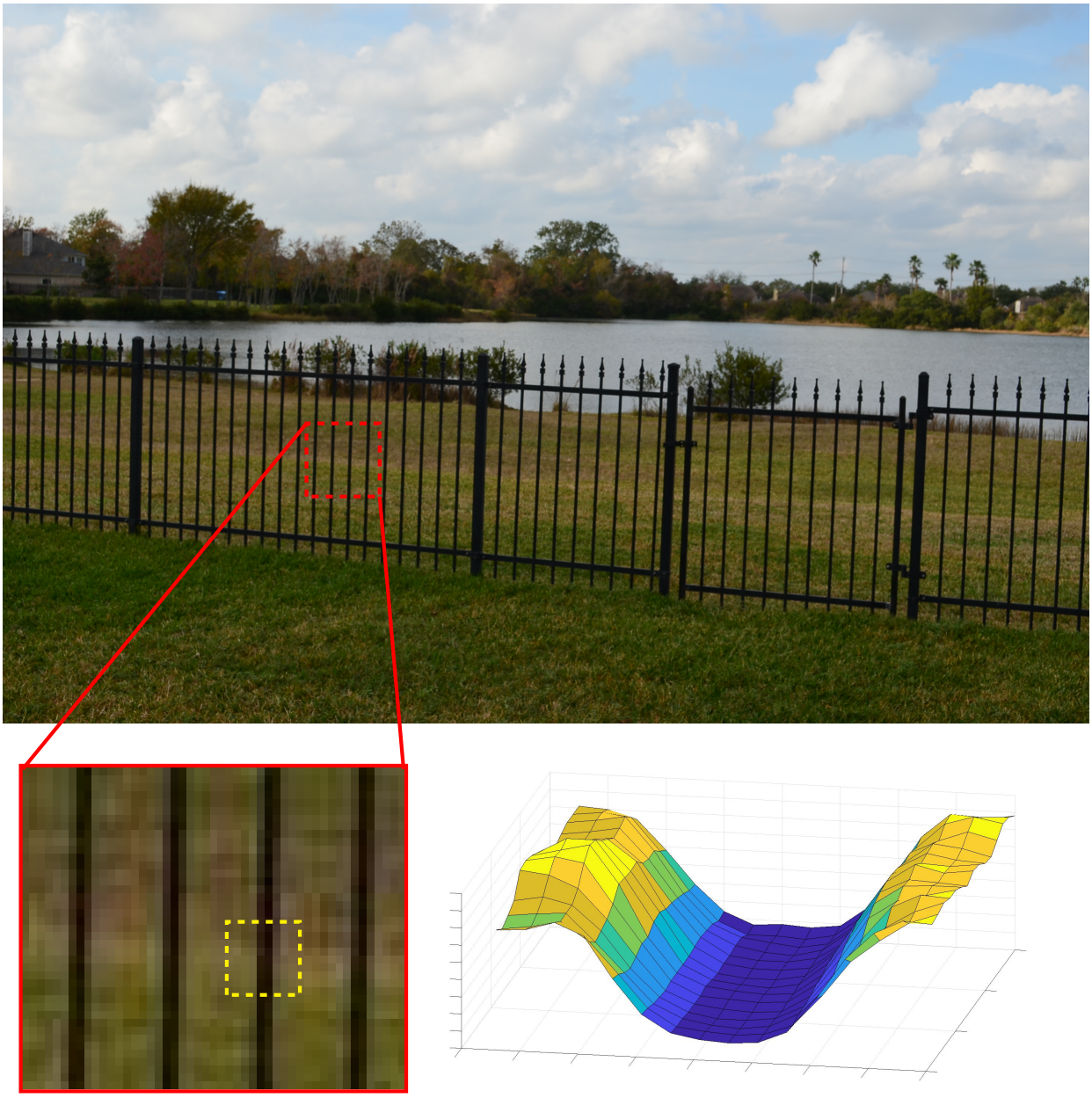

Figure 3.1: Example of streaks in an image. Bottom left image is area of original showing several streaks, and the bottom right shows the pixel intensity level for a selected patch. Note the gradual dip in pixel values where the fence overlays the grass background.

signal f [54], [73] yields the moment

$$\iint P_{nm}(x,y)f(x,y)\,dx\,dy \tag{3.1}$$

3.1 Zernike polynomials

One choice of basis functions is the Zernike polynomials. Zernike polynomials originally addressed spherical aberrations in optical lenses, and represent a complete orthogonal set of polynomials; orthogonality here implies that every element in the polynomial set is not a linear combination of the other elements and thus contains unique information. This orthongonality, however, only holds within a unit circle. This orthogonality eliminates redundant information and the requisite bookkeeping from duplicated information. The Zernike polynomial set can be written either in polar or Cartesian coordinates, with the latter being more common [44] and given as

$$P_{nm}(\rho,\theta) = R_{nm}(\rho)\ \exp\left(jm\theta\right) \tag{3.2}$$

where $j=\sqrt{-1}$, θ describes the streak's orientation, and

$$\rho^2 = x^2 + y^2 \tag{3.3}$$

$$R_{nm}(\rho) = \sum_{s=0}^{(n-|m|)/2} \frac{(-1)^s (n-s)!\,\rho^{n-2s}}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!} \tag{3.4}$$

where n and m—respectively denoting the order [42] and repetition (or angular dependence) [73]—can assume any values that satisfy

$$n \geq |m| \geq 0, \tag{3.5}$$

$$n-m \text{ even}. \tag{3.6}$$

We relate the 1D polynomials R_{nm} to their 2D counterparts $P_{nm}(x,y)$ for a sample set of combinations of n and m:

$$R_{00}(\rho) = 1 \quad \Rightarrow \quad P_{00}(x,y) = 1 \tag{3.7}$$

$$R_{11}(\rho) = \rho \quad \Rightarrow \quad P_{11}(x,y) = x + jy \tag{3.8}$$

$$R_{20}(\rho) = 2\rho^2 - 1 \Rightarrow \quad P_{20}(x,y) = 2x^2 + 2y^2 - 1 \tag{3.9}$$

$$R_{22}(\rho) = \rho^2 \quad \Rightarrow \quad P_{22}(x,y) = x^2 + y^2 \tag{3.10}$$

It is simple to understand that R_{00} represents the signal's mean, but visualizing the others might be more useful. Figure 3.2 displays the set of polynomials for the Eqns. 3.7–3.10, using open source Python code [74].

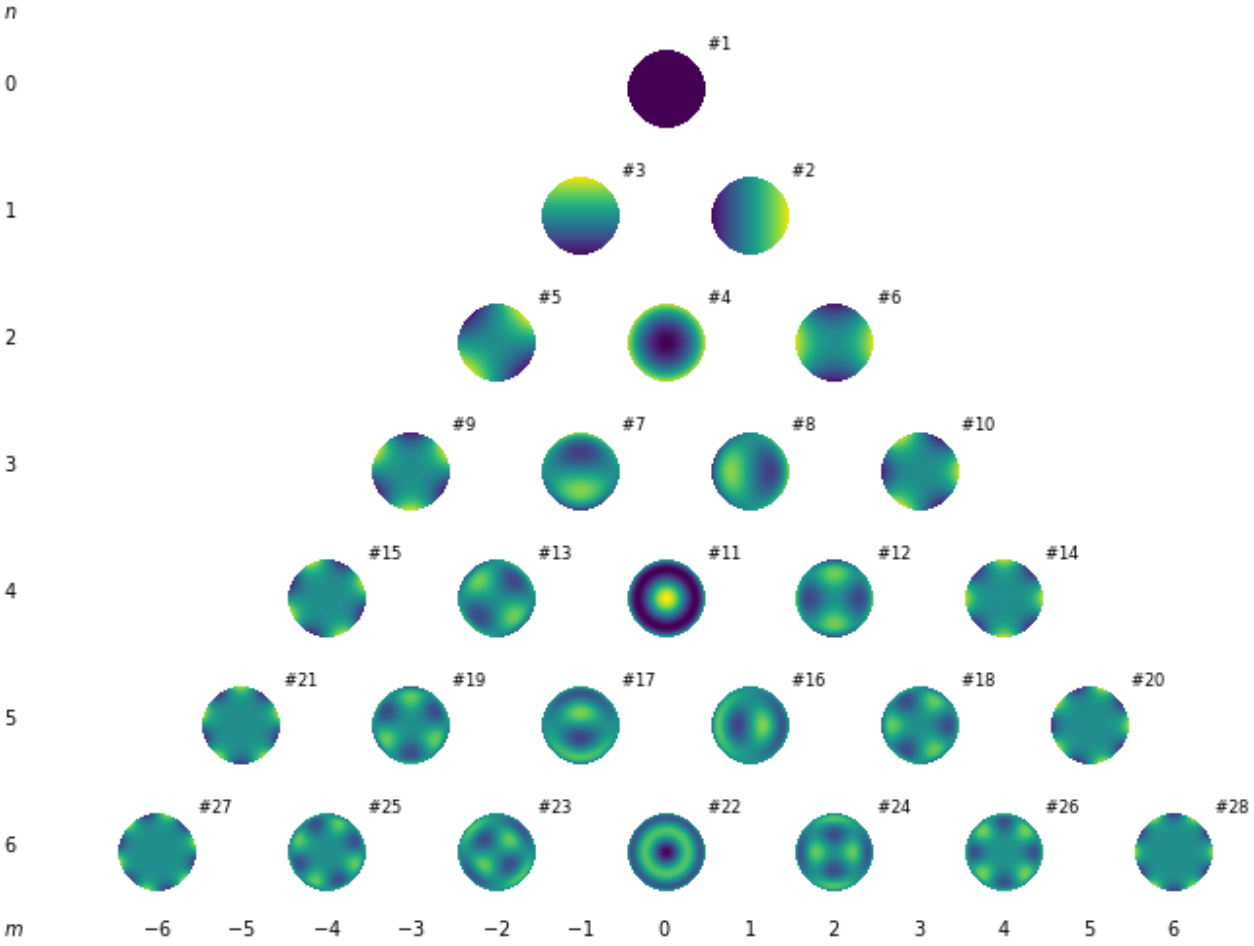

Figure 3.2: Selected set of Zernike polynomials in 2D. The row of the triangle represents the order n, and the column is the repetition or angular dependence (m). The ordering shown is one particular way to uniquely number each moment. This figure was produced by the author using Jacopo Antonello's Zernike polynomial function.

Since orthogonality for Zernike moments only exists within the unit circle, the moments can form from the corresponding polynomials only with a restriction of integration bounds and a scaling coefficient, given as

$$Z_{nm} = \frac{1}{Q_{nm}} \iint\limits_{x^2+y^2\leq 1} P_{nm}(x,y)\ f(x,y)\ \mathrm{d}x\ \mathrm{d}y \tag{3.11}$$

where

$$Q_{nm} = \frac{\pi}{n+1} \tag{3.12}$$

Scaling with the normalization coefficient is often not required, so we relate

$$A_{nm} = \iint\limits_{x^2+y^2\leq 1} P_{nm}(x,y)\ f(x,y)\ \mathrm{d}x\ \mathrm{d}y, \tag{3.13}$$

The relationship between Z_{nm} and A_{nm} therefore is simply the scalar factor Q_{nm}

$$A_{nm} = Q_{nm} Z_{nm} \tag{3.14}$$

Of particular interest to streak localization are two moments—Z_{11} and Z_{20}. It is also important to note that applying these moments to 2D signals like images produces complex-valued results. This forms the basis for determining incidence angle of the streak.

3.1.1 Rotational properties of Zernike moments

The value of m represents the angular dependence or the repetition of the moment. A value of $m = 0$ for any given moment means that there is no angular dependence, or that the moment is rotationally invariant. That is, the incidence angle of the signal (here, a streak) results in the same resultant moment. Any other value of m, conversely, depends also on incidence angle.

Any change in orientation of ψ of the underlying signal is related to the original signal

by

$$A'_{nm} = A_{nm} \exp(-jm\psi) \tag{3.15}$$

We exploit this angular dependence of Z_{11} (Z_{20} is rotationally invariant) to determine the incidence angle.

3.2 Discretized Zernike moments for digital images

A digital image $I(u, v)$ consists of an array of numbers, with each element indicating a pixel intensity value. Digital photographs are discretized representations of continuous signals $f(x, y)$. Computer vision and image processing convention uses integer values of u and v to represent the centers of pixels, with the origin $(0, 0)$ at the top left of the image, with the u-axis pointing right and the down direction as the v-axis. We can transform the signal in x,y coordinates into the pixel space by the transform given in Section 2.1.2.

After the transformation, we can replace the x,y in Eq. 3.13 with u,v with no loss in generality. We must now discretize the moment to apply it to the discretized image I. We divide the moment into a patch of odd size N_p

$$A_{nm}(\tilde{u}_i, \tilde{v}_i) \approx \sum_{k=-p}^{p} \sum_{s=-p}^{p} M_{nm}(p+s, p+k) I(\tilde{u}_i + s, \tilde{v}_i + k) \tag{3.16}$$

where $p = (N_p - 1)/2$ is a non-negative integer, and M_{nm} is a mask of size $N_p \times N_p$ corresponding to a discretized version of P_{nm}. Note that 3.16 is simply an image correlation operation

$$A_{nm} = M_{nm} * I \tag{3.17}$$

where $*$ is the 2D correlation operator. Applying Eq. 3.17 to moments Z_{11} and Z_{20}, we note an important distinction—Z_{20} is always real-valued, where Z_{11} is complex-valued

$$A_{20} = Q_{nm} Z_{20} = M_{20} * I \tag{3.18}$$

$$M_{11} = \mathrm{Re}\,[M_{11}] + j\,\mathrm{Im}\,[M_{11}] \tag{3.19}$$

Fortunately, the relationship of the real and imaginary parts of M_{11} permits discarding the imaginary component in practice since [42]

$$\mathrm{Im}\left[M_{11}\right] = \mathrm{Re}\left[M_{11}\right]^T \tag{3.20}$$

Computing the three required moments for any given image patch provides the information needed to compute the localization step.

3.2.1 Localization update

Given a set of pixel-level streak locations and the three moments for each point's neighborhood, this section demonstrates how to update or refine the pixel-level location. This step is true for not just streaks but also for edges using the same approach (with different values and approximation models) [42].

Let us use $\sim$ notation to represent a pixel-level boundary location and the caret notation to indicate the sub-pixel update, given in local coordinates relative to $(\tilde{u}, \tilde{v})$

$$\begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} = \ell \begin{bmatrix} \cos\psi \\ \sin\psi \end{bmatrix} \tag{3.21}$$

where ℓ and ψ are the magnitude and angle of the update, respectively. Details for computing ψ are in Section 3.2.2, and for ℓ in Section 3.2.3.

Rearranging Eq. 3.22 yields the complete update step in image space

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} + \frac{N_p\,\ell}{2} \begin{bmatrix} \cos\psi \\ \sin\psi \end{bmatrix} \tag{3.22}$$

3.2.2 Computing incidence angle

Moment A_{11} provides all of the information required to compute the incidence angle. Figure 3.3 shows the meaning of the moment calculated for A'_{nm}

where w is half the width of the streak, which is the sum of contributions from the actual width of the streak and defocus and other distortions.

Note that v' aligns with the direction of the streak and u' aligns with the vector normal to the incident streak. Using such a frame permits the imaginary component of the moment

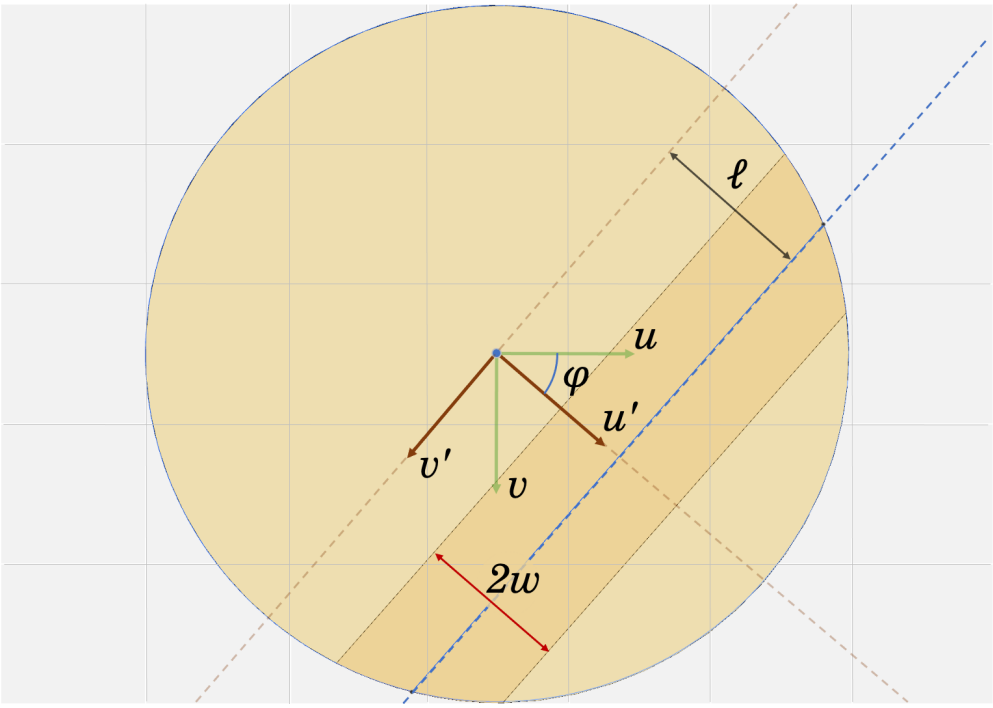

Figure 3.3: Illustration of the original frame and the prime frame, which is oriented such that the prime axes align with the direction of the streak and its normal.

A_{11} to be zero. Recall that since

$$\exp(-jm\psi) = \cos m\psi - j \sin m\psi \tag{3.23}$$

then Eq. 3.15 becomes

$$\mathrm{Im}[A'_{11}] = \mathrm{Im}[A_{11}]\cos(\psi) - \mathrm{Re}[A_{11}]\sin(\psi) = 0 \tag{3.24}$$

Rearranging Eq. 3.24 reveals the relationship between the real and imaginary components and the angle of incidence

$$\mathrm{Im}[A_{11}]\cos(\psi) = \mathrm{Re}[A_{11}]\sin(\psi) \tag{3.25}$$

The angle of incidence, therefore, is

$$\psi = \mathrm{atan2}\left(\mathrm{Im}[A_{11}],\, \mathrm{Re}[A_{11}]\right) \tag{3.26}$$

demonstrating that the angle is solely a relationship found within A_{11} between the real and

imaginary components.

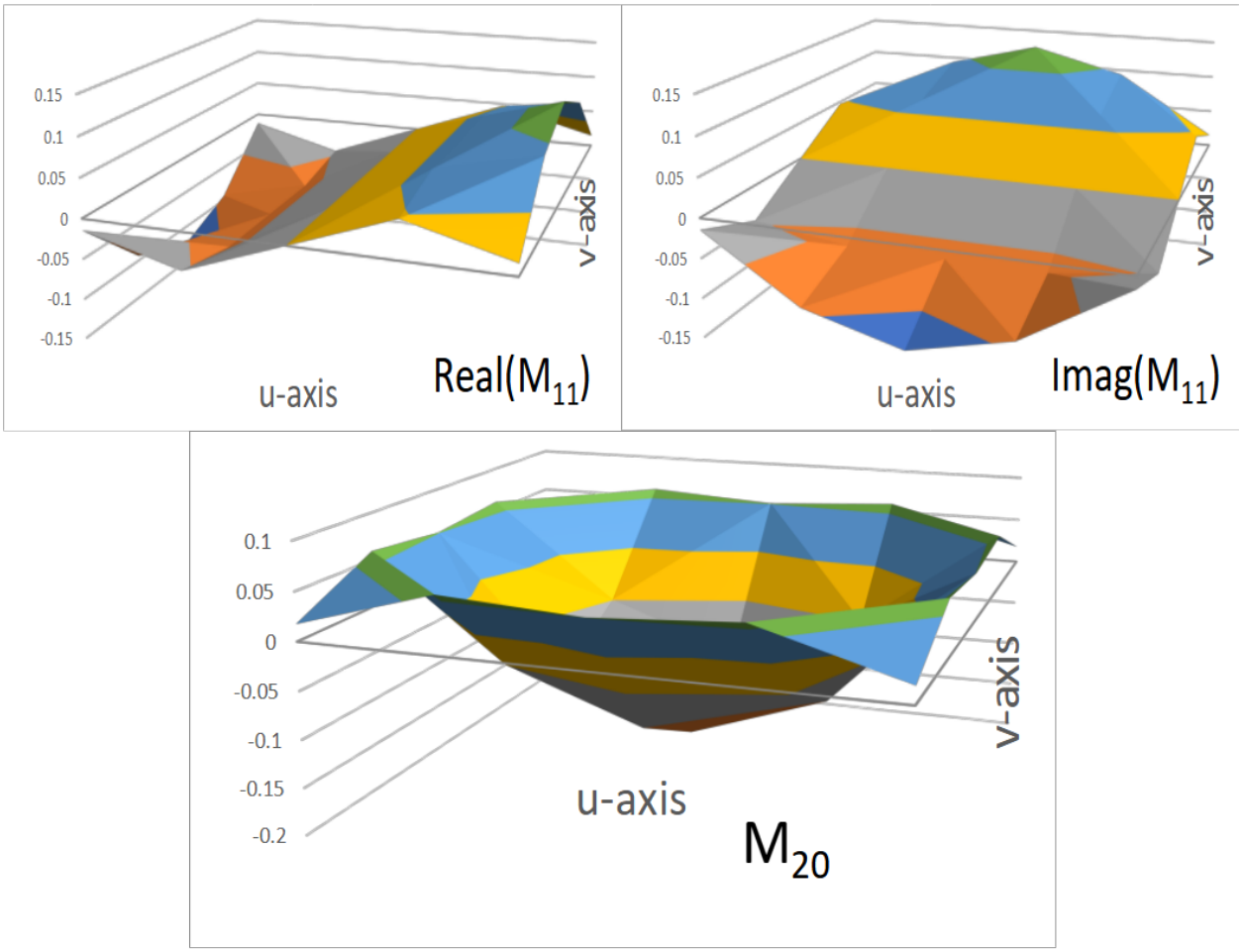

Figure 3.4: Moment masks for M_{11} (real and imaginary parts) and M_{20} (bottom). These are the masks to apply to the region centered on each pixel-level guess. It is the ratio of the imaginary to real that computes the angle and the ratio of the magnitude of M_{11} to M_{20} that gives the offset ℓ according to the equations in the next section. The vertical axis represents the magnitude, and the two horizontal axes (u and v), represent the local pixel coordinates.

3.2.3 Computing ℓ

After computing the incidence angle, solving for ℓ—the magnitude of the update—is the last necessary value. Doing so requires some further description of what streaks are. Being subject to distortions, aberrations, and camera PSF, the true representation of a streak—in contrast to an ideal streak—is a gradient and not a true discontinuity. The original approach

to Zernike moments for calculating edge refinements [43] represented the boundary as a step function: a sharp jump. This model produced biased results with real images.

Gopala and Jain [75] propose that the PSF is well-modeled as a 2D Gaussian function. Therefore, we can model the blurred image as a convolution of an ideal image (with no blur or distortions) and a Gaussian kernel K_G

$$I_{blur} = K_G * I_{ideal} \tag{3.27}$$

Taking a slice of the profile perpendicular to the intensity change produces the *edge spread function*, which takes the shape of a sigmoid function. Using the sigmoid function produces substantial complexity when integrating the Zernike moments, so [42] proposed approximating the sigmoid function as a ramp function. Conversely, Christian and the current author [54] model the ideal *line spread function* as a natural extension of the ESF to represent streaks, implementing instead a wedge function to mimic the profile of a thin strand.

The wedge model requires that we know the relationship of the wedge width and the associated Gaussian blur kernel used to blur the image

$$w \approx k_{streak}\sigma \tag{3.28}$$

where σ is the kernel blur width, and k_{streak} is the parameter of interest.

Quantitatively, we [54] determined that a value of $k_{streak} = 0.90$ produced the best results for a wide range of synthetic and real images with varying levels of noise and blur. Note that basic image processing techniques can determine w, the result of the blurring operation, for a given image.

Figure 3.5 shows the model of the unit disk with an incident streak at a distance ℓ from the center, and Figure 3.6 shows the constituent parts of the model.

For this formulation to work on a patch of size $N_p \times N_p$, we must scale down the image patch by a factor of $N_p/2$ to fit in the unit disk. Eq. 3.29 shows the different integration regions over which we calculate the volume represented by each of the sections denoted in Figure 3.6

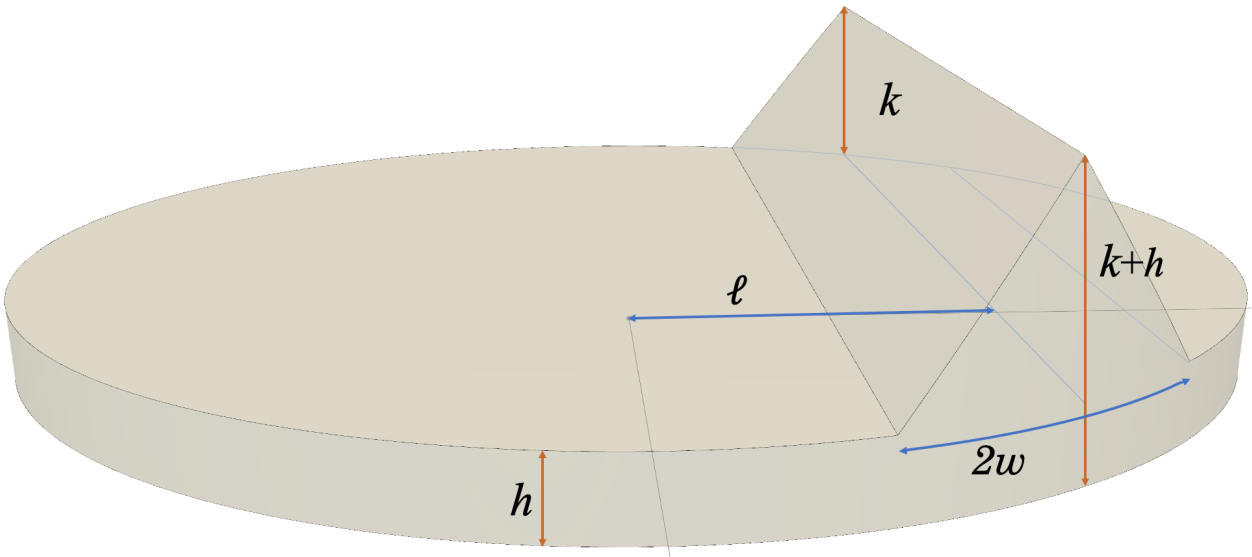

Figure 3.5: Wedge model of streak.

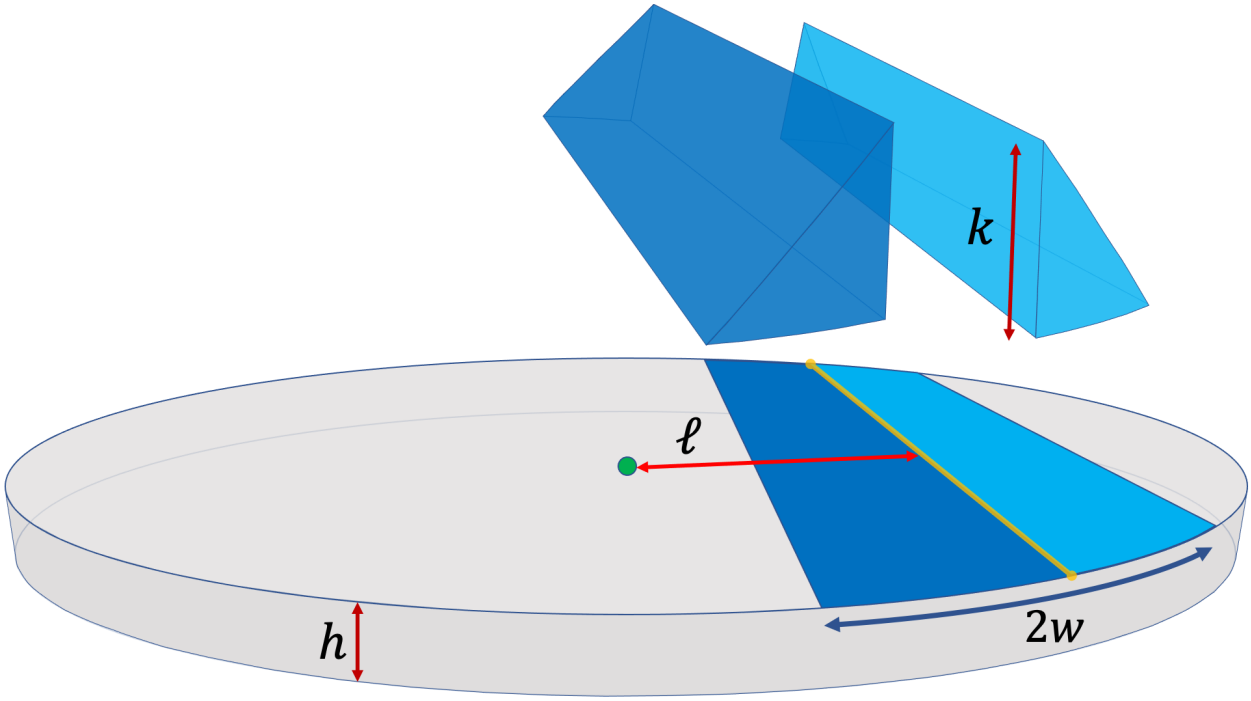

Figure 3.6: Components of wedge model. The gray section is the uniform background, dark blue the ascending ramp up to the maximum $k + h$ over a distance w, and light blue the descending ramp.

$$I_{streak}(\bar{u}', \bar{v}') = \begin{cases} h & \bar{u}' \leq \ell - w \\ h + k\frac{\bar{u}' - (\ell - w)}{w} & \ell - w < \bar{u}' \leq \ell \\ h + k - k\frac{\bar{u}' - \ell}{w} & \ell < \bar{u}' \leq \ell + w \\ h & \ell + w < \bar{u}' \end{cases} \tag{3.29}$$

We can analytically solve the double integral from Eq. 3.13 in the aligned or prime frame. Integrating each region yields the following analytical values for A'_{11} (Eq. 3.30)

$$\begin{aligned} A'_{11} = \frac{k}{12w}\Big[&6\arcsin\ell - 3\arcsin\ell_1 - 3\arcsin\ell_2 \\ &+ 2C\ell(5 - 2\ell^2) - B_1\ell_1(5 - 2\ell_1^2) - B_2\ell_2(5 - 2\ell_2^2)\Big] \end{aligned} \tag{3.30}$$

and A_{20} (Eq. 3.31)

$$A'_{20} = \frac{2k}{15w}\Big[B_1^5 + B_2^5 - 2C^5\Big], \tag{3.31}$$

where

$$\ell_1 = \ell - w \quad \text{and} \quad \ell_2 = \ell + w \tag{3.32}$$

$$B_1 = \sqrt{1 - \ell_1^2} \quad \text{and} \quad B_2 = \sqrt{1 - \ell_2^2} \tag{3.33}$$

$$C = \sqrt{1 - \ell^2} \tag{3.34}$$

The expressions for Eqs. 3.30 and 3.31 both have a leading term with the value k, representing the intensity, which taking the ratio of the two eliminates this dependency. This means that the magnitude of the update step is independent of the intensity of the streak.

Lyvers [43] found that it was the ratio of these two moments that produced the size of the update step, that is,

$$Q_s = \frac{A_{20}}{A'_{11}} \tag{3.35}$$

3.2.4 Equation parameter fitting

We are left with solving the problem with one unknown: ℓ. Despite many attempts, solving this equation analytically for ℓ failed to produce results for arbitrary values of w and ℓ; however, an approximation for this relationship provides highly accurate results.

The assumption for streaks is that the width of the streak is thin (i.e., $w < N_p$), so we explore the value of this ratio as w tends toward zero, providing a relationship for which solving for ℓ is simple,

$$Q_{S_0} = \lim_{w \to 0} Q_S = \frac{4\ell_{S_0}^2 - 1}{3\ell_{S_0}} \tag{3.36}$$

which, solving for ℓ_{S_0}, produces

$$\hat{\ell}_{S_0} = \frac{3}{8} Q_{S_0} \pm \sqrt{\left(\frac{3}{8} Q_{S_0}\right)^2 + \frac{1}{4}} \tag{3.37}$$

Choosing the correct quadratic root is crucial, so the following logic applies:

$$\lim_{\ell_{S_0} \to 0^+} Q_{S_0} = \lim_{\ell_{S_0} \to 0^+} \frac{4\ell_{S_0}^2 - 1}{3\ell_{S_0}} = -\infty \tag{3.38}$$

where we know to choose the right-hand limit since $\ell_S \geq 0$ by construction.

Observe the behavior of Q_{S_0} as $\hat{\ell}$ approaches zero, which tends to negative infinity. Thus we seek the root that is approximately zero when Q_{S_0} is a large negative number, which only happens when the plus sign is chosen in Eq. 3.37. Thus

$$\hat{\ell}_{S_0} = \frac{3}{8} Q_{S_0} + \sqrt{\left(\frac{3}{8} Q_{S_0}\right)^2 + \frac{1}{4}} \tag{3.39}$$

While this solves the problem when the width tends to zero, we still need to solve for arbitrary w. We can generalize this result to $w > 0$, which does not appear to permit an exact analytic solution.

Writing a parameterized expression for $\hat{\ell}_S$, as seen in Eq. 3.40, we sought the combinations of the moment ratio and the streak width most closely match that of the behavior of the analytic ratio of Eq. 3.30 and Eq. 3.31. Furthermore, we must ensure that the equation simplify to Eq. 3.39 when $w = 0$. We then performed a nonlinear polynomial fit that gave us the most robust fit for varying levels of noise and blur. Modeling this expression with additional parameters, we considered the form

$$\hat{\ell}_S \approx \sqrt{a_1^2 Q_S^2 + a_2 Q_S w + a_3 w^2 + a_4 w + a_5} + a_6 Q_S + a_7 w + a_8 \tag{3.40}$$

Extensive analysis of this form showed that the terms a_1, a_3, a_5, and a_6 dominated the estimation of ℓ, while changes in a_2, a_4, a_7, and a_8 produced little deviation. Furthermore, the terms a_1 and a_6 consistently converged to the same value, despite different initial conditions.

Eliminating the terms with little effect and equating a_1 and a_6, we form the following 3-parameter Eq. 3.41 to find the final fit coefficients

$$\hat{\ell}_S \approx \sqrt{a_1^2 Q_S^2 + a_3 w^2 + a_5} + a_1 Q_S. \tag{3.41}$$

The least squares solver determined that the values of a_1 and a_5 exactly aligned with the corresponding analytically derived coefficients for ℓ_{S_0} in Eq. 3.39

$$a_1 = 3/8, \quad a_5 = 1/4 \tag{3.42}$$

and empirically determined the value for a_3 as

$$a_3 \approx -1/10 \tag{3.43}$$

The parametric expression for the update for a given w as

$$\hat{\ell}_S = \frac{3}{8}Q_S + \sqrt{\left(\frac{3}{8}Q_S\right)^2 - \frac{1}{10}w^2 + \frac{1}{4}} \tag{3.44}$$

There is one final caveat about the update: if the update is too large (i.e., update puts point near the unit circle boundary) then we have little confidence in a step that large. Similarly, if the ratio of A_{20} to A_{11} is a very large negative value (e.g., -10^6), we know that the update size is very close to zero. Now with both the incidence angle and the update distance computed and the moments to determine them in hand, we can now apply this finding to images with streaks, both real and synthetic.

3.2.5 Steps to refine streak locations

As a quick summary, here are the basic steps to apply to an image to find and localize streaks in an image.

1. Find pixel level streak locations with standard method (e.g., Canny, Sobel)
2. For each pixel-level point:
   (a) Apply the three masks $Re[M_{11}]$, $Im[M_{11}]$, M_{20}
   (b) Compute incidence angle ψ (Eq. 3.26)
   (c) Compute the update distance ℓ (Eq. 3.44)
   (d) Ensure that update is not too small or too large ($0.001 < \ell < 2$ pixels)
3. Perform update according to Eq. 3.22

3.2.6 Limitations

While this method is ambivalent to streak curvature, it does have limitations. For example, the mathematical definition of the streak demands that the discontinuity that

describes the boundary must be only in one direction. That is, it is not well-suited for corners or isolated points. A method to facilitate this limitation using this same approach is possible but not entertained in this work.

Furthermore, this approach is not suited for direct application to multi-channel images, such as color images and images with alpha parameters. This, however, can be approached in several ways, both well developed methods in image processing. The first is converting the image to a grayscale image and applying the algorithm, which is generally the accepted method. Or applying this algorithm to each channel individually is possible, but could introduce artifacts due to color pixel detector locations.

Only the local neighborhood around a pixel-level estimate must behave similar to a streak, which is generally a good assumption except for areas near corners. This itself could be an area of investigation to determine the subpixel locations of corners using a method similar to this.

3.3 Numerical validation of estimation functions on synthetic data

To test the quality and consistency of this method, I applied the algorithm to synthetic images where the actual location and orientation of streaks were known by computing the location and orientation of a synthetic streak (either curved or straight) through a uniform background to obtain the truth locations and orientation of the streak.

Further pushing the algorithms, I introduced varying levels of noise and blur, simulating defocus, quantization, and noise. I swept out a range of blur levels by convolving zero-mean Gaussian kernels with the perfect image which affected performance and showed how to compute the width of the streak. Additionally, I introduced synthetic shot noise (as a Poisson process) and white Gaussian noise to simulate real sources of noise. Adding synthetic noise allowed me to extract the signal-to-noise ratio to verify performance as SNR changed.

Furthermore, knowing the SNR and blur level precisely permits comparison between this method and any number of other methods for streak localization. Section 3.3.1 contains the analysis of streak localization using Zernike moments against the alternative state-of-the-art methods.

Monte Carlo runs demonstrated this method's accuracy, shown in Figure 3.7. As the values for noise and blur increased, performance dipped predictably. Even where noise levels approached those of the actual image information, the RMS error stayed below a tenth of a

pixel.

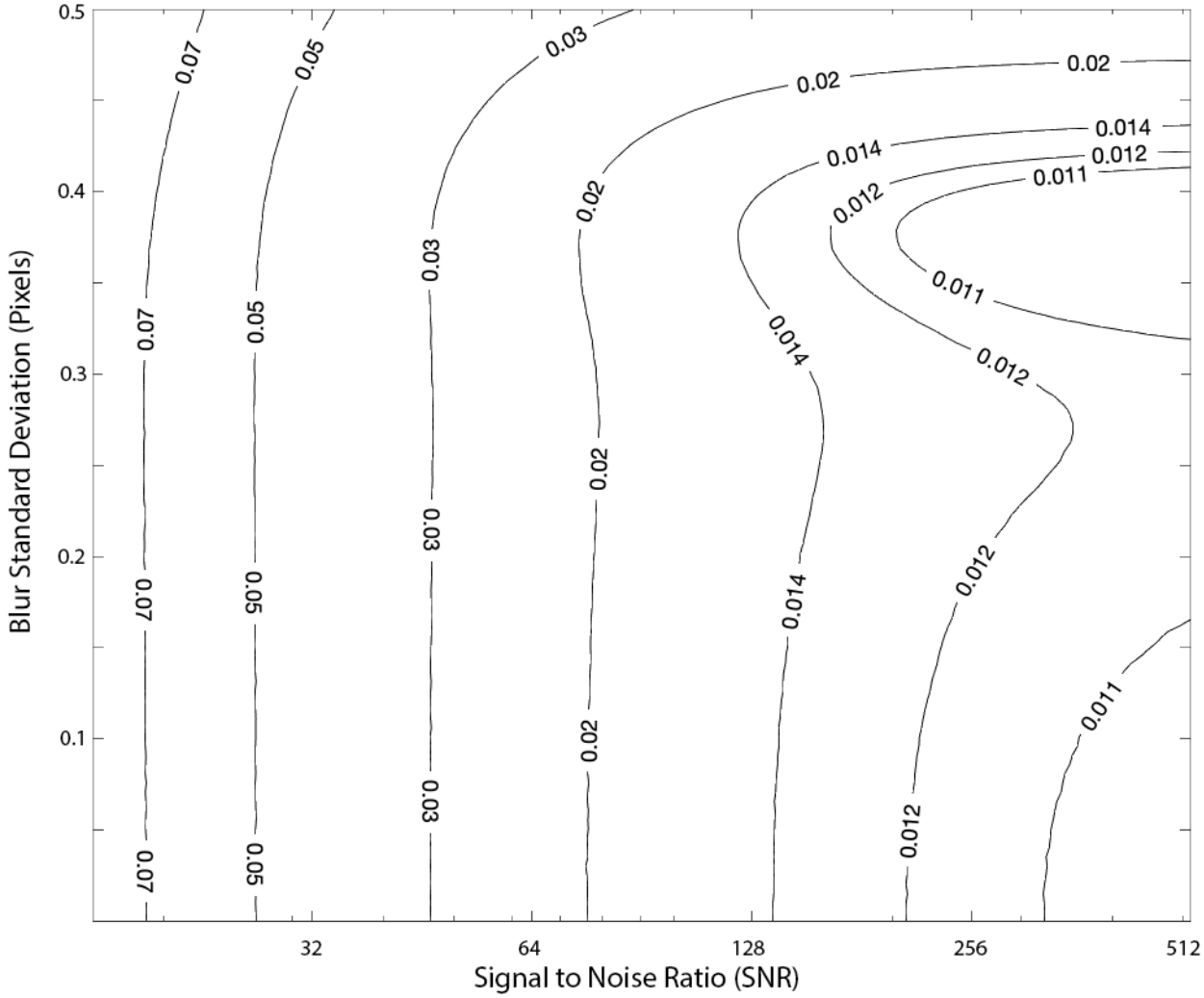

Figure 3.7: Assessment of streak localization RMS error using Zernike moments over a range of blur sizes and SNR. Note that the error never reaches 0.1 pixels, even in extreme cases where the noise approaches parity with signal level (i.e., high noise, low SNR).

3.3.1 Comparing alternative methods

The image processing algorithm proposed here offers fast results using analytic methods. We now turn our attention to alternatives within the same domain; that is, streak (line) localization methods. Of the various alternatives available, most deal explicitly only with straight lines or specific curves (e.g., circles). While streaks are vital primitives in machine vision, methods for edges (arguably a similar primitive) vastly outnumber streak techniques.

While this method works equally well for any type of streak, I had to settle for a comparison between algorithms dealing with straight lines (the Hough transform) and normalized sums of gradients [76]. The Hough transform uses a parameter space into which it converts shapes in 2D into a parameter space (of which the polar form currently appears to be the most common), and originally derives from work of Hough and his analysis of bubble chambers used in physics [77] for which he also owns the patent [78]. Later works generalized this into a more robust framework some ten years later [79]. While the Hough transform is a mainstay of image processing, it requires that the user know what shapes they want, be it a straight line, a circle, or an ellipse. Each of these has a distinct signature in the transform space [80], but we don't get the simplicity and general application that the Zernike method introduces. We don't need to know something about a streak (its shape or contour): we can just compute the two moments for the local region. This is why the comparison is much more appropriate against the normalized sums of gradients (NSG) method.

Simply put, the normalized sums of gradients approach uses the ratio of the sums of the gradients and the gradient lengths. As before, Seo notes the lack of research in regards to streaks in stark contrast to edges, which tend to dominate localization research studies. But this can signify a field rich in exploration potential, as this chapter notes. Most of the recent research outside Hough transforms on streaks discuss machine learning methods [81]–[83] and not analytical techniques like that of Steger [84] and Seo [76].

Seo's work is unique in the combination of not only subpixel localization but also "line linking" using the subpixel estimates, something that has dogged pixel-level estimates. We are not particularly concerned with such linking, which could also be performed post-localization here, but this can further improve the localization.

Figure 3.9 shows one sample of the Monte Carlo comparison run to assess performance and accuracy. Zernike moments performed all calculations anywhere from 0.9 milliseconds to 1.3 milliseconds, and the alternative method tallied in the range of 5 milliseconds to 10 milliseconds. Note that I did not implement any optimizations for this alternative method, but the same is true of the Zernike method; additionally, I only implemented NSG using the 3×3 ($k = 1$) window Seo uses, so this might have some effect on accuracy, but little changed when this increased to a 5×5 ($k = 2$) window, except for a slight increase in accuracy for NSG and a slight performance hit. Despite this, Zernike moments edge out NSG in both performance and in accuracy.

3.4 Validation on real images

While synthetic images were useful for determining performance and assessing accuracy against other methods, results on real images are the hallmark of successful image processing techniques. After satisfying benchmarks on synthetic images, we move now into using real digital imagery.

Being real images, there is no "truth" data available, and any information about objects in the image come from other analysis tools; however, we can still visually inspect the results for performance, meaning that the analysis becomes more qualitative and less quantitative.

Before continuing, it is imperative to understand that since this method relies on a rough estimate from another algorithm (e.g., medial-axis (bushfire) [41]), any points that this first layer misses will be absent in the final output. This method uses the first pass pixel coordinates and applies an update step to those coordinates. These first level estimates can be obtained through manual or automatic processes, but the refinement step is fully automatic.

Included in this section are several examples of real images processed through this algorithm. With examples from diverse fields I intend to show that this approach is application-agnostic, meaning that potential uses extend from space imaging to medical imaging [85], [86], from precision robotics to agriculture and land management. Disciplines like non-destructive evaluation and testing (where phenomenon like cracks and fissures within a material are of utmost importance) and GIS likewise benefit from more accurate streak localization. Other disciplines such as autonomous vehicles that rely on thin demarcations between regions (such as lanes) could benefit from a lightweight, analytic method to quickly discern safe and unsafe travel regimes.

Each sample image has three sub-images, with progressively more feature magnification. The left shows the original image, the middle the set of sub-pixel refinements, and the right the set of refinements vis-à-vis their original coarse locations. Contrast the smoother and more natural flow of the green refined points to the more jagged set of red $\times$'s.

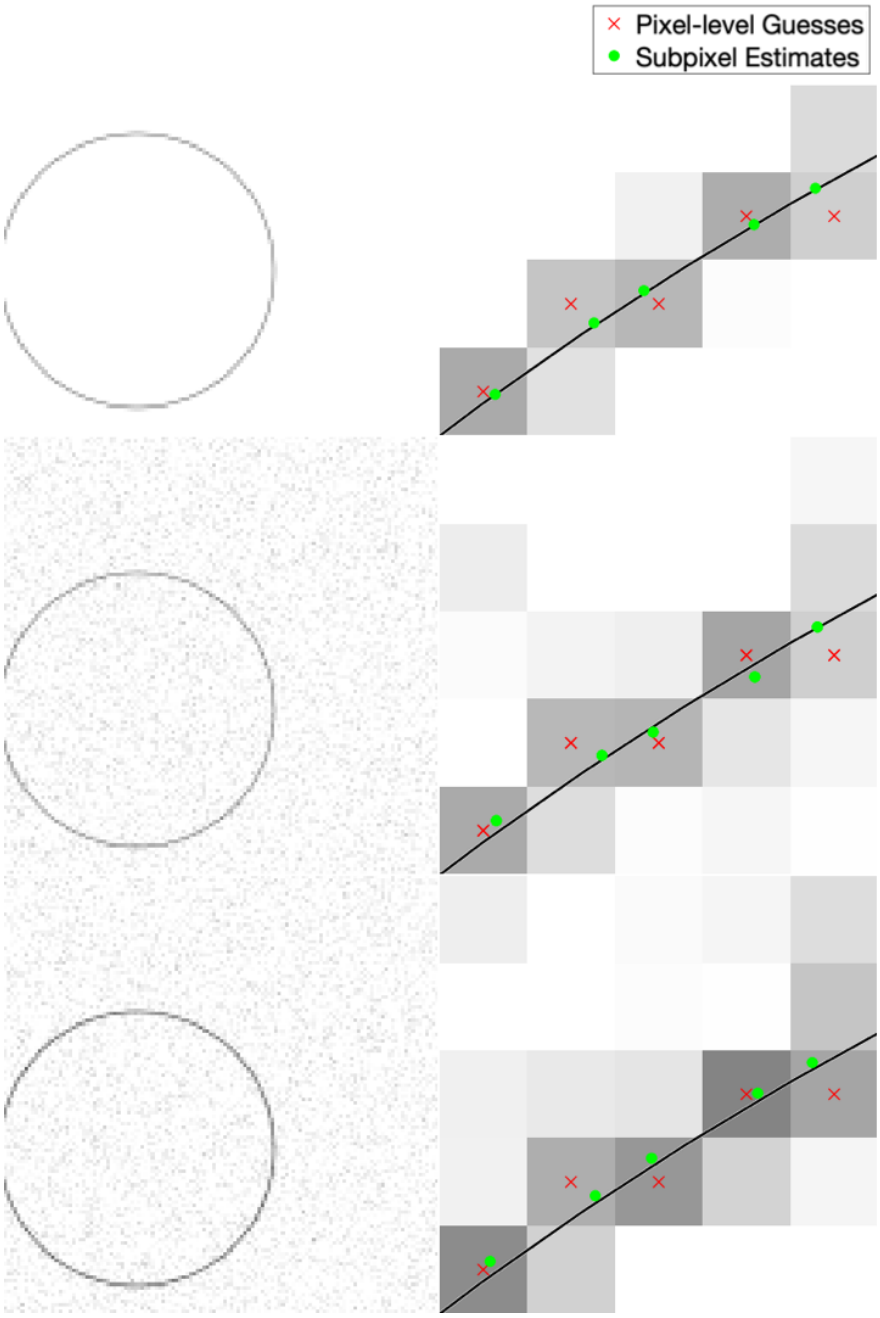

Figure 3.8: Example of a curved synthetic streak with varying levels of noise, with decreasing SNR as we move down. The red $\times$'s denote the pixel-level guesses, and the green circles the refined streak locations.

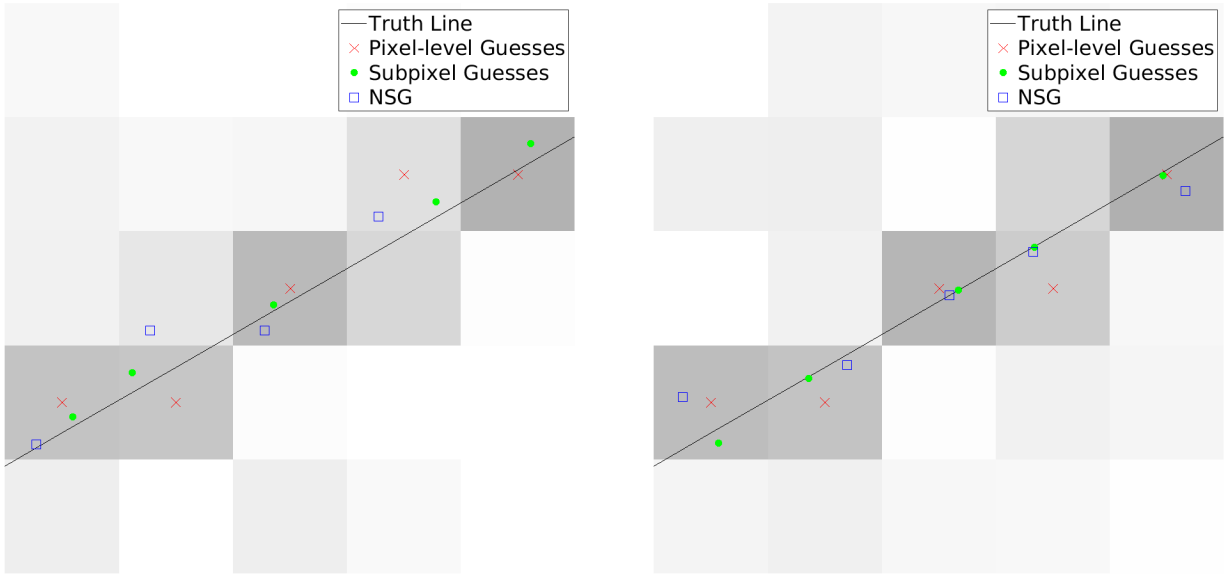

Figure 3.9: Illustration of a two comparison runs between the normalized sum of gradients approach (in blue) and the Zernike moments discussed here (in green). The left compares the NSG approach with a 3×3 window and the right with a 5×5 window. Both methods used the same starting pixel values for coarse estimates. The RMS error for Zernike moments was around 0.11 pixels and for NSG it was slightly less than 0.29 and 0.23, respectively.

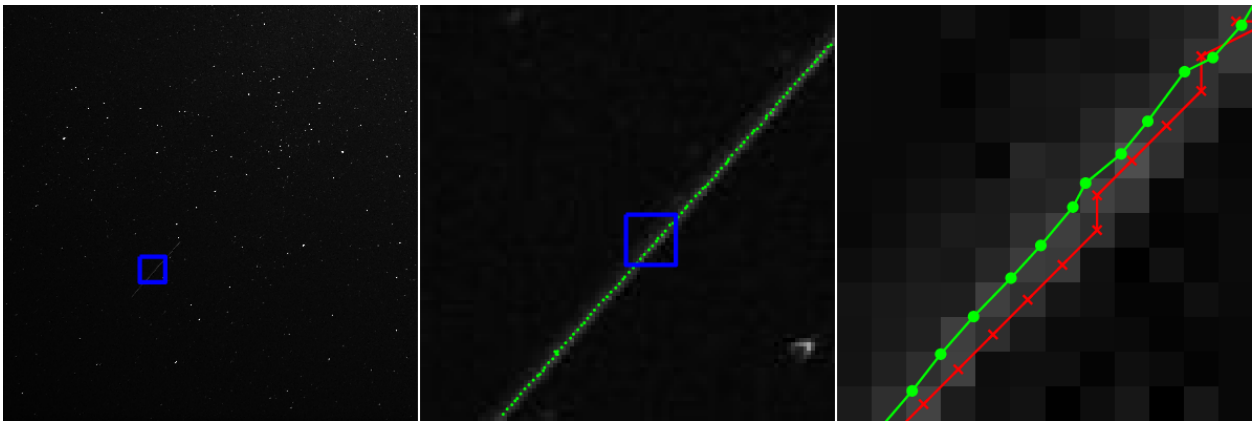

Figure 3.10: Inertially pointed star field image captured with with the Omnidirectional Space Situational Awareness (OmniSSA) system. This example image has a 10 second exposure time and contains a satellite that appears as a streak within the image. The original OmniSSA image is courtesy of Dr. Marcus Holzinger of University of Colorado Boulder.

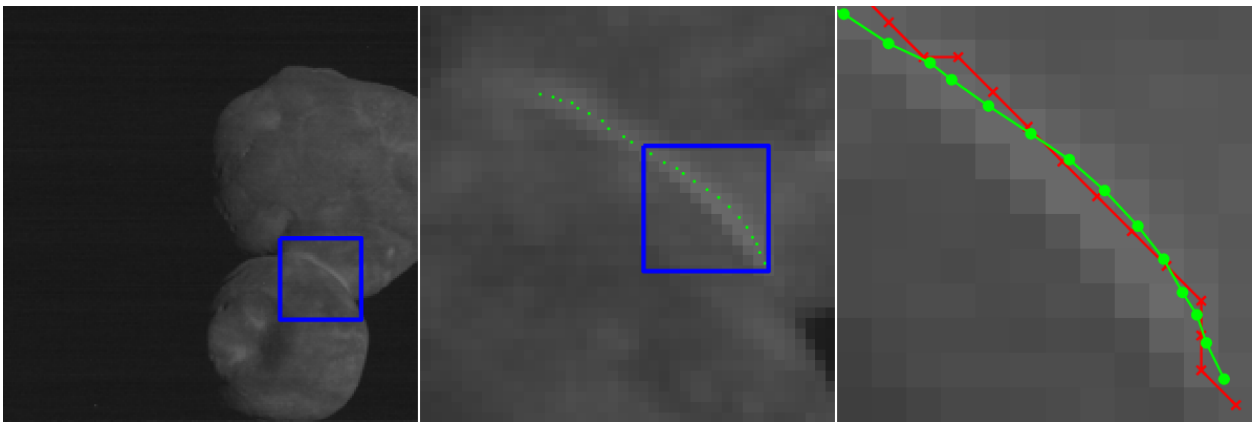

Figure 3.11: Image of Kuiper belt object Arrokoth (formerly called Ultima Thule) collected by the New Horizon spacecraft's Long Range Reconnaissance Imager (LORRI) during a flyby in early 2019 (credit for raw image: NASA/Johns Hopkins University Applied Physics Laboratory/Southwest Research Institute).

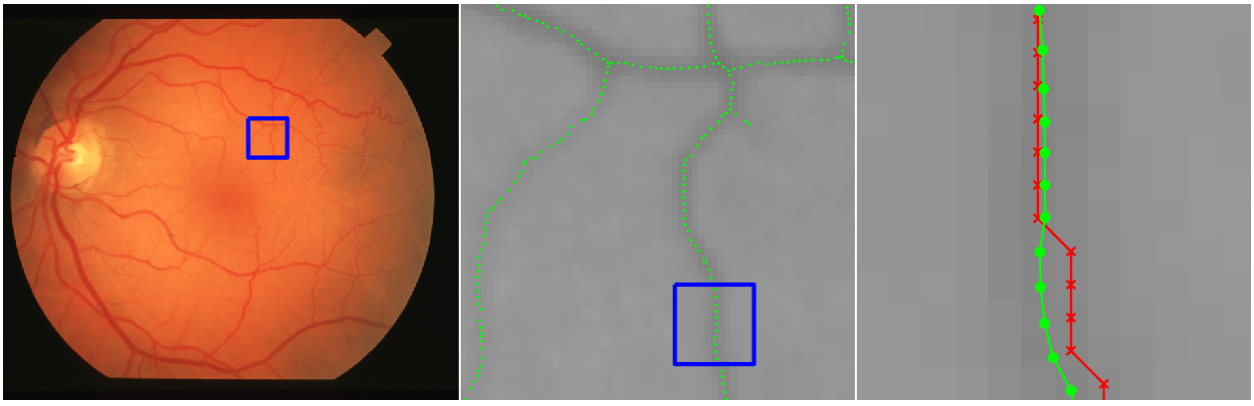

Figure 3.12: Image of a retinal scan for a healthy eye, where we seek to localize blood vessels. The original image is im00032 from the STARE database.

CHAPTER 4
INDIRECT P^2-INVARIANT INDEXING & SEARCH

One of the fundamental challenges of star identification is matching the observed data with that of a large catalog of known star information. The sheer size of the general catalogs of stars or craters can be prohibitively expensive in terms of storage size and query times.

For example, the Hipparcos [87] catalog contains information about over one hundred thousand stars; while this number alone is not overwhelming even for most spacecraft, the real size comes from combinations of stars. Many star identification catalogs contain not just the right ascension, declination, and other information about each star, they contain precomputed values of relationships with other stars, like the inter-star angle or the dihedral angle [21], both of which are geometric invariants under the action of the $SO(3)$ group (i.e., a rotation in 3D). This becomes not a linear system, but a combinatorial system: the roughly 118,000 stars in Hipparcos would become nearly 7 billion combinations just for pairwise matching. Matching a triad of stars in this catalog would require over 250 trillion entries!

Of course, some careful considerations can drastically reduce this number. Unless using omni-directional cameras for star identification purposes, clipping the catalog to only include inter-star angles less than the angular size of the camera field-of-view significantly diminishes the number of combinations, as well as rejecting any stars whose brightness is less than the sensitivity of the camera at a given setting.

The work Christian et al. developed in [20] discusses using invariants for crater identification. That work developed a framework for using invariant theory on crater identification based on the invariants of a triad of both coplanar and non-coplanar craters using precisely the number of independent invariants for each case (seven and three, respectively).

The question arises then about efficient storage and access to crater triad information. Each of the invariants of a triad are related by a permutation. If we can order them in a particular way (e.g., clockwise or counterclockwise), then they are more strictly related by a cyclic permutation, meaning that {a,b,c} should yield the same results as {b,c,a} and {c,a,b}, but not any other ordering. Therefore, blind or naïve placement into a catalog without exploiting this relationship adds needless computations and queries down the pipeline. We shall see how exploiting this relationship will reduce the number of searches required to find a match in the catalog.

4.1 What are P^2-invariants?

Some readers might not be entirely familiar with the term "P^2-invariants". The term refers to geometric relationships that are not only invariant to projective actions (e.g., perspective projection of a camera) but are also invariant to permutations. A previous section explained the cross ratio and its use in camera calibration and object recognition, but the invariance to permutations of the objects becomes important if we want to reduce the number of searches to a database (again, very expensive). We may see the usefulness if we consider the motivating example for crater identification: how do we uniquely identify craters in an image. If we want to use geometric invariants for identification, we must determine first which objects are under consideration and then which objects are which. The order in which we consider the objects also determines the invariant set that describes the relationship (e.g., triad *abc* yields one descriptor and *acb* yields another, but the two are related).

While this is useful for identification purposes, it also means that we might have to search for all the combinations of a given triad (or quad, etc.), which seems extraordinarily wasteful. If we could find a way to order them consistently and search we could drastically reduce the number of requisite searches. Thus, we want an invariant descriptor that is both projectively and permutatively invariant. Initially, this might seem that we lose order information, as Bergamasco et al. [88] lament, suggesting that the loss of correspondence to each marker makes it impossible to extract pose without stereo images or additional features. This, however, is what I intend to disprove in the coming section.

4.2 Databases and search

Most of us are familiar with at least some database structures; a list or an array is such a structure. Databases are containers for information that we not only want to store but to search—preferably in a timely manner. Databases like the list or array in programming can be sorted or unsorted—i.e., they do or do not contain information about order. Ordering can be by value like descending or ascending value or any number of alternatives. Ordered lists contain the same number of entries as the same list without ordering, but they contain more information. This information can speed up the search process considerably since we might not have to search the entire array to ensure we have the closest value to an input. This ordering, however, does come at a cost: the cost of ordering the array. What's more, the array might be dynamic, or subject to change. With each addition or deletion, the order

must therefore be maintained. The structure, ordering, and dynamics of a structure are all points of trade-offs that programmers must consider before using a database.

4.2.1 Database Management Systems (DBMS)

The structures listed in Section 4.2 are examples of structures that contain information, which we may broadly define as a database structure, but more sophisticated database structures exist. With the additional sophistication comes extra costs and benefits. One particular set of database structures with a stringent set of requirements is called database management systems (DBMS). DBMS vary widely in terms of implementation and focuses, but they all must adhere to four particular requirements, known collectively as ACID:

- atomicity
- consistency
- isolation
- durability

Atomicity is the idea that a transaction (e.g., insertion, query) either completes fully or leaves no effect on the database. That is, if an insertion command partially completes but does not finish, this is a violation of atomicity. Consistency means that the DBMS rejects any request from a user that does not follow the the database schema (a schema is how a relationship or set of relationships are defined). A DBMS must also isolate, meaning that multiple transactions operating simultaneously (which happens infrequently but enough to corrupt data) must be consistent with the same transactions occurring in sequence. This is similar to the idea in programming to concurrency and race conditions. Finally, durability is the idea that once a transaction has occurred the result of the operation persists, meaning that if upon restart the changes revert to previous values the database fails this requirement.

The ACID requirements for databases ensure (or, at least, try to ensure) that information within a database remains consistent across multiple platforms and operating systems and subject to many distinct users and transactions; this is obviously beneficial to data integrity, so we desire a system that meets these specifications to some degree. Some specialty systems purport to be a database management system but not strictly adhere to the ACID

requirements, but those discussions are based on specific architectures and implementations which will not inform our discussion here, so we skip that here.

A special case of database systems are spatial databases; these systems are designed to work with applications involving coordinate systems, like computer-aided design (CAD), geographic information systems (GIS), and surveying and mapping. We can, however, exploit the storage techniques made for 2D and 3D applications and extend them to higher dimensions for our purposes. Whenever we have a set of n-dimensional data, these are great applications to use despite being designed for spatial data.

Because GIS and mapping applications generally deal with enormous numbers of data points (e.g., points and shapes), they have developed an extensive array of database systems to cope with such vast amounts of information. Among those most well-studied and tested for n-dimensional data is the R-tree [89], which is an extension of the one-dimensional B-tree and well-suited for applications of higher dimensions. Also included in the assessment as a comparison is the *kd*-tree, which has become a staple in many disciplines for its simplicity and versatility.

4.3 R-trees

Called the R-tree due to its use of bounding rectangles, the R-tree is a powerful tool for storing and searching spatial data. Despite its extensive use in geospatial and similar domains, it sadly has not been part of spacecraft navigation applications, likely due to lack of interaction between navigators and the computer science communities. It nevertheless is a perfect domain for the R-tree's capabilities. In addition to searching n-dimensional points, R-trees store and search polygons and other higher order shapes just as easily. It does this by only storing the data in leaf (end) nodes and keeping only bounds information about the data within its children.

While there are many variants on the original R-tree, including packing capabilities (space utilization nearing 100%), overlap minimization, and other optimization techniques, I will discuss the basic, no-frills version of the R-tree. Although these optimizations no doubt imbue the resulting database with exceptional space usage and access times, I develop this idea fully within the basic R-tree so that a comparison between relatively similar algorithms as the *kd*-tree and others is fair and indicative of the benefits of this approach, as well as making implementation into existing codebases much simpler. However, the final algorithm

after comparison will include several of the optimizations such as packing using space-filling curves and improvements for static data.

4.3.1 R-tree alternatives

Due to the implementation overhead, many alternative algorithms are better for data sets with handfuls or a few hundred points or objects. When the number of objects increases ever upward, R-trees tend to outperform most other spatial database algorithms, though this advantage has several trade-offs worth considering. For database sizes numbering no more than several dozen, the linear search model is the obvious choice due to effectively non-existent overhead and dead simple implementation. Anything beyond a few hundred entries demands a more careful analysis depending on several objectives and the potential costs.

The most important of these is the complexity of implementing the R-tree, which should never be overlooked. In such a case, the *kd*-tree is a solid choice among many competitors, though performance suffers much more than that of the R-tree as the dimension increases beyond 2D. We also need to modify the basic *kd*-tree to work with intersections of bounding rectangles and *k*-nearest neighbor (knn) search, which adds complexity but is necessary for inexact searches. A general rule of thumb is that a *kd*-tree performance is great in 2D, reasonable in 3D, and drops off rapidly thereafter. A great *kd*-tree algorithm is the NanoFLANN repository [90], which I used for comparison for its well-implemented functions and ability to search based on distance—something that is not included on many of the simple implementations that only match those that are exact.

Another similar structure is the *k*-vector and its multidimensional extension known as the *nd k*-vector. This has been implemented in a 1D search for inter-star angles in the Pyramid Star Identification technique [91] to great success, which searches the database three times for angles within a range of the three computed angles. It is a variant of the hash table, but its extension into higher dimensions has marginal performance that falls below that of the *kd*-tree.

Another important note is that the benefits of R-trees become much more compelling when the data is disk-bound—that is, not held in memory but in non-volatile storage. This is another reason that for smaller sets R-trees would be a poor choice. But again, as the size increases, the need to store the full data on disk rather than in memory is more apparent

since storing several gigabytes of data in memory might hinder execution in other tasks. This disk-bound method moves from a choice to a necessity when we consider resource-constrained platforms like spacecraft, where RAM capacities might be well below the requirements to store the entire database in memory. While inordinately slower than memory, disk storage limits are often correspondingly larger than that of memory. Additionally, when placed on disk, a search becomes extraordinarily expensive, thus limiting the number of queries is paramount. Generally the performance drop is 50-200 times slower for disk than it is for memory [92]!

We might also consider that our searches for "spatial" information contain noise about the search criteria. When perusing databases with noisy data, it seems apparent that finding precise points becomes unlikely; this is a considerable strength of the R-tree, since it searches using bounding rectangles (or hyper-rectangles for higher dimensions) and we can expand our search area/volume/hyper-volume to account for a given amount of noise. Binary splits, as those in *kd*-trees, do not enjoy this luxury and are susceptible to noisy searches and k-nearest neighbor searches slow down and complicate the algorithm, though this can be necessary.

One last item to consider is a database update. This last piece is not strictly necessary for our purposes, but for some applications this might be a significant issue. Adding a node requires traversing the tree until it finds an open position: either a node with open children slots or a leaf node. This insertion sometimes requires re-balancing or reassembling the structure, but thanks to the R-tree framework, most insertions require only a slight reshuffling of nodes. It is possible, however, that a major node reshuffling is necessary, which is rather expensive. Note that a *kd*-tree suffers from this same drawback, but to an even greater extent. Again, for our application where updates are rare to non-existent (i.e., we are not adding new stars or craters often if at all), this is a non-issue after initial indexing.

4.3.2 R-tree algorithm

Like most tree structures in any dimension, R-trees begin with a root node. From this starting point, it links information about nearby data points into children nodes, and recurses until exhausting the list of data. For each set of nearby nodes, it groups them into a minimum bounding rectangle (MBR). That way, any search that does not lie in the MBR of a group cannot contain the search point, thus eliminating unnecessary traversals down incorrect paths. At the very bottom are the leaf nodes; each leaf contains the data of an

original object of the full set. Moving up the tree enlarges the bounding region around the leaf nodes.

Figure 4.1 provides a sample of what the R-tree structure would look like given a set of 2D data points.

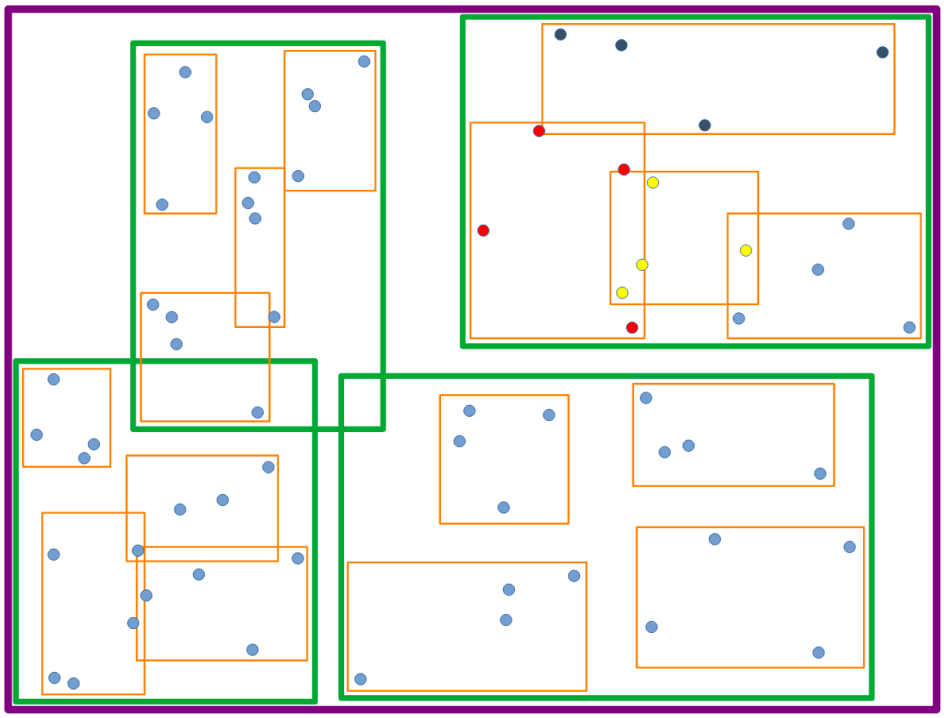

Figure 4.1: R-tree visualization given a set of 2D points shown with dots. This shows three levels of nodes (shown in descending order: purple, green, orange) when the number of child nodes is four. The image shows that overlapping regions are possible unless additional structuring takes place, which reduces redundancy within the tree. The nodes in the upper right of the image have color-coded values to illustrate which point belongs with which minimum bounding rectangle (MBR). Note that here a point only shows one color, but depending on implementation several MRBs may contain the same point.

4.4 Reforming the R-tree for invariants

Now the question arises: How do we exploit the R-tree structure to efficiently search for a descriptor? Since invariant theory and its applications lie almost entirely within the

realm of mathematics, little work tackles both the interrelatedness of invariants describing the same set as well as the combinatoric nature of the problem.

Christian et al. describe the problem for a triad of craters (lunar or otherwise). They employ a specific ordering of craters (namely, clockwise or counterclockwise) so that each crater can be positively and uniquely identified; this also reduces the number of permutations to search. They discuss several methods to address the unadulterated, ordering-specific descriptor, which can be solved by either searching the database several times (computationally expensive) or including all of the cyclic permutations (disk space expensive). They further investigate several methods to introduce P^2-invariants, or invariants that are immune to projective distortions and cyclic permutations; this covers different ordering schemes such as minimum or maximum value first [93], [94]. These, they note, are susceptible to noise and could easily (though not necessarily often) break down to the worst case of searching the catalog three times. An alternative presorts the interior triangle legs by shortest first [12], but this approach suffers from that of above as well as not being a projective invariant.

Finally, they propose a solution to this ambiguity under projective invariant theory using a particular class of functions. Both the idea and execution of this idea are discussed in the following sections. I call this idea the "indirect" method, namely "indirect insertion" and "indirect search". These names will become clear in the following section.

4.4.1 Symmetric and antisymmetric functions

Variable ordering is generally important. Swapping variables with one another typically leads to incorrect results—possibly catastrophic results. Mathematics, though generally a stickler for meticulous bookkeeping, provides us with a special class of polynomials that toss such oppressive restrictions as ordering without a second thought: the symmetric polynomials. Symmetric polynomials are a set of functions that are invariant to ordering changes. Such a function is as simple as

$$a = \sum_{i=1}^{n} x_i \tag{4.1}$$

or, slightly more complicated,

$$b = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_i x_j, \text{ for } i \neq j \tag{4.2}$$

There are many more such polynomials that share this property. One additional property that we would like is not just symmetry, but *anti*symmetry, which reflects how many swaps take place among the variables. One example of this is

$$c = x^4y^2z - x^4z^2y - y^4x^2z + y^4z^2x + z^4x^2y - z^4y^2x \tag{4.3}$$

Equation 4.3 is known as a monomial antisymmetric function, which means that a base monomial x^4y^2z acts as the form for the remaining monomials and simply permuting the variables and changing the sign according to the sign of the permutation (i.e., odd or even).

For example, the values $x = 1$, $y = 2$, and $z = 3$ result in -72. It is elementary to show that if we cyclically permute this sequence, say, $x = 2$, $y = 3$, and $z = 1$, we get the same answer: -72. If, however, we swap two variables—as in $x = 2$, $y = 1$, and $z = 3$—we get its negation: $+72$. The same is true for any odd number of swaps. This cyclic group is known as $\mathbb{Z}/3\mathbb{Z}$, and the antisymmetry also forms the alternating group A_3. We need only one function to serve as the antisymmetry equation since the sign here is sufficient to prove cyclic permutation for a set of three.

In the physical implementation of this algorithm, I use the polynomial generator functions provided in [20], but alternatives satisfying these criteria are potentially just as powerful and descriptive, though a careful study of performance and numerical accuracy would be necessary.

4.4.2 A new paradigm for database storage and query

With the descriptor equations and invariants in hand, it is now time to introduce a shift in the storage and retrieval paradigm in a database with a set of P^2-invariants. Obviously with the cost of query and data access, reducing the number of such operations correlates to a significant boost in performance for databases. While I have motivated the use of R-trees based on several important objectives, the fundamental idea of this paradigm applies just as easily to the aforementioned *kd*-trees or any other database structure used for storing multidimensional data.

Every database in multidimensional access structures stores the data in the form presented, with the exception of hash tables, which are the closest relation to this proposal, but with important differences which I shall discuss shortly. Even in the case of multidimensional hashing the table uses the data directly to form the hash, then store the hash.

The typical storage algorithm (and corresponding search algorithm) processes each point of data in a particular way, finds where in the structure this particular piece may fit in. It may traverse several nodes or branches along the way, then finally end up where the data scheme demands that it go. Some optimized algorithms (e.g., packed R-trees [95]–[98], R*-trees [99], [100], grid files [101], STR methods [102]) sort the data prior to this stage using a number of tricks, but the end result is the same: the location of the data is determined by the data itself. If the data is processed when creating the database, the search must also include the same processing algorithm. This makes sense in the case of spatial data, where the ordering matters (i.e., swapping x with y or z is a wholly unrelated point). But in the case of P^2-invariant data, this is not the case: each descriptor element in one configuration of the descriptor is related to elements of a permuted result. That is, all the possible descriptors from a cyclic permutation are related.

Consider the similar case of a set of five stars used to determine not only camera attitude but also camera calibration. Invariant theory demands that at least five stars be considered to form an invariant known as the cross ratio (recall Introduction Section 1).

The general form that produces the number of independent invariants is

$$n_{invariants} = \dim \mathbb{V} - s \tag{4.4}$$

where $\mathbb{V}$ represents the variety (object), which could be a point or conic in $\mathbb{P}^2$, d is the number of objects forming a set, and s is effectively the number of constraints.

The minimum number of points required to define a set of invariants in this 2D projective space is 5 since it is the first integer to produce a strictly positive value for projective space. The number of invariants is two, since [20], [24]

$$n_{invariants} = 5d - 8 \tag{4.5}$$

where d is the number of stars per set and where SO_3 (special orthogonal group 3, which contains rotations) acts on the group, and the dimension of this space is 8. Similarly, we know that for the group PGL_4 (projective general linear group 4, containing rotations, translations, and projections) we need at least 2, since

$$n_{invariants} = 8d - 15 \tag{4.6}$$

where the dimension of a general orbit is 15 for PGL_4 acting on quadrics (an 8-dimensional variety). Finally, since there are no invariants for a set of conics in arbitrary space, Christian et al. [20] propose placing these conics on a non-degenerate surface (a quadric), which they show to have the following formula for independent invariants

$$n_{invariants} = 3d - 6 \tag{4.7}$$

Thus, the number of invariants for this scenario is $3 * 3 - 6 = 9 - 6 = 3$ invariants for such a set of conics on a non-degenerate surface.

Again, this results from a proper framework of invariants suited to a particular space. For any d-tuple of points we define a descriptor, which is merely the set of invariants from that d-tuple of points. This descriptor now defines a particular d-tuple of points. While five stars is sufficient to produce a set of invariants, we find that additional points will produce more unique descriptors. The more points—and, consequently the more invariants—the more unique descriptor we obtain.

In the star identification algorithm, two invariants are insufficient to properly differentiate it from neighbors, so selecting six or possibly seven stars (producing four and six invariants, respectively) is likely a better choice, but considering the explosive growth of the combinations (i.e., `n choose k`), more than seven might be excessive, both for storage and access times. In a similar way to the triad of craters, we need to establish a P^2-invariant for the 4D and 6D case, something that the R-tree can handle natively, but for which the kd-tree becomes increasingly sluggish. These cases become one of the main reasons for using the R-tree in 3+ dimensions rather than the kd-tree since the algorithm remains largely the same for all cases.

4.4.2.1 Indirect search and store

Searching and storing functions must use the same data and methods. If we process the data and store the result in a particular way, then to compare the resultant we must have an inverse mapping. If, however, we process the data in a given way to point it to a certain portion of the structure and then store the raw data, then this inverse mapping function is unnecessary. In effect, this is the method that I have produced to eliminate repeated calls to a search function for a given invariant descriptor.

Note that the search can use a specific ordering of the invariant descriptor for both

storage and retrieval—one that remains consistent over noise corruption. This is not an essential step, though it can speed up matching once the R-tree has returned results from a given query. The real power of this algorithm is that only one search is necessary, whereas three would be necessary to ensure that all possible combinations have been found. This differentiation allows fewer accesses to storage (which, again, are extremely expensive—more so than memory accesses). This reduction means substantially curtailed runtimes compared to the naïve case where we search the database in triplicate or have a triply redundant database, as well as better resilience to noise and better disambiguation than using P^2-invariants directly.

This approach is a hybrid of hash tables (in this case in multiple dimensions) and R-trees and attempts to leverage the power of R-trees in searching and the idea of hashing a value and placing the result in a particular place. The drawbacks of the hashing method include unbalanced hash tables, where a vast majority of the values end up in the same bin while other bins have no entries; indeed the worst case scenario devolves into a linear search of all values in the table. Additionally, there have been no attempts at using hash tables to store P^2-invariants of which the author is aware, and hash tables are susceptible to noise [103]. Conversely, the R-tree has the inherent balancing, but is unable to effectively categorize the invariants if the values are permuted. This was not the original idea, but through several iterations of various ideas, this idea surfaced as the most viable solution for P^2-invariants, and the upcoming results section showcases the indirect search and store method.

Pseudocode for inserting data and retrieving data are provided in Algorithms 1 and 2.

Care is essential when using a group whose invariant descriptor is not a cyclic permutation of its objects. An example of this is the descriptor for three coplanar conics [20], that is, the seven-element descriptor of a three object group. The permutation of the triad is a permutation of the *pairs* of the elements, namely,

$$I = \{I_{ij}, I_{jk}, I_{ki}, I_{ji}, I_{kj}, I_{ik}, I_{ijk}\} \tag{4.8}$$

we have the mirror pairs

Algorithm 1 Indirect invariant insertion algorithm

$N \leftarrow$`length`$(catalog)$ ▷ Length of the catalog
$M \leftarrow$`length`$(objects)$ ▷ Number of objects
$L \leftarrow$`length`$(descriptor)$ ▷ Number of invariants
Require: $N, M, L \geq 0$ ▷ Non-empty check
Ensure: $Im[J_a^i] = Im[J_a^i] = Im[J_c^i] = 0$ ▷ No imaginary part
  for i in $1 : N$ **do**
    $idx \leftarrow 0$
    $\sigma \leftarrow NULL$ ▷ Initialize best ordering value
    $I_a \leftarrow I_a^i$ ▷ First object in catalog row i
    $I_b \leftarrow I_b^i$ ▷ Second object in catalog row i
    ... ▷ And so on...
    $Arr \leftarrow [I_a, I_b, ..., I_M]$
    $P2 \leftarrow$`P2_invariant`$(I_k, I_{k+1}, ..., I_{k-1})\ \forall\ k = a, b, ...$ ▷ Compute P^2-invariant
    for n in $1 : M$ **do** ▷ OPTIONAL; additional sorting
      $V \leftarrow$`permute`(Arr, n) ▷ Permute ordering
      $J_{inv} \leftarrow$`J_invariant`$(I_k, I_{k+1}, ..., I_{k-1})\ \forall\ k = a, b, ...$ ▷ Compute J^2-invariant
      $V \leftarrow$`order_values`(J_{inv})
      if $V > \sigma$ **then** ▷ Exceeds previous ordering value
        $idx \leftarrow n$
      end if
    end for
    $X \leftarrow$ `insert`$(J_{inv},$ `permute`$(Arr, idx))$ ▷ Insert descriptor
  end for

$$(I_{ij}, I_{ji}) \tag{4.9}$$

$$(I_{jk}, I_{kj}) \tag{4.10}$$

$$(I_{ki}, I_{ik}) \tag{4.11}$$

and one cyclic permutation produces

$$I = \{I_{jk}, I_{ki}, I_{ij}, I_{kj}, I_{ik}, I_{ji}, I_{jki}\} \tag{4.12}$$

with the final one

$$I = \{I_{ki}, I_{ij}, I_{jk}, I_{ik}, I_{ji}, I_{kj}, I_{kij}\} \tag{4.13}$$

where we note that the final entries (I_{ijk}, I_{jki}, or I_{kij}) are the the same value. Thus the

Algorithm 2 Indirect invariant search algorithm

$N \leftarrow$`length`$(catalog)$ ▷ Length of the catalog
$M \leftarrow$`length`$(objects)$ ▷ Number of objects
$L \leftarrow$`length`$(descriptor)$ ▷ Number of invariants
Require: $N, M, L \geq 0$ ▷ Non-empty check
Ensure: $Im[J_a^i] = Im[J_a^i] = Im[J_c^i] = 0$ ▷ No imaginary part
  for i in $1 : N$ **do**
    $idx \leftarrow 0$
    $\sigma \leftarrow NULL$ ▷ Initialize best ordering value
    $I_a \leftarrow I_a^i$ ▷ First object in catalog row i
    $I_b \leftarrow I_b^i$ ▷ Second object in catalog row i
    ... ▷ And so on...
    $Arr \leftarrow [I_a, I_b, ..., I_M]$
    $P2 \leftarrow$`P2_invariant`$(I_k, I_{k+1}, ..., I_{k-1})\ \forall\ k = a, b, ...$ ▷ Compute P^2-invariant
    for n in $1 : M$ **do** ▷ OPTIONAL; additional sorting
      $V \leftarrow$`permute`(Arr, n) ▷ Permute ordering
      $J_{inv} \leftarrow$`J_invariant`$(I_k, I_{k+1}, ..., I_{k-1})\ \forall\ k = a, b, ...$ ▷ Compute J^2-invariant
      $V \leftarrow$`order_values`(J_{inv})
      if $V > \sigma$ **then** ▷ Exceeds previous ordering value
        $idx \leftarrow n$
      end if
    end for
    $Y \leftarrow$ `search`$(J_{inv},$ `permute`$(Arr, idx))$ ▷ Search for descriptor
  end for

pseudo J-invariant for the whole series would then be two J-invariants in sequence with the final triad invariant (e.g., I_{ijk}). Or, Christian et al. [20] uses the discrete Fourier transform (DFT) for computing both the non-coplanar (with 3 invariants) and coplanar (with its 7). For example, the equations here demonstrate the DFT for a set of three craters [20]

$$F_1(x, y, z) = a = x + y + z \tag{4.14}$$

$$F_2(x, y, z) = \frac{b^2}{c} + \frac{c^2}{b} \tag{4.15}$$

$$F_3(x, y, z) = \frac{1}{j}\left(\frac{b^2}{c} - \frac{c^2}{b}\right) \tag{4.16}$$

They note that though the original descriptors are less susceptible to noise, they require searching the catalog 3 times—a hugely expensive operation—but that P^2-invariants, which permit single-shot searches, are correspondingly *more* susceptible to noise. This is the

shortcoming that my contribution ameliorates by using the P^2-indexing to point *where* the algorithm should place (and search for) the information (single-shot search), but ultimately placing the less noise-sensitive information in that location: potentially the optimal tradeoff.

At this point I must note that disambiguation must take place at some point: prior to query, within the search, or after the search. Choosing the first option, which possibly sorts the invariants prior to searching, *must* search the database the full number of valid permutations (in the current scenario of a triad of three craters, three times) to ensure that all possible triads are within the search results. Using P^2-invariants reduces this to a single query, much like the indirect invariant search; the difference here is that the result has no information on the original ordering, and some process must therefore disambiguate the ordering by some involved method to extract the correct ordering. The indirect search method strikes a balance between these two extremes, and the disambiguation process is a simple element-wise comparison of the catalog version and some permutation of the calculated invariants. This trade-off offers a simple alternative to general P^2-invariants and a fast search time to the triplicate search model.

Another benefit of the proposal is that if we store the P^2-invariants directly, then the search resultant provides no information as to the information of the original invariants and their order. When extracting the descriptor in this new manner, then the resulting descriptor has the original ordering or some cyclic permutation of it. Thus, when the search concludes, the user is given both the invariant information as well as the order of that triad.

The implementation of the indirect search R-tree uses the regular `insert` and `search` functions, with a single additional parameter: a pointer to the P^2-invariant function. This way the user determines which function they would like to calculate the P^2-invariant and not be dependent on a hard-coded function. It is important to note that the same function used to store the values *must* be used to search, else the index position will be incorrect. This function pointer must also agree with the number of elements in the descriptor.

To make this idea more tantalizing for those with history using other database systems (e.g., the *kd*-tree), I expand this concept to more than just the R-tree, though it provided the ideal structure for searching in n-dimensional space. For example, this same procedure is amenable to most familiar database structures, albeit with tweaks to storage and retrieval. For example, the *kd*-tree is a great candidate since it is simple to implement and familiar to those outside the DBMS community. Let us demonstrate how to implement such an idea in

an existing pipeline.

I show this in the *kd*-tree, but the idea applies to most common multidimensional databases. Some databases might require storing both the P^2-invariant and the original descriptor, which could double the size of the container, but it would still outperform the catalog reproduced in triplicate. The search would then search the database using the P^2-invariant against the same in the database, then extract the secondary part (i.e., the descriptor). While not as ideal as the indirect search method, it still should decrease search times and number of total searches.

4.5 Numerical results

Now that I've shown that this can feasibly apply to most of the database systems engineers and scientists are familiar with, let us delve into the performance of the R-tree implementation. I will compare a vanilla version of the R-tree (i.e., non-optimized) with my indirect indexing method as well as a comparison against a similar *kd*-tree. Some results here are not entirely representative of performance on real applications, but this works in the favor of R-trees generally since they are designed for disk storage. The R-trees and *kd*-trees will both be stored instead in memory at first since the databases are relatively small (400k entries of real lunar crater invariants); this set of data is included in Figure 4.2.

One optimization that I employed for both data structures is using `int`s instead of `float`s for the values; using `int` decreased the size of the structure by one-third with no other changes. Since all descriptor values ranged from 0 to 3, I applied a scalar to each value to bring all values to greater than one and less than the `int` data type maximum value; see Figure 4.2. While this loses some data, it compresses storage and saves on search time by around 20-30%. This mapping is also reversible (minus some loss in data type conversion less than 1% due to truncation or rounding). The only caveat is that all searches of the catalog must use the same scalar to map the calculated invariant values to the same scale, but based on the results moving from `float`s to `int`s I believe this is a beneficial tradeoff. Moreover, computing the P^2-invariant for the scaled set is a multiple of the original—for example,

$$sF(x, y, z) = F(sx, sy, sz) \tag{4.17}$$

where F performs the conversion on the P^2-invariant.

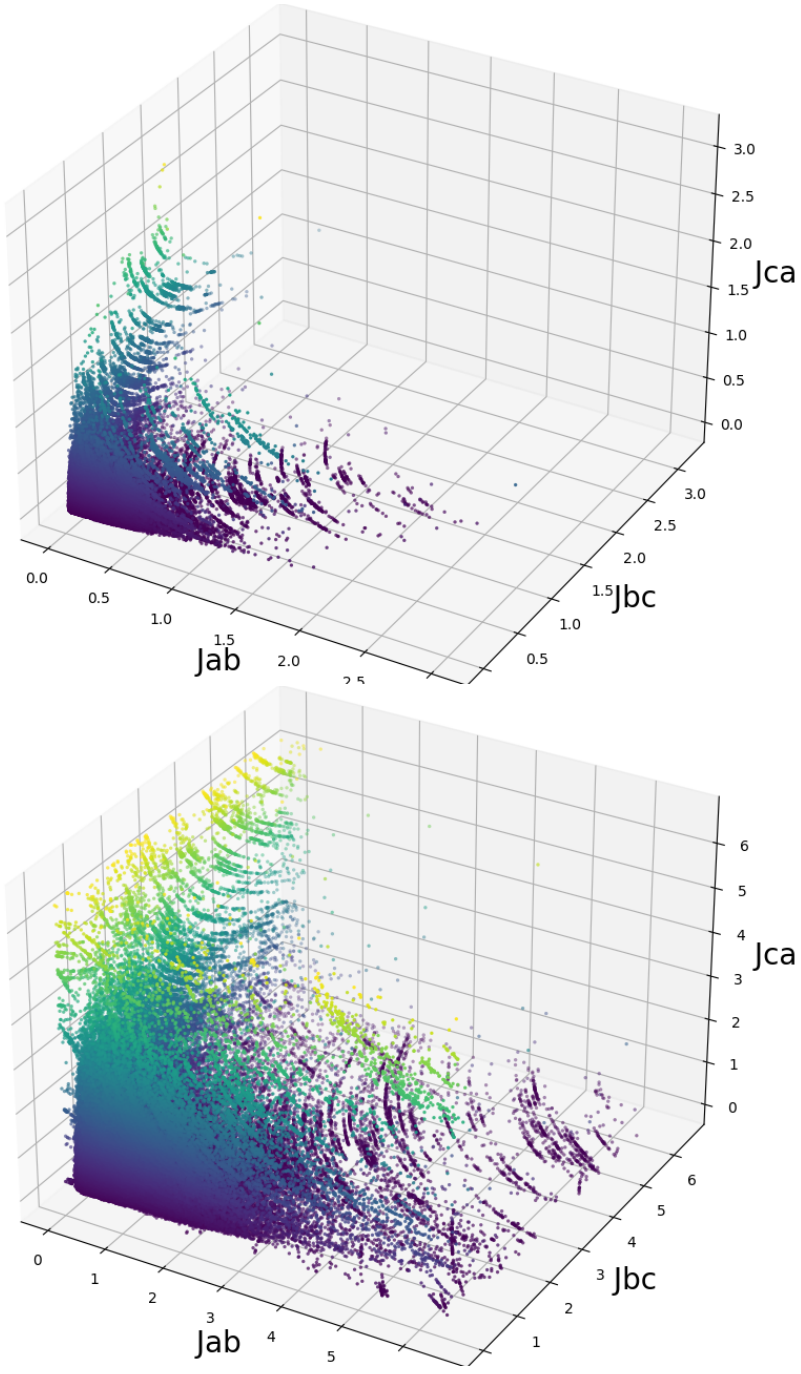

Figure 4.2: **Plots of the original set of over 400,000 invariants from the crater triads (craters ranging from 20 to 500 kilometers in radius). The top shows the original distribution of points, and the bottom shows the scaled values to ensure an integer representation for a smaller file size. The colors merely intend to differentiate points at different heights in the image. J_{ab}, J_{bc}, J_{ca} are the invariants associated with a crater triad ordering of A, B, and C.**

All versions of the R-tree and the alternative *kd*-tree have search options for rectangular search regions. Normally the *kd*-tree will either return only a single entry, or uses *k*-nearest neighbor to find nearby entries. The R-tree does this natively, but this is an add-on feature for the *kd*-tree. This feature drastically slowed performance (10-20x) on the *kd*-tree, so I decided to use *k*-nearest neighbor instead. Both programs were implemented in C++ using no optimizations, and were compiled using `g++` (version 11.2.0 on Linux Ubuntu); I designed the same algorithms in Python using existing packages for verification purposes. A preliminary run only searched using invariants in the correct permutation just to get an idea of the inner workings of the two distinct programs.

Table 4.1: Python tree creation time in seconds. Both flavors of the *kd*-tree outperformed the both R-tree variants (though only one is shown here). Creation times for both R-tree versions were sufficiently similar to include only one.

# Entries	Rtree_{I^3}	KD-Tree_{knn}	KD-TRee_{radius}
1000	0.08111	0.0007	0.0008
5000	0.5017	0.0038	0.0044
10000	1.2459	0.0048	0.0075
50000	6.5404	0.0369	0.0342
100000	12.9945	0.0686	0.0679
106615	14.100	0.076	0.069
461042	35.2430	0.3964	0.3928

The results without noise show the edge in 3D to the *kd*-tree versus the R-tree (though this algorithm has reached its effective maximum dimensions—in 4D and 5D this algorithm was rendered ineffectually slow). The time to create each storage container shows the *kd*-tree as the clear winner, taking as little as one percent the time as the R-tree, as seen in Table 4.1, though the performance in C++ was more near 20-50 times slower for the R-tree. With more entries this becomes even more apparent, since the R-tree naturally rebalances its structure when a leaf node becomes full. This upfront cost, though, pays dividends when searching in higher dimensional spaces (i.e., $n > 2$), which is a significant advantage in later stages. Note that the performance in 2D heavily favors the *kd*-tree to triumph over the R-tree due to its overhead, so we may not assume that this holds for even large datasets in 2D as opposed to 3D or greater. Furthermore, when the data contain no noise, the *kd*-tree scores another victory on timing.

Once we introduce noise, however, the *kd*-tree suffers much more than the R-tree and the two are roughly equal in computation time when the *kd*-tree uses k-nearest neighbor search, where k here is 5. Once we use the radial search (a search within a given radius about the query), which is more akin to the approach used by the R-tree, the two radically diverge and the *kd*-tree is about five to eight times slower than the other two algorithms. This is important to notice since for most of the invariant descriptors in Figure 4.2 huddle closely together near the origin, which means that a k-nearest neighbor search might not return the correct value if that region is particularly dense. Indeed, there were dozens of results (particularly at higher noise levels) where $k = 5$ failed to provide the correct match and a higher k value was necessary.

Table 4.2: Standard indexing R-tree vs *kd*-tree. Time (in milliseconds) to complete ten thousand random searches at varying levels of noise. Prior to each search, noise is added to each of the invariants of the descriptor by a Gaussian distribution with the standard deviation of σ and zero mean. The total number of descriptors in the catalog was 106615. No permutations occurred on the query data.

σ	Rtree	KD-Tree$_{knn}$	KD-TRee$_{radius}$
1.000	744	613	4396
1.585	762	626	5060
2.512	743	666	4901
3.981	742	676	4724
6.310	753	651	4719
10.000	715	732	5270
15.849	714	633	4507
25.119	760	723	5013
39.811	774	645	5012
63.096	724	650	5097
100.000	748	641	4935
158.489	712	654	4470
251.189	712	628	4469
398.107	706	630	4482
630.957	708	630	4457
1000.000	706	625	4459
1584.893	708	631	4426
2511.886	706	632	4451
3981.072	707	629	4461
6309.573	783	634	5418
10000.000	853	877	5634

I then introduced the indirect indexing R-tree, which used the P^2-invariant to find the search region. The difference is negligible between the all R-trees per search, but remember that the vanilla R-tree searches three times per descriptor. The indirect invariant search window might be enlarged to account for noise, but this is simple to execute in practice, though increasing the size of the windows to ten times the original value increases computation time by roughly 30%, but as seen in .

Table 4.3: Indirect indexing R-tree vs *kd*-tree. Time (in milliseconds) to complete ten thousand random searches at varying levels of noise. The same zero-mean Gaussian noise was added to each query as in the above table. Random cyclic permutations shifted the data which meant that both versions of the *kd*-tree and the vanilla R-tree all required three searches each.

σ	Rtree	KD-Tree$_{knn}$	KD-TRee$_{radius}$
1.000	748	2487	14934
1.585	713	2638	14154
2.512	709	2474	13946
3.981	700	2475	13897
6.310	702	2460	13879
10.000	697	2485	13886
15.849	696	2472	13787
25.119	716	2482	13673
39.811	700	2476	13653
63.096	695	2477	13700
100.000	698	2476	13849
158.489	696	2476	14016
251.189	696	2469	13850
398.107	695	2462	13806
630.957	705	2471	13747
1000.000	696	2471	13761
1584.893	696	2467	13766
2511.886	697	2465	13793
3981.072	693	2491	13891
6309.573	710	2476	14246
10000.000	739	2766	16593

The range for the P^2-invariants ranged from -4.4 to 5 using the DFT method from [20], so we can either apply a shift and a scaling to push the values positive and scale them up as we did before, or we can simply apply a scaling and keep the negative values. The only bookkeeping issue is moving from unsigned data values to signed values halves the effective

range (e.g., `uint8` has a range of 0-255, where signed `int8` ranges from -128 to 127 for 2's complement).

Searching the database using the indirect search method worked as well as the P^2-invariant for finding the correct descriptor, but significantly better than tripling the size of the database or using ordering alone. We expected as much, since tripling the size of the database means more space to search, but this difference was not as much as the *kd*-tree since the *kd*-tree will compare on average more nodes when traversing a tree of triple the size.

It is clear that the indirect invariant indexing (I^3) method has promise despite the aforementioned tradeoffs. It seems clear, in light of the evidence of Table 4.2, that the R-tree performs well in 3D space, and Table 4.3 excels when the data are permutations of the values in the catalog. Different techniques like scaling the descriptors to integer values provides a reduced footprint of the catalog while simultaneously boosting performance, albeit only slightly.

CHAPTER 5
QUADRIC POSE ESTIMATION

Quadratic surfaces, or quadrics, have seen extensive use in image processing [67], [104]–[109], and more specifically in space applications [20], [28], [42]. Before beginning this next step, we assume that the navigator has already identified three craters in the image as known craters for which we have information on position, orientation (in the case of elliptical craters), and size; no description or motivation about such identification is included here, though the interested reader is directed to Christian et al. in [20] for a robust framework of crater identification using geometric invariants particularly suited to this approach to spacecraft navigation.

As mentioned in background section 2.3, the relationship between a quadric surface and its image projected onto a plane is always a conic (including the special case the projection is a line, which is also a degenerate conic). We reiterate that relationship as in Eq. 5.1

$$C^* \propto PQ^*P^T \tag{5.1}$$

where the proportionality in this relationship illustrates the underlying polynomial structure equal to zero. That means that any nonzero scalar multiple of the above equation identically represents the conic. In fact, a scalar variable makes Eq. 5.1 an equality relation rather than proportional,

$$C^* = sPQ^*P^T \tag{5.2}$$

Using Eq. 5.2, we exploit the relationship between the two to obtain the camera transformation matrix—that is, the attitude and position of the camera relative to a reference frame. Let us use the Moon's inertial frame as the reference, since we will be using the Moon's craters in the analysis.

Originally, I used Gröbner bases [110]–[112] to determine this equivalency. Section 2.2 discusses the action of groups on varieties, and solving nonlinear systems of equations employ Gröbner bases and the Hilbert Nullstellensatz (or *theorem on locations of zeros*, discussed later), which rely heavily on understanding varieties. What, then, *is* a variety? An algebraic variety represents the common zeros of the polynomials $f_1, \ldots, f_r \in \mathbb{R}[x_1, \ldots, x_n]$ [112]. Those

familiar with the terms manifold, model, shape or space may be pleased to know that varieties convey the same idea, but without some of the accompanying implications (e.g., a manifold is smooth, but a variety has no such restriction). A simpler working definition here might be that it is a geometrical description of an object (e.g., an ellipsoid). Remember that a conic satisfies the following equation

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Fy + G = 0 \tag{5.3}$$

which indeed meets the definition of varieties as common zeros: any point on that ellipse, circle, parabola, or hyperbola must satisfy this equation. The same is true for quadrics and their mathematical representation. Both conics and quadrics are varieties of their respective spaces. The "action of a group" can be considered a transform; e.g., rotating a rectangle is an action of the Euclidean space. We can therefore consider the "action of a group on a variety" as transformations acting on an object.

While working with polynomials, we break apart the constituent elements of a polynomial that are separated by addition or subtraction. These elements are known as monomials, and are of the form [113]

$$x_1^{a_1} \cdot x_2^{a_2} \ldots x_{n-1}^{a_n - 1} \cdot x_n^{a_n}. \tag{5.4}$$

where the degree of the monomial is $\sum_{i=1}^{n} a_i \;\; \forall \;\; a_i \geq 0$.

Polynomials are simply linear combinations of monomials, and the sum and product of polynomials are also polynomials. Polynomials also satisfy all of the axioms of a field except the inverse multiplier element; they do, however, satisfy the requirements for a ring, and we refer to such structures as polynomial rings [26]. Refer to Appendix B for a short discussion on fields, rings, and commutative algebra.

Since polynomials can be formed as combinations of other polynomials, we want to find the minimal set of polynomials that comprise the current set of polynomials. We can think of this as the greatest common divisor, or `gcd` of two natural numbers. This set is known as the *ideal*. (Normally, this is done using the Euclidean algorithm [114], but the coming discussion in Appendix B.4 shows that the Euclidean algorithm is simply a subset of the Gröbner bases.) Given a set of polynomials F, a collection of polynomials generate the ideal

[113] (a special subset of F), given

$$\langle f_1, \ldots, f_m \rangle = \{\Sigma_{i=1}^{m} h_i f_i \mid h_i \in k[x_1, \ldots, x_n]\} \tag{5.5}$$

Any additional polynomials formed by a combination of the polynomials in F shares the ideal. Formally, an ideal satisfies the following properties of a ring R and its ideal I [114]:

i. $a + b \in I \ \ \forall \ \ a, b \in I$

ii. $ar \in I \ \ \forall \ \ a \in I$ and $r \in R$

Gröbner bases are akin to linear systems solvers like Gaussian elimination. In fact, such applications as Gaussian elimination, `gcd`, and integer programming are specialized forms of these Gröbner bases.

Gröbner bases can determine whether a system of linear or nonlinear systems have solutions, how many solutions, and what those solutions are. Nonlinear systems of equations are notoriously difficult to solve, and those that have general solutions are very well-defined and well-behaved. General nonlinear systems, such as our application, might not have solutions, so the first step is finding out: does this system have solutions?

The result of running Gröbner bases on the set of symbolic functions resulted in a group of 49 basis polynomials. Normally, one or more in this series would contain only one variable, albeit an equation with a high degree. This allows the system to be solved sequentially, with each new equation solution cascading into the variables of the other equations. This set, however, did not contain such a convenient solution.

Instead of trudging through different simplifying assumptions (such as the canonical position, which I discuss later in Section 5.1.2), I shifted my focus instead toward a geometric solution. Moreover, the time required to find a solution was often on the order of minutes to hours on a desktop computer—a far cry from the minuscule computing power of an embedded spacecraft computer. However, that a solution even existed was proof enough of the usefulness of Gröbner bases in this problem, and I hope that this powerful tool becomes more widespread in similar problems.

I instead exploit the geometric properties inherent in the problem. To make the problem more visually useful, I develop the reasoning behind the circular crater case before moving on to the problem solution.

5.1 Circular craters and elliptical projections

Let's begin with a circular crater on the surface of a sphere. Such a model is useful in the case of the Moon, where all three axes of the Moon's triaxial ellipsoid model are exceptionally similar, warranting its representation as a sphere at a global scale. Figure 5.1 shows how we can define the circular quadric disk as the intersection of the body sphere and a plane. This can be done in a myriad of ways, but some include algebraic geometry and geometric algebra. I will describe the former, but the latter is a particularly satisfying treatment of geometric objects.

We can describe a crater as a flattened quadric surface, known as a quadric disk; this degenerate quadric is the disk whose edge marks the intersection of the spherical body surface and a right circular cone with a vertex at the origin (the body center), shown in Figure 5.1. Conversely, it is also the intersection of the sphere and a plane in which the crater lies.

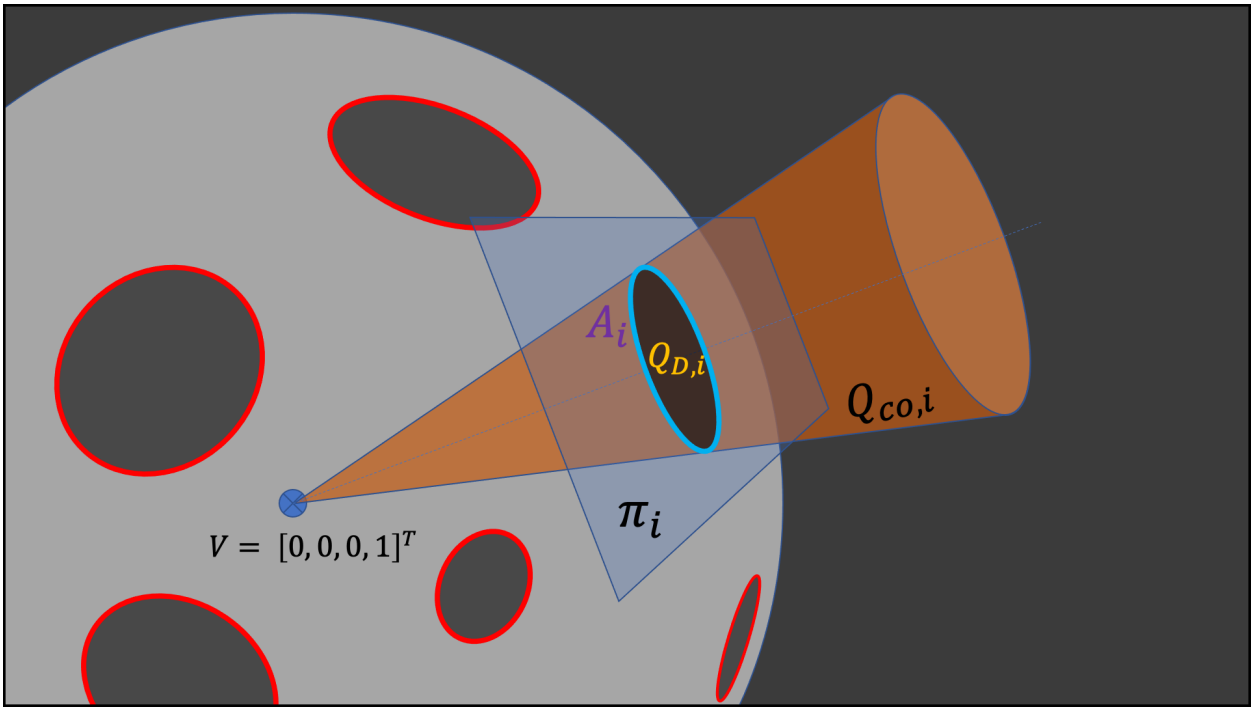

Figure 5.1: The disk quadric as the flat circular plate whose rim is the intersection of a sphere and a circular quadric right cone $Q_{co,i}$ or whose rim is also the intersection of the same quadric cone and the plane π.

While the solutions to the Gröbner bases were infinite, taking the time to verify points and attitudes that satisfy the equations resulted in some fruitful insights. The most important

outcome was that, at least for circular craters, the solutions to these equations lie in a plane parallel to the crater's plane. What's more is that those solutions all lay at a particular distance from the camera, meaning that those solutions construct a circle in 3D: another conic! This insight led to another circle that mirrored the first, meaning that one circle lay above and the other lay below at the same distance from the crater. This ambiguity is common in imaging applications and pose estimation with conics, so we need more information to determine which circle is the correct choice. This is what drives the selection of two other conics in the image; these other two conics along with the first disambiguate the location and orientation of the camera relative to the reference frame. The next section deconstructs this idea.

5.1.1 Quadrics and the intersections of circles

Since the set of solutions to this problem lie in a circle parallel to that of the circular crater, the intersection of the corresponding three 3D circles provides the position solution. In the absence of noise these three circles intersect at precisely the same point; working with noise and real images, the solution is the corresponding least squares solution to the intersection.

With the intersection of these three circles as the position solution, solving for the attitude component requires knowing the crater normals in both the reference frame and the camera frame. The crater locations in the Moon frame are given in the Robbins Crater Database [115] (or any other crater database) in latitude and longitude (equivalent to declination δ and right ascension α, respectively), which convert to Cartesian coordinates in the following calculation

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \rho \begin{bmatrix} \cos\delta\cos\alpha \\ \cos\delta\sin\alpha \\ \sin\delta \end{bmatrix} \tag{5.6}$$

where ρ is the body radius.

With these normals in the Moon frame, we can calculate the potential camera frame crater normals using the approach found in [109], [116]. In a later section, I discuss the issues inherent in the assumptions made during this phase, but this has produced consistent results despite its shortcomings.

Based on the geometry, each image conic has four mathematically valid normals and associated circle centers, two of which lie in front of the camera and two behind. Tossing aside the two behind the camera, we focus on the two alternatives in front. This means we have six candidate vectors, three of which are the correct choice. Since we know the orientations of each circle with respect to the other two, we know the homography from each crater to the other. These circles need not lie on a non-degenerate quadric or planar surface, which *is* required when computing the invariants used in the approach in [20].

5.1.2 Canonical position and orientation

We denote each plane with π_i for $i = 1, 2, 3, ...$ where i is the index of the crater (see Figure 5.1. With the craters and their planes in mind, we must determine the orientation of the circular crater with respect to the camera frame. Using either the approach from [109], [116] or [117] we exploit the relationship between the circular feature and its projection onto the image plane, both of which place the conic in the center of the image and orient the axes of the ellipse with the image axes using the transformation known as the *canonical transformation*, introduced by Kanatani [118], which consists of first a translation of the feature into the center of the image then a rotation to align its axes with that of the image. Figure 5.3 shows this transformation, which I call here the *canonical pose* This "canonical pose" greatly simplifies the enveloping cone used for backprojection. Figure 5.2 shows an example of backprojection from the camera focal point to the circular object in 3D space; this reprojection is the goal of the upcoming section.

This "canonical pose" of the ellipse can be thought of as a combination of a 2D translation (of a in x and b in y) and a 2D rotation of θ, represented by [119]

$$A = \begin{bmatrix} \cos\theta & \sin\theta 0 & \\ -\sin\theta & \cos\theta & 0 \\ \frac{a}{f} & \frac{b}{f} & 1 \end{bmatrix} \tag{5.7}$$

where f is the focal length of the camera (the form used here employs a focal-length normalized form, so that focal length is unity and can thus be ignored here). Applying transformation matrix A to a conic C is as follows [119]

$$C' = A^{-1} C A^{-T} \tag{5.8}$$

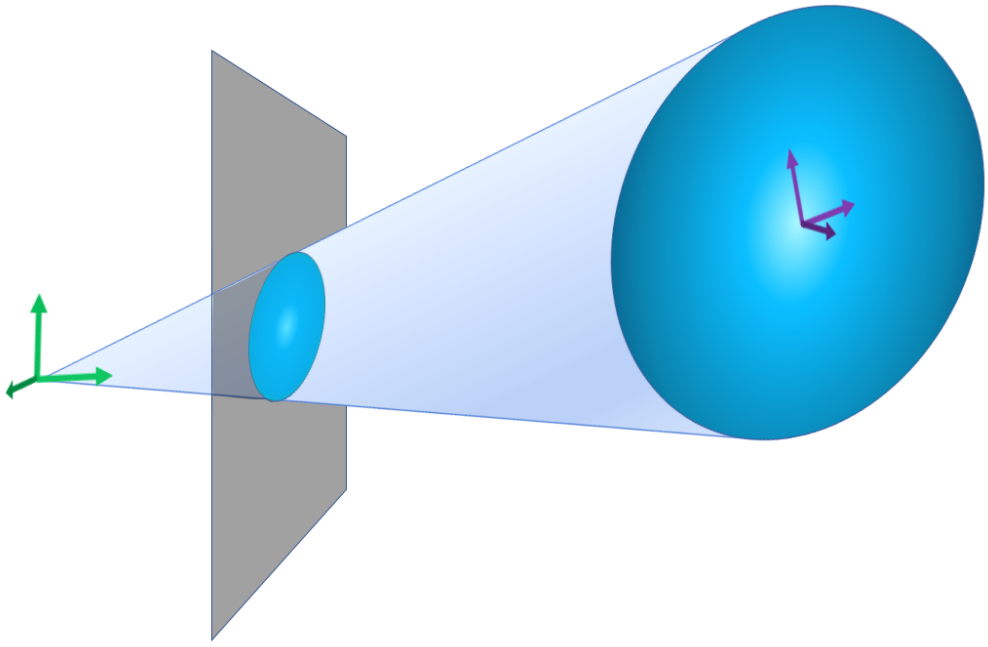

Figure 5.2: Example of conic backprojection given an ellipse on the image plane and the circular object in space. An elliptical cone (the "enveloping cone") has its vertex as the focal point of the camera and expands in the direction of the circular object; the conic it forms by intersecting the image plane *is* the projection of the disk quadric onto the image plane.

which performs the whole action of moving the conic to the center of the image and rotates it so that the axes are aligned with the principal axes, as seen in Figure 5.3, where the black conic ultimately rests in the red *canonical pose*

The idea behind both [109], [116] and [72], [117] is to trace the cone that emanates from the focal point of the camera (the vertex) and encapsulates the circular feature, and the intersection of the cone with the focal plane forms the image ellipse. I will call this the *enveloping cone*. This combination of rotation and translation does not merely simplify the geometry of the problem, as Kanatani [118] notes, but this is the crucial idea to making this problem tractable. The mathematical form of the cone (now a right elliptical cone rather than oblique) collapses down significantly, forming a diagonal matrix and eliminating cross terms in the quadratic equation.

I will apply [117], but both approaches solve the same problem, albeit in different ways. Both approaches assume that both the ellipse parameters and the radii of the circles are

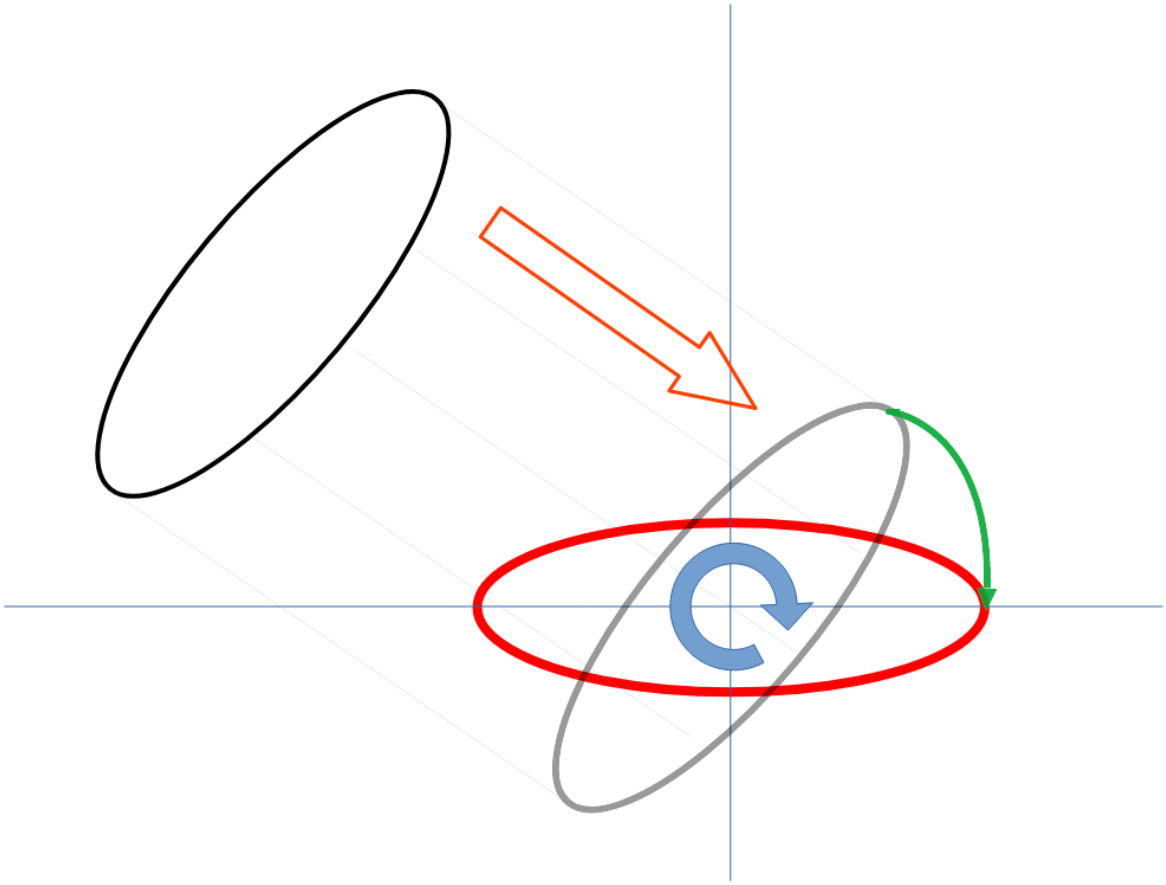

Figure 5.3: Illustration of the transformation of an ellipse (black) into the "canonical pose" (canonical position and orientation) (red) through an intermediate translation step (grey). This translation of the ellipse center to the coordinate frame origin followed by a rotation to align the ellipse axes with that of the coordinate frame.

known, which can be obtained in a myriad of methods outside the scope of this section, though Section 3 above discusses using the subpixel estimation method for streaks [54] or for its related edge-based algorithm [42] to obtain remarkably accurate ellipse parameters from the image.

The image conic matrix

$$C = \begin{bmatrix} A & \frac{B}{2} & \frac{D}{2} \\ \frac{B}{2} & C & \frac{E}{2} \\ \frac{D}{2} & \frac{E}{2} & F \end{bmatrix} \tag{5.9}$$

contains all the information we need about the ellipse.

We first compute the eigenvalues for C which we denote as μ_1, μ_2, μ_3 and the associated normalized eigenvectors as $\mathbf{f_1}$, $\mathbf{f_2}$, and $\mathbf{f_3}$. Since C is a conic and whose matrix is symmetric and of full rank, we know that two of the eigenvalues are of one sign and the third of the opposite sign.

Since any scalar multiple of this matrix also represents the conic, we order the eigenvalues with those of the same sign as μ_1 and μ_2 (subject to $|\mu_1| > |\mu_2|$), and the eigenvalue of opposite sign as μ_3. Of special note is when μ_1 and μ_2 are equal—this case indicates that the cone is right circular, and we know that the camera lies directly above the circle and need only compute the ratio of the radii to estimate the distance.

We desire a set of ordered eigenvalues of the conic at the canonical position and orientation, which we denote λ_1, λ_2, and λ_3, with the associated eigenvectors $\mathbf{e_1}$, $\mathbf{e_2}$, and $\mathbf{e_3}$. We want to ensure that the positive z-direction follows the convention [72], [117], so we need

$$\lambda_3 = \mu_3 \tag{5.10}$$

and

$$\mathbf{e_3} = \begin{cases} +\mathbf{f_3}, & \mathbf{k}^T\mathbf{f_3} > 0 \\ -\mathbf{f_3}, & \mathbf{k}^T\mathbf{f_3} < 0 \end{cases} \tag{5.11}$$

where $\mathbf{k}$ is the camera boresight direction in the camera frame

$$\mathbf{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.12}$$

The eigenvector λ_2 comes directly the remaining eigenvalues, satisfying

$$\lambda_2 = \begin{cases} \mu_1, & |\mu_1| < |\mu_2| \\ \mu_2, & |\mu_1| > |\mu_2| \end{cases} \tag{5.13}$$

and $\mathbf{e_1}$ is the eigenvector associated with the smaller magnitude eigenvalue ($\mathbf{f_1}$ or $\mathbf{f_2}$). This leaves the larger of the two same-sign eigenvalues—either μ_1 or μ_2. The remaining eigenvector, however, must form a right-handed system, so

$$\mathbf{e_1} = \mathbf{e_2} \times \mathbf{e_3} \tag{5.14}$$

Based on the relationship of these eigenvalues to the conic matrix, the resultant transformation matrix

$$M = \begin{bmatrix} \mathbf{e_1} & \mathbf{e_2} & \mathbf{e_3} \end{bmatrix} \tag{5.15}$$

orthogonalizes the conic matrix as follows (since M is orthonormal)

$$E = M^T C M \tag{5.16}$$

where E is the matrix with the λ_i as diagonal entries

$$E = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{5.17}$$

which is represents the following equation

$$\lambda_1 X + \lambda_2 Y + \lambda_3 Z = 0 \tag{5.18}$$

The resultant matrix is the standard or canonical form expression of a cone; the objective now is to determine the orientation of a circular disk given the intersection of a cone with its vertex at the camera focal point. We find that only two such orientations of a surface normal are possible [120], with one being the correct normal and the other its mirror reflection. This ambiguity is seen in Figure 5.4—note that the two possible circle centers are not coincident nor do they coincide with the ellipse center (except in the case of the right circular cone where only one normal is possible), which is a result of the projective geometry.

As an example of a triad of circles, see Figure 5.5, where an example of a real image I took using a calibrated camera shows several circular features in the field of view. The results here were promising, since only rough estimates of position ($\pm 3cm$) and approximate angle and orientation were known and still produced values within 15cm at a range of 2 meters and within about a degree of angular error. Additionally, stock camera parameters were used and no distortion correction was applied; only one such figure is significantly distant from

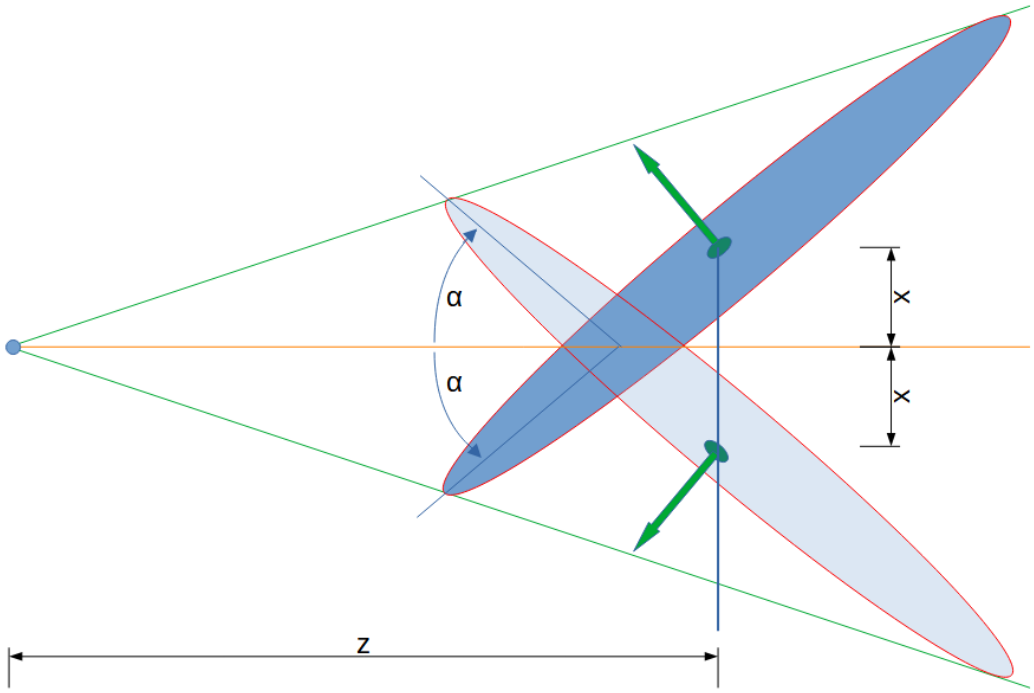

Figure 5.4: Example of where the centers of the circles and normals are calculated. Notice that the second ellipse is a mirror image of the first, and the angles and distances are the same, save the negative or positive angle.

the principal axis (the top disk in red) to be affected by distortion, but even a slight misfit can skew the results, as seen in the nearly 8 percent location error.

Figure 5.6 shows the camera in the reference frame as well as its view frustum. This custom Python visualization program was used to ensure proper orientations and positions, as well as provide visual confirmation when positioning and orienting the camera.

The points of each circle must now satisfy the equation of its corresponding enveloping cone

$$\frac{X^2}{k_x^2} + \frac{Y^2}{k_y^2} - \frac{Z^2}{k_z^2} = 0 \tag{5.19}$$

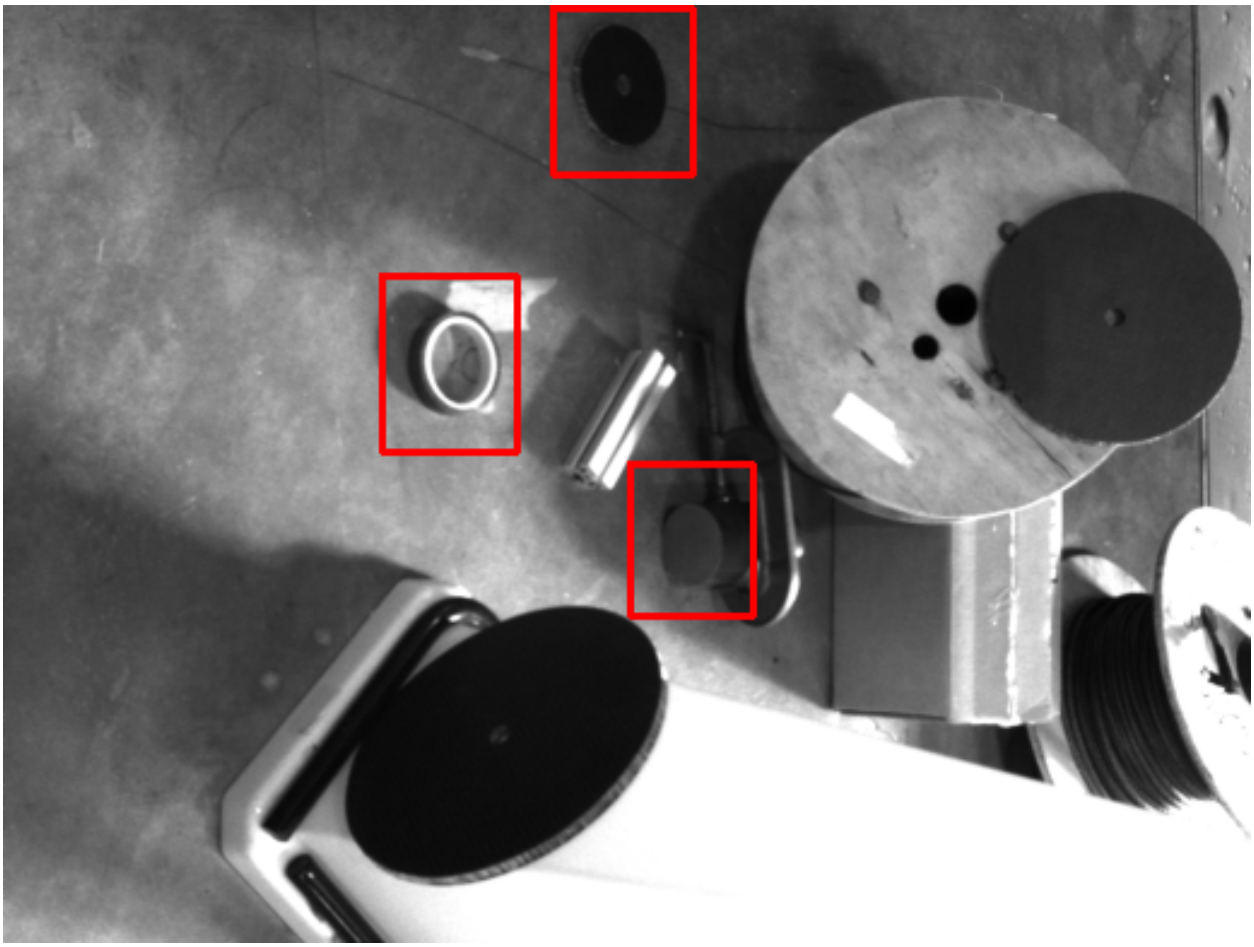

Figure 5.5: Real image with several circular objects with known pose; image personally taken with camera used at Intuitive Machines. The algorithm produced an accuracy of 15cm and within two degrees of the known camera pose. OpenCV's contour function computed ellipses after finding the contours in the image.

and the equation of a 3D circle

$$(X - X_0)^2 + (Y - Y_0)^2 + (Z - Z_0)^2 = R^2 \tag{5.20}$$

where R is the radius of the circle and X_0, Y_0, and Z_0 are the center of the circle. Note the relationship between Eq. 5.18 and Eq. 5.19, and we find the relationship between the eigenvalues of the cone and the cone equation, namely

$$k_x^2 = \frac{1}{|\lambda_1|} \quad k_y^2 = \frac{1}{|\lambda_2|} \quad k_z^2 = \frac{1}{|\lambda_3|} \tag{5.21}$$

Both [117] and [72] move the sectioning plane to the vertex of the cone (the origin),

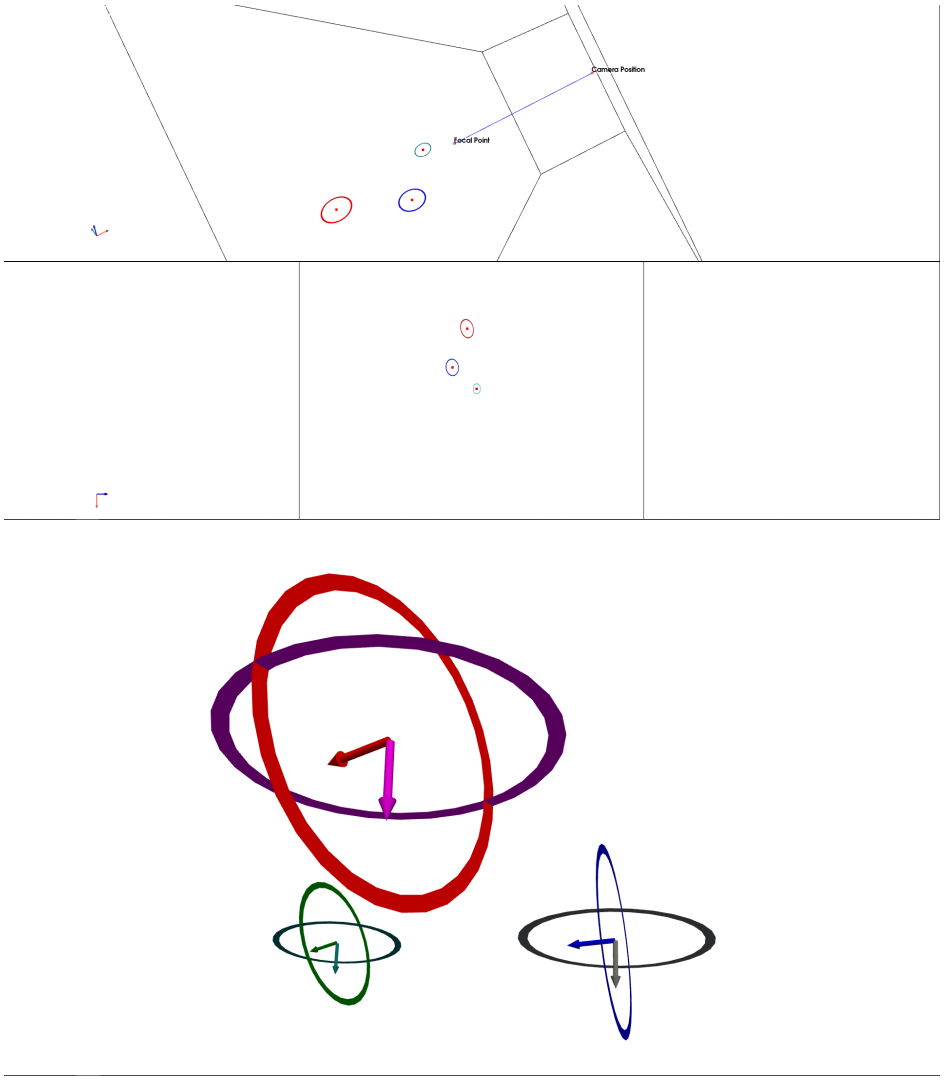

Figure 5.6: Top shows a custom Python visualization (using PyVista) of the camera position and its view frustum in the previous image. The two possible normals are shown at the bottom image, and we can see that the two geometrically indistinguishable options face in varying directions. That is, if we were to place a disk at either orientation for all of the disks in the image, it would be impossible to extract the correct disk without more information.

which simplifies Eq. 5.20 and the radius of Eq. 5.20 shrinks to zero, thus Eq. 5.20 becomes

$$X^2 + Y^2 + Z^2 = 0 \longrightarrow Y^2 = -X^2 - Z^2 \tag{5.22}$$

Substituting $(-X^2 - Z^2)$ for Y^2 from Eq. 5.22 into Eq. 5.19 yields the plane oriented to slice the cone into a circular disk, given as

$$\frac{X^2}{k_x^2} + \frac{-X^2 - Z^2}{k_y^2} - \frac{Z^2}{k_z^2} = 0 \tag{5.23}$$

Using the relations in Eq. 5.21 into Eq 5.23, we have

$$\left(\frac{1}{k_x^2} - \frac{1}{k_y^2}\right) X^2 = \left(\frac{1}{k_z^2} + \frac{1}{k_y^2}\right) Z^2 \tag{5.24}$$

which reduces to

$$\frac{k_y^2 - k_x^2}{k_y^2 k_x^2} X^2 = \frac{k_y^2 + k_z^2}{k_y^2 k_z^2} Z^2 \tag{5.25}$$

Rearranging, we get

$$\frac{1}{k_{xz}^2} \left(\frac{k_y^2 - k_x^2}{k_y^2 + k_z^2}\right) X^2 = Z^2 \tag{5.26}$$

where

$$k_{xz} = \frac{k_x}{k_z} \tag{5.27}$$

and finally

$$\pm \frac{1}{k_{xz}} \sqrt{\frac{k_y^2 - k_x^2}{k_y^2 + k_z^2}} X = Z \tag{5.28}$$

Looking at Eq.5.28, we can substitute the λ values for the k values [117] as shown in Eq. 5.19

$$\frac{\frac{1}{|\lambda_3|}}{\frac{1}{|\lambda_1|}} \left(\frac{\frac{1}{|\lambda_2|} - \frac{1}{|\lambda_1|}}{\frac{1}{|\lambda_2|} + \frac{1}{|\lambda_3|}}\right) X^2 = Z^2 \tag{5.29}$$

where we can rearrange the terms thus

$$\frac{|\lambda_3|}{|\lambda_1|}\left(\frac{\frac{|\lambda_1|-|\lambda_2|}{|\lambda_2||\lambda_1|}}{\frac{|\lambda_3|+|\lambda_2|}{|\lambda_2||\lambda_3|}}\right)X^2 = Z^2 \tag{5.30}$$

Finally, simplifying Eq. 5.30 we obtain the equivalent to Eq. 5.28 using λ instead of the related k values

$$Z = \pm\sqrt{\frac{|\lambda_1| - |\lambda_2|}{|\lambda_2| + |\lambda_3|}}X \tag{5.31}$$

Thus we have the orientation of the planes that section the cone into circular cross sections, but this is merely an orientation of two sets of parallel planes that create circular cross sections. We yet lack the precise pair that satisfy our crater size. We need to translate the sectioning planes out to a distance where the radius of the circles matches that of the actual crater. Remembering Eq. 5.18 and knowing that the circle center lies on the y-axis (thus, $y = 0$)

$$\lambda_1 X + \lambda_3 Z = 0 \longrightarrow Z = \pm\sqrt{\frac{\lambda_1}{\lambda_3}}X = \pm\frac{1}{k_{xz}}X \tag{5.32}$$

since λ_1 and λ_3 have opposite signs.

If we move the two planes down the boresight by a scalar s, then the equation becomes

$$Z = \pm\sqrt{\frac{|\lambda_1| - |\lambda_2|}{|\lambda_2| + |\lambda_3|}}X + s \tag{5.33}$$

Leveraging the geometry of the problem we can move the planes out a unit distance in $+z$ making Eq. 5.33 look like

$$Z = \pm\sqrt{\frac{|\lambda_1| - |\lambda_2|}{|\lambda_2| + |\lambda_3|}}X + 1 \tag{5.34}$$

What remains then is the required translation, s, of the projected conic to expand along the camera's z-axis until the diameter of the image conic $2r$ matches that of the quadric $2R$, which is given by the relationship

$$s = \frac{R}{r} \tag{5.35}$$

We now insert Eq. 5.32 into Eq. 5.28 to obtain

$$Z = \pm\sqrt{\frac{k_y^2 - k_x^2}{k_y^2 + k_z^2}}Z + 1 \tag{5.36}$$

The solutions to Eq. 5.36 are the two z-values in the camera frame of the circle centers. This is demonstrated in the previous figure where the two centers lie equidistant from the camera vertex and are effectively mirrors of the other. We therefore must solve for the remaining coordinate value in the rotated camera frame, of which only x remains to be solved since the canonical position of the feature aligns the semi-major axes with the y-axis, so the y-value for both solutions is 0.

We adopt a simplification similar to Christian [72] as follows

$$\alpha = \sqrt{\frac{k_y^2 - k_x^2}{k_y^2 + k_z^2}} \tag{5.37}$$

and now rewrite Eq. 5.36 as

$$Z = \pm\alpha Z + 1 \Longrightarrow Z = \frac{1}{1 \pm \alpha} \tag{5.38}$$

The two solutions are as follows, from Eq. 5.32 are

$$X_i = \pm\frac{k_{xz}}{1 \mp \alpha} \qquad\qquad i = 1, 2 \tag{5.39}$$

$$Z_i = \frac{1}{1 \mp \alpha} \tag{5.40}$$

where X_1 and Z_1 denote the positively sloped plane.

The distance between these two calculated centers in the camera frame is given by

$$\begin{aligned} dX &= X_2 - X_1 \\ &= \left| \frac{-k_{xz}}{1 + \sqrt{\alpha}} - \frac{k_{xz}}{1 - \sqrt{\alpha}} \right| \\ &= \frac{2k_{xz}}{1 - \alpha} \end{aligned} \tag{5.41}$$

and

$$
\begin{aligned}
dZ &= Z_2 - Z_1 \\
&= \left| \frac{-1}{1+\sqrt{\alpha}} - \frac{1}{1-\sqrt{\alpha}} \right| \\
&= \frac{2\sqrt{\alpha}}{1-\alpha}
\end{aligned}
\tag{5.42}
$$

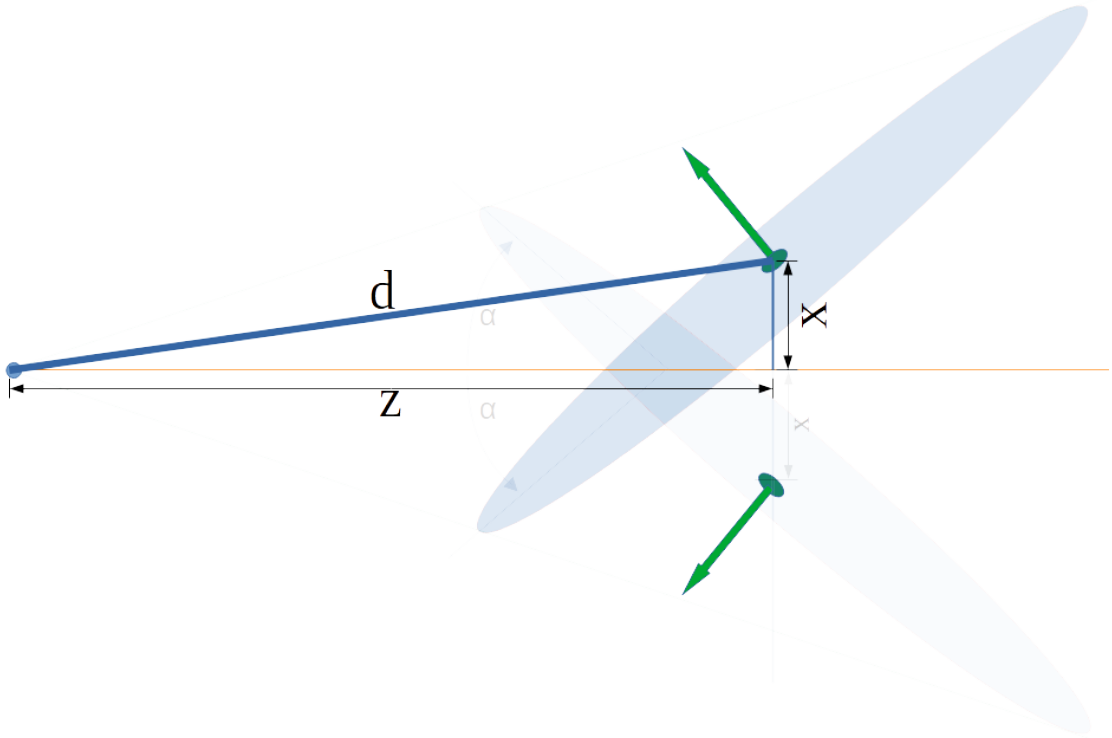

Figure 5.7: Illustration of the distance between the camera vertex and the circle center (rather than the ellipse center).

We can therefore use the values for dX and dZ from Eq. 5.41 and Eq. 5.42 to compute the length of the hypotenuse of the right triangle shown in Figure 5.7 as follows

$$\begin{aligned}(2d)^2 &= dX^2 + dZ^2 \\ &= \left(\frac{2k_{xz}}{1-\alpha}\right)^2 + \left(\frac{2\sqrt{\alpha}}{1-\alpha}\right)^2\end{aligned} \tag{5.43}$$

which leads to

$$4d^2 = \frac{4k_{xz}^2}{(1-\alpha)^2} + \frac{4\sqrt{\alpha}^2}{(1-\alpha)^2} \tag{5.44}$$

$$d^2 = \frac{k_{xz}^2}{(1-\alpha)^2} + \frac{\alpha}{(1-\alpha)^2} \tag{5.45}$$

$$d = \frac{\sqrt{k_{xz}^2 + \alpha}}{1-\alpha} \tag{5.46}$$

5.2 Solution using homography

The ambiguous results from the previous section (shown visually in Figure 5.8) mean that we must eliminate the extraneous solutions and extract the valid ones. The problem is that we don't necessarily want to restrict our problem to a non-degenerate quadric surface, like we needed to when constructing the invariant descriptor set (see Section 2.2 and [20]). That way, even though Figure 5.9 shows the original application to this approach, we do not need such a setup; instead, we can have arbitrary quadric orientations.

We are left with eight possible combinations of crater centers and normals (2^3, e.g., $A_1B_1C_1,\ A_1B_1C_2,\ A_1B_2C_1,\ ...$). Rather than trying all combinations and checking the errors on each, we can apply what we know from the catalog. Since we have information about the latitude and longitude and radius, we can extract the plane for each crater. Given these planes, we can compute the slopes (see Figure 5.10) that each plane makes with another, and find the correct triad of normals.

Given the set of three correct normals and their corresponding circle centers, we can now compute the homography matrix. The homography is a one-to-one mapping from one frame to another. A point lies in a plane if and only if

$$\pi^T \bar{x} = 0 \tag{5.47}$$

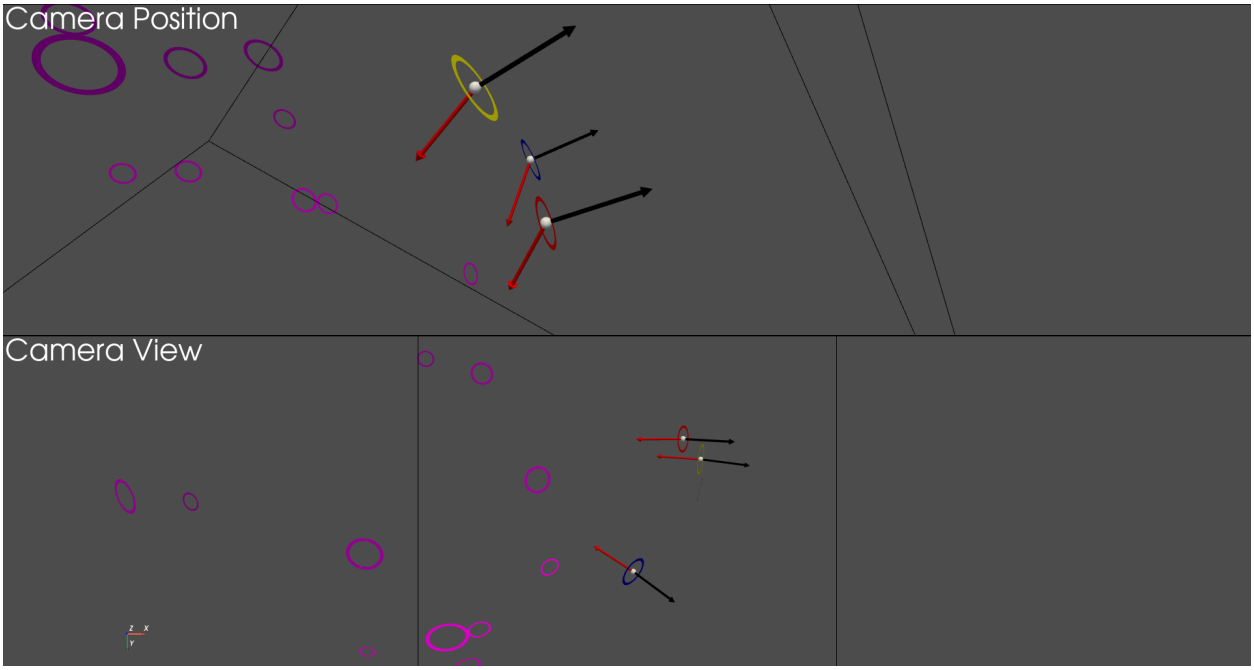

Figure 5.8: Visualization of the two possible surface normals given the camera view of each circular feature. The top view shows the location of the camera and the craters on the body surface, and the bottom shows the view from the camera. The true normals are shown in black and the counterfeit normals in red.

or, equivalently,

$$\bar{x}^T \Pi = 0 \tag{5.48}$$

for a given plane Π and homogeneous point $\bar{x}$. We want to know how to express that plane now in another frame given the homography between the two frames. Knowing how to transform a point using a homography, namely,

$$x' = Hx \tag{5.49}$$

where x' denotes the point in the new frame using the homography H. Knowing that a homography is a one-to-one mapping and is therefore an invertible matrix, we therefore may express Eq. 5.49 as

$$x = H^{-1}x' \tag{5.50}$$

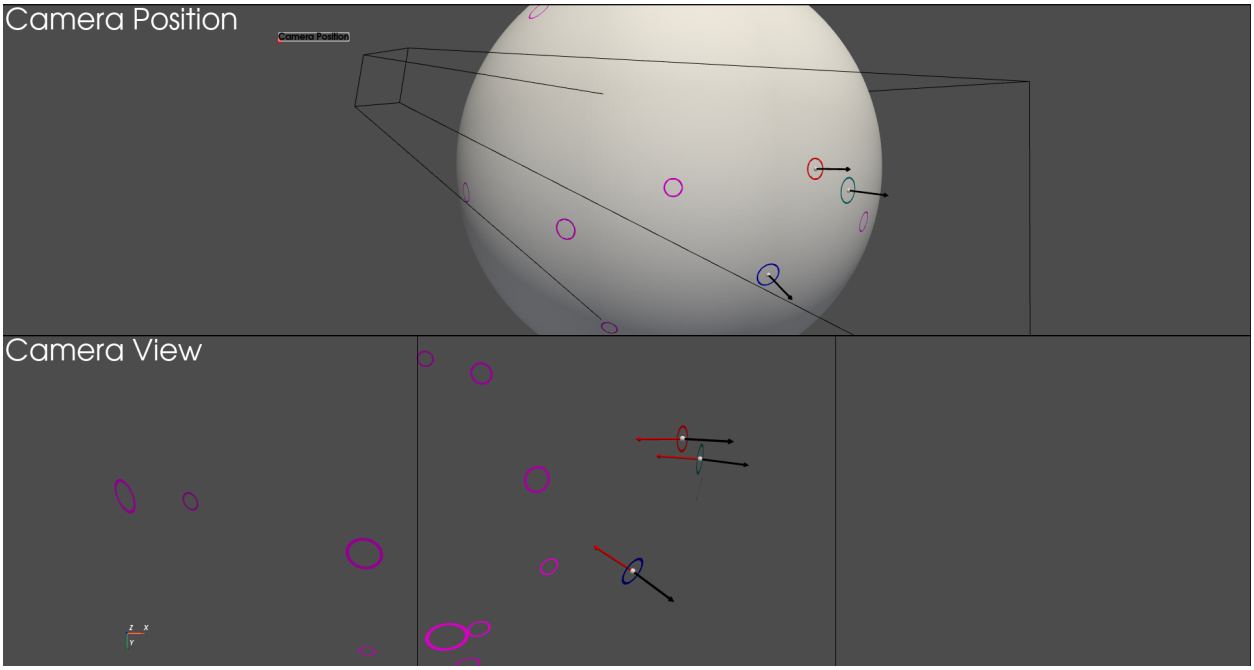

Figure 5.9: Spherical body showing the normals of the circular surface features on top and the corresponding dual normals (in red) given the camera view. Note that the restriction for circular features to lie on a non-degenerate surface is not a necessary condition for finding the pose.

Again, a point belonging to the plane π satisfies Eq. 5.47, so

$$\Pi^T H^{-1} \bar{x}' = 0 \tag{5.51}$$

$$(H^{-T}\Pi)^T \bar{x}' = 0 \tag{5.52}$$

Finally we see that the plane expressed in another from given the homography is

$$\Pi' = (H^{-T}\Pi)^T \tag{5.53}$$

Equation 5.53 is the crucial connection that resolves the equations we have that take us from the conic equations to the full pose: attitude and position. Let us construct the homography that takes us from the camera coordinate system (in which frame we have found the centers and normals of the craters) to the world coordinate system (in which we have the database information). The homography matrix H can be broken apart as follows:

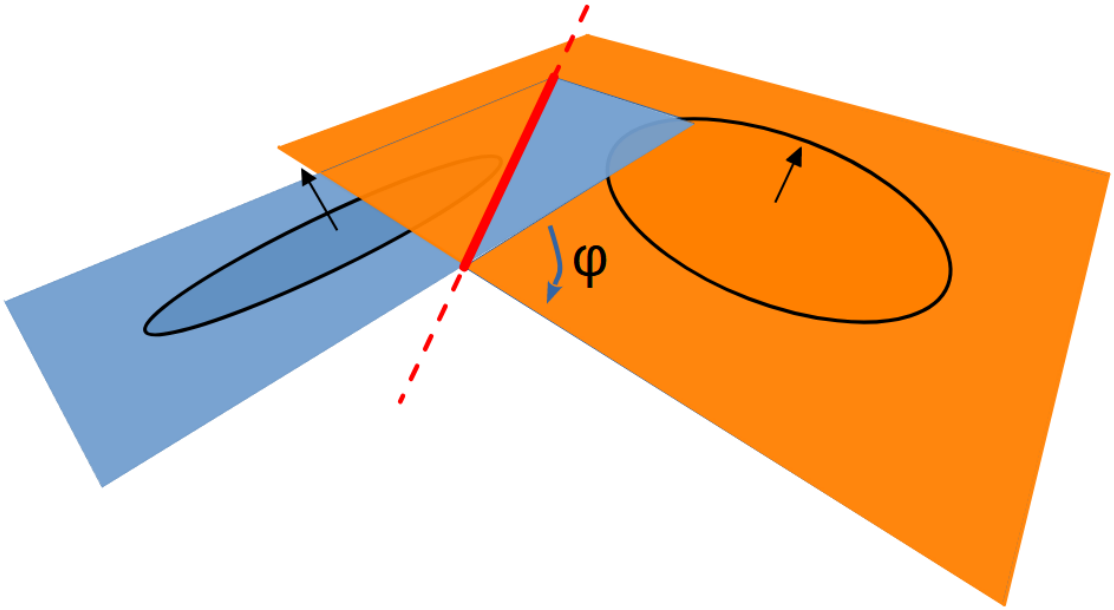

Figure 5.10: Given two disk quadrics, we define the angle ϕ as that between the two planes at the intersecting line (shown in red). Using the idea of homogeneous space, we can say that all planes in 3D intersect in a line, even parallel planes, where the line of intersection is the ideal line and the angle is 0 or 180 degrees.

$$H = \begin{bmatrix} R_C^M & t \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{5.54}$$

and

$$H^{-T} = \begin{bmatrix} R_M^C & \mathbf{0}_{3\times 1} \\ t^T & 1 \end{bmatrix} \tag{5.55}$$

where R_C^M is the transformation matrix, t is the camera position. R_b^a takes points expressed in the a frame into the b frame. This is the attitude. Given three planes expressed each in two ways, we have enough information to solve the simultaneous equations. That is, we can solve

$$\Pi'_i = H^{-T}\Pi_i \tag{5.56}$$

$$\Pi'_j = H^{-T}\Pi_j \tag{5.57}$$

$$\Pi'_k = H^{-T}\Pi_k \tag{5.58}$$

For each of the subscripts i, j, and k, the system is

$$\begin{bmatrix} \pi'_1 \\ \pi'_2 \\ \pi'_3 \\ \pi'_4 \end{bmatrix} = \begin{bmatrix} r_{x1} & r_{y1} & r_{z1} & 0 \\ r_{x2} & r_{y2} & r_{z2} & 0 \\ r_{x3} & r_{y3} & r_{z3} & 0 \\ X_M & Y_M & Z_M & 1 \end{bmatrix} \begin{bmatrix} \pi_{\mathbf{1}} \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{bmatrix} \tag{5.59}$$

where

$$\Pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{bmatrix} \tag{5.60}$$

We can therefore produce a set of 12 equations with the 12 unknown components of the homography. For instance, Eq. 5.56 creates the following equations:

$$\pi'_{i1} = r_{x1}\pi_{i1} + r_{y1}\pi_{i2} + r_{z1}\pi_{i3} \tag{5.61}$$

$$\pi'_{i2} = r_{x2}\pi_{i1} + r_{y2}\pi_{i2} + r_{z2}\pi_{i3} \tag{5.62}$$

$$\pi'_{i3} = r_{x3}\pi_{i1} + r_{y3}\pi_{i2} + r_{z3}\pi_{i3} \tag{5.63}$$

$$\pi'_{i4} = X\pi_{i1} + Y\pi_{i2} + Z\pi_{i3} + \pi_{i4} \tag{5.64}$$

Solving the equations simultaneously for all three (or more) plane pairs produces the least squares estimate of the location and orientation of the camera in the reference frame. The equations for the three are provided in Eq. 5.65, which, again, is not limited to three quadrics, but to any number of quadrics by just extending the same process above to determine the planes and compute the associated homographies. This is similar to vectorizing in the direct

linear transform [66].

One note about using planes and the homography mapping approach: we naturally lose information about the size of the craters when we use only the plane in which they reside, yet this loss *does* allow to express the homography solution in a linear system, rather than a high-order polynomial, such as that expressed in [104], which constrains the camera to lie at a certain point and only requires two quadrics. I posit that this tradeoff presents a simpler solution that relies on solving linear systems, especially since we already use three craters from which to identify the invariants. Furthermore, this approach is extensible to the n-crater problem: meaning that if we have identified more than three craters we can solve the linear problem in the least squares sense. The explanation herein does not rely on ideal conics, points at infinity, or nonlinear curves, but rather on a set of linear equations with which engineers and scientists are quite familiar. That said, this inherent tradeoff between an array of options must be considered before pressing on, and the approach in [104] presents such an alternative.

$$\begin{bmatrix} \pi'_{i1} \\ \pi'_{i2} \\ \pi'_{i3} \\ \pi'_{i4} - \pi_{i4} \\ \pi'_{j1} \\ \pi'_{j2} \\ \pi'_{j3} \\ \pi'_{j4} - \pi_{j4} \\ \pi'_{k1} \\ \pi'_{k2} \\ \pi'_{k3} \\ \pi'_{k4} - \pi_{k4} \end{bmatrix} = M \begin{bmatrix} r_{x1} \\ r_{x2} \\ r_{x3} \\ X \\ r_{y1} \\ r_{y2} \\ r_{y3} \\ Y \\ r_{z1} \\ r_{z2} \\ r_{z3} \\ Z \end{bmatrix} \tag{5.65}$$

where

$$M = \begin{bmatrix} \pi_{i1} & 0 & 0 & 0 & \pi_{i2} & 0 & 0 & 0 & \pi_{i3} & 0 & 0 & 0 \\ 0 & \pi_{i1} & 0 & 0 & 0 & \pi_{i2} & 0 & 0 & 0 & \pi_{i3} & 0 & 0 \\ 0 & 0 & \pi_{i1} & 0 & 0 & 0 & \pi_{i2} & 0 & 0 & 0 & \pi_{i3} & 0 \\ 0 & 0 & 0 & \pi_{i1} & 0 & 0 & 0 & \pi_{i2} & 0 & 0 & 0 & \pi_{i3} \\ \pi_{j1} & 0 & 0 & 0 & \pi_{j2} & 0 & 0 & 0 & \pi_{j3} & 0 & 0 & 0 \\ 0 & \pi_{j1} & 0 & 0 & 0 & \pi_{j2} & 0 & 0 & 0 & \pi_{j3} & 0 & 0 \\ 0 & 0 & \pi_{j1} & 0 & 0 & 0 & \pi_{j2} & 0 & 0 & 0 & \pi_{j3} & 0 \\ 0 & 0 & 0 & \pi_{j1} & 0 & 0 & 0 & \pi_{j2} & 0 & 0 & 0 & \pi_{j3} \\ \pi_{k1} & 0 & 0 & 0 & \pi_{k2} & 0 & 0 & 0 & \pi_{k3} & 0 & 0 & 0 \\ 0 & \pi_{k1} & 0 & 0 & 0 & \pi_{k2} & 0 & 0 & 0 & \pi_{i3} & 0 & 0 \\ 0 & 0 & \pi_{k1} & 0 & 0 & 0 & \pi_{k2} & 0 & 0 & 0 & \pi_{k3} & 0 \\ 0 & 0 & 0 & \pi_{k1} & 0 & 0 & 0 & \pi_{k2} & 0 & 0 & 0 & \pi_{k3} \end{bmatrix} \tag{5.66}$$

If the image has more than the required three craters, stacking the solutions similar to this method can also solve the problem in the least squares sense using this same computation (using, for example, *numpy*'s `lstsq` function in Python) , and indeed provides more robust localization in the presence of noise and error.

One final step is necessary to ensure that the solution produces a valid orthonormal attitude matrix, which the homography estimation does not guarantee. We do this by using singular value decomposition to orthonormalize the attitude matrix as follows

$$U, S, V^T = svd(T_{est}) \tag{5.67}$$

$$T^*_{est} = UV^T \tag{5.68}$$

where T_{est} is the computed attitude matrix of the homography solution, and T^*_{est} is the orthonormalized result of Eq. 5.67. This is similar to calculating the attitude matrix using the SVD in Algorithm 3, but without the need to ensure unity determinant. This approach is presented in the subsequent section.

5.2.1 Alternative solution to the homography

Another approach to the solution given the correct planes is using a standard approach to solving the attitude and position separately. This approach uses the same method as the previous up to solving the plane normals and angles to establish the correct orientations of the circles, but instead of solving the system of equations as before, we solve the pose estimation piecewise, solving separately for attitude then for position.

We can solve the attitude problem by forming an array of the normal vectors in the reference (Moon) frame,

$$n_{ref} = \begin{bmatrix} {}^r_i\mathbf{n} & {}^r_j\mathbf{n} & {}^r_k\mathbf{n} \end{bmatrix} = \begin{bmatrix} {}^r_i n_x & {}^r_j n_x & {}^r_k n_x \\ {}^r_i n_y & {}^r_j n_y & {}^r_k n_y \\ {}^r_i n_z & {}^r_j n_z & {}^r_k n_z \end{bmatrix} \tag{5.69}$$

where ${}^r_i n_x$ means the x-component normal of the ith crater expressed in the reference frame and in the camera frame

$$n_{cam} = \begin{bmatrix} {}^c_i\mathbf{n} & {}^c_j\mathbf{n} & {}^c_k\mathbf{n} \end{bmatrix} = \begin{bmatrix} {}^c_i n_x & {}^c_j n_x & {}^c_k n^c_x \\ {}^c_i n_y & {}^c_j n_y & {}^c_k n^c_y \\ {}^c_i n_z & {}^c_j n_z & {}^c_k n^c_z \end{bmatrix} \tag{5.70}$$

Note that we are currently only using three unit vector pairs, but this same process is extensible to the n-crater problem, similar to Section 5.2, also using linear least squares. With these two matrices, we can solve for the estimated attitude T_{est} using singular value decomposition (SVD) according to Algorithm 3.

Algorithm 3 Attitude estimation using SVD

$n_{ref} \leftarrow [{}^r_i\mathbf{n}, {}^r_j\mathbf{n}, {}^r_k\mathbf{n}]$
$n_{cam} \leftarrow [{}^c_i\mathbf{n}, {}^c_j\mathbf{n}, {}^c_k\mathbf{n}]$
$B \leftarrow n_{cam} * n_{ref}^T$
$U, S, V^T \leftarrow \texttt{svd}(B)$
$d \leftarrow \texttt{det}(U) * \texttt{det}(V^T)$
$M \leftarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{bmatrix}$
$T_{est} \leftarrow U * M * V^T$

With the attitude in hand, we turn our attention to the position estimation problem.

This problem can be solved using any number of trilateration (or multilateration in the case of $n > 3$) solutions. One relatively unknown approach uses geometric algebra (GA), which uses objects to represent actions on other objects. The GA approach uses the distance from each crater to the camera as the radius of a sphere centered at the crater center. The intersection of two spheres—calculated by taking the "meet" operation (effectively the intersection operation) of two of the spheres, which results in a circle (even if the spheres do not intersect). Taking the resulting circle and performing another meet operation with the remaining sphere results in a point pair, signifying the two possible positions for the camera. This two-point ambiguity also exists in other methods for trilateration, such as the Bancroft method [121]. Basic reprojection techniques (using the calculated attitude) or line-of-sight calculations resolve which of the two positions is valid. Figure 5.11 shows the resulting graphic using the `Ganja.js` geometric algebra toolkit.

Alternative methods for trilateration (or multilateration) include the least squares Bancroft method [121], as well as several others [122], [123], which result in the roughly the same result. With ideal information about the conics and the distances—as in the attitude estimation section—we calculate the position down to 10^{-8} meters. Again, this multilateration and SVD attitude approach is not restricted to the 3-crater case, and any more craters we can uniquely identify will aid in the pose estimation, similar to Section 5.2. The next section demonstrates the efficacy of both approaches in the presence of noise and in real and synthetic images.

5.3 Numerical Results

A full discussion of the results would be inadequate without a study of the effect of noise on various parameters. We first introduce noise intentionally to the ideal case by adding noise to the various ellipse parameters. After assessing the performance in this case, I investigate a scenario where the the ellipse parameters are drawn from synthetic and real images. This is the true mark of performance since any noise inherent in digital imagery is compounded by subsequent data handling and estimation and is a better measure for real-world performance than ideal or slightly less than ideal scenarios.

Compounding this study is that even the measurement of craters in real images is problematic, as demonstrated by [124] and later [125]. Within the studies are scientists with considerable clout and experience within the aerospace community, including S.J. Robbins

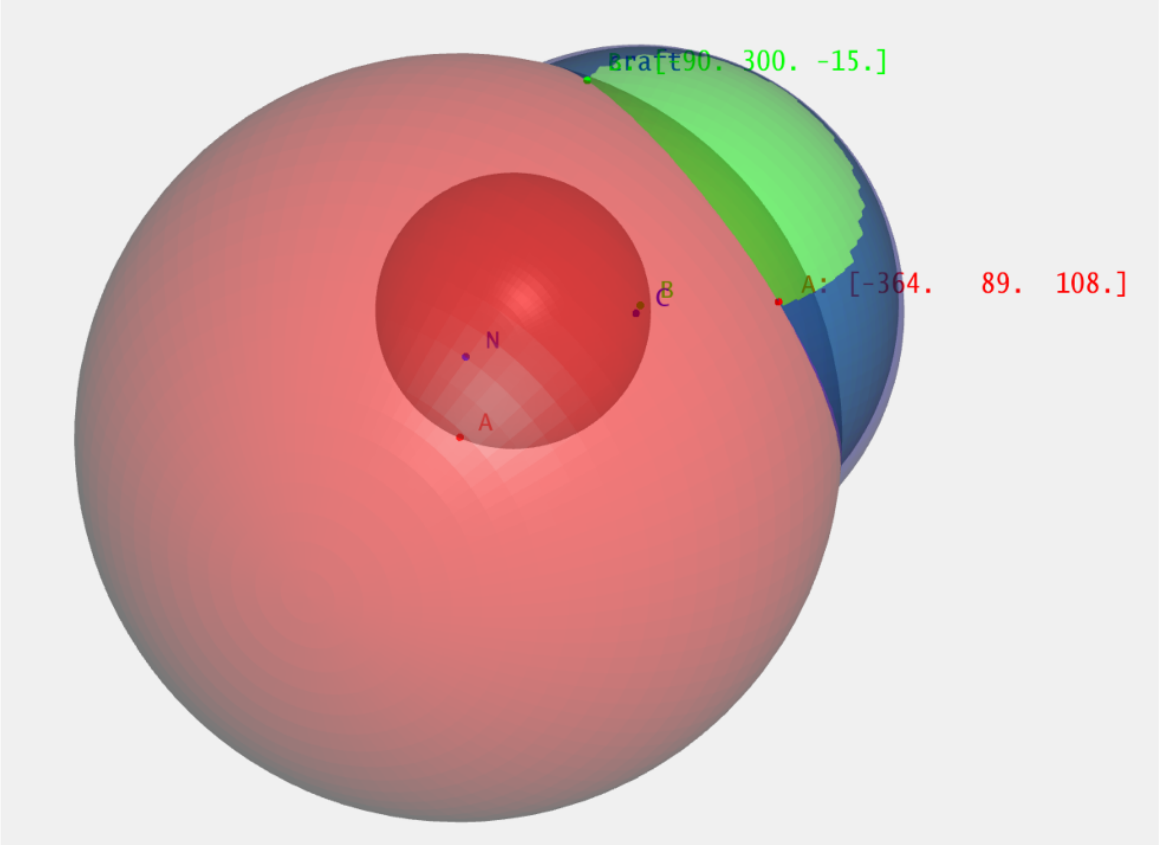

Figure 5.11: Graphic of geometric algebra solution to the position problem. Three spheres intersect at precisely two points: one of which is the correct solution. This approach is highly intuitive, but seemingly not familiar in engineering at large. Note that the intersection of two spheres results in a circle (in geometric algebra, two spheres have an intersection even if they do not meet, similar to how two non-intersecting conics intersect), and the intersection of a circle and another sphere has two solutions: here, the value in green.

of the indispensable Robbins lunar and Martian crater databases [115]. Rather than what might seem to be a straightforward assessment of objective facts, Robbins et al. explain [124] that crater identification is largely subjective, and agreement between discipline titans is not nearly as guaranteed as it might seem. Indeed, they find that dispersions (here, the expert's standard deviation from the ensamble opinion) among expert opinion varied from 13% for smaller craters (roughly 10 pixel diameters) to 32% for craters ten times that size. Some of the larger craters range from 30-40% dispersions!

With this in mind, several attempts at a more consistent approach to crater localization have been undertaken [115], [125] to minimize this variation, but bear in mind that such uncertainty is inherent in these methods, and, in turn, any approaches that use the data from that source (e.g., this work).

5.3.1 Ellipse parameter study

We first assess the accuracy of the quadric method by adding noise to various parameters: the axes, the center point, and the angle. We explore the effect that noise has on each parameter family and determine sensitivity to signal-to-noise ratios (SNR). This is done using a series of Monte Carlo runs varying the SNR for each parameter.

The first parameter we change will be the size of the semi-major and semi-minor axes. Figure 5.12 shows that a small amount that is barely noticeable at the full image level could place the camera a few hundred kilometers away from the actual position, though the attitude estimate is largely unaffected. Figure 5.13 shows just such an effect. The results show that the algorithm is largely susceptible to noise in just two of the five parameters: either axis of the ellipse. This is unsurprising since a change in either of the axes comes from a significant change in position. The semimajor axis causes even more volatility in position estimation, since a change in angle affects the semiminor axis proportionally more than it does for the semimajor axis. This underscores the importance of a highly accurate ellipse fitting algorithm.

Table 5.1 shows the relationship between error and accuracy, with σ being the Gaussian distributed noise for ellipse fitting. RMS error and standard deviation here are in kilometers. While these ranges exceed those in [20], the range is about 5000 kilometers above the surface (compared to 150 to 600 kilometers) and the algorithm here estimates both attitude and position, whereas [20] uses known inertial attitude in the position estimation.

Figures 5.14 and 5.15 demonstrate the synthetic images used in the current section to ascertain the effects of realistic images and scenarios. Most of the data collected used distances of well over 1000 kilometers from the lunar surface to a distance of nearly 10000 kilometers. The fits shown in Fig. 5.15 result in an error of 5.41 km (or 1.3%) total range error, and about 0.75 degrees of attitude error.

Knowing that localization attempts are susceptible to minute errors in ellipse fitting emphasizes the need for robust methods to minimize the error. This work does not favor

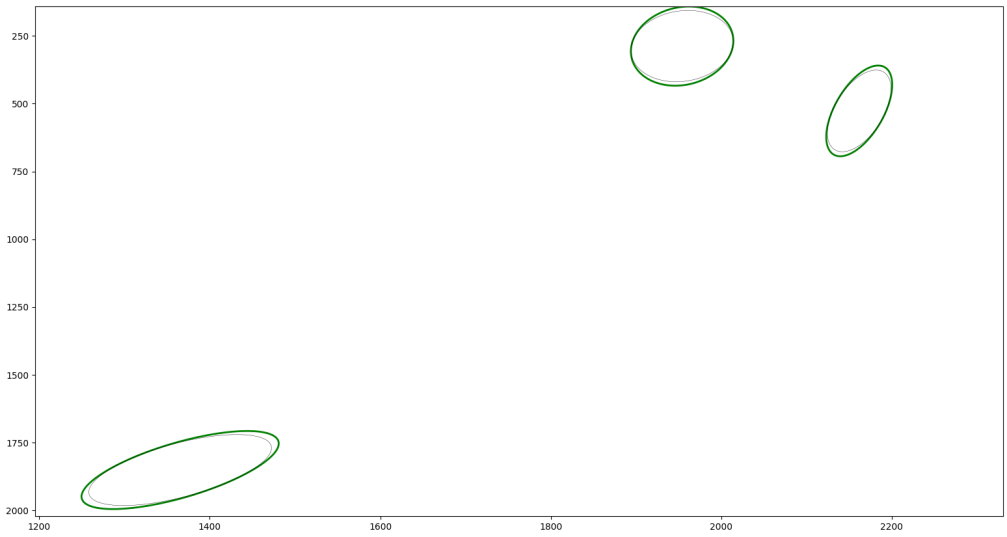

Figure 5.12: **This figure illustrates the change to a conic when the ellipse size increases by 10%. While these changes seem relatively insignificant, they can have drastic effects on the position estimation: the resulting position error from this 10% increase was nearly 600 kilometers!**

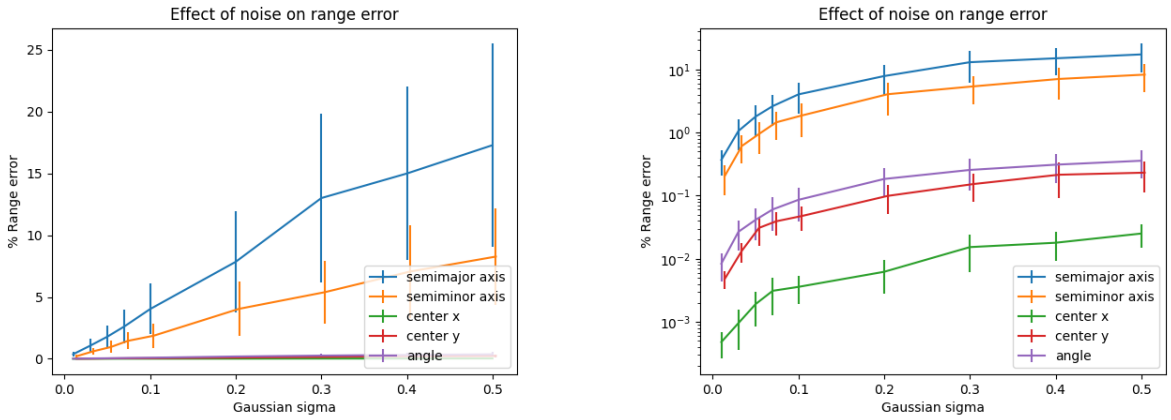

Figure 5.13: **Sensitivity plot of localization on noise on different ellipse parameters on both linear (top) and logarithmic (bottom) y-values. Both plots reflect the same Monte Carlo simulation, with 100 samples per data point. Note the heavy dependence on the ellipse axes but little sensitivity to the other ellipse parameters.**

Table 5.1: Quadric localization error table for varying levels of noise on the ellipse axis parameters since the axes shows the greatest sensitivity to noise. RMS error and standard deviation are given in kilometers, while percentage error shows the relationship between location error and closest distance to the lunar surface.

σ	% Error	RMS Error	StDev
0.5000	0.2865	13.71	6.8288
0.4000	0.2297	11.17	5.8206
0.3000	0.1543	7.55	3.9932
0.2000	0.1107	5.84	3.6123
0.1000	0.0533	2.71	1.5694
0.0900	0.0499	2.55	1.4824
0.0800	0.0501	2.51	1.4030
0.0700	0.0436	2.11	1.0824
0.0600	0.0321	1.59	0.8633
0.0500	0.0302	1.50	0.8162
0.0400	0.0193	0.96	0.5380
0.0300	0.0166	0.87	0.5319
0.0200	0.0105	0.51	0.2731
0.0100	0.0052	0.26	0.1398

one approach over another, only that the chosen method does so consistently and accurately within system constraints.

This method applies to any situation where the circular quadrics' positions and orientations are known. Just like Figure 5.6 shows such a scenario, this technique can be applied to autonomous vehicle navigation, where things like traffic signs, traffic lights, etc. are restricted to specific sizes and spacing, this is an area rich with potential applications. The entire circular feature need not be completely visible, since ellipse fitting can still take place with significant occlusion. Other areas such as robotics in extreme environments such as nuclear plants could feasibly navigate using this technique without the need for direct control, or in other GPS-denied environments.

Figure 5.14: Blender render of the Moon using NASA surface images and displacement maps. The main crater is found in the bottom left, and is known as the Copernicus Impact Crater, a well-studied lunar crater. The craters are outlined in the next figure, but this example and others like it were used to determine the precision of results in realistic situations. The camera here is approximately 5000 kilometers from the lunar surface.

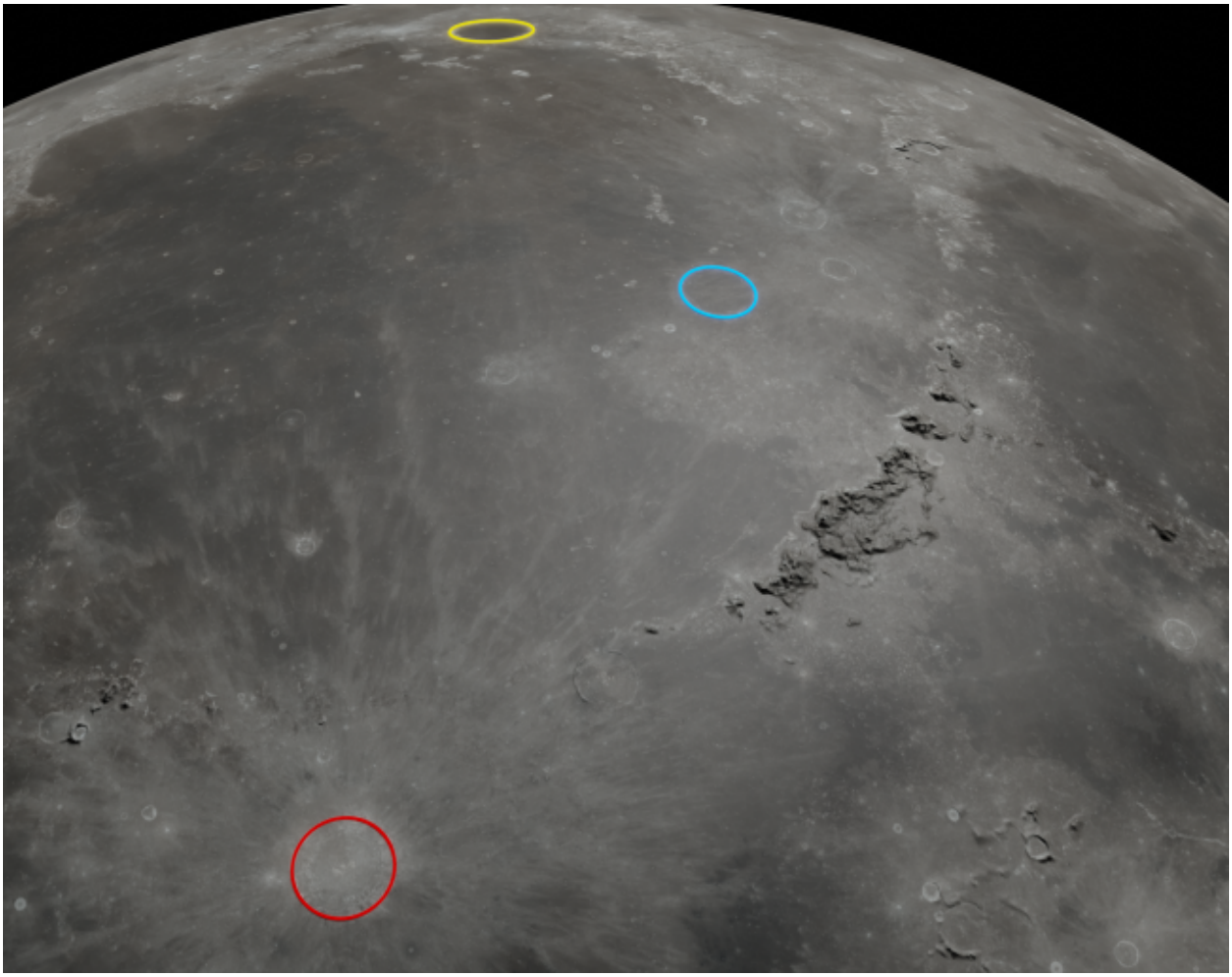

Figure 5.15: Blender render of the Moon with conics highlighting the crater rims. It is from this image that we may apply the full optical navigation approach to first detect then identify craters and work our way to determining the camera pose. A first pass edge detection algorithm can be used for crater detection and refined using subpixel edge localization for the edges of each crater. An ellipse fitting algorithm must then be applied to those points to estimate the ellipse parameters. Using invariant theory or other method, we can identify each of the chosen craters. Once we know the crater locations in the Moon frame using the catalog, we can estimate both the attitude and the position using either of the approaches discussed above.

CHAPTER 6
CONCLUSION

6.1 Putting it all together

While this work has explored largely disparate fields of engineering (image processing, computer databases, and geometric optical navigation), they are linked to a purpose: navigation. While each one is agnostic to what type of application uses it (e.g., medical imaging, robotic localization, object recognition), I would be remiss to revisit weaving them together into a toolkit for such a mission. Indeed, this is and has been the catalyst for investigating such distinct fields of study.

Firstly, let me return to the scenario I laid out immediately before delving into the ideas this work presents—see Figure 6.1. When the flight software commands the camera to capture an image of the surface, we can identify craters using different image processing techniques such as image segmentation. We now want to fit an ellipse to each of these rims, which can be accomplished using the streak localization technique in conjunction with a similar edge refinement technique using Zernike moments [42]. These more accurate rim points translate to better ellipse fits: as I discussed previously, even minute errors hinder identification and can trouble the localization algorithm later. We can then compute the invariants for each crater, either for the non-coplanar case or the coplanar case, using the indirect search technique, which reduces the number of searches required down from three to one. Finally, without any a priori information on attitude or position, we can use our knowledge of the crater positions and sizes to analytically determine our pose using a single snapshot (rather than using stereo vision or multiple images).

Again, this pipeline is only one example of putting them all together in a single package used for a specific purpose. Some users might need only one or two pieces, others might need all parts for an entirely different purpose entirely. For example, you could easily use all three major contributions in autonomous driving. The subpixel localization method determines the ellipse shapes, then—since we know that many objects like signage and stop lights abide by specified sizes and configurations, we could uniquely identify objects like a stop light based on its invariant set by looking up that set in a database. Thereafter the pose estimation algorithm can determine our location relative that stoplight, or to the reference frame if we know which stop light we have identified.

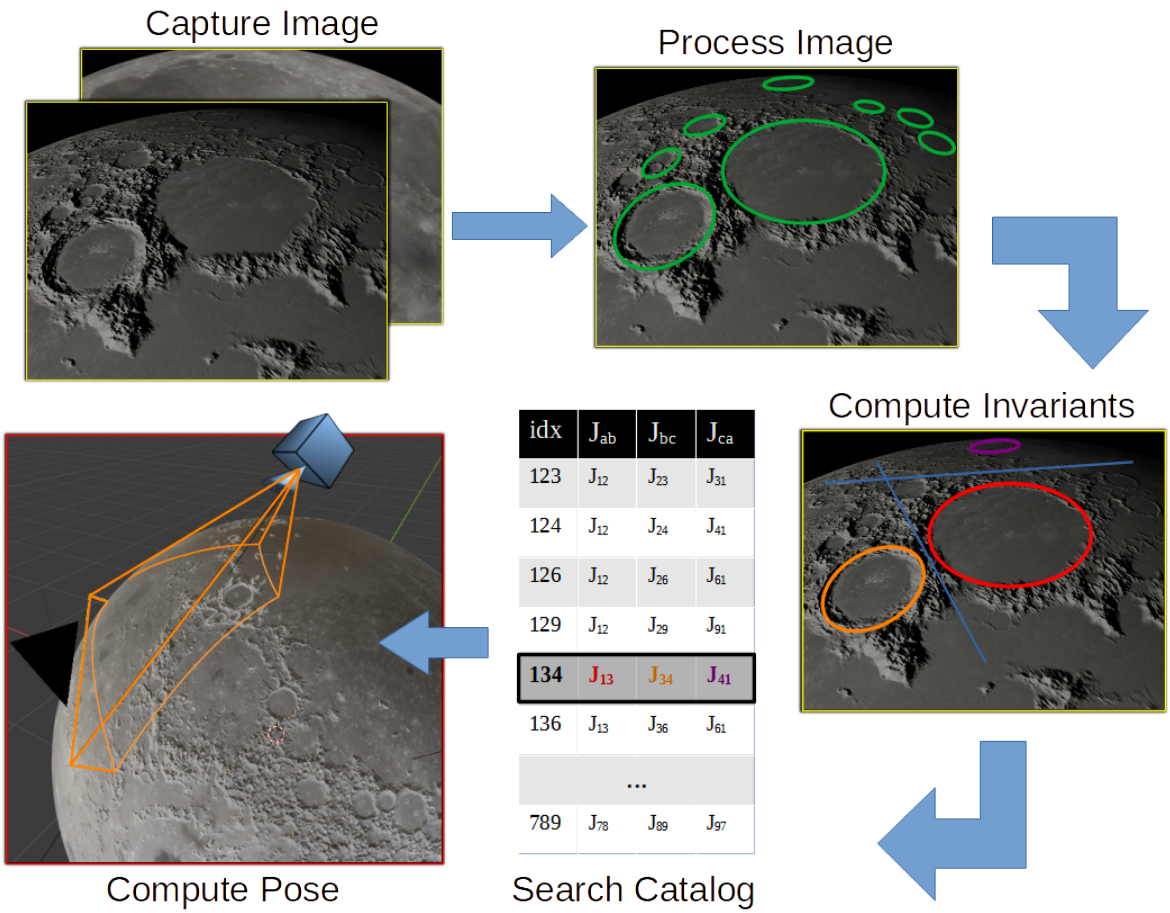

Figure 6.1: Reiteration of flowchart starting from image capture through to pose estimation. Ellipse fitting leads to computing the associated projective invariants, which we search the database for matches. We then have identified the individual craters and can reproject other ellipses in the image for verification. Any of the matched ellipses are then used to compute both attitude and position using either approach in the last chapter. The lunar images in this flowchart were all produced in Blender, where the images were taken from the position shown in the lower left, whose position and attitude we attempt to compute.

6.2 Summary and future work

We have seen throughout this work a myriad of applications targeted specifically at spacecraft navigation, yet the full range of applications can hardly be listed here. Robotics, medicine, and database management are just three branches that can find direct use of the findings here. The notable advancements range from image processing to database storage and search, covering a broad range of needs in distinct communities. Both computation

techniques, streak localization with Zernike moments and quadric-based pose estimation, are analytic and non-iterative; this means that we have greater insight into the geometry of the problem and not rely on iterative numerical methods, though these techniques surely have their place.

6.3 Future work

I am pleased to present this research, and display the value that each idea and implementation presents. While working through these contributions, several extensions to this work became apparent along the way, and I am actively pursuing some of them. Chief among them is the more general case of elliptical craters for localization. While the quadric-based pose estimation with circular or nearly circular craters produced great results, I am generalizing this approach to the more elusive and equally rewarding elliptical case. Solving this would satiate the incessant scientific curiosity on this problem, at least for a time. I have already greatly simplified the path forward by transforming each conic in the triad to the canonical position and rotation, thereby reducing the number of terms in the three sets of projections. I have been doing this for circular objects, and should soon be able to move into the more general elliptical case.

Additionally, adding optimizations to the indirect search and store method to multiple database algorithms has been a priority for some time, and only recently have I began dabbling in generalizing what I initially thought was application-specific (that is, only applicable to the R-tree). The R-tree has many optimizations that reduce the size and search times significantly, such as packed trees, static database (no need for deletion or reshuffling nodes after creation), etc. I also began to see a more general idea of this theory, which meant that it need not be tied to a specific database algorithm. There are many spatial database algorithms available, and I question if R-tree is squarely positioned as the pinnacle of spatial query methods as I had first imagined. Indeed there are many such alternatives that purport to outperform the R-tree under particular circumstances, and it would be invaluable to show the tradeoffs involved in choosing a DBMS in something as valuable as a spacecraft or any other such endeavor.

An interesting point that Seo [76] makes is that the contrast between two sides of a streak need not be of the same intensity, although the model I used and the one presented in that work both simplify this to an assumption that two sides are similar in intensity. But

this simplification, though true for many cases, is incomplete and can introduce bias. In this case, I would have thought about using an edge technique (such as the related Zernike edge technique [42]), but imagining that between the two regions exists a peak or trough that is greater than or less than either side of the discontinuity—this breaks our initial assumptions. Improving this model to account for differing background values on either side of a streak peak or trough might lead to better results, though likely at a cost of complexity.

Finally, throughout this journey I happened across a most satisfying branch of mathematics outside of the realm of vector algebra and calculus: geometric and Clifford algebra. I am struck at the natural elegance of the objects operations and at the intuitive nature that pervades almost everything about it. Take, for instance, the intersection of two spheres. This is obviously a circle when they overlap. Geometric algebra takes those two spheres and, using a "meet" operation, produces that very circle! This area has been largely unexplored, especially for cameras and projective transformations and particularly in aerospace. I have rarely seen such innate beauty and connectedness as I see there.

6.4 Final note

This has been a richly rewarding journey, and I reiterate that I have not nor could not do it alone. From my incredible mentor to my peers, from the faculty to my family, I have had incredible experiences of growth, new perspectives. and learning that are unique to research. Thank you for everything.

REFERENCES

[1] A. M. Smith, "Ptolemy and the foundations of ancient mathematical optics: A source based guided study," *Trans. Am. Philos. Soc.*, vol. 89, no. 3, pp. 1–172, Oct. 1999. DOI: 10.2307/3185879.

[2] H. Damisch and J. Goodman, *L'origine de la Perspective*. Cambridge, MA, USA: MIT Press, 1994.

[3] J. Alexander, *Pappus of Alexandria Book 7 of the Collection : Part 1. Introduction, Text, and Translation.* Vol. 8. London, UK: Springer, 2013.

[4] M. Kemp *et al.*, *The Science of Art: Optical Themes in Western Art from Brunelleschi to Seurat*. New Haven, CT, USA: Yale University Press, 1990.

[5] W. Isaacson, *Leonardo da Vinci: The Biography*. New York, NY, USA: Simon & Schuster, 2018.

[6] Leonardo da Vinci, *Il Cenacolo (The Last Supper)*, 1495.

[7] C. Porco. "Cassini Orbiter Saturn ISSNA/ISSWA 2 EDR Version 1.0." NASA Planetary Data System. (2005), [Online]. Available: pds.nasa.gov/ds-view/pds/viewDataset.jsp?dsid=CO-S-ISSNA%5C%2FISSWA-2-EDR-V1.0.

[8] M. Chasles, *Traité des Sections Coniques*. Paris, France: Gauthier-Villars, 1865.

[9] J. G. Semple, *Algebraic Projective Geometry*. Oxford, UK: Clarendon Press, 1952.

[10] D. P. Rodgers, "Improvements in multiprocessor system design," in *Proc. 12th Annu. Int. Symp. Comput. Architecture*, Boston, Massachusetts, USA, Jun. 1985, pp. 225–231. DOI: 10.1145/327070.327215.

[11] ——, "Improvements in multiprocessor system design," *SIGARCH Comput. Archit. News*, vol. 13, no. 3, pp. 225–231, Jun. 1985. DOI: 10.1145/327070.327215.

[12] B. Maass, S. Woicke, W. Oliveria, B. Razgus, and H. Krüger, "Crater navigation system for autonomous precision landing on the moon," *J. Guid., Control, and Dyn.*, vol. 43, no. 8, pp. 1414–1431, Apr. 2020. DOI: 10.2514/1.G004850.

[13] Y. Cheng and J. Miller, "Autonomous landmark based spacecraft navigation system," in *AAS/AIAA Astrodynamics Specialist Conf.*, Feb. 2003, pp. 1–15. DOI: hdl.handle.net/2014/6432.

[14] Y. Cheng and A. Ansar, "Landmark based position estimation for pinpoint landing on mars," in *Proc. 2005 IEEE Int. Conf. Robot. and Automat.*, Jan. 2006, pp. 1573–1578. DOI: 10.1109/ROBOT.2005.1570338.

[15] Y. Cheng, A. E. Johnson, L. H. Matthies, and C. F. Olson, "Optical landmark detection for spacecraft navigation," in *2003 AAS/AIAA Astrodynamics Specialist Conf.*, Jan. 2003, pp. 1–19. DOI: hdl.handle.net/2014/6431.

[16] W. Park, Y. Jung, H. Bang, and J. Ahn, "Robust crater triangle matching algorithm for planetary landing navigation," *J. Guid., Control, and Dyn.*, vol. 42, no. 2, pp. 402–410, Oct. 2019. DOI: 10.2514/1.G003400.

[17] H. W. Turnbull, *The Theory of Determinants, Matrices, and Invariants*. London, UK: Dover Publications, 1960.

[18] S. S. Koh, *Invariant Theory*. Berlin, Germany: Springer-Verlag, 1987.

[19] J. H. Grace and A. Young, *The Algebra of Invariants*. New York, NY, USA: Chelsea Pub. Co., 1965. DOI: 10.1017/CBO9780511708534.

[20] J. Christian, H. Derksen, and R. Watkins, "Lunar crater identification in digital images," *J. Astronaut. Sci.*, vol. 68, no. 1, pp. 1–89, Oct. 2021. DOI: 10.1007/s40295-021-00287-8.

[21] J. A. Christian and J. L. Crassidis, "Star identification and attitude determination with projective cameras," *IEEE Access*, vol. 9, no. 1, pp. 25 768–25 794, Jan. 2021. DOI: 10.1109/ACCESS.2021.3054836.

[22] A. Zisserman, C. Marinos, D. A. Forsyth, J. L. Mundy, and C. Rothwell, "Relative motion and pose from invariants," in *Proc. Brit. Mach. Vision Conf.*, 1990, pp. 1–6. DOI: 10.5244/C.4.4.

[23] V. S. Tsonis, K. V. Chandrinos, and P. E. Trahanias, "Landmark-based navigation using projective invariants," in *Proc. 1998 IEEE/RSJ Int. Conf. Intell. Robots and Syst. Innov. in Theory, Practice and Appl.*, vol. 1, Oct. 1998, pp. 342–347. DOI: 10.1109/IROS.1998.724642.

[24] D. Heisterkamp and P. Bhattacharya, "Invariants of families of coplanar conics and their applications to object recognition," in *Proc. 13th Int. Conf. Pattern Recognit.*, vol. 1, Aug. 1996, pp. 677–681. DOI: 10.1109/ICPR.1996.546110.

[25] B. Sturmfels, *Solving Systems of Polynomial Equations*. Providence, RI, USA: Amer. Math. Soc., 2002.

[26] D. Cox, J. Little, and D. OShea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Switzerland: Springer Science & Business Media, 2013. DOI: 10.1007/978-3-319-16721-3.

[27] J. A. Christian, L. Hong, P. McKee, R. Christensen, and T. P. Crain, "Image-based lunar terrain relative navigation without a map: Measurements," *J. Spacecr. Rockets*, vol. 58, no. 1, pp. 164–181, Oct. 2021. DOI: 10.2514/1.A34875.

[28] J. A. Christian, "A tutorial on horizon-based optical navigation and attitude determination with space imaging systems," *IEEE Access*, vol. 9, no. 1, pp. 19 819–19 853, Jan. 2021. DOI: 10.1109/ACCESS.2021.3051914.

[29] C. C. Liebe, "Star trackers for attitude determination," *IEEE Aerosp. and Electron. Syst. Mag.*, vol. 10, no. 6, pp. 10–16, Jun. 1995.

[30] C. C. Liebe, "Accuracy performance of star trackers: A tutorial," *IEEE Trans. Aerosp. and Electron. Syst*, vol. 38, no. 2, pp. 587–599, Apr. 2002.

[31] R. M. Haralick, "Ridges and valleys on digital images," *Comput. Vision, Graphics, and Image Process.*, vol. 22, no. 1, pp. 28–38, Apr. 1983. DOI: 10.1016/0734-189X(83)90094-4.

[32] I. Sobel and G. Feldman, "An isotropic 3x3 image gradient operator," *Pattern Classification and Scene Anal.*, pp. 271–272, Feb. 2014.

[33] J. Prewitt, "Object enhancement and extraction," in *Picture Process. and Psychopictorics*, New York, NY, USA, 1970, pp. 75–149.

[34] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. Roy. Soc. London. Ser. B, Biol. Sciences*, vol. 207, no. 1167, pp. 187–217, Feb. 1980.

[35] J. F. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. and Mach. Intell.*, vol. 8, no. 6, pp. 679–698, Nov. 1986.

[36] P. Dollár, Z. Tu, and S. Belongie, "Supervised learning of edges and object boundaries," in *IEEE Conf. Comput. Vision and Pattern Recognit. (CVPR)*, Oct. 2006, pp. 1964–1971. DOI: 10.1109/CVPR.2006.298.

[37] G. Bertasius, J. Shi, and L. Torresani, "Deepedge: A multi-scale bifurcated deep network for top-down contour detection," in *IEEE Conf. Comput. Vision and Pattern Recognit. (CVPR)*, Oct. 2015, pp. 4380–4389. DOI: 10.1109/CVPR.2015.7299067.

[38] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang, "Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection," in *IEEE Conf. Comput. Vision and Pattern Recognit. (CVPR)*, Oct. 2015, pp. 3982–3991. DOI: 10.1109/CVPR.2015.7299024.

[39] S. Xie and Z. Tu, "Holistically-nested edge detection," in *IEEE Int. Conf. Comput. Vision (ICCV)*, Apr. 2015, pp. 1395–1403. DOI: 10.1109/ICCV.2015.164.

[40] Y. Wang, X. Zhao, Y. Li, and K. Huang, "Deep crisp boundaries: From boundaries to higher-level tasks," *IEEE Trans. Image Process.*, vol. 28, no. 3, pp. 1285–1298, Oct. 2018. DOI: 10.1109/TIP.2018.2874279.

[41] H. Blum, "A transformation for extracting new descriptors of shape," in *Models for the Perception of Speech and Visual Form*, Cambridge, MA, USA: MIT Press, 1967, pp. 362–380.

[42] J. A. Christian, "Accurate planetary limb localization for image-based spacecraft navigation," *J. Spacecr. Rockets*, vol. 54, no. 3, pp. 708–730, May 2017. DOI: 10.2514/1.A33692.

[43] E. P. Lyvers, O. R. Mitchell, M. L. Akey, and A. P. Reeves, "Subpixel measurements using a moment-based edge operator," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 12, pp. 1293–1309, Dec. 1989. DOI: 10.1109/34.41367.

[44] S. Ghosal and R. Mehrotra, "Orthogonal moment operators for subpixel edge detection," *Pattern Recognit.*, vol. 26, no. 2, pp. 295–306, Feb. 1993. DOI: 10.1016/0031-3203(93)90038-X.

[45] T. Hermosilla, E. Bermejo, A. Balaguer, and L. Ruiz, "Non-linear fourth-order image interpolation for subpixel edge detection and localization," *Image and Vision Comput.*, vol. 26, no. 9, pp. 1240–1248, Sep. 2008. DOI: 10.1016/j.imavis.2008.02.012.

[46] J. Ye, G. Fu, and U. Poudel, "High-accuracy edge detection with blurred edge model," *Image and Vision Comput.*, vol. 23, pp. 453–467, May 2005. DOI: 10.1016/j.imavis.2004.07.007.

[47] A. Trujillo-Pino, K. Krissian, M. Alemán-Flores, and D. Santana-Cedrés, "Accurate subpixel edge location based on partial area effect," *Image and Vision Comput.*, vol. 31, no. 1, pp. 72–90, Jan. 2013. DOI: 10.1016/j.imavis.2012.10.005.

[48] F. von Zernike, "Beugungstheorie des schneidenver-fahrens und seiner verbesserten form, der phasenkontrastmethode," *Physica*, vol. 1, no. 7, pp. 689–704, May 1934. DOI: 10.1016/S0031-8914(34)80259-5.

[49] A. Khotanzad and Y. H. Hong, "Invariant image recognition by zernike moments," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 5, pp. 489–497, May 1990. DOI: 10.1109/34.55109.

[50] T.-W. Lin and Y.-F. Chou, "A comparative study of zernike moments," in *Proc. IEEE Conf. on Web Intell.*, Oct. 2003, pp. 516–519. DOI: 10.1109/WI.2003.1241255.

[51] C. Teh and R. T. Chin, "On image analysis by the methods of moments," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 10, no. 4, pp. 496–513, Jul. 1988. DOI: 10.1109/34.3913.

[52] W. J. Tango, "The circle polynomials of zernike and their application in optics," *Appl. Phys.*, vol. 13, no. 4, p. 327, Aug. 1977.

[53] S. Peng, W. Su, X. Hu, C. Liu, Y. Wu, and H. Nam, "Subpixel edge detection based on edge gradient directional interpolation and zernike moment," in *Int. Conf. Comput. Sci. and Softw. Eng. (CSSE)*, Oct. 2018, pp. 106–116.

[54] D. T. Renshaw and J. A. Christian, "Subpixel localization of isolated edges and streaks in digital images," *J. Imag.*, vol. 6, no. 5, pp. 1–20, May 2020. DOI: 10.3390/jimaging6050033.

[55] P. Jackson, *The Lord of the Rings: The Fellowship of the Ring*. New Line Home Entertainment, 2001.

[56] Euclid, *The Elements of Euclid*, 2nd ed. New York, NY, USA: Dover Publications Inc., 1956.

[57] T. Sorell, *Descartes: A Very Short Introduction*. New York, NY, USA: Oxford University Press, 2000.

[58] R. Descartes, *La Géométrie de René Descartes*. Paris, France: A. Hermann, 1886.

[59] C. E. Springer, *Geometry and Analysis of Projective Spaces*. New York, NY, USA: Freeman, 1964.

[60] ——, "Geometry and analysis of projective spaces," *The Math. Gazette*, vol. 51, no. 375, pp. 76–76, Feb. 1967. DOI: 10.2307/3613656.

[61] J. L. Mundy and A. Zisserman, *Geometric Invariance in Computer Vision*. Cambridge, MA, USA: MIT Press, 1992.

[62] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. DOI: 10.1145/358669.358692.

[63] L. Quan and Z.-D. Lan, "Linear n-point camera pose determination," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 8, pp. 774–780, Aug. 1999. DOI: 10.1109/34.784291.

[64] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *Int. J. Comput. Vision*, vol. 81, no. 155, pp. 155–166, Feb. 2009. DOI: 10.1007/s11263-008-0152-6.

[65] A. Penate-Sanchez, J. Andrade-Cetto, and F. Moreno-Noguer, "Exhaustive linearization for robust camera pose and focal length estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 10, pp. 2387–2400, Oct. 2013. DOI: 10.1109/TPAMI.2013.36.

[66] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685.

[67] L. Quan, "Conic reconstruction and correspondence from two views," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 2, pp. 151–160, Feb. 1996. DOI: 10.1109/34.481540.

[68] W. Wang, X. Zhang, Y. Qi, and W. Cheng, "An algorithm of pose estimation based on conic correspondences," in *2017 IEEE Int. Conf. on Robot. and Biomimetics (ROBIO)*, Mar. 2018, pp. 314–319. DOI: 10.1109/ROBIO.2017.8324436.

[69] P. K. Mudigonda, C. Jawahar, and P. Narayanan, "Geometric structure computation from conics," in *Indian Conf. on Comp. Vision, Graphics and Image Process. (ICVGIP)*, Dec. 2004, pp. 9–14.

[70] J. Kannala, M. Salo, and J. Heikkilä, "Algorithms for computing a planar homography from conics in correspondence," in *Brit. Machine Vision Conf.*, Sep. 2006, pp. 77–86. DOI: 10.5244/C.20.9.

[71] J. A. Christian, "Optical navigation using iterative horizon reprojection," *J. Guid., Control, and Dyn.*, vol. 39, no. 5, pp. 1092–1103, Jan. 2016.

[72] ——, "Perspective projection of ellipses and ellipsoids with applications to spacecraft navigation," Unpublished, 2017.

[73] M. Born and E. Wolf, *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*. New York, NY, USA: Cambridge University Press, 1997. DOI: 10.1017/CBO9781139644181.

[74] J. Antonello and T. Lequesne, *Zernike*, Accessed Jan 15, 2022, GitHub, 2021. [Online]. Available: github.com/jacopoantonello/zernike.

[75] U. V. G. Rao and V. K. Jain, "Gaussian and exponential approximations of the modulation transfer function," *J. Opt. Soc. Am.*, vol. 57, no. 9, pp. 1159–1160, Sep. 1967. DOI: 10.1364/JOSA.57.001159.

[76] S. Seo, "Subpixel line localization with normalized sums of gradients and location linking with straightness and omni-directionality," *IEEE Access*, vol. 7, no. 1, pp. 180 155–180 167, Jan. 2019. DOI: 10.1109/ACCESS.2019.2959320.

[77] P. V. C. Hough, "Machine analysis of bubble chamber pictures," *Conf. Proc. C*, vol. 590914, pp. 554–558, Sep. 1959.

[78] ——, "Method and means for recognizing complex patterns," U.S. Patent 3 069 654, Dec. 1962.

[79] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. of the ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972. DOI: 10.1145/361237.361242.

[80] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern Recognit.*, vol. 13, no. 2, pp. 111–122, Mar. 1981. DOI: 10.1016/0031-3203(81)90009-1.

[81] K. Zhao, Q. Han, C.-B. Zhang, J. Xu, and M.-M. Cheng, "Deep hough transform for semantic line detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 3, pp. 249–265, Mar. 2020.

[82] Z. Li, Y. Liu, R. Walker, R. Hayward, and J. Zhang, "Towards automatic power line detection for a uav surveillance system using pulse coupled neural filter and an improved hough transform," *Mach. Vision and Appl.*, vol. 21, no. 5, pp. 677–686, Sep. 2010. DOI: 10.1007/s00138-009-0206-y.

[83] S. J. Lee, J. P. Yun, H. Choi, W. Kwon, G. Koo, and S. W. Kim, "Weakly supervised learning with convolutional neural networks for power line localization," in *2017 IEEE Symp. Ser. on Comput. Intell. (SSCI)*, Dec. 2017, pp. 1–8. DOI: 10.1109/SSCI.2017.8285410.

[84] C. Steger, "Subpixel-precise extraction of lines and edges," *Int. Archives of Photogrammetry and Remote Sens.*, vol. 33, no. 3, pp. 141–156, Jan. 2000. DOI: 10.1364/OSAC.402101.

[85] A. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piece-wise threshold probing of a matched filter response," *IEEE Trans. Med. Imag.*, vol. 19, no. 3, pp. 203–210, Mar. 2000. DOI: 10.1109/42.845178.

[86] A. Hoover and M. Goldbaum, "Locating the optic nerve in a retinal image using the fuzzy convergence of the blood vessels," *IEEE Trans. Med. Imag.*, vol. 22, no. 8, pp. 951–958, Aug. 2003. DOI: 10.1109/TMI.2003.815900.

[87] E. S. Agency. "The Hipparcos and Tycho Catalogues," ESA SP–1200. (1997), [Online]. Available: cosmos.esa.int/web/hipparcos/catalogues.

[88] F. Bergamasco, A. Albarelli, and A. Torsello, "Pi-tag: A fast image-space marker design based on projective invariants," *Mach. Vision and Appl.*, vol. 24, no. 6, pp. 1295–1310, Aug. 2013. DOI: 10.1007/s00138-012-0469-6.

[89] A. Guttman, "R-trees:a dynamic index structure for spatial searching," in *ACM Special Interest Group on Manage. of Data (SIGMOD) Rec.*, vol. 14, Jun. 1984, pp. 47–57. DOI: 10.1145/971697.602266.

[90] J. L. Blanco and P. K. Rai, *nanoflann: A C++ header-only fork of FLANN, a library for nearest neighbor (NN) with KD-trees*, Accessed Sep 27, 2021, 2014. [Online]. Available: github.com/jlblancoc/nanoflann.

[91] D. Mortari, M. A. Samaan, C. Bruccoleri, and J. L. Junkins, "The pyramid star identification technique," *Navigation*, vol. 51, no. 3, pp. 171–183, Sep. 2004. DOI: 10.1002/j.2161-4296.2004.tb00349.x.

[92] J. Bonér, *Latency numbers every programmer should know*, Accessed Feb 10, 2022, GitHub, 2012. [Online]. Available: gist.github.com/jboner/2841832.

[93] C. Hanak, T. Crain, and R. Bishop, "Crater identification algorithm for the lost in low lunar orbit scenario," in *AAS Guid. and Control Conf.*, vol. 137, Feb. 2008, pp. 1–27. DOI: hdl.handle.net/2152/ETD-UT-2009-05-126.

[94] F. Hanak, "Lost in low lunar orbit crater pattern detection and identification," Ph.D. dissertation, University of Texas at Austin, Austin, TX, USA, 2009.

[95] I. Kamel and C. Faloutsos, "Hilbert r-tree: An improved r-tree using fractals," in *Proc. 20th Int. Conf. Very Large Data Bases*, San Francisco, CA, USA, Feb. 1994, pp. 500–509. DOI: 10.5555/645920.673001.

[96] ——, "On packing r-trees," in *Proc. 2nd Int. Conf. Inf. and Knowledge Manage.*, Dec. 1993, pp. 490–499. DOI: 10.1145/170088.170403.

[97] N. Roussopoulos and D. Leifker, "Direct spatial search on pictorial databases using packed r-trees," *SIGMOD Rec.*, vol. 14, no. 4, pp. 17–31, May 1985. DOI: 10.1145/971699.318900.

[98] ——, "Direct spatial search on pictorial databases using packed r-trees," in *Proc. 1985 ACM SIGMOD Int. Conf. Manage. of Data*, Austin, Texas, USA, May 1985, pp. 17–31. DOI: 10.1145/318898.318900.

[99] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *Proc. 1990 ACM SIGMOD Int. Conf. Manage. of Data*, Atlantic City, NJ, USA, May 1990, pp. 322–331. DOI: 10.1145/93597.98741.

[100] ——, "The r*-tree: An efficient and robust access method for points and rectangles," *SIGMOD Rec.*, vol. 19, no. 2, pp. 322–331, May 1990. DOI: 10.1145/93605.98741.

[101] H. Samet, "Multidimensional data structures for spatial applications," in *Algorithms and Theory of Computation Handbook: General Concepts and Techniques*, 2nd ed. London, UK: Chapman and Hall//CRC, 2010, p. 6.

[102] S. Leutenegger, M. Lopez, and J. Edgington, "Str: A simple and efficient algorithm for r-tree packing," in *Proc. 13th Int. Conf. Data Eng.*, Apr. 1997, pp. 497–506. DOI: 10.1109/ICDE.1997.582015.

[103] H. Wolfson and I. Rigoutsos, "Geometric hashing: An overview," *IEEE Comput. Sci. Eng.*, vol. 4, no. 4, pp. 10–21, Oct. 1997. DOI: 10.1109/99.641604.

[104] S. J. Maybank, "The projection of two non-coplanar conics," in *Geometric Invariance in Computer Vision*. Cambridge, MA, USA: MIT Press, 1992, pp. 105–119.

[105] R. Uribe-Vargas, "On polar duality, lagrange and legendre singularities and stereographic projection to quadrics," *Proc. London Math. Soc.*, vol. 87, no. 3, pp. 701–724, Oct. 2003. DOI: 10.1112/S0024611503014230.

[106] M. Ferri, F. Mangili, and G. Viano, "Projective pose estimation of linear and quadratic primitives in monocular computer vision," *CVGIP: Image Understanding*, vol. 58, no. 1, pp. 66–84, Jul. 1993. DOI: 10.1006/ciun.1993.1032.

[107] S. Rahmann, "Reconstruction of quadrics from two polarization views," in *Iberian Conf. Pattern Recognit. and Image Anal.*, Jun. 2003, pp. 810–820. DOI: 10.1007/978-3-540-44871-6_94.

[108] C. A. Rupp, "Stereographic projection of a quadric," *Amer. Math. Monthly*, vol. 35, no. 8, pp. 415–421, Oct. 1928. DOI: 10.1080/00029890.1928.11986866.

[109] R. Safaee-Rad, K. Smith, B. Benhabib, and I. Tchoukanov, "Constraints on quadratic curves under perspective projection," in *Proc. 1991 IEEE Int. Conf. Syst., Man, and Cybern.*, vol. 1, 1991, pp. 57–62. DOI: 10.1109/ICSMC.1991.169661.

[110] B. Sturmfels, "Computational algebraic geometry of projective configurations," *J. Symbolic Comput.*, vol. 11, no. 5-6, pp. 595–618, May 1991. DOI: 10.1016/S0747-7171(08)80121-6.

[111] ——, "What is... a grobner basis?" *Notices Am. Math. Soc.*, vol. 52, no. 10, p. 1199, Jan. 2005.

[112] P. Breiding, S. Kališnik, B. Sturmfels, and M. Weinstein, "Learning algebraic varieties from samples," *Revista Matemática Complutense*, vol. 31, no. 3, pp. 545–593, Aug. 2018. DOI: 10.48550/arXiv.1802.09436.

[113] D. A. Cox, B. Sturmfels, and D. N. Manocha, *Applications of Computational Algebraic Geometry*. Providence, RI, USA: American Mathematical Society, Jan. 1998. DOI: 10.1090/psapm/053.

[114] T. Becker, V. Weispfenning, and H. Kredel, *Gröbner Bases: A Computational Approach to Commutative Algebra*. Berlin, Germany: Springer-Verlag, 1993. DOI: 10.1137/1036089.

[115] S. J. Robbins, "A new global database of lunar impact craters ¿1–2 km: 1. crater locations and sizes, comparisons with published databases, and global analysis," *J. Geophys. Res.: Planets*, vol. 124, no. 4, pp. 871–892, Feb. 2019. DOI: 10.1029/2018JE005592.

[116] R. Safaee-Rad, I. Tchoukanov, K. Smith, and B. Benhabib, "Three-dimensional location estimation of circular features for machine vision," *IEEE Trans. Robot. and Automat.*, vol. 8, no. 5, pp. 624–640, Oct. 1992. DOI: 10.1109/70.163786.

[117] Y. Shiu and S. Ahmad, "3d location of circular and spherical features by monocular model-based vision," in *IEEE Int. Conf. Syst., Man and Cybern.*, vol. 2, Nov. 1989, pp. 576–581. DOI: 10.1109/ICSMC.1989.71362.

[118] K.-I. Kanatani, "Constraints on length and angle," *Comput. Vision Graph. Image Process.*, vol. 41, no. 1, pp. 28–42, Jan. 1988. DOI: 10.1016/0734-189X(88)90115-6.

[119] K. Kanatani and W. Liu, "3d interpretation of conics and orthogonality," *CVGIP: Image Underst.*, vol. 58, no. 3, pp. 286–301, Nov. 1993. DOI: 10.1006/ciun.1993.1043.

[120] G. Salmon, *A Treatise on the Analytic Geometry of Three Dimensions*. Dublin, Ireland: Hodges, Figgis, & Company, 1882.

[121] S. Bancroft, "An algebraic solution of the gps equations," *IEEE Trans. on Aerosp. and Electron. Syst.*, vol. 21, no. 1, pp. 56–59, Jan. 1985. DOI: 10.1109/TAES.1985.310538.

[122] W. Navidi, W. S. Murphy, and W. Hereman, "Statistical methods in surveying by trilateration," *Comput. Statist. and Data Anal.*, vol. 27, no. 2, pp. 209–227, Apr. 1998. DOI: 10.1016/S0167-9473(97)00053-4.

[123] R. Bajaj, S. Ranaweera, and D. P. Agrawal, "Gps: Location-tracking technology," *Comput.*, vol. 35, no. 4, pp. 92–94, Apr. 2002. DOI: 10.1109/MC.2002.993780.

[124] S. J. Robbins, I. Antonenko, M. R. Kirchoff, *et al.*, "The variability of crater identification among expert and community crater analysts," *Icarus*, vol. 234, pp. 109–131, May 2014, ISSN: 0019-1035. DOI: 10.1016/j.icarus.2014.02.022. [Online]. Available: http://dx.doi.org/10.1016/j.icarus.2014.02.022.

[125] S. J. Robbins, J. D. Riggs, B. P. Weaver, *et al.*, "Revised recommended methods for analyzing crater size-frequency distributions," *Meteoritics & Planet. Sci.*, vol. 53, no. 4, pp. 891–931, Jan. 2018. DOI: 10.1111/maps.12990.

[126] D. Brown, "Decentering distortion of lenses," in *Photogramm. Eng.*, vol. 32, May 1966, pp. 444–462.

[127] A. E. Conrady, "Decentred lens-systems," *Monthly Notices of the Roy. Astronomical Soc.*, vol. 79, no. 5, pp. 384–390, Mar. 1919. DOI: 10.1093/mnras/79.5.384.

[128] J. Kollár, "Algebraic hypersurfaces," *Bull. of the Am. Math. Soc.*, vol. 56, no. 4, pp. 543–568, Oct. 2019. DOI: 10.1090/bull/1663.

[129] T. Hanning, R. Schoene, and S. Graf, "A closed form solution for monocular re-projective 3d pose estimation of regular planar patterns," in *2006 Int. Conf. Image Process.*, Feb. 2007, pp. 2197–2200. DOI: 10.1109/ICIP.2006.312976.

[130] C. Doignon, G. Abba, and E. Ostertag, "Recognition and localization of solid objects by a monocular vision system for robotic tasks," in *Proc. IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, vol. 3, Sep. 1994, pp. 2007–2014. DOI: 10.1109/IROS.1994.407588.

[131] T. Shih and W. Faig, "A solution for space resection in closed form," *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. 27, no. 1, pp. 547–556, Mar. 1988.

[132] W. C. Karl, G. C. Verghese, and A. S. Willsky, "Reconstructing ellipsoids from projections," *Graphical Models and Image Process. (CVGIP)*, vol. 56, no. 2, pp. 124–139, Mar. 1994. DOI: 10.1006/cgip.1994.1012.

[133] T. Yamamoto and X. Chen, "An existence and nonexistence theorem for solutions of nonlinear systems and its application to algebraic equations," *J. Comput. and Appl. Math.*, vol. 30, no. 1, pp. 87–97, 1990. DOI: 10.1016/0377-0427(90)90008-N.

[134] S. M. Goldfeld and R. E. Quandt, "Nonlinear simultaneous equations: Estimation and prediction," *Int. Econ. Rev.*, vol. 9, no. 1, pp. 113–136, Feb. 1968. DOI: 10.2307/2525617.

[135] B. Buchberger, "Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal)," Ph.D. dissertation, Universitat Innsbruck, Innsbruck, Austria, 1965.

[136] J.-C. Faugère, "FGb: A Library for Computing Gröbner Bases," in *Math. Softw. - ICMS 2010*, vol. 6327, Kobe, Japan, Sep. 2010, pp. 84–87. DOI: 10.1007/978-3-642-15582-6_17.

APPENDIX A
CAMERA MODELING

Imaging systems like the camera are a major focus within this work; thus, some introduction to such systems is essential. First we start with likely the simplest model: the pinhole camera. We will slowly augment this model with additional real-world effects, ranging from lens effects to distortions and aberrations. Nevertheless, this model provides beneficial insight into imaging systems. The ideal pinhole camera model permits a simple, intuitive understanding of camera mechanisms while still representing the basic of a wide array of various camera configurations. See [28] for more information on how other cameras may be represented as a pinhole model.

A.1 Pinhole camera model

For cameras with simple lenses, we may consider the effect of a perfect lens as a refraction of rays as seen in Figure A.1. The light, which we assume to be collimated (that is, parallel rays), passes through the lens from a given direction and focuses at a particular point on the focal plane at a distance f from the lens. Since all of these directional rays each focus on a unique location on the focal plane, we consolidate all of the rays from a given direction as one single ray. Since we can choose which particular ray represents its cohort, let us choose the ray that passes through the optical center, which we denote as the chief ray. This ray, as seen in Figure A.1, effectively experiences no diffraction along its path, which will simplify the mathematics and lend greater intuition.

All of the chief rays now pass through a single point in the lens: the optical center. Since all rays pass through this single point and those rays are undeterred from their original paths, let us shrink the lens down to that single point along its center axis. It is as if light only passes through this single point with no lens. This, in essence, is the pinhole camera model.

This approximation abstracts away the lens and other effects. Pinhole cameras do exist and are often used to show the inner workings of cameras; in fact, they are relatively simple to make as a home project.

Despite the benefits bestowed by this model, we lose some important information along the way. Gone are the real-world effects like lens aberrations and distortions. For example,

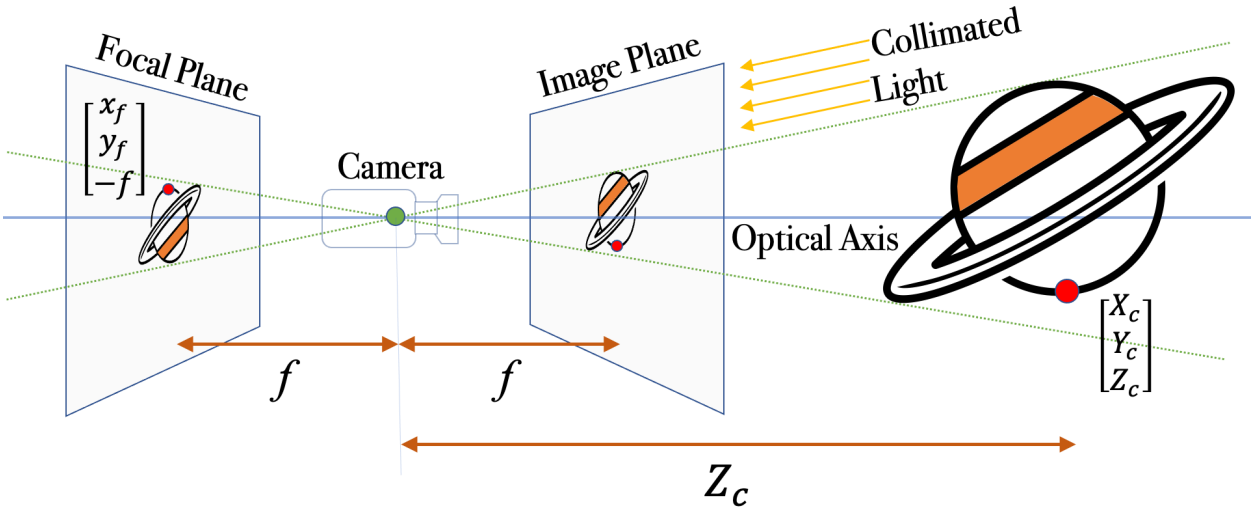

Figure A.1: Pinhole camera model. All light crossing the tiny aperture passes undistorted on to the detector array on the left.

the pinhole camera is not an adequate model when using wide-angle lenses, especially when the field of view exceeds 180 degrees. The mapping between a point in front of the camera and a point behind the lens along the path are expressed in the same way. Nevertheless, this model provides an adequate framework for most of the work in spacecraft navigation (star trackers typically have a field of view of only a handful of degrees or less).

A.2 A distorted view of the world

The pinhole camera neglects effects like distortions, which are important physical phenomenon to consider (think: fish-eye lenses). Distortions range from lens effects to misalignment to chromatic aberration. Misalignment and certain lens distortions dominate the discussion here, but this does not imply that other such effects are less important to consider.

The purpose of the lens is to focus light in a particular way at a particular point or plane. Considering an ideal camera, such as that in Figure A.2, we see that a perfect lens placed at the correct distance would cause only a single pixel to light up since all the light from that ray focuses to that pixel.

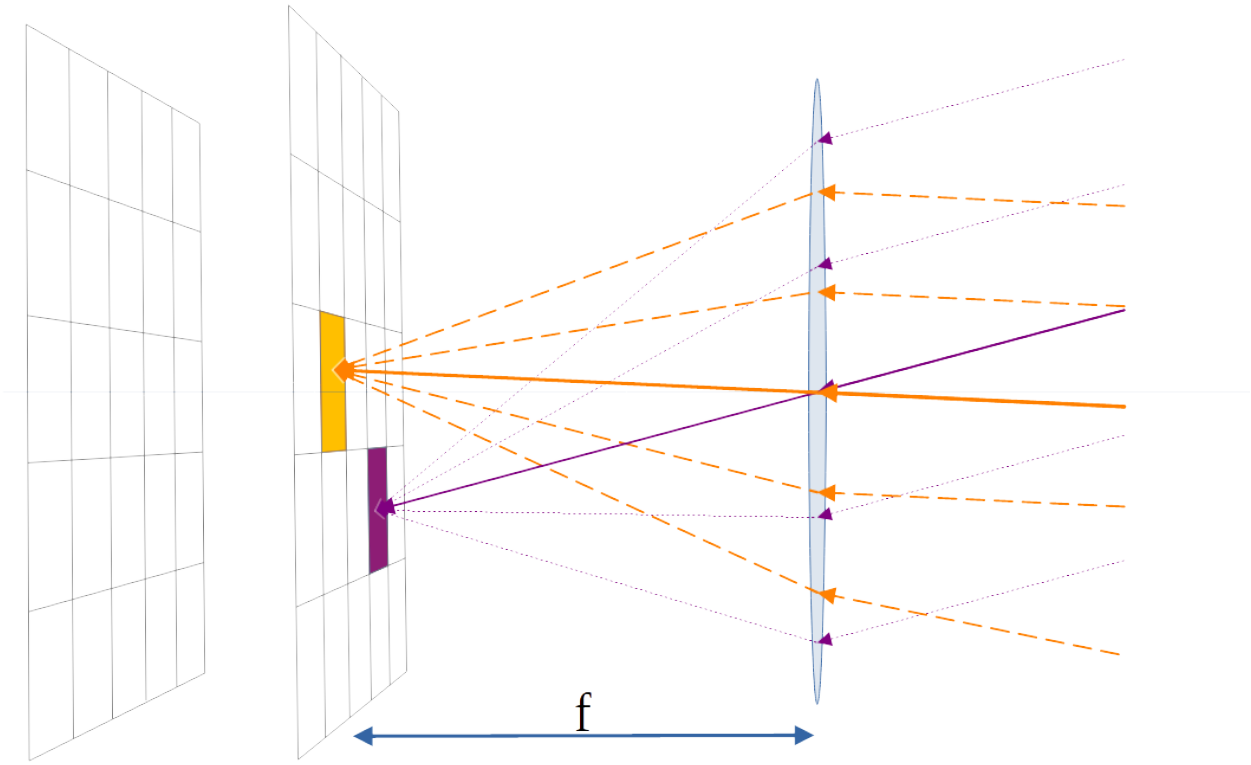

Figure A.2: Ideal camera lens. All light from a given direction (assumed as collimated light) focuses at a particular point on the focal plane at a distance f from the lens, as shown by the orange and purple lines and the lit portion of the focal plane array.

If, however, the focal detector array were not precisely at a distance f from the lens, we would start to see those rays diverge and impinge on nearby pixels, as seen in Figure A.3, where the focal detector is the grid to the left of the focal plane. This effect, known as defocus distortion, is often undesirable; certain applications, however, use this phenomenon to pinpoint the centroid of a small object better than a perfectly crisp image would allow. For example, star trackers exploit defocus when imaging stars, which allows a much more precise location of the star centroid than a single lit pixel could provide (see Figure A.4).

Apart from this displacement, the focal detector array might be rotated in any of the axes, which produces a skew in the resulting image and produces exaggerated perspective effects.

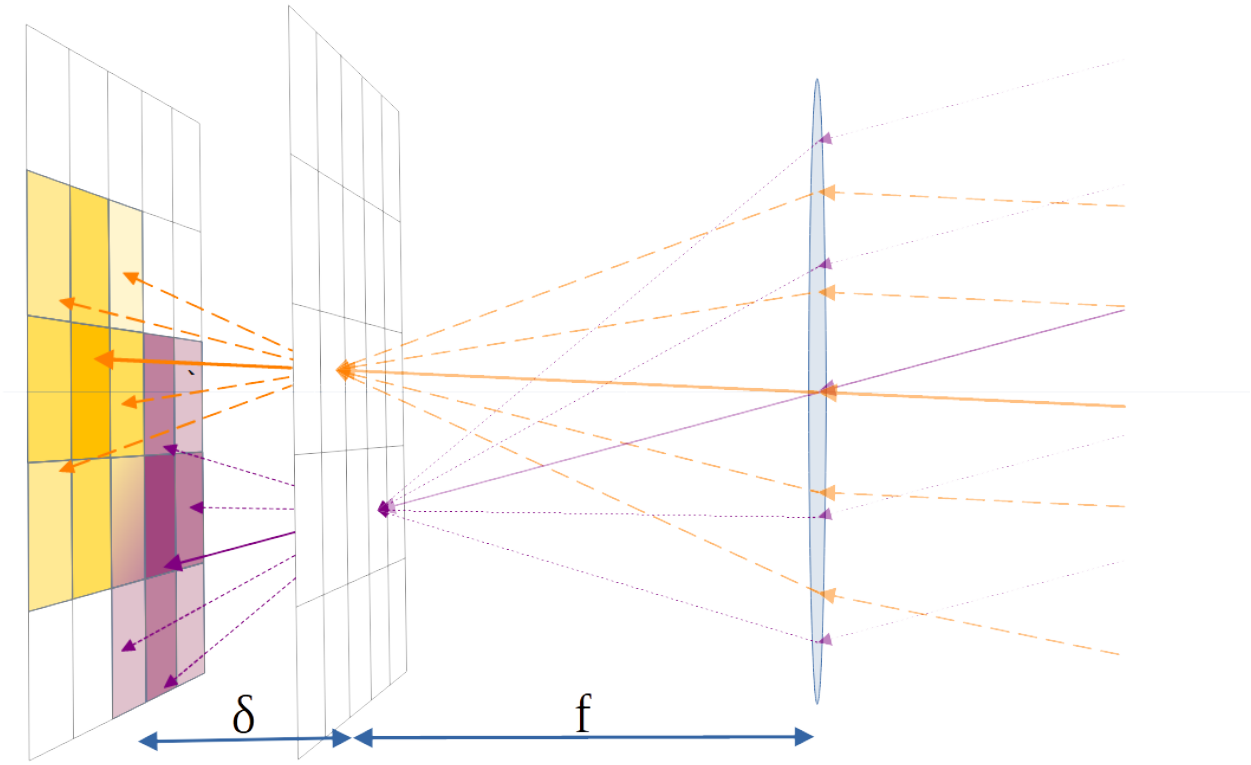

Figure A.3: Camera detector misalignment. All light previously focused on a single pixel now spread to a local neighborhood, thereby diffusing the intensity of the single pixel to adjacent cells.

In addition to detector installation, the lens introduces some effects of its own. Lens imperfections result in aberrations affecting the way that light bends on its way to the detector. Some effects, like radial distortion, bend light near the edges in a different way than near the center of the image, which warps straight lines near the border of the image into curved lines. These effects are known as *barrel* (if the lines curve away from the center) or *pincushion* (curving towards the center). There can also be a hybrid of the two.

In addition to radial distortion, tangential distortion also occurs. This is associated with misalignment of the detector array. Both tangential and radial distortion effects are so common that the popular Brown [126] (or Brown-Conrady [127]) distortion model approximates both with a series of coefficients and powers that represent the type and amount of distortion. The

Figure A.4: Example raw star image taken in upstate New York on a Pixelink camera. The single points of light are hot pixels, or pixels whose values are stuck at a certain value (usually 255 for 8-bit images). The center blob is a real star; note the gradual intensity change, which is the defocus effect spreading the intensity of this single star to a few dozen nearby pixels.

purpose of camera calibration (at least using Brown's model) is to estimate these coefficients up to a certain degree (usually two to three terms can be sufficient for many applications).

APPENDIX B
PROJECTIVE GEOMETRY AND INVARIANTS

We start at the most restrictive level of the geometry hierarchy: Euclidean geometry. Known for millennia simply as geometry (they never imagined needing anything else!), basic axioms established the behavior of points, lines, and angles in 2D and 3D. Euclid's *Elements* [56] is a foundational work in mathematics: a compilation of existing knowledge into a strong framework used for millennia afterwards. Certain additions followed that aided in analytic treatment within the space, such as the coordinate system often attributed to Descartes (though coordinate systems predated him by millennia) [57], [58], allowing algebraic treatment of geometric properties. Certain transformations, known as groups, preserve particular properties of objects despite these objects undergoing a transformation. These constant properties, known as invariants, provide valuable, unchanging information about the object.

Translation and rotation comprise Euclidean transformation group, and the invariants are lengths and areas [21]. We can see that despite undergoing both transformations, the polygons' sides in the upper left of Figure B.1 sides remain the same lengths, and the enclosed area remains exactly the same [9]. Geometers often label this group $E(n)$, where n denotes the number of dimensions of the space.

Let's consider scaled versions of the polygons in the upper right Figure B.1. It is clear that areas and lengths have changed, but corresponding angles remain the same. Additionally, the ratio of the areas remains constant. These transformation is the *similarity* transform, often denoted $S(n)$.

We now move up the hierarchy into affine transformations, which we denote $A(n)$. Here, general transformations preserve parallel lines and incident lines. Area ratios and the ratios of lengths of parallel lines are also maintained, but we lose angle invariance as objects like rectangles might morph into a parallelogram or vice versa under an affine transformation, as seen in the lower left of Figure B.1.

We arrive at the base of the hierarchy and the least restricted transformation: the projective transformation group, denoted $PGL(n+1)$ (projective general linear group). Here, we lose all concept of parallel lines and invariant areas and volumes, as seen in the bottom right of Figure B.1. At this point the question arises: What is preserved under such a transformation?

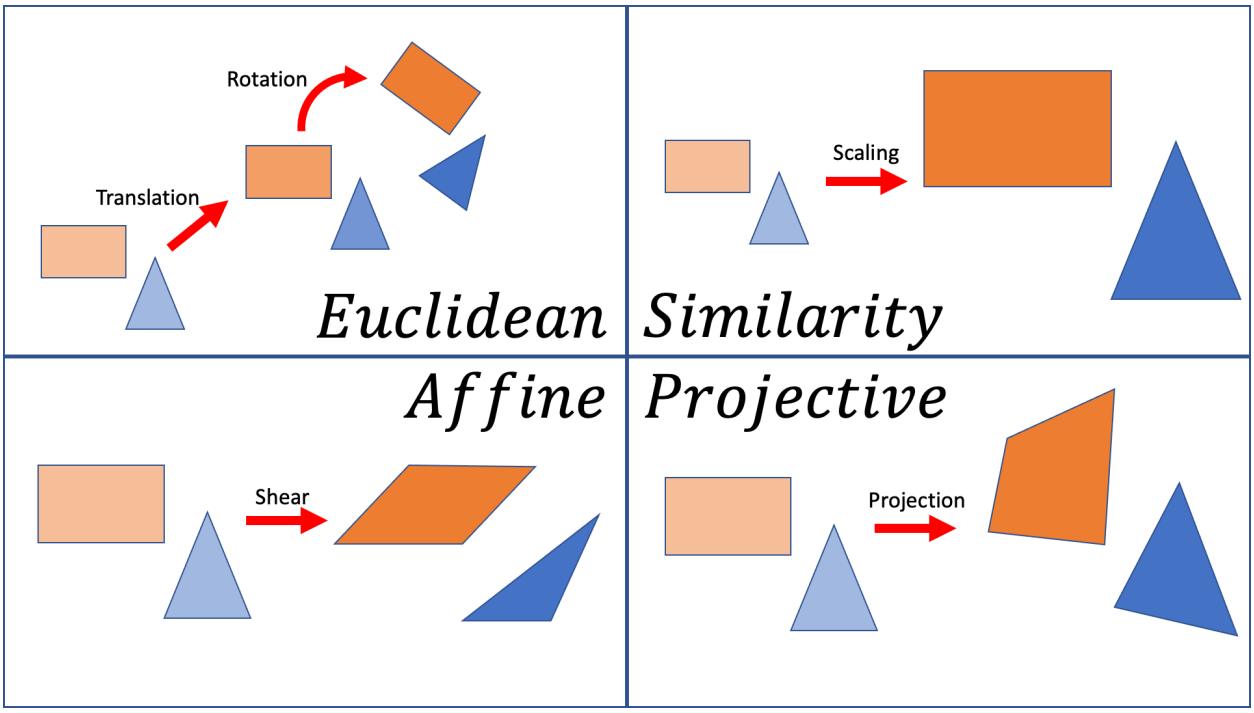

Figure B.1: The projective geometry transformations. Illustrates the groups of transformations allowed in each group. Every group inherits allowable transformations from those above it in the hierarchy.

B.1 Commutative algebra

Commutative algebra is a field of mathematics dealing with varieties, groups, and ideals [114], [26]. In a nutshell, you can imagine varieties as objects, groups as transformations that act on varieties [25], [112]. This is a gross simplification, but for our purposes this explains a good deal without too much mathematical jargon.

B.1.1 Fields, rings, and ideals

The discussion in Chapter 5 relies heavily on invariant geometry and invariant algebra, and some terms might be unfamiliar to the reader. Thus we first introduce several terms that may or may not be commonly used within the engineering domain. Varieties, groups, rings, and fields are just a handful of ideas within commutative algebra that might be outside the experience of many engineers and scientists. This section briefly describes these ideas, starting with the basic concept of fields.

A field is a fundamental concept in algebra—it is simply any number set (real or complex) together with a set of operations—$\times$, $+$, $-$, $\div$ (though more strictly only $+$ and

$\times$)—that, when any two members combine with an operator, the resultant is also a member of that set (except the division by zero) [17], [26]. The natural numbers, $\mathbb{N}$, clearly do not constitute a field, since $1/3$ does not result in a natural number. Rational numbers, however, do form a field, as well as complex numbers.

We've already introduced an idea of groups, but here we refine the idea into a more concrete concept. A group is a set of numbers G together with a binary operation ($\diamond$) and an element e of G that satisfy the following axioms [17], [114]:

i. Associativity: $a \diamond (b \diamond c) = (a \diamond b) \diamond c \quad \forall \quad a, b, c \in G$

ii. Identity element: $e \diamond a = a \quad \forall \quad a \in G$

iii. Inverse element: $\exists \quad z \in G$ such that $a \diamond z = e$

Furthermore, if the group also satisfies a commutative property (i.e., $a \diamond b = b \diamond a$), the group is called an Abelian group [114]. The natural numbers—$\mathbb{N}$—form an Abelian group with the multiplication operator, and the integers—$\mathbb{Z}$—form an Abelian group with the addition and multiplication operators.

Rings are similar to fields and groups, with a requirement to satisfy both $+$ and $\times$ (addition and multiplication, respectively) and a neutral element 0 [114]. Additionally, distributive laws are maintained—that is, $a \cdot (b + c) = a \cdot b + a \cdot c$. If multiplication within the ring is commutative, then the ring is called a commutative ring.

Section 2.2 discusses the action of groups on varieties, and solving nonlinear systems of equations employ Gröbner bases and the Hilbert Nullstellensatz (or *theorem on locations of zeros*, discussed later), which rely heavily on understanding varieties. What *is* a variety, then? An algebraic variety is the common zeros of the polynomials $f_1, \ldots, f_r \in \mathbb{R}[x_1, \ldots, x_n]$ [112]. Those familiar with the terms manifold, model, shape or space may be pleased to know that varieties convey the same idea, but without some of the accompanying connotations (e.g., a manifold is smooth, but a variety has no such restriction). A simpler working definition here might be that it is a geometrical description of an object (e.g., an ellipsoid).

While working with polynomials, we break apart the constituent elements in a polynomial that are separated by addition. These elements are known as monomials, and are of the form [113]

$$x_1^{a_1} \cdot x_2^{a_2} \ldots x_{n-1}^{a_n - 1} \cdot x_n^{a_n}. \tag{B.1}$$

where the degree of the monomial is $\sum_{i=1}^{n} a_i \ \ \forall \ \ a_i \geq 0$.

Polynomials are simply linear combinations of monomials, and the sum and product of polynomials are also polynomials. Polynomials also satisfy all of the axioms of a field except the inverse multiplier element; they do, however, satisfy the requirements for a ring, and we refer to such structures as polynomial rings [26].

Since polynomials can be formed as combinations of other polynomials, we want to find the minimum set of polynomials that comprise the current set of polynomials. We can think of this as the greatest common divisor, or `gcd` of two natural numbers. This set is known as the *ideal*. (Normally, this is done using the Euclidean algorithm [114], but the coming discussion in Chapter 5 shows that the Euclidean algorithm is simply a subset of the Gröbner bases.) Given a set of polynomials F, a collection of polynomials generate the ideal [113] (a special subset of F), given

$$\langle f_1, \ldots, f_m \rangle = \{\Sigma_{i=1}^{m} h_i f_i \mid h_i \in k[x_1, \ldots, x_n]\} \tag{B.2}$$

Any additional polynomials formed by a combination of the polynomials in F shares the ideal. Formally, an ideal satisfies the following properties of a ring R and its ideal I [114]:

i. $a + b \in I \ \ \forall \ \ a, b \in I$

ii. $ar \in I \ \ \forall \ \ a \in I$ and $r \in R$

This discussion continues in Chapter 5, where the idea of exploiting nonlinear systems of polynomials is expounded and the pose estimation problem is introduced.

B.2 The pose problem

The question that the quadric pose problem attempts to answer is: Can we solve for attitude and position simultaneously using only knowledge about those quadrics?

The problem poses several obstacles, the most important being nonlinearity. The pose estimation problem is quadratic in P (which represents the rotation and translation of the camera), stemming from the action of a projective camera on a quadric surface given by Eq. B.3 [66]

$$C^* \propto PQ^*P^T \tag{B.3}$$

What we have is the inverse of the problem: we have the conic C^* corresponding to the projection of the quadric Q^*, and we wish to backproject [66], [108], [128], [106] this conic onto the quadric disk. Several articles discuss solving this problem using conic correspondences [68], [129], [105], [130], [22], [131], but these methods are intended for coplanar conics. One particularly discusses the backprojection of a conic onto an ellipsoid [132], but ellipsoids have full rank and our disk quadric has rank 3, and deals only with reconstructing the ellipse and not with pose estimation. Others work with multiple images [66], [107]. We may wish to simplify this equation to eliminate unnecessary variables within the computation, leading to the conic-quadric pair i equivalence

$$C_i^* \propto (KT)Q_i^*(KT)^T = KTQ_i^*T^TK^T \Rightarrow K^{-1}C_i^*K^{-T} \propto TQ_i^*T^T \tag{B.4}$$

since all values of K and C_i^* are known. Similarly for craters j and k

$$C_j^* \propto (KT)Q_j^*(KT)^T = KTQ_j^*T^TK^T \Rightarrow K^{-1}C_j^*K^{-T} \propto TQ_j^*T^T \tag{B.5}$$

$$C_k^* \propto (KT)Q_k^*(KT)^T = KTQ_k^*T^TK^T \Rightarrow K^{-1}C_k^*K^{-T} \propto TQ_k^*T^T \tag{B.6}$$

What remains is a system of quadratic equations in T as a function of Q^*.

While the concept of linear algebra is widely used in engineering, a system of nonlinear equations poses a significant problem [133]. Often, students linearize a system around a set of equilibrium points and solve using this linearized set. Other times iterative solutions or optimization approaches are the preferred approach. Maybe we employ elimination theory? Some applications may not be amenable to any of these approaches. When such methods fail, we may look to the Gröbner bases [110], [111]. While Gröbner bases were not part of the final solution, their insights into the problem coalesced into a more tractable problem, and I think it is important to explain a little more about this method which is likely to provide an alternative solution to the problem I solved in Chapter 5.

B.3 Gröbner bases

Solving systems of equations, linear and nonlinear, is a standard operation for engineers and scientists. While systems of linear equations often have neat, clear solutions, nonlinear equations are, at the very least, troublesome [134]. Compounding this problem is that most

real-world systems are nonlinear. How we solve these systems depends on the system. We might think of ways to remove the nonlinearity, such as linearizing about specific points. Perhaps elimination theory can remove a few polynomials. Or maybe the polynomials fit a very precise structure that is amenable to a known solution. When these simplifying methods fail, it may spell doom for the application. Furthermore, solutions may not even exist. How can we find solutions if we don't even know that they exist?

Building on the particular geometric and algebraic definitions from B.1.1, we now motivate our use of the Gröbner basis. While the name might be unfamiliar to most engineers and mathematicians, elementary linear algebra uses a linear version of Gröbner bases when solving a system of linear equations, known as Gaussian elimination. In addition to generalizing two familiar solutions—Gaussian elimination and the Euclidean algorithm—the process also performs the same operation as the simplex algorithm in linear programming.

Apart from generalizing the three algorithms above, the bases also show whether or not a solution exists (the polynomials share common zeros), *how many* solutions exist, and what the solutions are. The following example demonstrates the power of Gröbner bases.

Assume that we have a system of univariate polynomials F, given by

$$f_1 : x^2 + xy = 10 \tag{B.7}$$

$$f_2 : x^3 + xy^2 = 26 \tag{B.8}$$

$$f_3 : x^4 + xy^3 = 70 \tag{B.9}$$

and must find the solutions satisfying the equations. To compute the zeros, set each function to zero as follows:

$$f_1 : x^2 + xy - 10 = 0 \tag{B.10}$$

$$f_2 : x^3 + xy^2 - 26 = 0 \tag{B.11}$$

$$f_3 : x^4 + xy^3 - 70 = 0 \tag{B.12}$$

Using the Gröbner bases, the ideals generating this set are $(x - 2)$ and $(y - 3)$; we also know that only one solution exists, which is that $x = 2$ and $y = 3$.

Computing the Gröbner bases relies on the Buchberger Algorithm [135]; Buchberger, who proposed the concept of Gröbner bases, named the bases after his advisor, Wolfgang

Gröbner. The most efficient algorithms today for computing these bases are `F4` amd `F5`, developed by Faugére [136]. When a set of polynomials do not share common zeros, then the result is {1}, meaning no solution exists. Section B.3 will develop this idea further and apply it to the problem of pose estimation.

B.4 Applying Gröbner basis

The system of polynomials formed by Eq. B.4 is quadratic in the terms of P, which has 6 degrees of freedom (3 for rotation, 3 for translation). Note that since C_i^* is symmetric, meaning that each set of functions formed by Equations B.4 and B.5 each have 6 independent polynomials. We must determine whether the solutions exist before moving forward with solving this set. The Gröbner basis method applies to arbitrary systems of polynomials [25], yet its generality is often undesirable. This generality is apparent when computing the bases for only a subset of our system of equations, namely Eq. B.4

What we notice is that just for the first quadric and conic pair, the runtime is on the order of tens of minutes. Obviously, this approach is unacceptable on real-time systems, but it does provide the answer to whether the solution even exists, and we obtain a notion of the number of possible solutions. For the quadric-conic pairs, each had at least one solution, potentially many solutions. The trickier part is to determine the solutions. The Gröbner basis functions for the 6 polynomials formed from Eq. B.4 form 49 polynomials; each of these equations is also of higher degree than any of the quadratic equations formed originally, but each of these equations separates the variables more than the intertwined versions formed by Eq. B.4. The path forward is solving for the zeros of the equations to determine both the number of solutions and the solutions themselves. Several promising avenues are discussed in [114], [113], [25], including fewnomials, Berstein's Theorem, and polytopes.

While ultimately I set this approach aside for another using backprojection of an elliptical cone, Gröbner bases continue to be a viable contender as an alternative. Indeed, they provided some much needed insight into the problem, especially in the more generally elliptical case, which remains to be solved analytically.