NEW APPROACHES TO RECONSTRUCTING GEOMETRIC MODELS FROM NOISY MEASUREMENTS

JIEQI YU

A DISSERTATION PRESENTED TO THE FACULTY OF PRINCETON UNIVERSITY IN CANDIDACY FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE BY THE DEPARTMENT OF ELECTRICAL ENGINEERING ADVISERS: SANJEEV R. KULKARNI AND H. VINCENT POOR

SEPTEMBER 2012
Abstract

This dissertation is comprised of four different studies in the areas of shape/surface/trajectory reconstruction based on point cloud data (potentially with time stamps) and sensor localization (with measurements of distances among neighboring sensors) in a noisy environment on a distributed system. A noise resistant ellipse/spheroid fitting algorithm is discussed first, with an innovative objective function that provides more accurate axial direction estimation in noisy environments. This new objective function is combined with an efficient iterative algorithm with a correction term so that it can obtain accurate axial estimation as well as accurate fitting of the size of the ellipse/spheroid. Secondly, to better deal with outliers in ellipse fitting, and more generally, in curve and surface fitting, a hybrid outlier detection algorithm is proposed, combining both proximity-based and model-based outlier detection techniques. This hybrid technique can effectively eliminate outliers of various types, and considerably improve the robustness of ellipse/spheroid fitting for scenarios with large portions of outliers and high levels of inlier noise. Thirdly, the shape reconstruction is generalized to shape-trajectory reconstruction of rigid bodies, from distributively collected, asynchronous point cloud data with time stamps. An energy-minimization scheme is first proposed to solve the trajectory reconstruction problem of rigid bodies with known shape parameters, assuming that the rigid body moves in an energy efficient manner, with an acceleration upper limit. Then, this method is generalized to the case with unknown rigid body shape parameters, employing cross-validation techniques to determine the best parameter hypothesis. Finally, a series of techniques to improve the spring-model-based sensor localization algorithm are proposed, including dimension expansion, which solves a 2-D sensor localization problem in 3-D space to reduce the chance of “folding” phenomena, an $L_p$ spring potential function that generalizes quadratic potential of Hooke spring to arbitrary power functions, and a customized spring force with “lock-in” mode that provides a compromise between incremental sensor localization and concurrent sensor localization to achieve rapid convergence.
Acknowledgements

This dissertation was made possible by the support of both my advisors, Professor Sanjeev R. Kulkarni and Professor H. Vincent Poor. I am very grateful for their generous, but unimposing guidance, which offered me freedom to explore many interesting research areas, including robust shape/surface reconstruction, rigid body trajectory reconstruction, spring model sensor localization, and a touch of partial-clustering and its applications in Sybil user detection. Moreover, they had given me a lot of advice and freedom in my course selections, which turned out to be very helpful to my entire research experience in Princeton.

I would like to thank Professor Bede Liu for being a valuable member of my general exam committee, as well as my dissertation reader, for his many enlightening advices and discussions.

I am also grateful for Ms. Beth Jarvie and Ms. Dorothy Coakley who patiently helped me with many conference, travel and publication issues.

Moreover, I would like to show my gratitude to Sarah McGovern, who helped me overcome many small but essential issues of my life as an Electrical Engineering graduate student.

I owe my deepest gratitude to my parents, to whom this dissertation is dedicated. Being an accountant and an engineer themselves, they instilled the values and power of knowledge and science to me so that I could eventually embark on this memorable journey in another continent, 8000 miles and 12 hours distant from my hometown.

Finally, this dissertation, as well as my entire life, became truly meaningful when I met my husband, Haipeng Zheng. His knowledge, wit, discipline and companionship are the things I value the most. Without his support and encouragement, I would not have had the patience and strength to go through so many highs and lows in my life.
To my parents, Yuewen Zhang and Qingfeng Yu,
and my husband, Haipeng Zheng.
Contents

Abstract .............................................................................................................. iii
Acknowledgements ........................................................................................... iv

1 Introduction .......................................................................................... 1
    1.1 Robust Ellipse and Spheroid Fitting ......................................................... 2
    1.2 Two-Stage Outlier Elimination for Robust Curve and Surface Fitting .... 3
    1.3 Trajectory Reconstruction for Rigid Bodies .............................................. 3
    1.4 Improved Distributed Spring Model Algorithms for Sensor Localization .... 4

2 Robust Ellipse and Spheroid Fitting ............................................................ 6
    2.1 Review of Previous Methods ................................................................. 6
    2.2 A New Geometric Objective Function .................................................... 8
    2.3 Modified Algorithm for Ellipse Fitting: Weighted Objective Function ....... 12
    2.4 Spheroid Fitting .................................................................................... 16
    2.5 Experimental Results ........................................................................... 17
        2.5.1 Comparison of the proposed algorithms .......................................... 17
        2.5.2 Comparison with Algebraic Fitting, OLSF, and ML ....................... 19
        2.5.3 Spheroid Fitting ............................................................................ 19
    2.6 Conclusion ............................................................................................ 21

3 Two-Stage Outlier Elimination for Robust Curve and Surface Fitting ....... 22
    3.1 Background .......................................................................................... 22
    3.2 Model and Assumptions ......................................................................... 25
    3.3 Proximity-Based Algorithm .................................................................... 26
        3.3.1 Graph-Component Based Outlier Detection .................................. 27
        3.3.2 Eigenspace-based Algorithm ......................................................... 32
4.8.1 A more efficient solution for the rod and ellipse problems ....... 72
4.8.2 Introducing Noise ..................................................... 73
4.8.3 Sequential Algorithms .............................................. 73
4.9 Conclusion .................................................................. 73

5 Improved Distributed Spring Model Algorithms for Sensor Localization ... 74

5.1 Review of Previous Methods ........................................ 74
5.2 Spring Model with Anchors .......................................... 77
  5.2.1 Basic Model .......................................................... 77
  5.2.2 Distributed Algorithm Solving the Spring Model for Sensor Localization ... 77
  5.2.3 Introducing Anchors: Long-range Anchor Scenario .......... 78
5.3 Dimension Expansion ................................................ 80
  5.3.1 “Folding” Phenomenon and Dimension Expansion .......... 80
  5.3.2 Efficacy of Dimension Expansion ................................ 81
5.4 Optimal Spring Potential ............................................ 82
5.5 Customizing the Potential for Faster Convergence .............. 86
  5.5.1 Spring with “Lock-in” Mode ..................................... 87
  5.5.2 Numerical Caveats ................................................... 88
  5.5.3 Simulation Results .................................................. 88
5.6 Conclusion .................................................................. 90
List of Figures

2.1 Typical ellipse fitting result based on gradient descent on the objective function (2.5).
Left: a successful ellipse fitting result when noise is relatively small. Right: estimated foci slip away along the major axis when noise is large. ................................. 9

2.2 Derivation of the expected contribution of a noisy observation to the objective function of OLSF. ................................................................. 10

2.3 Derivation of the expected contribution of a noisy observation to our objective function: the change of $g_{z_i}$ after an infinitesimal displacement $\Delta y$. ......................... 12

2.4 Derivation of the expected contribution of a noisy observation to our objective function: the change of $g_{z_i}$ after an infinitesimal displacement $\Delta x$. ......................... 12

2.5 Ellipse fitting result by optimizing the original objective function (2.5) and the objective function (2.8). ................................................................. 13

2.6 Value of the objective function (2.8) in each iteration until convergence when optimizing according to the original objective function, weighted objective function with $\beta$ linearly increasing, and weighted objective function with $\beta$ increase according to a step function. ................................................................. 13

2.7 Accuracy of the algorithms: average error rate under a wide range of noise levels for the four proposed algorithms. The lower bound and upper bound of the error bars are the 20% and 80% quantiles of 50 trials. ................................................. 18

2.8 Two typical sequences of the weighting coefficients $\beta$ for different levels of noise, with $\sigma^2 = 0.2$ and $\sigma^2 = 0.5$. ................................................................. 18

2.9 Average error rate under a wide range of noise levels for algebraic fitting, OLSF, FNS and our method. ................................................................. 20

2.10 Average error rate for algebraic fitting, OLSF and our method with 20% quantile and 80% quantile error bars. ................................................................. 20

2.11 Spheroid fitting result. ................................................................. 20
2.12 Average error rate under a wide range of noise levels for spheroid fitting using weighted objective with adaptive $\beta$. The lower bound and upper bound of the error bars are the 20% and 80% quantiles of 50 trials.

3.1 Two different types of outliers. Proximity-based methods are good at detecting outliers similar to point A, while model-based methods can better detect outliers similar to point B.

3.2 $k$-NN algorithm failure example.

3.3 Different choices of connection radius lead to different results. The figure on the left depicts the situation in which the radius is so small that even the inliers do not connect sufficiently with each other. The figure on the right illustrates the case in which the radius is so large that almost all the observations are connected with each other, and thus makes it impossible to distinguish inliers from outliers. The figure in the middle shows that with a properly selected radius of connection, the inliers form a large connected component, leaving most of the outliers unconnected.

3.4 Outlier detection results for ellipse fitting. The circled observations are classified as outliers by the algorithm. The figure on the left demonstrates the detected outliers after the proximity-based algorithm, in which there are several missed outliers close to the ellipse, while a few inliers are misclassified as outliers. The figure on the right shows that these mistakes are corrected by the model-based algorithm.

3.5 Comparison between ellipse fitting algorithms with and without outlier detection using the hybrid detection algorithm.

3.6 Comparison of performance (measured in relative area difference) between RANSAC and the hybrid algorithm, given the maximum of number of iterations set to 1000.

3.7 Typical fitting results for the RANSAC-EIS-Metropolis Algorithm. Left: a successful fit in 500 iterations, with inlier noise level $\sigma_0 = 0.02$. Right: an unsuccessful fit in 500 iterations, with inlier noise level $\sigma_0 = 0.08$.

3.8 The performance of our two-stage algorithm under a variety of outlier noise levels. Our algorithm performs consistently well for outliers with very different magnitudes.
3.9 Outlier detection results for ellipsoid fitting. Similarly to the ellipse fitting case, the
proximity-based algorithm eliminates most of the distant outliers, yet misclassifies
some inliers as outliers, with several close-by outliers missed, as illustrated on the
left figure. These errors are corrected by the second stage model-based algorithm, as
shown in the right-hand figure. ................................................................. 43

4.1 Shape and trajectory reconstruction problem based on point cloud data (points on
the boundary of the ellipse). ................................................................. 46

4.2 Illustration of the trajectory recovery problem. ........................................ 47

4.3 Trajectory recovery problem for a disk. .................................................... 48

4.4 Trajectory recovery problem for a rod. ..................................................... 51

4.5 Left: the direction of the center velocity is perpendicular to the orientation of the rod,
in which case \( S_{\theta,v} = L \). Middle: the direction of the center velocity is the same as the
orientation of the rod, in which case \( S_{\theta,v} = W \). Right: illustration of the calculation
of \( S_{\theta,v} \), where \( S_{\theta,v} = W \cos \alpha + L \sin \alpha \). ........................................ 52

4.6 Left: A sensor collecting a surface observation. Right: the trajectory of the center of
the disk and the observations collected by the four sensors. ........................ 65

4.7 Trajectory recovery result for a disk with know shape parameter \( r \) on synthetic data
generated by a dynamical model. ............................................................. 65

4.8 The trajectory of the rod and the observations collected at either end of the rod with
equal probability. .............................................................................. 67

4.9 Simulated annealing result for rod trajectory recovery problem. Black dots denote
the observations; blue lines represent the true rod positions at sampling times, while
the red lines represent the recovered rod positions. Left: entire trajectory recovery
result. Right: enlarged result at the beginning of the trajectory. ................. 67

4.10 Left: A sensor collecting a surface observation. Right: Trajectory of the ellipse and
observations collected by the four sensors. ...................................... 69

4.11 Simulated annealing result for ellipse trajectory recovery problem. Black stars de-
note the observations; green ellipses represent the true ellipse positions at sampling
times, while the red ellipses represent the reconstructed ellipse positions. Left: entire
trajectory recovery result. Right: enlarged result at the middle of the trajectory. 69

4.12 The trajectory of the center of the rigid body and the observations collected by the
four sensors. .............................................................................. 70
4.13 Trajectory reconstruction for different values of \( r \) ........................................ 71
4.14 The relative position of observations in the validation set with respect to the reconstructed center of the rigid body. ................................................................. 72

5.1 A case of four long-range anchors with large communication radii. ............... 79
5.2 Illustration of the folding phenomenon. Left: correct displacement of the sensors, global minimum; middle: folding phenomenon, system reaches local minimum - forces within the plane cannot unfold the system; right: the system unfolds easily with the introduction of the third dimension. ................................................................. 80
5.3 Average localization errors (of 100 trials) for 2-D and 3-D algorithms. .............. 81
5.4 Comparison of convergence speeds for 2-D and 3-D algorithms. ......................... 82
5.5 Setting and resulting \( \text{dist}_2(x_S) \) of a particular run of experiment, where the range measurements are corrupted by additive Guassian noise: \( \xi_i \sim \mathcal{N}(0, 0.2), i = 1, ..., 10 \). Ten anchors are used and 10,000 rounds of simulations are run to generate \( \text{dist}_2(x_S) \). Left: anchor positions and color plot of “hits” received by each grid point; Right: 3-D mesh plot of the left hand graph, representing \( \text{dist}_2(x_S) \) .................................................... 85
5.6 Comparison of \( \Phi_p(r) \) for distributions \( \text{dist}_p(x_S) \) with various \( p \) values, where the x-axis represents the radius \( r \). Left: \( \Phi_p(r) \) under Gaussian noise; Right: \( \Phi_p(r) \) under Laplacian noise. ................................................................. 85
5.7 Comparison of quantiles for distributions \( \text{dist}_p(x_S) \), where the x-axis represents the \( p \) values and the y-axis shows the radius \( r \) of the circle. 20\%, 50\%, 80\% and 90\% quantiles of distributions with various \( p \) values are compared. Left: quantile comparison under Gaussian noise; Right: quantile comparison under Laplacian noise. ............... 86
5.8 An example potential function with customized strength. ................................. 88
5.9 Comparison of convergence rates. Left: range measurements with no noise; Right: range measurements corrupted with Gaussian noise with zero mean and \( \sigma = 0.1 \) ........................................ 89
5.10 Comparison of integrated convergence rate under Gaussian noise with standard deviation \( 0.00, 0.05 \) and \( 0.10 \). ................................................................. 90
List of Tables

3.1 Average number of iterations of RANSAC and the hybrid algorithm for different
percentages of outliers. .................................................. 40
3.2 Failure rates of RANSAC-EIS-Metropolis and the hybrid algorithm. ............. 42
4.1 Initial condition and propelling force for disk trajectory recovery. ............... 64
4.2 Initial condition and propelling force for rod trajectory recovery. ............... 66
4.3 Initial condition and propelling force for ellipse trajectory recovery. .......... 68
5.1 Comparison of average convergence rates under different noise levels ......... 89
Chapter 1

Introduction

This dissertation is dedicated to reconstructing geometric models from noisy location or range measurements. A geometric model is a general idea: it can be a curve, a surface, a trajectory, or a graph structure. We look for algorithms that are noise resistant (robust to large noise and outliers), efficient in computation (faster convergence) and energy (local in communication). Different chapters of this dissertation emphasizes different aspects of this general idea.

In Chapter 2, we concentrate on shape/surface (ellipse and spheroid) reconstruction, using a noise-resistant objective function that provides accurate axial estimation. In Chapter 3, we propose a hybrid algorithm that combines proximity-based and model-based outlier detection algorithm to eliminate two types of outliers for curve and surface fitting. In Chapter 4, we discuss the problem of reconstructing the trajectory and shape of rigid bodies simultaneously. In Chapter 5, several techniques to improve a spring-model-based sensor localization algorithm (a graph structure reconstruction problem) are discussed with extensive simulation.

Although the challenges change over different problems described above, one common theme is present throughout this dissertation: To use the domain specific knowledge in order to improve the performance of general algorithms. For instance, instead of using general least squares fitting methods to solve ellipse/spheroid fitting problems, we propose a definition-based objective function, combined with a correction term, to achieve better noise-resistance for ellipse/spheroid fitting, especially for high eccentricity cases. Another example is the design of a “lock-in” mode in the spring model algorithm for sensor localization. The vanilla version of the algorithm simply minimizes a quadratic potential in a damping environment in a iterative way until convergence. However, under local communication constraints, there are many local structures among a few sensors that are co-
directly configured long before the global convergence. With this knowledge, we design a spring model with “lock-in” mode so that the local structures are preserved once they are formed, thus improving the convergence speed of the entire system.

The motivation and basic ideas for each chapter are summarized in the following four sections.

1.1 Robust Ellipse and Spheroid Fitting

Curve and surface fitting have a broad range of applications. For example, in computer vision, curves and surfaces are important geometric primitives and shape descriptors. As a result, curve and surface fitting is commonly carried out in applications such as object reconstruction and feature extraction. In related fields such as image processing, pattern recognition and machine learning, it is consequently an important tool.

Ellipse and ellipsoid fitting plays an important role in curve/surface fitting. An ellipse can serve as a geometric prototype in computer vision and pattern recognition, which allows reduction and simplification of data for higher level processing. Ellipse fitting is also useful in applied sciences from observational astronomy to structural geology. Furthermore, an ellipse/ellipsoid can serve as a geometric prototype for a wide range of objects. By obtaining observations from the surface of the vehicle and fitting an ellipsoid to them, the shape and orientation of the vehicle can be well estimated.

Although ellipse and ellipsoid fitting has been extensively researched, fitting in high-noise environment, potentially with outliers, is still a challenging task. Traditional ellipse fitting methods provide accurate estimation of ellipse parameters in low noise environment. However, their performance is compromised when the noise level is high or the ellipse eccentricity is large.

In this dissertation, we first propose an innovative objective function based on the geometric definition of an ellipse/spheroid (a special class of ellipsoid) [55][53]. This objective function has the advantage of providing better accuracy for estimating the direction of the semi-major axis of an ellipse/spheroid. Moreover, we further improve the algorithm by introducing an weighing term, so that the proposed method can be accurate in estimating both axial direction and size of the ellipse/spheroid. Combined with gradient descent, our algorithm performs well in high-noise, and high-eccentricity cases, and outperforms several popular classical methods in accuracy, with reasonable computational cost.

In Chapter 2, we discuss this new objective function and corresponding algorithms and improvements, with some analysis and extensive simulations.
1.2 Two-Stage Outlier Elimination for Robust Curve and Surface Fitting

Designing noise-resistant objective functions and adaptive algorithms can enable our curve and surface fitting methods to withstand more noise and some outliers. However, noise-resistant objective functions may not be sufficient for the fitting algorithms to be robust against a large number of outliers. These outliers can significantly degrade the performance of the shape/surface fitting algorithms.

Therefore, in Chapter 3, an outlier elimination algorithm for curve and surface fitting is proposed [58][56]. This two-stage hybrid algorithm employs a proximity-based outlier detection algorithm, to greatly reduce the number of outliers in the data set, followed by a model-based stage, to take advantage of the knowledge of the existence of a pattern in the point cloud data.

To be more specific, a proximity graph is generated first. Depending on the use of a hard/soft threshold of the connectivity of observations, two algorithms are developed, one graph-component-based and the other eigenspace-based. Second, a model-based algorithm, taking the classification of inliers/outliers of the first stage as its initial state, iteratively refits and retests the observations with respect to the curve/surface model until convergence. These two stages compensate for each other so that outliers of various types can be eliminated with a reasonable amount of computation.

Compared to pure proximity-based and model-based outlier elimination algorithms, this hybrid algorithm considerably improves the robustness of ellipse/ellipsoid fitting for scenarios with a large portion of outliers and high level of inlier noise, as demonstrated by extensive simulations.

Details of this algorithm are described in Chapter 3.

1.3 Trajectory Reconstruction for Rigid Bodies

The methods described in Chapter 2 and Chapter 3 work only for static cases, i.e., we assume that the point cloud data points are sampled from the boundaries/surfaces of a shape that is static in the space. If the shape moves as time elapses, a “tracking” capability has to be introduced into the shape reconstruction problem. To be more specific, we need to step forward from pure curve/surface fitting to developing methods that accomplish shape fitting and trajectory tracking simultaneously. Literature on this topic is rather limited.

In Chapter 4, we propose an innovative method for reconstructing the trajectory of a rigid body in a damped environment from distributively collected, asynchronous data [57]. The specification of
the environment provides us with realistic constrains on the possible trajectories of the rigid body (e.g., no infinite acceleration is allowable in reality). In our discussion, a disk, a rod and an ellipse are considered as the rigid body prototypes and both the situations of known and unknown rigid body shape parameters are investigated.

With known shape parameters, the shape/trajectory recovery problem is modeled as a minimization of energy dissipation under geometric and acceleration constraints. In order to solve this problem, a convex relaxation for the geometric constraint is introduced, and the energy minimization problem is solved using convex optimization techniques in the disk case, and by numerical methods in the rod and ellipse cases.

When shape parameters of the rigid body are unknown, we propose different parameters and fix the shape to reduce the problem to the previous case with know shape parameters. Then we employ cross-validation to determine the best shape parameter hypothesis to choose the best model and shape parameter. Extensive simulations demonstrate the efficacy of the algorithms in Chapter 4.

1.4 Improved Distributed Spring Model Algorithms for Sensor Localization

Localization is an essential tool in many sensor network applications. Over the years, a rich literature has been developed to solve the sensor localization problem from different perspectives. Franceschini et al. characterized them according to four criteria in their survey paper [18]: anchor-based vs. anchor free, incremental vs. concurrent, fine-grained vs. coarse-grained, and centralized vs. distributed. The spring model algorithm discussed is a distributed, concurrent, fine-grained algorithm to solve the sensor localization problems. In Chapter 5, we concentrate on an anchor-based scenario so as to obtain absolute coordinates of the sensors in the network.

The central ideas of Chapter 5 are composed of three techniques to improve the spring model algorithm for solving localization problems in a wireless sensor network with anchors [54].

First, the two-dimensional localization problem is solved in a three-dimensional space. This “dimension expansion” technique can effectively prevent the spring model algorithm from falling into local minima.

Secondly, the Hooke spring force (linear force, or quadratic potential) is generalized to spring with $L_p$ potential function. The optimality of different values of $p$ is compared under different noise environment. Our simulation results show that $p \approx 2$, i.e. Hooke force, is the optimal choice of
spring for Gaussian noise, yet for Laplacian noise, $p \approx 1.3$ outperforms other choices of $L_p$ potential functions.

Thirdly, a customized spring force function with “lock-in” mode, which has larger strength when the estimated distance between two sensors is close to the true length of the spring, is introduced, which accelerates the convergence speed of the algorithm.

All these techniques significantly improve the noise resistance and efficiency of the spring model algorithm. In Chapter 5, the efficacy of these algorithms is demonstrated by multiple simulations, especially in a scenario with anchor points of longer broadcasting radius than other sensors.
Chapter 2

Robust Ellipse and Spheroid Fitting

2.1 Review of Previous Methods

As a classical problem, ellipse fitting has a rich literature. Various algorithms have been proposed from very different perspectives [19]. We start our discussion by formulating the ellipse fitting problem and reviewing several classes of the most important ellipse fitting algorithms.

Let \( \{ z_i = (x_i, y_i) \}_{i=1}^n \) denote a set of observations from the boundary of an ellipse corrupted by noise. The objective of ellipse fitting is to fit an ellipse with unknown parameters \( \alpha \) to these observations, so that the total error \( \mathcal{E}(\{ z_i \}_{i=1}^n, \alpha) \) is minimized. \( \mathcal{E}(\cdot) \), the measure of error, differs for different classes of algorithms.

The most intuitive class of ellipse fitting algorithms is algebraic fitting. An ellipse can be described by \( P(\alpha) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \), with \( \alpha = (A, B, C, D, E, F) \). Algebraic fitting uses algebraic error as a measure of error: \( \mathcal{E}(z_i, \alpha) = Ax_i^2 + Bx_iy_i + Cy_i^2 + Dx_i + Ey_i + F \). The most widely used algorithm in this category was proposed by Fitzgibbon et al. [17]. It is a least squares optimization problem with respect to the algebraic criterion:

\[
\min_{\alpha} \sum_{i=1}^{n} (Ax_i^2 + Bx_iy_i + Cy_i^2 + Dx_i + Ey_i + F)^2 \tag{2.1}
\]

s.t. \( B^2 - 4AC = 1 \).

The constraint \( B^2 - 4AC = 1 \) ensures the problem to be ellipse specific and prevents the parameters
from free scaling. The problem is reduced to a generalized eigenvalue problem, which can be efficiently solved. The algorithm was improved into a numerically stable version by Halir and Flusser [23]. Algebraic fitting has the advantage of low computational complexity. However, the algebraic criterion lacks geometric interpretation, and it is difficult to generalize to three dimensions due to its non-linear constraint.

To overcome the shortcomings of algebraic fitting, Ahn et al. [2] proposed orthogonal least squares fitting (OLSF). OLSF employs the Euclidean distance from the observations to the ellipse as a measure of error, and solves:

\[
\min_\alpha \sum_{i=1}^n \|z_i - z'_i(\alpha)\|^2, \tag{2.2}
\]

where \(z'_i(\alpha)\) is the orthogonal contact point, which is the point on the ellipse that has the shortest distance to \(z_i\). OLSF has a clear geometric interpretation and exhibits high accuracy. Moreover, it can be generalized to the three-dimensional case as shown by Ahn et al. [1]. Unfortunately, OLSF is computationally intensive. It employs the iterative Gauss-Newton algorithm, and at each iteration, the orthogonal contact points have to be found for each observation, which also is performed iteratively.

There are various extensions to OLSF. Angular information is incorporated into the OLSF algorithm in the work of Watson [51], in which the orthogonal geometric distance is replaced by the distance along the known measurement angle. Moreover, instead of the \(l_2\) norm of OLSF, \(l_1\), \(l_\infty\) and \(l_p\) norms have been considered as well by Al-Subaihi and Watson [4], Atieg and Watson [6], and Watson [52].

The third class of algorithms consists of Maximum Likelihood (ML) algorithms, proposed by Chojnacki et al. [13] and Leedan et al. [33]. The key steps of two ML algorithms, the fundamental numerical scheme (FNS) in the work of [13] and the heteroscedastic errors-in-variables scheme discussed in [33], have been proved to be equivalent by Chojnacki et al. [14].

Chojnacki et al. [13] assume that the observations are independent and follow a multivariate normal distribution: \(z_i \sim \mathcal{N}(z'_i, \Lambda_{z_i})\). The ML solution is then reduced to an optimization problem based on Mahalanobis distance:

\[
\min_\alpha \sum_{i=1}^n (z_i - z'_i(\alpha))^T \Lambda_{z_i}^{-1} (z_i - z'_i(\alpha)). \tag{2.3}
\]

The FNS is implemented to solve a variational equation iteratively by solving an eigenvalue problem at each iteration until it converges. The ML algorithms are accurate with moderate computational
cost. However, when the noise is large or the eccentricity of the ellipse is large, the algorithm breaks down, because when any observation is close to the center of the estimated ellipse, one of the matrices in the algorithm has elements that tend to infinity.

All three classes of algorithms described above have their advantages and perform well when the noise level is low. However, they share a common disadvantage of being non-robust to large noise, and/or when the eccentricity of the ellipse is large. In this chapter, we propose an algorithm that is robust to both high noise levels and high ellipse eccentricity. Moreover, our algorithm can be easily generalized to three-dimensions, with competitive accuracy and moderate computational cost.

The structure of this chapter is as follows. In Section 2.2, a new objective function based on the geometric definition of ellipse and its properties are introduced, and it is modified to form a noise resistant ellipse fitting algorithm in Section 2.3. Then a spheroid fitting algorithm is proposed in Section 2.4, followed by simulation results in Section 2.5.

2.2 A New Geometric Objective Function

The new objective function is inspired by the geometric definition of an ellipse. An ellipse is the locus of all points in the plane whose distances to two fixed points (called the foci of the ellipse) add to the same constant. I.e., \( z \) is a point on the ellipse if and only if

\[
\|z - c_1\| + \|z - c_2\| = 2a, \tag{2.4}
\]

where \( \| \cdot \| \) denotes the \( l_2 \) norm, \( c_1 \) and \( c_2 \) are the two foci, and \( a \) is the length of the semi-major axis.

Based on the geometric definition, the deviation of a data point \( z \) from a given ellipse can be quantified as \( \|z - c_1\| + \|z - c_2\| - 2a \). Thus, the ellipse fitting problem can be naturally formulated as an optimization problem with a new geometric objective function:

\[
\min_{c_1, c_2, a} \frac{1}{n} \sum_{i=1}^{n} (\|z_i - c_1\| + \|z_i - c_2\| - 2a)^2, \tag{2.5}
\]

where \( n \) denotes the number of observations.

Rosin [44] analyzed the error measure of a similar objective function, but concentrated only on reducing the curvature bias of the objective without exploiting its advantages or proposing an algorithm, which we discuss next.
The objective function in (2.5) has several advantages. First, it has a clear geometric interpretation: it is the expected squared deviation of the observations from the ellipse. Second, the parameters, $c_1, c_2$ and $a$, are intrinsic parameters of ellipses, which are translation and rotation invariant. To be more specific, if an ellipse translates and rotates, the new set of $c_1$ and $c_2$ can be obtained following the same transformation. This property contrasts with the conic coefficients of the algebraic expression, which can change drastically when the ellipse translates or rotates even by a small amount. Therefore, the performance of our method is not affected by the position or orientation of the ellipse. Finally, the objective function is ellipse specific, so that no extra constraints are needed. As a result, although the objective function (2.5) is non-convex, as are all the objective functions in the traditional methods, it can be readily solved by gradient descent algorithms for a local minimum. The gradients can be expressed as

\[ G_{c_1,x} = \frac{1}{n} \sum_{i=1}^{n} \frac{2(p_i + q_i - 2a)(c_{1,x} - x_i)}{p_i}, \]
\[ G_{c_1,y} = \frac{1}{n} \sum_{i=1}^{n} \frac{2(p_i + q_i - 2a)(c_{1,y} - y_i)}{p_i}, \]
\[ G_{c_2,x} = \frac{1}{n} \sum_{i=1}^{n} \frac{2(p_i + q_i - 2a)(c_{2,x} - x_i)}{q_i}, \]
\[ G_{c_2,y} = \frac{1}{n} \sum_{i=1}^{n} \frac{2(p_i + q_i - 2a)(c_{2,y} - y_i)}{q_i}, \]
\[ G_a = \frac{1}{n} \sum_{i=1}^{n} -4(p_i + q_i - 2a), \]

where $p_i = ||x_i - c_1||$, $q_i = ||x_i - c_2||$, $x_i = (x_i, y_i)$, $c_1 = (c_{1,x}, c_{1,y})$ and $c_2 = (c_{2,x}, c_{2,y})$.

A typical ellipse fitting result based on gradient descent on the objective function (2.5) is displayed in the left hand graph of Figure 2.1, which shows a satisfying result.

![Figure 2.1: Typical ellipse fitting result based on gradient descent on the objective function (2.5). Left: a successful ellipse fitting result when noise is relatively small. Right: estimated foci slip away along the major axis when noise is large.](image-url)
However, the benefit comes with a price. The objective function in (2.5) has a global minimum at infinity. When the foci move away from each other toward infinity and the semi-major axis length tends to infinity, the value of the objective function approaches zero. So when the noise level is high, the local minimum (which we desire in this case) is smeared out by the noise, resulting in the estimated foci slipping away, which is shown in the right hand graph in Figure 2.1.

Although this is an undesirable situation, it also reveals a very important property of the objective function (2.5). As depicted in the right hand graph of Figure 2.1, the foci slip away along the major axis, which indicates that our algorithm provides a noise resistant and accurate major axis orientation estimate.

To analyze this property, we compare (2.5) with the objective function of OLSF. Assuming that the observations $z_i, i = 1, ..., n$ are independent and have a multivariate normal distribution $z_i \sim \mathcal{N}(z_0, \sigma^2 I)$, we analyze $E[f_{z_i}]$, the expected contribution of a noisy observation $z_i$ to the objective functions of (2.2) and (2.5).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig2_2.png}
\caption{Derivation of the expected contribution of a noisy observation to the objective function of OLSF.}
\end{figure}

In the OLSF case, suppose that $z_i = z_0 + \Delta z$, in which $\Delta z = (\Delta x, \Delta y)$, then $\Delta z \sim \mathcal{N}(0, \sigma^2 I)$. Let $ll'$ be a line that is tangent to the ellipse at point $z_0$, as shown in Figure 2.2. We approximate $z_i'$ with $z_i^*$, which is a point on $ll'$ such that $z_i, z_i^*$ is perpendicular to $ll'$. Thus, the expected contribution of $z_i$ to the OLSF objective function can be approximated as follows:

\begin{align*}
E[f_{z_i}] &= E[||z_i - z_i'||^2] \\
&\approx E[||z_i - z_i^*||^2] \\
&= E[(\Delta y \cos \theta - \Delta x \sin \theta)^2] \\
&= E[\Delta y^2 \cos^2 \theta + E[\Delta x^2] \sin^2 \theta] \\
&= \sigma^2 (\cos^2 \theta + \sin^2 \theta) \\
&= \sigma^2.
\end{align*}

10
The second equality in (2.6) is due to the geometric relationship shown in Figure 2.2, and the third equality is due to the fact that $\Delta x$ and $\Delta y$ are uncorrelated. The same result is obtained if $\Delta y$ or $\Delta x$ is pointing in the other direction.

On the other hand, in the case of our objective function, define $g_{z_i} = \|z_i - c_1\| + \|z_i - c_2\| - 2a$. Assume that there is an infinitesimal displacement $\Delta y$ from $z_0$, as shown in Figure 2.3, the resulting approximated change in $g_{z_i}$ is $\Delta y(\sin \theta_{i1} + \sin \theta_{i2})$, which is the sum of the lengths of the two red line segments in Figure 2.3. That is to say,

$$\frac{\partial g_{z_i}}{\partial y} \approx \sin \theta_{i1} + \sin \theta_{i2}.$$  

Similarly, assuming that there is an infinitesimal displacement $\Delta x$ from $z_0$, as shown in Figure 2.4, the resulting approximated change in $g_{z_i}$ is $\Delta x(\cos \theta_{i1} - \cos \theta_{i2})$, which is the difference in the lengths of the two red line segments in Figure 2.4. As a result,

$$\frac{\partial g_{z_i}}{\partial x} \approx \cos \theta_{i1} - \cos \theta_{i2}.$$  

Since

$$\Delta g_{z_i} = \frac{\partial g_{z_i}}{\partial x} \Delta x + \frac{\partial g_{z_i}}{\partial y} \Delta y,$$

the expected contribution of $z_i$ to our objective function can be approximated as follows:

$$E[f_{z_i}] = E[(g_{z_i} - 0)^2] = E[(\Delta g_{z_i})^2] \approx E[(\Delta y(\sin \theta_{i1} + \sin \theta_{i2}) + \Delta x(\cos \theta_{i1} - \cos \theta_{i2}))^2] = 2\sigma^2 + 2\sigma^2(\sin \theta_{i1} \sin \theta_{i2} - \cos \theta_{i1} \cos \theta_{i2}) = 2\sigma^2(1 - \cos(\theta_{i1} + \theta_{i2})) = 2\sigma^2(1 + \cos \zeta_i),$$

where $\zeta_i$ is the angle $\angle c_1 z_i c_2$. The third equality is due to the fact that $\Delta x$ and $\Delta y$ are uncorrelated. The same result is obtained if $\Delta y$ or $\Delta x$ is pointing in the other direction.

According to the analysis (2.6) and (2.7), it is clear that the expected contribution $E[f_{z_i}]$ is homogeneous for all the observations in the case of OLSF. However, $E[f_{z_i}]$ is heterogeneous around the periphery of the ellipse in the case of our objective. Our objective function puts a large weight
on the observations located at two ends of the major axis, with \( \cos \zeta_i = 1 \) being an extreme. As a result, our algorithm provides a highly noise resistant and accurate major axis orientation estimate.

![Figure 2.3: Derivation of the expected contribution of a noisy observation to our objective function: the change of \( g_{zi} \), after an infinitesimal displacement \( \Delta y \).](image)

![Figure 2.4: Derivation of the expected contribution of a noisy observation to our objective function: the change of \( g_{zi} \), after an infinitesimal displacement \( \Delta x \).](image)

In order to take advantage of this noise resistant and accurate major axis orientation estimation while overcoming the problem of having the foci slipping away, we propose a modified algorithm in the next section.

### 2.3 Modified Algorithm for Ellipse Fitting: Weighted Objective Function

The analysis (2.6) and (2.7) in the previous section suggests a way to get an accurate ellipse shape estimation in addition to the robust major axis orientation estimation: namely, by emulating the objective function of OLSF. Therefore, we apply an angle dependent weight \( 1 + \cos \zeta_i \) to the objective function (2.5):

\[
\min_{c_1, c_2, a} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{1 + \cos \zeta_i} \left[ ||x_i - c_1|| + ||x_i - c_2|| - 2a \right]^2.
\] (2.8)

Directly minimizing the objective function (2.8) sometimes leads to undesirable results (shown in Figure 2.5), since the objective function loses the ability to find an accurate ellipse orientation (the benefit) and slips away to a stray local minimum.

Before exploring further modifications, it is worth mentioning that the objective function in (2.8)
could be a good indicator of the accuracy of the ellipse fitting result. The point-dash curve in Figure 2.6 shows the value of the objective function (2.8) at each iteration when we optimize according to the original objective function (2.5). The curve shows that the value of (2.8) decreases rapidly during the first few iterations when the algorithm successfully finds the accurate orientation of the ellipse. The slow increase afterwards indicates that the estimated foci are slipping away along the major axis.

![Figure 2.5: Ellipse fitting result by optimizing the original objective function (2.5) and the objective function (2.8).](image)

![Figure 2.6: Value of the objective function (2.8) in each iteration until convergence when optimizing according to the original objective function, weighted objective function with $\beta$ linearly increasing, and weighted objective function with $\beta$ increase according to a step function.](image)

In order to take advantage of the noise resistant ellipse orientation estimation and obtain an accurate size estimation as well (preventing the foci from slipping away), we propose the following **weighted objective function**:

$$\min_{c_1, c_2, a} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{1 + \beta \cos \zeta_i} \left( ||x_i - c_1|| + ||x_i - c_2|| - 2a \right)^2,$$

where $\beta$ is a tuning parameter which varies from 0 to 1.
When $\beta = 0$, (2.9) is the same as the objective function (2.5), so that we can obtain accurate ellipse orientation estimation. On the other hand, when $\beta = 1$, the angle dependent weight $1 + \cos \zeta_i$ is applied to mimic the objective function of OLSF so as to obtain an accurate size estimation. By varying $\beta$ from 0 to 1, we avoid stray local minima at first, and aim for accuracy in the end.

For instance, two types of very simple methods can be employed to increase $\beta$ from 0 to 1. The first type is to implement a step function, with the first stage optimizing according to (2.5) ($\beta = 0$), and the second stage optimizing according to (2.8) ($\beta = 1$). The intuition is to first insure an accurate orientation estimation, and aim for accurate size afterwards. The second type is to change $\beta$ gradually, with linear increment at each iteration as a typical example. The small $\beta$ value during the first few iterations does not affect the ellipse orientation search much, and the larger $\beta$ in the later iterations prevents the foci from slipping away. Figure 2.6 shows clear improvement of the two methods compared to the original objective function. Again, we use the objective function in (2.8) as an indicator of the fitting accuracy. By varying $\beta$ from 0 to 1, the function value no longer increases during the later iterations. Note that the weighted objective function with linear $\beta$ increment converges faster, which agrees with the fact that it prevents the foci from slipping.

However, these ad-hoc ways of increasing $\beta$ from 0 to 1 are not sufficient. To increase the stability of the algorithm, $\beta$ should be changed slowly enough so that when the gradient descent algorithm is performed, $\beta$ can be assumed to be fixed.

Therefore, we propose an adaptive way to change the value of $\beta$ to achieve this goal. Define $O_0$ as a threshold. If, in two consecutive iterations, the change in the value of the objective function, $\Delta O$, is comparable to $O_0$, then we deem the objective function to be close to convergence so that we could increase $\beta$. Therefore, for $\Delta O \gg O_0$, $\beta$ should be close to 0; for $\Delta O \ll O_0$, $\beta$ should be close to 1. Moreover, we should also change $\beta$ in a way so that it is non-decreasing and the value of $\beta$ changes smoothly.

Based on the requirements above, we propose a scheme for updating the value of $\beta$ as a function of the iteration number $j$. For the $j$-th iteration, Let

$$\hat{\beta} = \rho \beta_{j-1} + (1 - \rho) \left( \frac{2}{\pi} \tan^{-1} \left\{ \frac{O_0}{|\Delta O|} \right\} \right), j \geq 1,$$

then $\beta_j$ can be expressed as:

$$\beta_j = \begin{cases} 
0, & j = 0 \\
\beta_{j-1} & \hat{\beta} \leq \beta_{j-1} \\
\hat{\beta}, & \hat{\beta} > \beta_{j-1} 
\end{cases}$$
Notice that in (2.10), ρ is a smoothing factor, which controls the rate of change of βj. When ρ is close to 1, βj changes slowly, which is good for the stability of the objective function for the gradient descent; on the other hand, when ρ is close to 0, βj changes rapidly, which can potentially accelerate the convergence of the algorithm. As a result, ρ should be chosen so that stability and efficiency are balanced properly.

With this sequence of β adapted to the change of the objective function value, we can take advantage of the accurate orientation estimation when β is close to 0, as well as the more accurate size estimation when β is close to 1. Moreover, this scheme also adapts to the noise level. For low noise cases, we can reach the threshold O₀ rapidly and switch to a high β value in very few iterations. For high noise cases, the algorithm can spend more time in the low β regime, for better ellipse orientation estimation, and switch to high β at proper time. Supporting simulation results will be provided in Section 2.5.

Assuming β is fixed during gradient descent, the gradient of the objective function can be expressed as follows:

\[
G_{c_{1,x}} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{2q_i(p_i + q_i - 2a)^2(c_{1,x} - x_i)}{p_i(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))} - \frac{4q_i(p_i + q_i - 2a)^2(q_i(c_{1,x} - x_i) + \beta p_i(c_{2,x} - x_i))}{(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))^2} \right) \\
G_{c_{1,y}} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{2q_i(p_i + q_i - 2a)^2(c_{1,y} - y_i)}{p_i(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))} - \frac{4q_i(p_i + q_i - 2a)^2(q_i(c_{1,y} - y_i) + \beta p_i(c_{2,y} - y_i))}{(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))^2} \right) \\
G_{c_{2,x}} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{2p_i(p_i + q_i - 2a)^2(c_{2,x} - x_i)}{q_i(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))} - \frac{4p_i(p_i + q_i - 2a)^2(p_i(c_{2,x} - x_i) + \beta q_i(c_{1,x} - x_i))}{(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))^2} \right) \\
G_{c_{2,y}} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{2p_i(p_i + q_i - 2a)^2(c_{2,y} - y_i)}{q_i(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))} - \frac{4p_i(p_i + q_i - 2a)^2(p_i(c_{2,y} - y_i) + \beta q_i(c_{1,y} - y_i))}{(2p_i q_i + \beta(p_i^2 + q_i^2 - r^2))^2} \right)
\]
\[ G_a = \frac{1}{n} \sum_{i=1}^{n} \frac{-8p_iq_i(p_i + q_i - 2a)}{2p_iq_i + \beta(p_i^2 + q_i^2 - r^2)}. \]

Here \( p_i = ||x_i - c_1||, \ q_i = ||x_i - c_2||, \ r = ||c_1 - c_2||, \ x_i = (x_i, y_i), \ c_1 = (c_{1,x}, c_{1,y}) \) and \( c_2 = (c_{2,x}, c_{2,y}). \)

The efficacy of this algorithm is shown by simulation results in Section 2.5.

### 2.4 Spheroid Fitting

We have proposed a noise-resistant ellipse fitting algorithm based on the geometric definition of an ellipse. In this section, we generalize our method to the three-dimensional case.

Unfortunately, general ellipsoids do not have a natural geometric definition similar to that of ellipses. Nonetheless, we can still generalize our algorithm to three-dimensions in the case of a spheroid. A spheroid is defined as a quadric surface obtained by rotating an ellipse about one of its principal axes; in other words, a spheroid is an ellipsoid with two equal semi-diameters.

According to the definition, a spheroid has the same basic geometric property as an ellipse. This suggests that the weighted objective function (2.9) can be applied to spheroid fitting directly, with the only change being that \( z_i, \ c_1, \ c_2 \in \mathbb{R}^3 \). The gradient is very similar to that of the ellipse case, with \( G_{c_1,x}, G_{c_2,x}, G_{c_1,y} \) and \( G_{c_2,y} \) identical to the ellipse case, and

\[
G_{c_{1,z}} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{2q_i(p_i + q_i - 2a)(c_{1,z} - z_i)}{p_i(2p_iq_i + \beta(p_i^2 + q_i^2 - r^2))} \right)
- \frac{4q_i(p_i + q_i - 2a)^2(q_i(c_{1,z} - z_i) + \beta p_i(c_{2,z} - z_i))}{(2p_iq_i + \beta(p_i^2 + q_i^2 - r^2))^2}
\]
\[
+ \frac{4q_i(p_i + q_i - 2a)(c_{1,z} - z_i)}{2p_iq_i + \beta(p_i^2 + q_i^2 - r^2)} \right),
\]  
\[
G_{c_{2,z}} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{2p_i(p_i + q_i - 2a)^2(c_{2,z} - z_i)}{q_i(2p_iq_i + \beta(p_i^2 + q_i^2 - r^2))} \right)
- \frac{4p_i(p_i + q_i - 2a)^2(p_i(c_{2,z} - z_i) + \beta q_i(c_{1,z} - z_i))}{(2p_iq_i + \beta(p_i^2 + q_i^2 - r^2))^2}
\]
\[+ \frac{4p_i(p_i + q_i - 2a)(c_{2,z} - z_i)}{2p_iq_i + \beta(p_i^2 + q_i^2 - r^2)} \right).
\]

We will demonstrate the spheroid fitting results at the end of the next section.
2.5 Experimental Results

To demonstrate the efficacy of the algorithms proposed above, we describe a series of experiments in different settings. Synthetic data has been used for the simulations.

A set of points on the perimeter of the ellipse were drawn according to a uniform distribution in angle. The observations were obtained by corrupting the true points with independent and identically distributed (i.i.d.) additive Gaussian noise, with mean $0$ and covariance $\sigma^2 I$. The error rate is defined as the normalized area of the symmetric difference between the true ellipse $E_t$ and the fitted ellipse $E_f$:

$$\text{error rate} = \frac{S_{E_t \cup E_f} - S_{E_t \cap E_f}}{2S_{E_t}},$$

(2.12)

where $S_{E_t \cup E_f} - S_{E_t \cap E_f}$ is the area of the symmetric difference and $S_{E_t}$ denotes the area of the true ellipse.

2.5.1 Comparison of the proposed algorithms

In this subsection, we compare the performance of the algorithms based on the original objective function and the weighted objective function (with the adaptive $\beta$ implementation) to show the improvement achieved by the weighted objective function.

An ellipse in standard position (centered at the origin and without rotation) with semi-major and semi-minor lengths 5 and 3 was used to generate the synthetic observations. Fifty observations were drawn for each trial for a total run of 50 trials per noise level. Figure 2.7 shows the mean error rate under a range of noise levels ($\sigma^2$ from 0 to 0.5). The error bar stands for the 20% quantile and 80% quantile of 50 trials. For the weighted version with adaptive $\beta$, we choose $O_0 = 0.002$ and the smoothing coefficients $\rho = 0.6$. The comparison of the two algorithms are shown in Figure 2.7.

Generally, the weighted objective with adaptive $\beta$ outperforms the original objective, both in mean error rate and consistency (the width of the error bars). Note that for low levels of noise ($\sigma^2 < 0.2$), both algorithms perform similarly and equally well. This indicates that for low noise cases, the proposed algorithm with the original objective function can fall into the desirable local minimum, and hence has no difference from the weighted version, which is closer to the OLSF objective. However, when the noise level becomes higher, the noisy data points smear out the desired local minimum in the objective function. Therefore, the foci of the fitted ellipse of the original objective function slip off as shown in the right half Figure 2.1 and the error rate deteriorates significantly. On the contrary, the weighted objective performs consistently well for all levels of
Figure 2.7: Accuracy of the algorithms: average error rate under a wide range of noise levels for the four proposed algorithms. The lower bound and upper bound of the error bars are the 20% and 80% quantiles of 50 trials.

noise. Therefore, the weighting revision is highly important for increasing the noise-resistance of the algorithm.

Moreover, it is also interesting to observe the change of $\beta$. Our algorithm is designed to be adaptive to different levels of noise. When the noise level is low, the objective function converges fast and $\beta$ should be increased to 1 faster than in the high noise level cases, in which more time is needed for the algorithm to align the fitted ellipse to the correct direction, thus $\beta$ should stay in close-to-zero values longer. Two typical sequences of $\beta$ are shown in Figure 2.8, which demonstrates this phenomenon.

Figure 2.8: Two typical sequences of the weighting coefficients $\beta$ for different levels of noise, with $\sigma^2 = 0.2$ and $\sigma^2 = 0.5$.

In Figure 2.8, both sequences of $\beta$ reach 1 eventually. However, in the low-noise case, $\beta$ approaches 1 much faster than its high-noise counterpart. This illustrates that $\beta$ is adaptive to the
noise level, and hence the weighted objective function can be adjusted accordingly.

2.5.2 Comparison with Algebraic Fitting, OLSF, and ML

To demonstrate the efficacy of the proposed algorithms, we describe simulations to compare our algorithm with the major ellipse fitting algorithms reviewed (algebraic fitting, OLSF and ML). We implemented the numerically stable version of algebraic fitting proposed by Halir and Flusser [23], the OLSF algorithm proposed by Ahn et al. [2], and the FNS ellipse fitting algorithm from Chojnacki et al. [13] respectively. And we use the weighted objective function with adaptive $\beta$ (with the same configuration as in the previous subsection) as a representative for our algorithms.

We compare the algorithms in terms of accuracy, under a high-eccentricity situation (an ellipse in standard position with semi-major and semi-minor lengths 8 and 2, respectively), over a wide range of noise levels ($\sigma^2$ from 0 to 0.8). Fifty observations were drawn for each trial for a total run of 200 trials per noise level for our algorithm, algebraic fitting and OLSF, and a total run of 1000 trials for FNS.

Figure 2.9 shows the simulation result. Although our algorithm performs slightly worse than the other three algorithms when the noise level is very low, it outperforms them when the noise level increases, which shows the noise resistant property of our algorithms. The curve with triangle markers represents the FNS algorithm. It has high accuracy when the noise level is low, yet it breaks down when there are observations near the center of the estimated ellipse, which happens often for moderate or high noise levels. The FNS curve represents the average error on those trials (out of 1000 trials) for which the algorithm produced an estimate. The dotted segment of the FNS plot indicates that the algorithm failed to produce an estimate in more than 90% of the trials.

Figure 2.10 shows the comparison with error bars (20% quantile and 80% quantile of 100 trials) for the algebraic fitting, OLSF and our method. This demonstrates the noise resistance of our algorithms.

As for the computational cost, algebraic fitting is the most efficient algorithm. Our algorithms perform almost the same as the ML algorithms and are much more efficient than the OLSF algorithms in a typical setting.

2.5.3 Spheroid Fitting

We first present an example of a typical spheroid fitting result to demonstrate the efficacy of our spheroid fitting algorithm based on the weighted objective function. Fifty true points were generated
Figure 2.9: Average error rate under a wide range of noise levels for algebraic fitting, OLSF, FNS and our method.

Figure 2.10: Average error rate for algebraic fitting, OLSF and our method with 20% quantile and 80% quantile error bars.

Figure 2.11: Spheroid fitting result.
from the surface of a $10 \times 2 \times 2$ spheroid centered at the origin without rotation. The observations were generated from the true points with additive Gaussian noise with mean $0$ and variance $0.2I$.

Figure 2.11 shows the fitting result of our algorithm.

In Figure 2.12, the simulation results for spheroid fitting under different noise levels are shown. In this simulation, we generate 50 samples points on the surface of the $5 \times 3 \times 3$ spheroid in a standard position, and uses adaptive $\beta$ with $\theta_0 = 0.01$ and $\rho = 0.5$. The average error rates and the corresponding 20\% and 80\% quantiles are demonstrated. For noise as high as $\sigma^2 = 0.5$, the algorithm still generate reasonably accurate fits, even for as few as 50 data samples.

![Spheroid Fitting Error Rate v.s. Noise Level](image)

**Figure 2.12**: Average error rate under a wide range of noise levels for spheroid fitting using weighted objective with adaptive $\beta$. The lower bound and upper bound of the error bars are the 20\% and 80\% quantiles of 50 trials.

### 2.6 Conclusion

In this chapter, a new objective function for ellipse/spheroid fitting has been proposed, which emphasizes accuracy in the axial direction, and is hence more resistant to noise, especially for the case of large eccentricity ellipses. The global minimum of this new objective occurs when the foci tends to infinity, and this is compensated by a weighting term, so that the proposed method can be accurate in estimating both axial direction and size of the ellipse/spheroid. The sequence of weights is increased adaptively for different levels of noise.
Chapter 3

Two-Stage Outlier Elimination for Robust Curve and Surface Fitting

3.1 Background

As an extensively studied field, curve and surface fitting has a rich literature. Various algorithms have been proposed from several different perspectives. Most of the algorithms are based on minimizing an $l_2$ norm, which results in least-squares fitting methods. There are two main categories of least-squares fitting algorithms, algebraic fitting [9] [49] [17] and geometric fitting [2] [1] [6], depending on the definition of error distances.

Least-squares methods are efficient and accurate in situations in which observations are accurate or are contaminated by a moderate amount of noise. However, when the noise level is high or when the set of observations contains wildly erroneous observations, which is often the case when dealing with real environments, least-squares algorithms break down, and more robust algorithms are required to accomplish the task.

Under those circumstances, $l_1$ distance is often employed instead of Euclidean distance to tackle the outliers. There is a considerable literature focusing on this case as well. Al-Subaihi and Watson [3] proposed a series of curve fitting algorithms using the $l_1$ norm and a Gauss-Newton step, which includes examples of fitting lines in three-dimensional space, as well as fitting circles and ellipses in two-dimensional space. Al-Subaihi [5] further investigated an algorithm that fits circular arcs to observations using the $l_1$ norm. As shown by numerical examples, the algorithm can produce satisfying results with the presence of a small portion of outliers. Although fitting curves and surfaces
using the $l_1$ norm is an effective way of dealing with outliers, it still requires a concrete model for the curve and surface to be fitted. In particular, a specific algorithm has to be designed for each model based on its mathematical form, and it usually results in high computational cost. As a result, it is still desirable to remove the majority of the outliers from the observations first without the knowledge of the specific model, and then to fit the observations to the model by more commonly used and efficient algorithms, such as least-squares.

Being a well-developed field itself, outlier detection has been widely studied from very different perspectives, such as statistics, neural networks and machine learning. Hodge and Austin [25] provides a comprehensive survey of outlier detection algorithms. Among all these methods, the proximity-based outlier detection algorithms have the advantage that they do not require prior knowledge about the distribution of the observations. The most widely applied group of proximity-based algorithm is $k$-Nearest Neighbor ($k$-NN). For example, Byers and Raftery [11] proposed a method that determines the distance $D_k$ from an observation to its $k^{th}$ nearest neighbor and compares it with a threshold $d$. If $D_k > d$, then the observation is classified as an outlier. In 2000, Ramaswamy et al. [42] proposed a similar algorithm that ranks the observations according to $D_k$ in descending order, and marks the top $n$ observations in the ranking as outliers. In addition, Knorr and Ng [28] introduce an algorithm that classifies an observation as an outlier if less than $m$ of its $k$ nearest neighbors lie within a disk of radius $d$. Hautamaki et al. [24] proposed an algorithm based on a weighted directed graph ($k$-nearest neighbor graph). The graph is constructed in such a way that each vertex (observation) has $k$ directed edges to its $k$ nearest neighbors with edge weight being the distance between the observation and its neighbor. Any vertex that has indegree less than a threshold $T$ is classified as an outlier. $k$-NN flavored algorithms can be applied to various outlier detection problems, e.g. spatial outlier detection [31].

Despite a large number of proximity based algorithms available, we design a new outlier elimination algorithm, due to the special properties of the curve and surface fitting problem. Proximity-based algorithms alone are not sufficient to eliminate all the outliers for the fitting problem. One of the difficulties is that the outliers and inliers usually intertwine in a complex manner, as shown in Fig. 3.1. Some of the outliers are distant from the curve (like point A), yet others may fall closer to the curve (like point B). The outlier detection algorithms based on proximity may not be effective for detecting the B-type outliers. Moreover, even for A-type outliers, $k$-NN flavored algorithms fail in certain scenarios (see Section 3.3 for further details).

In order to better handle the B-type outliers, we may resort to model-based outlier detection algorithms like random sample consensus (RANSAC) [16]. RANSAC works nicely in various appli-
cations, such as [34], in which the percentage of outliers is relatively low and the inliers have little noise. However, with a high percentage of outliers, when the number of the parameters of the model is large, even if the inlier noise is negligible, the computational cost of RANSAC is a serious problem. On defining $w$ as the portion of the inliers, $d$ as the minimum number of observations needed to fit a model, and $p$ as the probability of successfully finding the correct model after running RANSAC $k$ times, we have the following relationship:

$$p = 1 - (1 - w^d)^k.$$  \hspace{1cm} (3.1)

When $w$ is small (the percentage of outliers is high), we need $k$ to be large in order to have a sufficiently high $p$. Unfortunately, the increase in $k$ with respect to $d$ is exponential. If we assume $w = 0.5$, to guarantee $p = 0.99$, we need $k = 16$ for a simple two-dimensional straight line model; yet for ellipse fitting, where $d = 5$, we need $k = 146$. For an ellipsoid ($d = 9$), we need $k = 2356$. Moreover, when $d$ is large, the fitting algorithm itself becomes rather expensive to run even once.

In a recent variation of RANSAC [15], the number of iterations needed can be dramatically decreased by replacing pure random sampling (which is the origin of the name RANSAC) by guided sampling, in which promising “inliers” are given more weight at resampling. Moreover, unlike the vanilla version of RANSAC, which uses the number of observations in accordance to the suggested model to evaluate its goodness, a more robust measure, (weighted) median absolute deviation (MAD/WMAD) is introduced to reduce the influence of outliers on model evaluation. In terms of total numbers of fittings needed, this algorithm performs very well in scenarios of low inlier noise. However, the algorithm is susceptible to falling into local minima and generating models completely different from the desired one. This phenomenon is aggravated when the level of noise on the inliers is high and the portion of outliers is large. The details of this phenomenon can be found in the simulation results of Section 3.5.

Another approach to robustly fit curves and surfaces is by enumerating all possible solutions. For example, in the case of ellipse fitting, Five Point Fit Ellipse Fitting (FPFEF) [45] is an algorithm in which all five-tuples of observations are selected and fit by ellipses, the median of the parameters of which provides the final fitting result. In terms of robustness, this algorithm performs competitively; however, the computational complexity is unacceptable when the number of parameters is large. Specifically, for a model with $d$ parameters, suppose the total number of observations (including
both inliers and outliers) is \( K \), then the total number of iterations needed for FPFEF is

\[
\binom{K}{n} = O(K^n).
\]  

(3.2)

For a reasonable ellipse fitting problem with 100 observations, the total number of parameters is \( n = 5 \), and thus we need approximately 75 million iterations to enumerate all 5-tuples in order to fulfill the requirement of the FPFEF algorithm.

To counter these problems, in this chapter, we develop a two-stage hybrid outlier detection algorithm for curve and surface fitting that combines proximity-based and model-based outlier detection algorithms. First, with the help of algebraic graph theory [36], we employ an outlier detection algorithm based on the graph formed by the observations, using their proximity information. Second, after reducing the portion of outliers within the remaining set of observations, we employ a model-based outlier detection algorithm to efficiently refine the results. In the first stage, a large portion of the most distant, isolated outliers, which can be a great hazard to model-based outlier detection methods, are detected and eliminated. In the second stage, the subtler, closer outliers that go against the curve/surface model are detected, and some of the misclassified inliers are retrieved. These two stages compensate for each other to form a more efficient and accurate outlier detection algorithm. To our knowledge, there are no such hybrid outlier elimination algorithms discussed in the literature for curve/surface fitting on point cloud data.

The structure of the chapter is as follows. In Section 3.2, the model of our outlier detection problem is specified with several important assumptions. Section 3.3 describes our proximity-based outlier detection algorithm, the first stage of our hybrid outlier detection algorithm. By employing hard and soft thresholds, the graph-component based and eigenspace based algorithms are described respectively. As a necessary supplement, the estimation of the length of a curve based on proximity information is also introduced in Section 3.3. Section 3.4 summarizes the model-based outlier detection algorithm. The simulation results, taking ellipse and ellipsoid fitting as examples, are described in Section 3.5. Section 3.6 concludes our work.

### 3.2 Model and Assumptions

First, we need to specify the model by stipulating a few assumptions about the data, both for inliers and outliers. Let \( \mathcal{M} \) be a \( d \)-dimensional manifold with finite volume \( V \) in a \( d^* \)-dimensional space
Figure 3.1: Two different types of outliers. Proximity-based methods are good at detecting outliers similar to point A, while model-based methods can better detect outliers similar to point B.

\[ \mathbb{R}^{d^*} \ (d \leq d^*) \] Specially, an ellipse is a 1-dimensional manifold in 2-dimensional space. We define the inliers as observations that are drawn from \( \mathcal{M} \) independently with a certain distribution, and then possibly corrupted by a small amount of noise. Due to the noise, the inliers might fall outside of \( \mathcal{M} \). Outliers are different in the sense that they are at a larger distance from \( \mathcal{M} \). We define \( \delta \)-outliers as outliers whose distance to \( \mathcal{M} \) is at least \( \delta \) in the space \( \mathbb{R}^{d^*} \).

In addition, we need to assume that the inliers are the majority of the observations, more than 50% of all the observations.

The basic idea of our proximity based outlier detection is that if the inliers are all distributed over a manifold \( \mathcal{M} \) of finite volume, then when the number of the inliers \( N \) is sufficiently large, with high probability, the distances between inliers become sufficiently small such that if we connect all the observations within distance \( \delta \), the inliers will form a strongly connected component in a proximity graph. Therefore, by selecting the largest strongly connected component formed by connecting observations within radius \( \delta \), we can effectively detect \( \delta \)-outliers, i.e., observations at least \( \delta \)-distant from \( \mathcal{M} \).

To tackle the outliers that lie close enough to \( \mathcal{M} \) (distance less than \( \delta \)), we use a model-based algorithm so that only the observations that agree with the model of \( \mathcal{M} \) are classified as inliers. Since the proximity-based outlier detection algorithm can guarantee that that all remaining outliers are at most \( \delta \) distant from \( \mathcal{M} \), it is rather easy to use implementation-friendly \( l_2 \) loss functions for our model-based algorithm, instead of resorting to other more computationally costly algorithms.

### 3.3 Proximity-Based Algorithm

Although there are many proximity based algorithm available, they may not be appropriate in the context of curve/surface fitting. Take the vanilla version of the \( k \)-NN algorithm [11] as an example.
In Fig. 3.2, a collection of observations (inliers) on a circle together with a small cluster of outliers are fed to the $k$-NN outlier elimination algorithm. Suppose we set $k = 2$. Then as shown in the left hand side of the figure, when the radius of $d$ is set to be large enough so that all observations on the circle are classified as inliers, the three outliers are also classified as inliers.

![Figure 3.2: $k$-NN algorithm failure example.](image)

On the other hand, as illustrated in the right-hand side of Fig. 3.2, when $d$ is sufficiently small so that the observations in the small cluster disqualify as inliers, so do all the true inliers. In this scenario given $k = 2$, no matter how we select $d$, there is no way to eliminate the outliers.

However, if we take into consideration the connectivity of graph components formed by links within radius $d$ in the case shown in the left hand side of Fig. 3.2, then the entire set of observations can be easily separated into two connected components, one much larger than the other, which can be used as an argument for classifying the observations in the small cluster as outliers. Therefore, we need a proximity based algorithm that takes into consideration the connectivity information among observations in a more holistic way beyond the $k$-NN flavored algorithms, which only use information of local neighborhoods of observations. Consequently, it is desirable to develop a graph-based algorithm to fully exploit the proximity information for outlier detection.

### 3.3.1 Graph-Component Based Outlier Detection

**Detecting outliers from graph structure**

As described in Section 3.2, when the number of inliers is sufficiently large, with high probability, the inliers will form a strongly connected graph component. On the other hand, the $\delta$-outliers, distant from the inliers as well as other outliers, will not form connected components with many vertices. As a result, we can effectively detect $\delta$-outliers.

For the special case when $\mathcal{M}$ is a closed curve, we have been able to derive a bound on the
probability of detecting $\delta$-outliers. It shows that on a closed curve of circumference $L$, if the $N$ independent inliers are uniformly distributed on the curve, then the probability that they fail to form a strongly connected graph component (with all points within distance $\delta = \alpha L$ connected) decays asymptotically as $O(N e^{-\alpha N})$. The details of the bound is described as follows.

On a closed curve of length $L$, on which a direction is defined, if there are $N$ independent uniformly distributed observations, then with probability no less than $1 - \epsilon$, each observation has a neighbor within on-curve distance (curve length) $\delta = \alpha L (0 < \alpha < 1)$ along the curve direction, where

$$
\epsilon = \frac{N e^{-\alpha N}}{1 - \alpha}.
$$

The derivation of the bound is as follows. Define an event $A$ as

$$
A = \{ \forall \text{ observation}, \exists \text{ a neighbor within distance } \delta \text{ along the curve direction} \}.
$$

Then by the union bound, and the inequality $(1 - x)^n \leq e^{-nx}$, we have

$$
P(A) = 1 - P(A^c) \geq 1 - N(1 - \delta/L)^N - 1 = 1 - N(1 - \alpha)^N - 1 \geq 1 - \frac{N e^{-\alpha N}}{1 - \alpha}.
$$

Therefore, the probability of $A$ is lower bounded by $1 - \epsilon$, where $\epsilon = \frac{N e^{-\alpha N}}{1 - \alpha}$.

According to this bound, with probability $1 - \epsilon$, each observation on the curve has a neighbor within on-curve distance $\delta = \alpha L$ along the curve direction, and thus its Euclidean distance to its neighbor is less than $\delta$. So each observation is connected to its neighbor. Therefore, in the proximity graph formed by connecting all neighbors within radius $\delta$, there exists a single loop that connects all the observations. As a result, the generated graph is strongly connected. Therefore, for a given $\epsilon$, by properly selecting $\delta$ according to the bound, we can effectively eliminate $\delta$-outliers by simply classifying large connected components as inliers and others as outliers.

The graph-component based outlier detection algorithm for curves is based on the reasoning above. After selecting an appropriate value of $\alpha$, and taking $\delta = \alpha L$, we form a proximity graph by connecting observations if the Euclidean distance between them is less than $\delta$. Then we classify all the connected components that have less than $\beta K$ vertices as outliers, with $K$ being the total
number of observations. Practically, $\beta$ ranging from 0.05 to 0.1 is an appropriate choice. Algorithm 1 summarizes the details of detecting $\alpha L$-outliers for a closed curve of circumference $L$ in 2-dimensional space.

**Algorithm 1:** Graph-Component based Algorithm

**Data:** $K$ observations $\{(x_i, y_i)\}_{i=1}^{K}$
**Result:** $I$, indices of inliers in the dataset

```
begin
  forall the $i, j \in 1, 2, ..., K$ do
    $Q_{ij} \leftarrow$ Euclidean distance between $(x_i, y_i)$ and $(x_j, y_j)$
  end

  Estimate the curve length $L$ based on $Q$ according to Algorithm 2

  forall the $i, j \in 1, 2, ..., K$ do
    if $Q_{ij} - \alpha L > 0$ then
      $W_{ij} \leftarrow 0$, no edge between vertexes $i$ and $j$
    else
      $W_{ij} \leftarrow 1$, edge added between vertexes $i$ and $j$
    end
  end

  Get the set of connected components $\{C_k = (V_k, E_k)\}_{k=1}^{S}$ of the graph corresponds to matrix $W$
  $I \leftarrow \{ i : i \in V_k, |V_k| > \beta K \}$
end
```

It is worth pointing out that the length $L$ of the curve, or more generally, the volume of $\mathcal{M}$ needs to be estimated from the observations. This is described in the following subsection.

### Estimating the Volume of $\mathcal{M}$

The efficacy of our outlier detection algorithm is directly related to the choice of $\delta$. A small value of $\delta$ leads to a small probability of forming a strongly connect component of all the inliers, yet a large value of $\delta$ results in including outliers, which is rather disastrous to the second stage model-based outlier detection algorithm.

The choice of $\delta$ is directly related to the volume of the manifold $\mathcal{M}$ in which the inliers reside. In order to select a proper $\delta$, one needs to have an estimate of the volume $V$ of $\mathcal{M}$, without even having an accurate model fitted to it. Specifically, in the one-dimensional case, the length of the curve $L$ is essential to outlier detection. As a result, we provide an estimate of the volume of $\mathcal{M}$ as follows.

Suppose $\mathcal{M}$ is a $d$-dimensional manifold, and that $N$ observations are drawn independently from
If the probability density of the observations is given by $\rho(x)$, $x \in M$, then

$$
\mathbb{P}(\text{Nearest neighbor distance} \leq y) \approx (N - 1)\mathcal{V}(B_y^{(d)}) \int_M \rho^2(x)dx,
$$

(3.4)

where $\mathcal{V}(B_y^{(d)})$ stands for the volume of a $d$-dimensional ball with radius $y$. Specifically, if the observations are uniformly distributed, then

$$
\mathbb{P}(\text{nearest neighbor distance} \leq y) \approx \frac{(N - 1)\mathcal{V}(B_y^{(d)})}{V}.
$$

(3.5)

In the special case in which a curve of length $L$ is considered, and in which $N$ observations are independent and uniformly distributed on the curve, then

$$
L \approx \frac{2(N - 1)y}{\mathbb{P}(\text{nearest neighbor distance} \leq y)}.
$$

(3.6)

Now we provide the details of the estimates (3.4) (3.5) and (3.6).

Suppose the inliers are independently sampled from a one-dimensional manifold of length $L$ with probability density $\rho(x)$. If $N$ observations are sampled, then for any observation $P$ located at $x_0$, the probability that its nearest neighbor on the manifold falls outside of the $B_y(x_0)$ ball of radius $y$ is

$$
\mathbb{P}(\text{Distance to nearest neighbor of } P > y) = \left(1 - \int_{x_0-y}^{x_0+y} \rho(x)dx\right)^{N-1}.
$$

(3.7)

Then the probability that a randomly chosen data point has a nearest neighbor within the radius of $y$, defined as $F(y)$, is given by

$$
F(y) = \int_0^L \rho(x_0) \left(1 - \left(1 - \int_{x_0-y}^{x_0+y} \rho(x)dx\right)^{N-1}\right)dx_0.
$$

(3.8)

Applying the fact that the integral of the density on the entire manifold is one, (3.8) can be simplified into

$$
F(y) = 1 - \int_0^L \rho(x_0) \left(1 - \int_{x_0-y}^{x_0+y} \rho(x)dx\right)^{N-1}dx_0.
$$

(3.9)
If we assume that $y$ is sufficiently small, then we may make a series of approximations:

$$
F(y) \approx 1 - \int_0^L \rho(x_0) (1 - 2y\rho(x_0))^{N-1} \, dx_0
$$

(3.10)

$$
\approx 1 - \int_0^L \rho(x_0) (1 - 2(N - 1)y\rho(x_0)) \, dx_0
$$

(3.11)

$$
= 2(N - 1)y \int_0^L \rho^2(x_0) \, dx_0.
$$

(3.12)

Notice that $\int_0^L \rho(x_0) \, dx_0$ is directly related to the volume of the manifold. If we assume that the observations are uniformly distributed, then $\rho(x_0) = 1/L$, and hence $\int_0^L \rho(x_0) \, dx_0 = 1/L$. In this case we have the result in (3.6):

$$
L \approx \frac{2(N - 1)y}{F(y)}.
$$

(3.13)

The method above can be directly generalized to the case of a $d$-dimensional manifold $M$, which leads to the result in (3.4) and (3.5).

In practice, the distribution $P($nearest neighbor distance $\leq y)$ can be estimated from the histogram of the nearest-neighbor distances of each observation (or resampled observations, if performed in a bootstrapping manner). To ensure that all the nearest-neighbor distances are between the inliers, it is desirable to truncate the tail of the histogram. In the worst case, we can cut off 50% of the nearest-neighbor distances to roughly ensure that no outliers mix in. Then the value of $N$ is simply the remaining number of observations after cutting off the tail. With properly selected quantile $y$, the curve length $L$ can be readily estimated.

The reason that we only use nearest-neighbor distances in our estimate is because nearest-neighbor distances are fairly good approximations (from above) to the true distances on the manifold. Other distances among the inliers could be informative, yet not compatible with our measure on the manifold.

If the inliers are contaminated by noise, then even the nearest-neighbor distances are not very accurate estimates of the distances on the manifold. Estimate based on (3.6) could deteriorate. Estimates given by (3.6) in the noisy case are larger than the true volume of the manifold.

The algorithm for estimating the length of a curve is summarized in Algorithm 2. The basic idea is to first take a vector $m$ of nearest-neighbor distances, sorted in ascending order. Then cut off 50% of the tail of the vector, and use the histogram of the remaining nearest-neighbor distances to estimate the curve length $L$ according to (3.6).
Algorithm 2: Circumference Estimation of a Curve (One Dimensional Manifold)

**Data:** $Q$, with $Q_{ij}$ being Euclidean distance between $(x_i, y_i)$ and $(x_j, y_j)$

**Result:** Curve length estimate $L$

begin
    for $i \in 1, 2, ..., K$ do
        $m_i \leftarrow \min\{Q_{ij}, j \in \{1, 2, ..., K\}\ \backslash \{i\}\}$
    end
    $m' \leftarrow$ sort $m$ in ascending order
    $m^* \leftarrow$ take the first $\lfloor K/2 \rfloor$ elements of $m'$
    $q \leftarrow p$-quantile of $m^*$
    $L \leftarrow 2q(\lfloor K/2 \rfloor - 1)/p$
end

### 3.3.2 Eigenspace-based Algorithm

In practice, graph-component based outlier detection works very well when there is no noise or a very small amount of noise on the inliers, since in this case, our estimate of the volume of $M$ is very accurate and we are able to choose $\delta$ properly. However, due to the noise on the inliers, we usually cannot determine a very accurate estimate of the size of $M$. In this case, if a hard connection rule is applied to generate the graph, it might lead to an over-connected graph in which too many outliers are connected to inliers, or an under-connected graph, in which the inliers are isolated in several pieces, as shown in Fig. 3.3.

![Figure 3.3: Different choices of connection radius lead to different results. The figure on the left depicts the situation in which the radius is so small that even the inliers do not connect sufficiently with each other. The figure on the right illustrates the case in which the radius is so large that almost all the observations are connected with each other, and thus makes it impossible to distinguish inliers from outliers. The figure in the middle shows that with a properly selected radius of connection, the inliers form a large connected component, leaving most of the outliers unconnected.](image)

To solve this problem, we choose to use a soft threshold (heat kernel) to determine the connectivity among observations, and then analyze the eigenspace (ideally, the null space) of the graph Laplacian of the resulting weighted undirected graph to detect outliers.
Constructing proximity graph and its Laplacian

According to algebraic graph theory, the connectivity information about a graph is reflected in the graph Laplacian and its corresponding eigenvalues and eigenvectors. Instead of using an unweighted graph with a hard connection rule as in the case of the previous subsection, the graph Laplacian is constructed using heat kernels in the following manner:

1. Form a matrix $Q$ with $Q_{ij}$ being the Euclidean distance between the observations $(x_i, y_i)$ and $(x_j, y_j)$;
2. Form a fully connected proximity graph with edge weight $W_{ij} = e^{-Q_{ij}^2/t}$ (heat kernel);
3. Construct the graph Laplacian matrix: $L = D - W$, where $D$ is a diagonal matrix with its diagonal entries as column sums (or row sums) of $W$;
4. Compute eigenvalues and eigenvectors for the generalized eigenvector problem, $Lf = \lambda Df$.

Then, the set $\{\lambda | \lambda \approx 0, \lambda \in \{\lambda_i\}_{i=1}^K\}$ and its corresponding eigenvectors form the cornerstone of our proximity-based outlier detection algorithm. The algorithm for constructing the graph Laplacian is summarized in Algorithm 3.

**Algorithm 3: Construct Graph Laplacian**

<table>
<thead>
<tr>
<th>Data:</th>
<th>$K$ observations ${(x_i, y_i)}_{i=1}^K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result:</td>
<td>The eigenspace of the Graph Laplacian $L$</td>
</tr>
</tbody>
</table>

begin

forall the $i, j \in 1, 2, ..., K$ do

$Q_{ij} \leftarrow$ Euclidean distance between $(x_i, y_i)$ and $(x_j, y_j)$
end

Choose radius of connection $t$ based on $Q$

forall the $i, j \in 1, 2, ..., K$ do

$W_{ij} \leftarrow e^{-Q_{ij}^2/t}$
end

$D \leftarrow \text{diag}(1^T W)$

$L \leftarrow D - W$

Solve $Lf = \lambda Df$

end

The success of proximity-based outlier detection depends heavily on the choice of the “radius of connection” (i.e., $\sqrt{t}$ in the heat kernel). Thus, the value of $t$ is of essential importance. Yet since the heat kernel is a soft threshold, it is more tolerant to this choice compared to the choice of $\delta$ in the graph-component based algorithm. Still, the choice of $t$ is a rather delicate problem. According to algebraic graph theory, $t$ is closely related to the algebraic connectivity of the graph (the smallest non-zero eigenvector of the graph Laplacian). Here we present an empirical rule for determining a
good value of $t$. Sort all the elements of $Q$ in ascending order and denote the resulting vector as $q$. Then take the $pK$-th element of $q$ as $\sqrt{t}$, i.e. assign $\sqrt{t}$ as the $p$-quantile of the distribution of all the elements of $Q$. This bears some similarity to the way we estimate $L$ and select $\delta$.

Intuitively, most of the small elements of $Q$ are composed of the distances between inliers, according to our assumptions. By taking the $pK$-th element of $q$ as $\sqrt{t}$, we approximately guarantee a connected subgraph among the inliers of average degree $p - 1$, if we assume $e^{-1}$ is the threshold for a “connection”. Empirically, $p = 4$ works well.

**Selecting outliers in eigenvectors**

Given an appropriate value of $t$, the inliers form a strongly connected component, leaving most of the outliers loosely connected to it and each other. As a result, with a proper interchange of rows and columns, the matrix $Q$, and thus the Laplacian matrix $L$, is close to a block diagonal matrix. Each block corresponds to a strongly connected component, with the largest one corresponding to the inliers.

For each block, there is an eigenvalue approximately equal to zero, and the corresponding eigenvector is almost a binary vector composed of 0’s and 1’s, with only the elements aligned with the block being 1’s. Here, we assume that the $l^\infty$ norms of all the eigenvectors are normalized to 1. Among all the approximate binary eigenvectors that correspond to the close-to-zero eigenvalues, some contain more 1’s than others. The key step of outlier elimination is to eliminate those observations that correspond to the non-zero elements of binary eigenvectors with very few 1’s, because the components composed of these observations are very weakly connected to other components, and thus are more likely to be outliers.

Therefore, finding outliers is approximately equivalent to finding protruding 1’s in the eigenvectors with close-to-zero eigenvalues. Thus, we have reduced a problem of high-dimensional outlier detection with a complex hypothesis into a one-dimensional outlier-detection problem. However, since the eigenvectors that correspond to close-to-zero eigenvalues are not perfectly binary, more elaborate methods need to be employed to detect outliers.

First, eligible eigenvectors (approximate binary eigenvectors) are selected from the collection of eigenvectors. Specifically, we take the eigenvectors that correspond to eigenvalues less than 0.1 as the candidates. Then, keeping only the binary eigenvectors, the “high frequency” eigenvectors are excluded. Note that $f$ is a “high frequency” eigenvector if $(\sum_j |f_j| - |\sum_j f_j|)/\sum_j |f_j|$ is sufficiently large, i.e. those eigenvectors with both large positive and negative elements are excluded. After that, a one-dimensional outlier detection algorithm is employed, as shown in Algorithm 4.
Algorithm 4: Select Outliers in Eigenvectors

Data: Eligible eigenvectors \( \{f_i\}_{i=1}^l, f_i \in \mathbb{R}^k \)

Result: \( O \), indices of outliers in the set of observations

begin
  \textbf{foreach} \( f_i, i \in \{1, 2, \ldots, l\} \) \textbf{do}
    \begin{align*}
    &\text{Initialize:} \\
    &\quad \mathcal{I}_i' \leftarrow \text{indices of randomly chosen } \left\lfloor \frac{k}{2} \right\rfloor \text{ elements of } f_i \\
    &\quad \mathcal{I}_i \leftarrow \text{null} \\
    &\text{while } \mathcal{I}_i \neq \mathcal{I}_i' \text{ do} \\
    &\quad \mathcal{I}_i \leftarrow \mathcal{I}_i' \\
    &\quad \alpha_{1/4} \leftarrow 25\% \text{ quantile of } \{f_{i,j} \in \mathcal{I}_i\} \\
    &\quad \mu \leftarrow 50\% \text{ quantile of } \{f_{i,j} \in \mathcal{I}_i\} \\
    &\quad \alpha_{3/4} \leftarrow 75\% \text{ quantile of } \{f_{i,j} \in \mathcal{I}_i\} \\
    &\quad I \leftarrow [\mu - \gamma(\mu - \alpha_{1/4}), \mu + \gamma(\alpha_{3/4} - \mu)] \\
    &\quad \mathcal{I}_i' \leftarrow \{j : f_{i,j} \in I\} \\
    &\text{end}
    \end{align*}

  \text{end}

  \( O_i \leftarrow \{j : j = 1, 2, \ldots, K, j \notin \mathcal{I}_i\} \)

end

Output \( O \leftarrow \bigcup_{i=1}^l O_i \)

Algorithm 4 does not guarantee the “correct” result. It can be helpful to run it several times and choose the result with minimum intra-class deviation (as in the case of RANSAC). However, in our simulations, we ran the routine only once, which was good enough for our data.

3.4 Model-Based Algorithm

The algorithm described in the previous section uses only the proximity information of the observations. However, it is rather incapable of detecting type-B outliers in Fig. 3.1, and more specifically, outliers that lies within the range of less than \( \delta \) from \( \mathcal{M} \), even if they obviously deviate from the true model.

To eliminate type-B outliers, we need to use the additional prior knowledge that the inliers are located on a manifold with certain shape and size, for example, an ellipse. Since we have greatly reduced the portion of outliers in the first step, we can of course achieve this goal by running a model based outlier detection algorithm.

Moreover, since our proximity-based algorithm also guarantees, with high probability, that the remaining outliers are close enough to the true model, we can even be more efficient by initializing our model using all the observations classified as inliers in the first step, and iteratively eliminating the ones whose distances to the fitted model exceeds a threshold, until convergence. We can even retrieve misclassified inliers, if any, given that they lie right on the model fitted in the second step,
the model-based algorithm.

We summarize the model-based outlier detection algorithm as follows:

1. Fit a model $h$ to the “inliers” selected by the proximity-based outlier detection algorithm;

2. Test all the observations with respect to $h$, classify the observations that saliently deviate from $h$ (above a threshold) as outliers and then classify other observations as inliers;

3. Refit model $h$ based on the updated inliers;

4. Repeat step 2 and step 3 until the classification does not change any further.

Specifically, for algebraic fitting of curves, the second step of the algorithm is shown in Algorithm 5.

Algorithm 5: Model-based Algorithm

<table>
<thead>
<tr>
<th>Data:</th>
<th>Inliers selected by a proximity-based algorithm, ${(x_i, y_i)}_{i \in I}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result:</td>
<td>$I$, indices of inliers in the data set</td>
</tr>
<tr>
<td>begin</td>
<td>Initialize: $I' \leftarrow I$, $I \leftarrow \text{null}$</td>
</tr>
<tr>
<td>while $I \neq I'$ do</td>
<td>$I \leftarrow I'$</td>
</tr>
<tr>
<td>fit a model $h(x_i, y_i) = 0$ to ${(x_i, y_i)}_{i \in I}$</td>
<td></td>
</tr>
<tr>
<td>compute average fitting deviation $\sigma = \sqrt{\frac{1}{</td>
<td>I</td>
</tr>
<tr>
<td>$I' \leftarrow {i : h(x_i, y_i) &lt; \alpha \sigma}$</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>Output $I$</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

With this model-based outlier detection algorithm, we are able to detect most of the missed outliers in the first stage (usually harder ones very close to the inliers), and also correct the labels for misclassified inliers.

In addition, due to the fact that the number of outliers is rather small for the second step, and they are all close enough to the true model, the convergence of the second step is usually very fast.

### 3.5 Simulation Results

Among many curve/surface fitting problems of possible interest, we choose ellipse and ellipsoid fitting as our subject for simulation, because these two fitting problems are widely useful in various application scenarios, yet are sufficiently difficult, in terms of the number of parameters and the
computational cost for each individual fitting, so that RANSAC is rather costly when the fraction of outliers is large.

In the outlier detection problem for ellipse and ellipsoid fitting, we assume that there are a total number of $K = M + N$ observations $\{(x_i, y_i)\}_{i=1}^{K}$ with $N$ inliers and $M$ outliers. The inliers are sampled on an ellipse and then contaminated by Gaussian noise with standard deviation $\sigma_0$; and we simply model the outliers as points on the ellipse added by a significantly intense Gaussian noise, with standard deviation $\sigma_1 > \sigma_0$.

Since we are interested in the case in which inliers are contaminated by non-negligible noise, the eigenspace-based algorithm (which employs soft links among observations) is used as the algorithm in the proximity-based stage. The value of $\sqrt{7}$ is selected as the $4K$-th element of $q$ as in the explanation of Algorithm 3. Moreover, for the second stage of our algorithm, we choose $\alpha = 3$ in Algorithm 5 throughout our simulation.

### 3.5.1 A Typical Ellipse Fitting with Outlier Elimination

It is of interest to inspect a typical simulation for outlier detection on an ordinary ellipse to see the performance of our two-stage algorithm. In the simulation, we have $N = 100$ inliers and $M = 50$ outliers. The standard deviation of the independent Gaussian additive noise of inliers in both $x$ and $y$ directions is $\sigma_0 = 0.1$, and that of the outliers is $\sigma_1 = 2$. The true ellipse has an eccentricity $\epsilon = 0.95$ with semi-major length $a = 5$, and takes the standard position (centered at the origin, with semi-major axis aligned with the $x$ axis).

The outlier detection results of our algorithm on an ellipse are shown in Fig. 3.4. Note that the first stage of our algorithm makes several mistakes, with several outliers missed (the ones inside and closely outside the ellipse) and a few inliers misclassified (several circled observations on the perimeter of the ellipse). These mistakes are corrected by the second-stage model-based algorithm. And the final result leaves us a group of purged, low-noise observations from an ellipse.

### 3.5.2 Improvement in Performance

It is worthwhile to compare the performance of fitting procedures with and without outlier elimination for different numbers of outliers. In this simulation we choose $N = 100$, $M$ ranging from 1 to 55, $\sigma_0 = 0.1$, and $\sigma_1 = 3$, keeping the true model for the ellipse the same as in the previous simulation.

To measure the performance of ellipse fitting algorithms, we employ the non-overlapped area of
the fitted and the true ellipses (normalized by the area of the true ellipse). To be more specific, the fitting error is defined as the normalized area difference between the true ellipse $E_t$ and the fitted ellipse $E_f$:

$$
\text{error rate} = \frac{S_{E_t \cup E_f} - S_{E_t \cap E_f}}{2S_{E_t}},
$$

(3.14)

where $S_{E_t \cup E_f} - S_{E_t \cap E_f}$ is the area difference and $S_{E_t}$ denotes the area of the true ellipse. This measure is henceforth called relative area difference. This simulation result is shown in Fig. 3.5.

![Figure 3.4: Outlier detection results for ellipse fitting. The circled observations are classified as outliers by the algorithm. The figure on the left demonstrates the detected outliers after the proximity-based algorithm, in which there are several missed outliers close to the ellipse, while a few inliers are misclassified as outliers. The figure on the right shows that these mistakes are corrected by the model-based algorithm.](image)

![Figure 3.5: Comparison between ellipse fitting algorithms with and without outlier detection using the hybrid detection algorithm.](image)

Each point in the figure is the average performance of 200 independent trials. It is obvious that
the outlier detection algorithm effectively eliminates most of the outliers, even when the fraction of outliers is close to 50%.

### 3.5.3 Comparison with Vanilla RANSAC

Not only does the hybrid algorithm perform well in terms of small fitting error, it is also attractive computationally. Given that the model is complex enough, fitting model to the observations can be deemed as the most costly step. Therefore, the total number of times that the model fitting step is implemented can be a good measure of computational complexity of the outlier eliminating algorithms. A desirable algorithm should be able to achieve a reasonably good result in a small number of iterations. Since in the hybrid algorithm the first stage proximity based algorithm eliminates a large portion of the outliers, empirically the second stage converges very fast, and is to some extent, irrelevant to the percentage of outliers. On the contrary, to guarantee a similar performance, the number of iterations needed for the vanilla version of RANSAC increases rapidly as the percentage of inliers $w$ decreases. According to (3.1), given that $w^d$ is sufficiently small, the total number of iterations $k$ needed satisfies

$$k \approx C \cdot \left(\frac{1}{w}\right)^d. \tag{3.15}$$

Obviously when $d$ is large ($d = 5$ for ellipse fitting) and $w$ is small, the number of iterations $k$ can be enormous.

Therefore, we demonstrate the comparison of our hybrid algorithm and vanilla RANSAC in two ways. First, given a maximum number of iterations for vanilla RANSAC, which almost ensures the same computational complexity as our hybrid algorithm (both are almost constant-time), we compare the average fitting errors (in relative area difference) of the two methods. Second, given that our algorithm and the vanilla version of RANSAC have the same performance measured in terms of relative area difference, we compare the number of average iterations needed for the hybrid algorithm (second stage) and the vanilla RANSAC.

In this simulation, we choose inlier noise level as $\sigma_0 = 0.1$ and outlier noise level as $\sigma_1 = 3$, keeping the geometry of the ellipse unchanged. Moreover, we fix the total number of inliers as $N = 100$, and change the number of outliers to achieve different outlier percentage of the total points, $\frac{M}{M+N}$. For the first purpose, we fix the total number of iterations of vanilla RANSAC to 1000, and vary the outlier percentage from 0% to 45%. For the second purpose, we enforce the constraint that the RANSAC algorithm reach the same performance (in terms of relative area difference) as our hybrid algorithm, and record the average of the number of iterations of both algorithms, with the outlier
percentage ranging from 0% to 25%. For computational reasons, we set up an upper limit of 20000 iterations for both algorithms.

For each percentage of outliers, we run the two algorithms 200 times, and record the averaged results in Fig. 3.6 and Table 3.1.

![Comparison of Our Method with Direct RANSAC](image)

**Figure 3.6:** Comparison of performance (measured in relative area difference) between RANSAC and the hybrid algorithm, given the maximum of number of iterations set to 1000.

<table>
<thead>
<tr>
<th>Percentage of Outliers (%)</th>
<th>5</th>
<th>7.5</th>
<th>10</th>
<th>12.5</th>
<th>15</th>
<th>17.5</th>
<th>20</th>
<th>22.5</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Relative Area Difference (%)</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Average Iterations of RANSAC</td>
<td>174</td>
<td>221</td>
<td>288</td>
<td>386</td>
<td>1246</td>
<td>1534</td>
<td>1739</td>
<td>2351</td>
<td>3364</td>
</tr>
<tr>
<td>Average Iterations of Hybrid (2-nd Stage)</td>
<td>2.1</td>
<td>2.2</td>
<td>2.4</td>
<td>2.5</td>
<td>2.8</td>
<td>3.0</td>
<td>3.2</td>
<td>3.7</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 3.1:** Average number of iterations of RANSAC and the hybrid algorithm for different percentages of outliers.

In the simulation, our hybrid algorithm has a considerable advantage in terms of the total number of fittings. To achieve the same performance, the second stage of the hybrid algorithm needs only a few iterations, as compared to thousands of iterations for vanilla RANSAC. If we force RANSAC to terminate in 1000 iterations, its average performance deteriorates rapidly with respect to the percentage of outliers.

Moreover, close inspection indicates that the RANSAC algorithm, when the percentage of the outliers is high, constantly reaches the maximum number (20000) of iterations, which gives the hybrid algorithm an advantage in terms of the number of iterations.
3.5.4 Comparison with RANSAC of Guided Sampling

To further demonstrate the efficacy of our algorithm, especially its robustness to inlier noise, we compare it with one of the variations of the RANSAC algorithm: RANSAC with efficient sampling using Ensemble Inlier Sets [15] (RANSAC-EIS-Metropolis).

RANSAC-EIS-Metropolis converges very rapidly compared to its predecessors. It works very effectively when the noise level of inliers is low, as shown in the left hand side of Fig. 3.7, in which case the inlier noise level is $\sigma_0 = 0.02$. However, when the inlier noise level is high and the fraction of outliers is large, the criterion for evaluating the goodness of the model (MAD/WMAD) becomes less effective and there is a high probability that the algorithm will fail to find the correct model, as shown in the left hand side of Fig. 3.7, in which case the inlier noise level is $\sigma_0 = 0.08$ (lower than all our previous simulations).

To show the influence of the inlier noise level on the effectiveness of the RANSAC-EIS-Metropolis algorithm, and to demonstrate the robustness of our algorithm with respect to the inlier noise, we compare our algorithm to the RANSAC-EIS-Metropolis algorithm in terms of failure rate under different inlier noise levels. We use 100 inliers and 80 outliers, with inlier noise level $\sigma_0$ increasing from 0 to 0.32 and outlier noise level $\sigma_1 = 10$ fixed. The failure rate is measured as the percentage of fitting results having relative area difference beyond 0.3 in 200 trials. For the RANSAC-EIS-Metropolis algorithm, we set the maximum number of iterations as 500, much higher than the average iterations needed for a similar ellipse fitting problem described in [15]. Keeping everything else the same, the following results are obtained, as displayed in Table 3.2.

For low inlier noise levels, the two algorithms perform competitively. However, as the inlier noise
level increases, the RANSAC-EIS-Metropolis algorithm fails to find the true model at a considerable rate, while the hybrid algorithm is not affected until the noise level is as high as $\sigma_0 = 0.32$. Although RANSAC-EIS-Metropolis is superior for low levels of inlier noise (which is usually the case for computer vision applications), as demonstrated in [15], it is not very resistant to the noise on inliers. Our hybrid algorithm, on the other hand, shows greater robustness to inlier noise level, which makes it more appropriate for application with inlier noise.

This simulation results have another implication: if the model of the inliers is not identical to the true model, then even if there are no observational errors, there still exist systematic errors, which can also be a hazard to the RANSAC-EIS-Metropolis algorithm. Therefore, for applications in which the inlier data cannot be accurately captured by the model, it might be safer to use a hybrid algorithm, which is resistant to the inaccuracy in the fitting model.

### 3.5.5 Robustness to Different Types of Outliers

Since our outlier detection algorithm is a hybrid, it is able to tackle a broad variety of outliers and the fitting error can be bounded. Here we run our algorithm for different types of outliers by adjusting the noise level of outliers $\sigma_1 \in [0.1, 1.2]$, with $N = 120$, $M = 90$, $\sigma_0 = 0.1$, and the true model for the ellipse unchanged. An interesting result is shown in Fig. 3.8, where for outliers closer to the inliers as well as outliers distant from the inliers, our algorithm performs consistently well, with fitting error tightly bounded below a low level. This shows the strong robustness of our scheme.

### 3.5.6 Generalization to 3-D: Ellipsoid Fitting

The basic set-up for the ellipsoid fitting simulation is as follows: for inliers, $N = 300$ and $\sigma_0 = 0.1$; for outliers, $M = 50$ and $\sigma_1 = 5$; the ellipsoid takes the standard position, with semi-axis lengths $a = 5$, $b = 4$, and $c = 3$. The outlier detection results are as shown in Fig. 3.9.
Figure 3.8: The performance of our two-stage algorithm under a variety of outlier noise levels. Our algorithm performs consistently well for outliers with very different magnitudes.

Figure 3.9: Outlier detection results for ellipsoid fitting. Similarly to the ellipse fitting case, the proximity-based algorithm eliminates most of the distant outliers, yet misclassifies some inliers as outliers, with several close-by outliers missed, as illustrated on the left figure. These errors are corrected by the second stage model-based algorithm, as shown in the right-hand figure.
3.6 Conclusion

Proximity-based outlier detection algorithms are good for situations in which outliers are wildly erroneous and large in number. However, such algorithms work poorly for outliers that are close to the inliers, even though these outliers are obviously not consistent with the model; on the contrary, model-based algorithms are very good at detecting a small portion of outliers that are not consistent with the model. Yet if the percentage of outliers is high and the number of parameters for the model is large, the implementation of model-based algorithms can be costly. In the problem of curve and surface fitting with many outliers, by combining these two types of algorithms, we have found a promising hybrid method that performs robustly with high accuracy for a variety of types and numbers of outliers. The hybrid algorithm can effectively reduce the total number of iterations of fitting model to the data (selected inliers). Consequently, although the proximity based step requires computation of all pairwise distances among all the data points, when the model is so complicated that fitting model to the data becomes the critical step, the hybrid algorithm can be more efficient compared to other outlier elimination algorithms that require a large number of model re-fittings.
Chapter 4

Trajectory Reconstruction for Rigid Bodies

4.1 Motivations

In the previous chapters, we concentrated on shape reconstruction. To be more specific, we proposed several methods to fit a static ellipse/ellipsoid with point cloud data in noisy environment, with outlier elimination techniques. In this chapter, we will discuss a more challenging problem: estimating the shape parameters and reconstructing the trajectories of the moving objects simultaneously.

The field of reconstructing the trajectories of moving objects, is well developed, with point target tracking based on various types of noisy observations being a major branch. This field has developed a number of effective algorithms, for instance, Kalman filtering [48] and particle filtering [43] are two of the important methods. Some recent work concentrates on position reconstruction based on range data [30], which may be distributively and asynchronously collected.

In all of the works described above, the target object is assumed to be a point. However, when the size of the object is non-negligible compared to its trajectory, and the observations are collected on different parts of the surface of the rigid body, it is necessary to take the shape of the rigid body into consideration when reconstructing the trajectory.

There is a rich literature in the computer vision community on shape tracking, in which the shape of the “rigid body” is usually known and deformable due to different directions of projection. The shape is tracked with the help of additional information, such as intensity, color, or even specially designed markers [40]. This model is quite different from the scenario of interest here, in which the
parameters of the shape are unknown and invariant, and the observations form a point cloud data with time stamps, sometimes with fewer data points than the number of shape parameters at each time stamp.

If we consider two extreme cases of this problem, it reduces to two problems that are well researched: if the location and orientation of the rigid body are invariant with respect to time, the problem is reduced to curve/surface reconstruction based on point cloud data [1]; if the rigid body shrinks to a single point, then it degenerates to a pure tracking/localization problem. The mixture of the two, however, creates a more interesting yet difficult problem, as shown in Fig. 4.1. Even if we merely need the shape or the trajectory information, we still have to combine both the dynamic and geometric properties of the moving rigid body.

In this chapter, we propose a method for reconstructing the trajectory of a moving rigid body based on point cloud data (with time stamps) on the surface of the rigid body. We consider three types of rigid bodies in two dimensional space: a disk, a rod and an ellipse. We will discuss both the situations in which the shape parameter is known and unknown.

The structure of the chapter is as follows. In Section 4.2, basic models and problems are established. In Section 4.3, the trajectory recovery problem for a disk with known shape parameter is discussed. The problem is further generalized to the case of a rod and an ellipse in Sections 4.4 and 4.5. In Section 4.6, the trajectory recovery problem for a disk with unknown shape parameter is investigated. It is followed by simulation results in Section 4.7. In Section 4.8, future work and directions are discussed, and Section 4.9 concludes the work.

### 4.2 Problem Formulation

Assume that the rigid body is moving in a two dimensional space, centered at \( c(t) \) at time \( t \). The orientation of the rigid body is described as a unit vector \( \theta(t) \). Denote the center velocity and
angular velocity of the rigid body, respectively, as

\[ \mathbf{v}(t) = \dot{\mathbf{c}}(t) \]  

(4.1)

and

\[ \omega(t) = ||\dot{\theta}(t)||. \]  

(4.2)

Suppose that \( p_j \) is an observation sampled from the surface of the rigid body at time \( t_j \), \( j = 0, ..., N \), as shown in Fig. 4.2. In the case where the shape parameters are known, our goal is to reconstruct \( \mathbf{c}(t) \) and \( \theta(t) \), given the observations \( p_j \) and sampling time \( t_j \), while in the case where the shape parameters are unknown, our goal is to reconstruct \( \mathbf{c}(t), \theta(t), \) and the shape parameters simultaneously.

Figure 4.2: Illustration of the trajectory recovery problem.

To tackle this problem, if we simply assume that the trajectory of the object needs to satisfy merely the geometric constraints, then there are infinitely many different solutions, even if the shape of the rigid body is completely known. Therefore, extra constraints are needed to obtain a meaningful result. In most real-world cases, such as a submarine, a land vehicle or an airplane, the rigid body moves in an environment of friction. If the system is designed and operated energy-efficiently, it should dissipate a minimal amount of energy into the environment due to friction. Therefore, we propose an energy minimization model to solve this trajectory/shape recovery problem. We will discuss the formulation of the optimization problems in the cases of a disk, a rod, and an ellipse respectively in the next few sections.

### 4.3 Trajectory Recovery for a Disk with Known Shape Parameter

First, we consider the trajectory recovery problem for a moving disk with known radius \( r \), as shown in Fig. 4.3. To simplify this problem, the friction due to the rotation of the disk is ignored.
4.3.1 Problem Formulation

As described in the previous section, we tackle this problem via an energy minimization principle. On assuming that the friction is proportional to the magnitude of the velocity and in the opposite direction, the total work done by the propelling force in time interval $[0, T]$ is shown in (4.3). It equals the dissipation of energy due to friction (friction times the distance traveled) plus the total change of kinetic energy of the disk:

$$\int_0^T \beta S \|v(t)\|^2 \, dt + \frac{1}{2} m \left( v^2(T) - v^2(0) \right),$$

where $\beta$ is the viscosity of the medium in which the disk is moving, $S$ is the equivalent cross-sectional area of the disk, and $m$ is its mass.

The trajectory of the disk is subject to geometric constraints, i.e. the distance between the center of the disk and the observation equals the radius $r$. Assuming that there is no observational noise, we can summarize the constraints as

$$\|p_j - c(t_j)\| = r, \quad j = 0, \ldots, N.$$  \hspace{1cm} (4.4)

Moreover, for a realistic dynamical system, the acceleration is closely related to the propelling force, which is usually bounded. We address this constraint by imposing an upper-bound on the magnitude of the acceleration:

$$\|\dot{v}(t)\| \leq a_{\text{max}}, \quad \forall t \in [0, T].$$  \hspace{1cm} (4.5)

Therefore, the trajectory reconstruction problem can be summarized as an optimization with (4.3)
as the objective and (4.1), (4.4) and (4.5) as constraints:

\[
\begin{align*}
\min_{c(t), r} & \quad \int_0^T \|v(t)\|^2 dt + \frac{m}{2\beta S} v^2(t) \bigg|_0^T \\
\text{s.t.} & \quad \|p_j - c(t_j)\| = r, \quad j = 0, \ldots, N \\
& \quad \|\dot{v}(t)\| \leq a_{\max} \\
& \quad v(t) = \dot{c}(t). 
\end{align*}
\]

(4.6)

It is rather difficult to solve (4.6) directly due to the non-convexity in both the objective and the constraints. In some cases, \(a_{\max}\) is unknown, which makes the problem even more difficult. Therefore, we first assume that \(a_{\max}\) is known, and also adjust both the objective and the constraints so that the problem is tractable. The value of \(a_{\max}\) will be estimated later on.

### 4.3.2 Convex Relaxation

The objective (4.6) is non-convex due to the second term \(v^2(t)|_0^T\). If we further assume that the trajectory is long enough and the friction is sufficiently strong, then the change of kinetic energy is negligible compared to the frictional dissipation, and hence the second term can be neglected.

The geometric constraint (4.4), on the other hand, is also non-convex. To make it convex, we relax this equality to an inequality as follows:

\[
\|p_j - c(t_j)\| \leq r, \quad j = 0, \ldots, N.
\]  

(4.7)

In other words, instead of requiring every observation to fall strictly onto the surface of the disk, we allow them to fall onto the surface and into the interior of the disk.

### 4.3.3 Discretization

To avoid infinite dimensionality of the continuous time domain, we discretize the time interval and represent both \(v(t)\) and \(\dot{v}(t)\) in terms of \(c(t)\).

For simplicity, we assume that \(c(t)\) is evenly sampled with time interval \(\Delta t\), and for every \(M\) samples, we have an observation \(p_j\) that corresponds to \(c_{jM}\). To be more specific,

\[
\Delta t = \frac{T}{MN},
\]  

(4.8)
and the sample times are
\[ t_j = jM\Delta t, \quad j = 0, \ldots, N. \] (4.9)

Now for \( i = 1, \ldots, MN \), define, \( c(i\Delta t) = c_i \), \( v(i\Delta t) = v_i \), and \( \dot{v}(i\Delta t) = \dot{v}_i \). Then, on using the approximations
\[ v_i \approx \frac{c_{i+1} - c_i}{\Delta t} \] (4.10)
and
\[ \dot{v}_i \approx \frac{c_{i+1} + c_{i-1} - 2c_i}{\Delta t^2}, \] (4.11)
we can represent everything in terms of \( c_i \). On denoting \( \mathcal{I} = \{0, \ldots, MN - 1\} \) and \( \mathcal{J} = \{0, \ldots, N\} \), the convex-relaxed, discretized optimization problem can thus be approximated as
\[
\min_{c_i, i \in \mathcal{I}} \sum_{i \in \mathcal{I}} \|c_{i+1} - c_i\|^2 \quad \text{(4.12)}
\]
\[
\text{s.t.} \quad \|p_j - c_{jM}\| \leq r, j \in \mathcal{J} \quad \text{(4.13)}
\]
\[
\|c_{i+1} + c_{i-1} - 2c_i\| \leq L, i \in \mathcal{I}\setminus\{0\}, \quad \text{(4.14)}
\]

where \( L = a_{\max}\Delta t^2 \). Note that given \( L \) or \( a_{\max} \), problem (4.12), with its constraints, is a convex problem, which can be readily solved (or proved to have no feasible solution). Nonetheless, we are left with the remaining problem of finding a reasonable \( a_{\max} \), which we discuss next.

### 4.3.4 Estimation of \( a_{\max} \)

The physical interpretation of \( a_{\max} \) is simple: it is the maximum magnitude of the acceleration of the rigid body. If the value of \( a_{\max} \) is too small, there is no feasible solution to (4.12), i.e., to satisfy geometric constraints, the rigid body is required to have enough agility.

On the other hand, a well designed/evolved system should be efficient in the sense that its maximum output power should not be larger than necessary for its environment. Therefore, for the set of values of \( a_{\max} \) with feasible solutions, we generally prefer the smallest one.

As a result, to estimate the value of \( a_{\max} \), we simply choose the minimum value of \( a_{\max} \) such that (4.12) has a feasible solution. This value makes the rigid body agile enough to satisfy the geometric constraints, but not over-provisioned.

In order to find the minimum feasible value of \( a_{\max} \), we first find any value \( a_0 \) that gives (4.12) a feasible solution as the upper bound of \( a_{\max} \), and choose 0 as the lower bound of \( a_{\max} \). Then, we employ bisection search to find the desirable \( a_{\max} \) with reasonable accuracy \( \epsilon \). The algorithm is
Algorithm 6: Bisection Search for $a_{\text{max}}$

\[
\overline{a} \leftarrow a_0
\]
\[
a \leftarrow 0
\]
\[
\text{while } |\overline{a} - a| > \epsilon \text{ do}
\]
\[
a_{\text{max}} = (\overline{a} + a)/2
\]
\[
\text{if (4.12) with } a_{\text{max}} \text{ has feasible solution then}
\]
\[
\overline{a} \leftarrow a_{\text{max}}
\]
\[
\text{else}
\]
\[
a \leftarrow a_{\text{max}}
\]
\[
\text{end}
\]
\[
\text{end}
\]
\[
a_{\text{max}} \leftarrow a
\]

summarized as in Algorithm 6.

The energy minimization problem is then solved with the chosen $a_{\text{max}}$. The efficacy of this algorithm will be demonstrated in the simulation section.

4.4 Trajectory Recovery for a Rod with Known Shape Parameters

In this section, we assume that the rigid body is a rod of length $L$ and width $W$, with $W \ll L$. The observations $p_j, j \in J$ are assumed to be on either end of the rod, as shown in Fig. 4.4. The rod trajectory recovery problem appears to be simple. However, it is in fact more complex compared to the sphere case, due to the introduction of rotation and orientation of the rod.

![Figure 4.4: Trajectory recovery problem for a rod.](image)

4.4.1 Problem Formulation

To formulate the energy minimization problem, we assume that the energy dissipation can be divided into two parts, with one related to shift and the other related to rotation of the rod. With the same
assumption as the disk case, the first part of the energy dissipation can be described as

$$\int_0^T \beta S_{\theta,v} \|v(t)\|^2 dt, \quad (4.15)$$

where $\beta$ is the viscosity of the medium in which the rod is moving, and $S_{\theta,v}$ is the equivalent cross-sectional area of the rod. $S_{\theta,v}$ is approximated as (shown in the right hand graph of Fig. 4.5)

$$S_{\theta,v} = W |\cos \alpha| + L |\sin \alpha|, \quad (4.16)$$

where $\alpha$ is the angle between the center velocity of the rod $v(t)$ and the orientation of the rod $\theta(t)$. $\alpha$ can be calculated as

$$\alpha = \cos^{-1} \left| \frac{v}{\|v\|} \cdot \theta \right|. \quad (4.17)$$

Intuitively, $S_{\theta,v}$ achieves its maximum when the direction of the center velocity is perpendicular to the orientation of the rod, in which case the energy dissipation is the greatest (shown in the left hand graph of Fig. 4.5); On the other hand, $S_{\theta,v}$ is small when the direction of the center velocity agrees with the orientation of the rod, in which case the energy dissipation is much smaller (shown in the middle graph of Fig. 4.5).

![Figure 4.5](image)

Figure 4.5: Left: the direction of the center velocity is perpendicular to the orientation of the rod, in which case $S_{\theta,v} = L$. Middle: the direction of the center velocity is the same as the orientation of the rod, in which case $S_{\theta,v} = W$. Right: illustration of the calculation of $S_{\theta,v}$, where $S_{\theta,v} = W |\cos \alpha| + L |\sin \alpha|$.

The second part of the energy dissipation is related to the rotation of the rod. On assuming that the torque of the resistance is proportional to the angular velocity, this part of the energy dissipation can be expressed as

$$\int_0^T \omega^2(t) dt. \quad (4.18)$$

Now that we have both parts of the energy dissipation, by ignoring the change of kinetic energy
as in the disk case, the optimization problem can be summarized as follows:

\[
\begin{align*}
\min_{c(t), \theta(t)} & \quad \int_0^T S_{\theta, v} \|v(t)\|^2 dt + \lambda \int_0^T \omega^2(t) dt \\
\text{s.t.} & \quad p_j = c(t_j) + \frac{L u}{2} \theta(t_j), \quad j \in J \\
& \quad |u| = 1 \\
& \quad \nabla(t) = \dot{c}(t) \\
& \quad \omega(t) = \|\dot{\theta}(t)\| \\
& \quad \|\dot{\theta}(t)\| = 1,
\end{align*}
\]

(4.19)

where \(\lambda\) is a tuning parameter. Constraints (4.20) and (4.21) are the geometric constraints (i.e. the observations \(p_j, j = 0, \ldots, N\) be on either end of the rod).

It is now evident from (4.19) that the problem becomes highly non-convex due to the introduction of rotation and orientation of the rod into the model. Even if we relax the geometric constraints as in the disk case, such that the observations can locate on the rod (constraint (4.21) becomes \(|u| \leq 1\)), and we over-simplify the problem such that the equivalent cross-sectional area \(S_{\theta, v}\) is considered constant, the optimization problem (4.22) is still non-convex due to the constraint (4.24):

\[
\begin{align*}
\min_{c(t), \theta(t)} & \quad \int_0^T S \|v(t)\|^2 dt + \lambda \int_0^T \omega^2(t) dt \\
\text{s.t.} & \quad p_j = c(t_j) + \frac{L u}{2} \theta(t_j), \quad j \in J \\
& \quad |u| \leq 1 \\
& \quad \nabla(t) = \dot{c}(t) \\
& \quad \omega(t) = \|\dot{\theta}(t)\| \\
& \quad \|\dot{\theta}(t)\| = 1.
\end{align*}
\]

(4.22)

There are no easy ways to get around this constraint since (4.24) transforms into a series of constraints after discretization, and techniques such as relaxation, considering the dual problem, or iterative projection do not apply here. As a result, we seek a numerical method, simulated annealing [27], to address this energy minimization problem.
4.4.2 Simulated Annealing

Simulated annealing is a numerical optimization method inspired by a physical process, annealing in metallurgy, which involves heating and slow cooling of a material, to achieve a state with lower potential.

In the simulated annealing method, each state \( S \) of the space we search on is analogous to a state of the physical system, and \( E(S) \), the function to be minimized, is analogous to the potential of the system in that state. At each iteration, the algorithm proposes a random neighboring state \( S' \) of the current state \( S \), and replacing \( S \) with \( S' \) according to a probability \( P \) associated with \( E(S), E(S') \) and a gradually decreasing temperature \( T \). The final objective is to bring the system to a state \( S_{\text{min}} \) with minimum possible potential \( E(S_{\text{min}}) \). The basic simulated annealing algorithm is illustrated in Algorithm 7, where the function \( \text{uniform}(a, b) \) outputs a number drawn from a uniform distribution between \( a \) and \( b \).

**Algorithm 7: Basic Simulated Annealing Algorithm**

<table>
<thead>
<tr>
<th>Data: Search space of the optimization problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: State ( S_{\text{end}} ) and the corresponding energy ( E_{\text{end}} ) achieved at the end of the annealing process</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\kappa & \leftarrow 0; \ T \leftarrow \text{temperature}(\kappa) \\
S & \leftarrow S_0 \\
\text{while} \ \kappa < \kappa_{\text{max}} \ \text{do} \\
& \quad S' \leftarrow \text{neighbor}(S) \\
& \quad \text{if} \ E(S') > E(S) \ \text{then} \\
& \qquad \text{if} \ P(E(S), E(S'), T) > \text{uniform}(0, 1) \ \text{then} \\
& \qquad \quad S \leftarrow S' \\
& \qquad \text{end} \\
& \text{else} \\
& \quad S \leftarrow S' \\
& \text{end} \\
& \quad \kappa \leftarrow \kappa + 1; \ T \leftarrow \text{temperature}(\kappa) \\
\text{end} \\
S_{\text{end}} & \leftarrow S; \ E_{\text{end}} \leftarrow E(S)
\end{align*}
\]

Simulated annealing can be easily adapted to solve our problem. For our rod trajectory recovery problem, a state \( S \) is defined as an arrangement of positions and orientations of the rod at each time stamp \( t_j, j \in J \) such that \( p_j \) is on the rod. Note that we are using the relaxed geometric constraint \( 4.23 \) instead of \( 4.21 \) so that neighboring states would not be too far away from each other in the perturbation stage, which will be discussed later.

We initialize the algorithm with a random state \( S_0 \). Denote the positions of the two ends of the rod at time \( t_j \) as \( x_{1,j} \) and \( x_{2,j} \). At each time stamp \( t_j, j \in J \), a random angle \( \zeta_j \) is chosen according
to uniform distribution to represent the orientation $\theta(t_j) = (\cos \zeta_j, \sin \zeta_j)$ of the rod at time $t_j$. The rod is then placed on the line with angle $\zeta_j$ passing through $p_j$, such that $x_{1,j}$ is $\gamma_j L$ away from $p_j$, $\gamma_j \in [0, 1]$. The initialization stage is summarized in Algorithm 8.

**Algorithm 8: Simulated Annealing for Rod: Initialization**

**Data:** Observations $p_j$, $j \in J$ and rod length $L$

**Result:** Initial state $S_0$

begin
    for $j \leftarrow 0$ to $N$ do
        $\zeta_j \leftarrow \text{uniform}(-\pi, \pi)$
        $\gamma_j \leftarrow \text{uniform}(0, 1)$
        $\theta(t_j) \leftarrow (\cos \zeta_j, \sin \zeta_j)$
        $x_{1,j} \leftarrow p_j - \gamma_j L \theta(t_j)$
        $x_{2,j} \leftarrow p_j + (1 - \gamma_j) L \theta(t_j)$
    end
end

The energy function $E(S)$ is calculated according to the objective function (4.19). The center velocity $v(t_j)$ and angular velocity $\omega(t_j)$ are calculated as

$$v(t_j) = \frac{1}{\Delta t}(c(t_j + 1) - c(t_j)) = \frac{1}{2}(v_{1,j} + v_{2,j}), \; j \in J \backslash \{N\},$$

(4.25)

and

$$\omega(t_j) = \frac{2}{L}\|v_{1,j} - v(t_j)\|, \; j \in J \backslash \{N\},$$

(4.26)

where

$$v_{1,j} = \frac{1}{\Delta t}(x_{1,j+1} - x_{1,j}), \; j \in J \backslash \{N\}$$

(4.27)

and

$$v_{2,j} = \frac{1}{\Delta t}(x_{2,j+1} - x_{2,j}), \; j \in J \backslash \{N\}.$$  

(4.28)

The equivalent cross-sectional area $S_{\theta,v}(t_j)$ is obtained as follows:

$$S_{\theta,v}(t_j) = W|\cos \alpha_j| + L|\sin \alpha_j|,$$

(4.29)

where

$$\alpha_j = \cos^{-1} \left( \frac{v(t_j)}{\|v(t_j)\| : \theta(t_j)} \right).$$

(4.30)
Finally, the objective function (potential) $E(S)$ is obtained by

$$
E(S) = \sum_{j \in \mathcal{J}\setminus\{N\}} S^{\theta(t_j)} \|\mathbf{v}(t_j)\|^2 + \lambda \omega(t_j)^2.
$$

(4.31)

Now that we have the states $S$ and the potential $E(S)$ at hand, we further specify the method of obtaining a random neighboring state $S'$ from $S$. Recall that state $S$ is defined as an arrangement of positions and orientations of the rod at each time stamp $t_j, j \in \mathcal{J}$. We can use a different point of view here. The state $S$ can be viewed as if there are $N + 1$ different rods, present at the same time, with rod $j$ being at the same position and orientation as the actual rod at time $t_j, j \in \mathcal{J}$. With this interpretation, $S'$ is obtained from $S$ in a way that each rod $j$, with probability $p_{ptb}$, is perturbed according to a randomized rule. The perturbation of rod $j$ consists of two parts, rotation around $\mathbf{p}_j$ for a random angle $\Delta \zeta_j$, and shift for a random amount while keeping $\mathbf{p}_j$ on the rod. After the action (or inaction) of each rod $j$, the candidate state $S'$ is reached. The detailed procedure of generating a neighboring state $S'$ from $S$ is illustrated in Algorithm 9, where the function $\text{Laplace}(\mu, \sigma)$ outputs a number drawn from a Laplace distribution with mean $\mu$ and variance $2\sigma^2$:

$$
f(x|\mu, \sigma) = \frac{1}{2\sigma} \exp \left( -\frac{|x - \mu|}{\sigma} \right).
$$

(4.32)

**Algorithm 9: Simulated Annealing for Rod: Generate Neighboring State $S'$ from $S$**

Data: Current state $S$; parameters for rotation and shift, $\sigma_\zeta$ and $\sigma_\gamma$.

Result: Neighboring state $S'$

begin

begin

for $j \leftarrow 0$ to $N$

if $p_{ptb} > \text{uniform}(0, 1)$ then

$\Delta \zeta_j \leftarrow \text{Laplace}(0, \sigma_\zeta)$ \\
$\zeta_j \leftarrow \zeta_j + \Delta \zeta_j$ \\
while $\zeta_j > \pi$ do $\zeta_j \leftarrow \zeta_j - 2\pi$ \\
while $\zeta_j < -\pi$ do $\zeta_j \leftarrow \zeta_j + 2\pi$ \\
$\theta(t_j) \leftarrow (\cos \zeta_j, \sin \zeta_j)$

$\Delta \gamma_j \leftarrow \text{Laplace}(0.5, \sigma_\gamma)$ \\
$\gamma_j \leftarrow \gamma_j + \Delta \gamma_j$ \\
while $\gamma_j > 1$ do $\gamma_j \leftarrow \gamma_j - 1$ \\
while $\gamma_j < 0$ do $\gamma_j \leftarrow \gamma_j + 1$ \\
$x_{1,j} \leftarrow \mathbf{p}_j - \gamma_j L\theta(t_j)$ \\
$x_{2,j} \leftarrow \mathbf{p}_j + (1 - \gamma_j) L\theta(t_j)$

end

end

end

The candidate neighboring state $S'$ is accepted if the energy decreases, i.e., $E(S') < E(S)$.
Otherwise, if the energy increases, the state $S'$ is accepted with probability

$$P(E(S), E(S'), T) = \exp \left( -\frac{E(S') - E(S)}{T} \right), \quad (4.33)$$

with $T = C/\kappa$, where $C$ is a constant and $\kappa$ is the current iteration of the simulated annealing algorithm.

With the elements described, the simulated annealing algorithm can be readily implemented, the efficacy of which will be demonstrated in the simulation section.

### 4.5 Trajectory Recovery for an Ellipse with Known Shape Parameters

In this section, we consider the trajectory recovery problem for an ellipse with semi-major length $a$ and semi-minor length $b$. The observations $p_j, j \in J$ are assumed to be on the surface of the ellipse, as shown in Fig. 4.2.

#### 4.5.1 Problem Formulation

The ellipse trajectory recovery problem is quite similar to the rod case, in the sense that they have the same degrees of freedom. As a result, with the same assumptions at hand, we formulate the energy minimization problem the same way and obtain the same objective function as (4.19):

$$\min_{c(t), \theta(t)} \int_0^T S_{\theta, \nu} \|v(t)\|^2 dt + \lambda \int_0^T \omega^2(t) dt,$$

with $S_{\theta, \nu}$ approximated by

$$S_{\theta, \nu} = b|\cos \alpha| + a|\sin \alpha|, \quad (4.35)$$

However, the ellipse problem has a set of more complicated geometric constraints:

$$\|p_j - c_{1,j}\| + \|p_j - c_{2,j}\| = 2a, \; j \in J, \quad (4.36)$$

where $c_{1,j}$ and $c_{2,j}$ are the locations of the two foci of the ellipse, which can be further expressed in terms of the center location $c(t_j)$ and orientation $\theta(t_j)$:

$$c_{1,j} = c(t_j) + r\theta(t_j), \; j \in J, \quad (4.37)$$
\[ c_{2,j} = c(t_j) - r\theta(t_j), \quad j \in J, \]

where \( r = \sqrt{a^2 - b^2}. \)

Thus, the energy minimization problem for an ellipse can be formulated as follows:

\[
\min_{c(t), \theta(t)} \int_0^T S_{\theta, v} |v(t)|^2 dt + \lambda \int_0^T \omega^2(t) dt \tag{4.39}
\]

s.t.

\[
S_{\theta, x} = b|\cos \alpha| + a|\sin \alpha| \\
\alpha = \cos^{-1}\left( \frac{v \cdot \theta}{||v||} \right) \\
||p_j - c(t_j) - r\theta(t_j)|| + ||p_j - c(t_j) + r\theta(t_j)|| = 2a \\
v(t) = \dot{c}(t) \\
\omega(t) = ||\dot{\theta}(t)|| \\
||\theta(t)|| = 1,
\]

Similar to the rod case, the non-convex constraints drive us to turn again to a numerical method, specifically simulated annealing (Algorithm 7), to tackle the energy minimization problem (4.39).

### 4.5.2 Simulated Annealing

For the ellipse trajectory recovery problem, a state \( S \) is defined as an arrangement of positions and orientations of the ellipse at each time stamp \( t_j, j \in J \) such that \( p_j \) is on the surface of or inside the ellipse. The geometric constraint (4.40) is again relaxed to

\[
||p_j - c(t_j) - r\theta(t_j)|| + ||p_j - c(t_j) + r\theta(t_j)|| \leq 2a \tag{4.40}
\]

The simulated annealing algorithm is initialized with state \( S_0 \). At each time stamp \( t_j, j \in J \), the ellipse is placed at a random orientation \( \theta(t_j) = (\cos \zeta_j, \sin \zeta_j) \) according to a uniform distribution in angle \( \zeta_j \), with the observations \( p_j, j \in J \) at the center of the ellipse. Then the ellipse of each time stamp \( t_j, j \in J \) is perturbed with \( p_{ptb} = 1 \) using the technique discussed next, and state \( S_0 \) is thereby obtained.

The major difference between the simulated annealing algorithm for an ellipse and that of the rod is the method of obtaining a random neighboring state \( S' \) from \( S \), since the ellipse problem has a more complicated geometric constraint. To better explain this procedure, we again view the state \( S \) as if there are \( N + 1 \) different ellipses, presenting at the same time, with ellipse \( j \) being at the
Algorithm 10: Simulated Annealing for Ellipse: Generate Neighboring State $S'$ from $S$

**Data:** Current state $S$; maximum rotation angle $\Delta \zeta_{\text{max}}$, $\Delta \zeta_{\text{max}} \leq \pi$; maximum shift amount $\Delta c_{\text{max}}$, $\Delta c_{\text{max}} \leq 2b$

**Result:** Neighboring state $S'$

begin
  for $j \leftarrow 0$ to $N$ do
    if $p_{ptb} > \text{uniform}(0,1)$ then
      $S_{j,old} \leftarrow$ current state of ellipse $j$
      $\xi_j \leftarrow \text{uniform}(-\pi, \pi)$
      $\gamma_j \leftarrow \text{Laplace}(0, \sigma_\gamma)$
      while $\gamma_j > 1$ do $\gamma_j \leftarrow \gamma_j - 1$
      while $\gamma_j < -1$ do $\gamma_j \leftarrow \gamma_j + 1$
      $\Delta c_j \leftarrow (2a\gamma_j \cos \xi_j, 2a\gamma_j \sin \xi_j)$
      $c(t_j) \leftarrow c(t_j) + \Delta c_j$
      $\Delta \zeta_j \leftarrow \text{Laplace}(0, \sigma_\zeta)$
      $\zeta_j \leftarrow \zeta_j + \Delta \zeta_j$
      while $\zeta_j > \pi$ do $\zeta_j \leftarrow \zeta_j - 2\pi$
      while $\zeta_j < -\pi$ do $\zeta_j \leftarrow \zeta_j + 2\pi$
      $\theta(t_j) \leftarrow (\cos \zeta_j, \sin \zeta_j)$
      $c_{1,j} = c(t_j) - r\theta(t_j)$  // test stage
      $c_{2,j} = c(t_j) + r\theta(t_j)$
      if $||p_j - c_{1,j}|| + ||p_j - c_{2,j}|| > 2a$ then
        recover state of rod $j$ to $S_{j,old}$
      end
    end
  end
end
same position and orientation as the actual ellipse at time $t_j$, $j \in J$. Each ellipse $j$ is perturbed with probability $p_{ptb}$ to generate the neighboring state $S'$. The ellipse perturbation contains three stages. The ellipse is first shifted along a random angle with a random amount on the first stage. Then on the second stage, the ellipse is rotated around its center for a random angle. After the first two stages, there is a probability that the observation $p_j$ is no longer on or inside the ellipse. Therefore, on the last stage, we test whether or not $p_j$ satisfies (4.40). If not, the ellipse is recovered to its original position and orientation. After the action (or inaction) of each ellipse $j$, the candidate state $S'$ is generated. The detailed procedure is illustrated in Algorithm 10.

The calculation of the potential $E(s)$ is similar to the rod case since they share the same form of the energy function. The two foci $c_{1,j}$ and $c_{2,j}$ are used instead of the two ends of the rod, $x_{1,j}$ and $x_{2,j}$, to obtain the center velocity and angular velocity:

$$v(1_j) = \frac{1}{2}(v_{1,j} + v_{2,j}), \quad j \in J \setminus \{N\} \quad (4.41)$$

and

$$\omega(t_j) = \frac{1}{r} \|v_{1,j} - v(t_j)\|, \quad j \in J \setminus \{N\}, \quad (4.42)$$

where

$$v_{1,j} = \frac{1}{\Delta t}(c_{1,j+1} - c_{1,j}), \quad j \in J \setminus \{N\} \quad (4.43)$$

and

$$v_{2,j} = \frac{1}{\Delta t}(c_{2,j+1} - c_{2,j}), \quad j \in J \setminus \{N\}. \quad (4.44)$$

The choice of using the two foci is consistent with our model, if we consider two extreme cases of the ellipse problem. When the eccentricity of the ellipse is close to 1, the shape of the ellipse becomes similar to a rod. In this case, the locations of the two foci are very close to the two ends of the ellipse semi-major axis, which is consistent with the rod model. On the other hand, when the eccentricity is zero, i.e., when the ellipse degenerates to a disk, the two foci coincide with the center of the disk, which is consistent with the fact that the rotation of the disk is ignored.

The temperature function and the acceptance probability are set to the same values as in the rod case. With these choices made, the simulated annealing algorithm can be readily implemented, the efficacy of which are demonstrated in the simulation section.
4.6 Trajectory Recovery with Unknown Shape Parameters

In the previous sections, we have discussed the trajectory recovery problem for a disk, a rod and an ellipse with known shape parameters. It is of interest to explore the cases in which the shape parameters are unknown. Combining the trajectory reconstruction and shape parameter estimation together creates an interesting yet challenging problem. Simply setting the shape parameters as variables and solving the same optimization problem is not viable in this case, and more sophisticated algorithms need to be developed.

4.6.1 Trajectory Recovery for a Disk with Unknown Shape Parameters

First, we consider the problem of trajectory recovery for a disk with unknown radius $r$. The problem setting is the same as in Section 4.3, except that the radius $r$ of the disk is unknown and needs to be estimated in this case.

**Optimizing $r$**

Recall that in Section 4.3, by relaxing the original geometric constraints from (4.4) to (4.7), we successfully convert (4.6) into a convex optimization problem. However, with $r$ as a variable, the problem loses its convexity and hence cannot be easily solved. If we optimize both $r$ and $c_j$ in (4.12) simultaneously, the optimal value is achieved when $r = \infty$. In this situation, the rigid body is infinitely large so that every observation lies inside the disk and the center does not need to move at all. This result is obviously undesirable.

We want to optimize $r$ while incorporating the original constraints (4.4). To solve this problem, we borrow the idea of validation from machine learning. Notice that $r$ is actually a model parameter that needs to be selected for a best learning result (the trajectory). If we could find a way to test the performance of different trajectories obtained under different assumptions of $r$, then we can select the optimal $r$ as the final shape parameter.

We want to optimize $r$ while incorporating the original constraints (4.4). To solve this problem, we borrow the idea of validation from machine learning. Notice that $r$ is actually a model parameter that needs to be selected for a best learning result (the trajectory). If we could find a way to test the performance of different trajectories obtained under different assumptions of $r$, then we can select the optimal $r$ as the final shape parameter.

Note that the geometric constraint (4.4) is very difficult to incorporate into the optimization stage, but it is very easy to be substituted and tested. Therefore, we can hold out part of the observations $p_j$ with indices $j \in V$ as a validation set, and use the geometric deviation

$$D_V(r) = \frac{1}{|V|} \sum_{j \in V} (\|p_j - \hat{c}_{j,M}\| - r)^2$$

(4.45)

as a measure of the performance of the trajectory $\hat{c}_i$, $i \in I$ obtained by solving (4.12) with the
geometric constraint (4.13) replaced by

\[ \| \mathbf{p}_j - \mathbf{c}_{jM} \| \leq r, j \in \mathcal{J} \setminus \mathcal{V}. \]  

(4.46)

Eventually, the value of \( r \) is determined by

\[ r^* = \arg \min_r \mathcal{D}_V(r). \]  

(4.47)

Intuitively, \( \mathcal{D}_V(r) \) is a reasonable measure for determining the value of \( r \). When \( r \) is under-estimated, most of the observations in the test set fall outside the disk \( \mathcal{B}_r(\mathbf{c}_{jM}), j \in \mathcal{V} \), and hence result in a large value of \( \mathcal{D}_V(r) \). On the other hand, if \( r \) is over-estimated, then most of the observations in the test set fall inside the disk \( \mathcal{B}_r(\mathbf{c}_{jM}), j \in \mathcal{V} \), and hence also result in large \( \mathcal{D}_V(r) \). This effect will be better demonstrated in the simulation section. The detailed algorithm is illustrated in Algorithm 11.

**Algorithm 11:** Trajectory recovery for a disk with unknown radius \( r \)

**Data:** Set of observations \( \mathcal{P} = \{ \mathbf{p}_j: j \in \mathcal{J} \} \)

**Result:** Optimal shape parameter \( r^* \); trajectory estimation \( \hat{\mathbf{c}}^*_i, i \in \mathcal{I} \)

**begin**

Divide set \( \mathcal{P} \) into training set \( \mathcal{T} \) and validation set \( \mathcal{V} \)

Generate a set \( \mathcal{R} \) of candidate radius \( r \)

**foreach** \( r \in \mathcal{R} \) **do**

\[ \hat{\mathbf{c}}_i, i \in \mathcal{I} \leftarrow \text{solution of optimization (4.12) with constraints (4.14) and (4.46) on } \mathcal{T} \]

using current \( r \)

\[ \mathcal{D}_V(r) \leftarrow \frac{1}{|\mathcal{V}|} \sum_{j \in \mathcal{V}} (\| \mathbf{p}_j - \hat{\mathbf{c}}_{jM} \| - r)^2 \]

**end**

\[ r^* \leftarrow \arg \min_r \mathcal{D}_V(r) \]

\[ \hat{\mathbf{c}}^*_i, i \in \mathcal{I} \leftarrow \text{solution of optimization (4.12) with constraints (4.13) and (4.14) on } \mathcal{P} \text{ using current } r^* \]

**end**

Note that we have not specified how to divide the data set into the training set and validation set, or how to generate a candidate radius \( r \), both of which will be discussed next.

**Cross Validation Approach**

If the number of observations is sufficient, we can simply use part of them as the validation set and the rest as the training set to solve (4.12). However, when the total number of observations is relatively small, we may resort to \( k \)-fold cross validation [29], and choose the \( r \) that generates, on average, the smallest value of geometric deviation. More specifically, the optimal value \( r^* \) is given
by
\[ r^* = \arg \min_r \sum_{j=1}^{k} D_{V_j}(r), \quad (4.48) \]
where \( V_j \) is the \( j \)th validation set. As specified by \( k \)-fold cross validation, \( V_j, j = 1, \ldots, k \) is an equal partition of \( \mathcal{J} \).

### Generating Candidate Radius \( r \)

If there is no special structure for \( D_V(r) \), the best thing to do is a full search to find the optimal value of \( r \). Unfortunately, for each value of \( r \), fully estimating \( D_V(r) \) is a computationally intensive process. Fortunately, in the simulation section, we will see empirically that \( D_V(r) \) appears to be a convex function. Therefore, we can use the surrogate algorithm [8] or Brent algorithm [10] to expedite the search for the optimal value of \( r \) and reduce the total computation required. This is extremely useful when we have more than one shape parameter and have to search for the optimal shape model in a higher dimensional space.

### 4.6.2 Trajectory Recovery for a Rod or an Ellipse with Unknown Shape Parameters

The cross validation method used to estimate the radius \( r \) of a disk can be naturally applied to the problems of estimating shape parameters of a rod or an ellipse. To reconstruct the trajectory and shape parameters simultaneously for both a rod and an ellipse, a validation set \( V \) is first held out. Then we assume different values of parameter pairs, \((W,L)\) in the rod case, and \((a,b)\) in the ellipse case. For each parameter pair, the simulated annealing algorithm is employed to obtain the estimated trajectory. Finally, we validate the performance of each trajectory on \( V \) and choose the parameter pair that has the smallest geometric deviation as the final model, based on which the trajectory is reconstructed.

The procedure described above is a viable way of solving the trajectory recovery problem for a rod and an ellipse with unknown shape parameters. However, it is computationally challenging. We have to search in a two-dimensional space for an optimal pair of shape parameters, and for each pair, the trajectory recovery problem is solved via simulated annealing, which itself is computationally intensive.
4.7 Simulation

In this section, the algorithms proposed in the previous sections are tested on synthetic data. A dynamical model is used to generate the center and orientation of the rigid body and observations are obtained from its surface. The proposed algorithms are then implemented to reconstruct the trajectory/shape of the rigid body, the results of which are compared to the true trajectory/shape generated by the dynamical model.

The trajectory recovery algorithms for a disk with known/unknown radius, a rod and an ellipse with known shape parameters are tested. The test cases of trajectory recovery for a rod and an ellipse with unknown shape parameters are not covered in the simulation, due to the intensiveness of the computation.

4.7.1 Disk Trajectory Recovery with Known Shape Parameter

System Specification

First we design a dynamical model to generate the trajectory of the disk.

Suppose there is a disk $\rho$ of mass $m = 1$ and radius $r = 0.8$. It moves in a medium with damping proportional to line speed. We neglect the rotation of $\rho$. In addition, an external force $f(t)$ propels $\rho$, and hence the speed of the center of $\rho$ is given by

$$\dot{v}(t) = -\alpha v(t) + \frac{1}{m} f(t), \quad (4.49)$$

where $\alpha$ is a damping constant determined by the cross section of $\rho$, mass $m$ and the viscosity $\beta$. In our simulation, $\alpha$ is set to 0.4. Moreover, the initial condition and the propelling force of $\rho$ are as shown in Table 4.1.

<table>
<thead>
<tr>
<th>$c(0)$</th>
<th>$v(0)$</th>
<th>$f(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(-2, -2)$</td>
<td>$(1, -1)$</td>
<td>$(2 \cos(t), \sin(t/2))$</td>
</tr>
</tbody>
</table>

Table 4.1: Initial condition and propelling force for disk trajectory recovery.

Second, we need to obtain observations from the surface of the disk. Assume that the observations are collected distributively by four sensors. The four sensors are located at $(10, 0)$, $(0, 10)$, $(-10, 0)$ and $(0, -10)$. To run our algorithm, we require these sensors to collect the coordinates of the point on the surface of $\rho$ on the line connecting the sensor and the center of $\rho$. The sampling process is shown in the left hand graph of Fig. 4.6, in which the observation $p_j$ lies on the line formed from
the center \( c(t_j) \) to the sensor.

We assume that at each sampling time \( t = jM \Delta t, j \in J \), one of the four sensors is randomly selected to take an observation \( p_j \). In our simulation \( M = 4, N = 50, \text{ and } \Delta t = 0.05 \). The trajectory of the center of \( \rho \) and the observations are shown in the right hand graph of Fig. 4.6.

![Diagram of sensor and observations](image)

**Figure 4.6:** Left: A sensor collecting a surface observation. Right: the trajectory of the center of the disk and the observations collected by the four sensors.

### Simulation Result

With the simulation settings described above, we implemented the algorithm described in Section 4.3 to solve the optimization problem (4.12). We used CVX, a package for specifying and solving convex programs [21, 22]. The obtained results are shown in Fig. 4.7.

![Trajectory recovery result](image)

**Figure 4.7:** Trajectory recovery result for a disk with known shape parameter \( r \) on synthetic data generated by a dynamical model.

Fig. 4.7 shows that our algorithm can reconstruct the trajectory accurately. The reconstructed trajectory is almost identical to the true trajectory generated by the dynamical model, which demonstrates the efficacy of our algorithm.
4.7.2 Rod Trajectory Recovery with Known Shape Parameter

System Specification

First we specify the dynamical model used to generate the trajectory of the rod. Suppose the rod \( \rho \) is of mass \( m = 1 \), width \( W = 0.2 \) and length \( L = 2.0 \).

For simplicity, we assume that the orientation of the rod always aligns with the velocity of the rod, i.e., \( \mathbf{v}(t) \times \theta(t) = 0 \), so that the mechanism of rotation does not need to be considered in the trajectory generation process. In addition, an external force \( f(t) \) propels \( \rho \), and hence the center velocity and orientation of \( \rho \) is given by

\[
\dot{\mathbf{v}}(t) = \frac{1}{m} (-\beta W \mathbf{v}(t) + f(t))
\]

and

\[
\theta(t) = \frac{\mathbf{v}(t)}{\| \mathbf{v}(t) \|},
\]

where \( \beta = 1 \). Moreover, the initial conditions and the propelling force of \( \rho \) are as shown in TABLE 4.2.

<table>
<thead>
<tr>
<th>( c(0) )</th>
<th>( v(0) )</th>
<th>( \theta(0) )</th>
<th>( f(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((-2, -2))</td>
<td>((0, 1))</td>
<td>((0, 1))</td>
<td>((2 \cos(t), \sin(t/2)))</td>
</tr>
</tbody>
</table>

Table 4.2: Initial condition and propelling force for rod trajectory recovery.

Now that the trajectory of the rod is obtained, 50 observations are collected from the rod. At each sampling time \( t = j\Delta t, j \in J \), an observation \( \mathbf{p}_j \) is taken from either end of the rod with equal probability, (as shown in (4.52)), where \( \Delta t = 0.05 \). The trajectory of the rod and the observations obtained are shown in Fig. 4.8. The lines represent the rod positions at time \( t_j, j \in J \), and the dots are the observations at each time:

\[
\mathbf{p}_j = \begin{cases} 
\mathbf{c}(t) + \frac{L}{2} \mathbf{\theta}(t) & \text{with probability } \frac{1}{2} \\
\mathbf{c}(t) - \frac{L}{2} \mathbf{\theta}(t) & \text{with probability } \frac{1}{2}.
\end{cases}
\]

Simulation Result

With the simulation settings described above, we implemented the simulated annealing algorithm discussed in Section 4.4 to solve the trajectory recovery problem for a rod with known shape parameters \((W = 0.2 \text{ and } L = 2.0)\). In particular, parameters in the perturbation stage are set to \( \sigma_c = \frac{\pi}{4} \),

\[
\]
\[ \sigma_\gamma = 0.25 \text{ and } p_{ptb} = 0.1 \text{ respectively. The value of } \lambda \text{ in } E(S) \text{ is set to } 0.6. \]

In our simulation, the constant \( C \) in the temperature function takes the value of \( 5 \times 10^4 \). A relatively large number is chosen so that the temperature decreases slowly and the algorithm is more likely to converge to a global minimum. We ran the algorithm 10 times (10 different random initializations), each time with \( 2.5 \times 10^7 \) iterations (\( \kappa_{\max} = 2.5 \times 10^7 \)), and the trajectory with the lowest energy function value among the 10 runs is chosen as the final result.

A typical trajectory recovery result is shown in Fig. 4.9. The black dots in the figure denote the observations given. Blue lines represent the true rod positions at sampling times, while the red lines represent the recovered rod. Fig. 4.9 shows that the trajectory recovery results are almost perfect, which demonstrates the effectiveness of our algorithm.
4.7.3 Ellipse Trajectory Recovery with Known Shape Parameter

System Specification

The dynamical model used to generate the trajectory of an ellipse is similar to the one utilized to generate the trajectory of a rod.

An ellipse $\rho$ of mass $m = 1$, semi-major length $a = 2$ and eccentricity $0.97$ (so that the semi-minor length $b$ is $0.486$) is considered in the simulation. Again, we assume that the orientation of $\rho$ always align with its velocity, and an external force $f(t)$ propels $\rho$. The center velocity and orientation of $\rho$ is given by

$$\dot{v}(t) = \frac{1}{m}(-2a\beta v(t) + f(t))$$

and

$$\theta(t) = \frac{v(t)}{\|v(t)\|},$$

where $\beta = 0.4$. Moreover, the initial conditions and the propelling force of $\rho$ are shown in TABLE 4.3.

<table>
<thead>
<tr>
<th>$c(0)$</th>
<th>$v(0)$</th>
<th>$\theta(0)$</th>
<th>$f(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-2, -2)</td>
<td>(1, 1)</td>
<td>(1, 1)</td>
<td>$(2 \cos(t), \sin(t/2))$</td>
</tr>
</tbody>
</table>

Table 4.3: Initial condition and propelling force for ellipse trajectory recovery.

Given the trajectory of $\rho$, obtaining the observations from an ellipse is more complicated than in the disk and rod cases. Assume that the observations are collected sequentially by four sensors $\{s_i\}_{i=1}^4$, which are located at $(15, 15)$, $(-15, 15)$, $(-15, -15)$ and $(15, -15)$ respectively. At each sampling time $t = j \Delta t, j \in \mathcal{J}$ and $t = 0.1$, sensor $s_i$ ($i = j \mod 4$), collects a point $p_j$ from the surface of the ellipse, in a way that $p_j$ is the point on the ellipse that has the smallest Euclidean distance from sensor $s_i$. $p_j$ is called the orthogonal contacting point of $s_i$ and can be calculated using a generalized Newton method [2]. The sampling process is shown in the left hand graph of Fig. 4.10. The trajectory of $\rho$ (blue ellipses) and the observations (red dots) are shown in the right hand graph of Fig. 4.10.

Simulation Result

With the simulation settings described above, we implemented the simulated annealing algorithm discussed in Section 4.5 to solve the trajectory recovery problem for an ellipse with known shape parameters ($a = 2$ and $b = 0.486$). In particular, parameters in the perturbation stage are set to
Figure 4.10: Left: A sensor collecting a surface observation. Right: Trajectory of the ellipse and observations collected by the four sensors.

\[ \sigma_\zeta = \frac{\pi}{4}, \quad \sigma_\gamma = 0.125 \text{ and } p_{\text{thb}} = 0.1, \text{ respectively.} \]  
The value of \( \lambda \) in \( E(S) \) is set to 5. In our simulation, the constant \( C \) in the temperature function again takes the value of \( 5 \times 10^4 \). The algorithm is run 10 times, each time with \( 1.5 \times 10^7 \) iterations, and the trajectory with the lowest energy function value among the 10 runs is chosen as the final result.

A typical ellipse trajectory recovery result is shown in Fig. 4.11. The black stars denote the observations given; the green ellipses are the true ellipse positions at sampling times, while the red ellipses are the reconstructed ellipse positions. Fig. 4.11 shows that the trajectory recovery results are almost perfect, which demonstrates the efficacy of our algorithm.

Figure 4.11: Simulated annealing result for ellipse trajectory recovery problem. Black stars denote the observations; green ellipses represent the true ellipse positions at sampling times, while the red ellipses represent the reconstructed ellipse positions. Left: entire trajectory recovery result. Right: enlarged result at the middle of the trajectory.
4.7.4 Disk Trajectory Recovery with Unknown Shape Parameter

System Specification

The same dynamical model is used here to generate the trajectory of a disk as in Subsection 4.7.1. The disk \( \rho \) is of mass \( m = 1 \) and \( r = 0.8 \). The damping constant is set to \( \alpha = 0.4 \). The initial condition and the propelling force is still set to the same values as in Table 4.1. With the four sensors located at the same positions, \( N = 100 \) observations are obtained from the surface of the disk at sampling times \( t = jM\Delta t, j \in J \), where \( M = 4 \) and \( \Delta t = 0.025 \). The trajectory of the center of \( \rho \) and the observations are shown in Fig. 4.12.

![Figure 4.12: The trajectory of the center of the rigid body and the observations collected by the four sensors.](image)

Validation Data

In order to use the validation algorithm to estimate the radius \( r \), we need to hold out part of the data as the validation set. In our simulation, we choose those \( p_j \) with even indices \( j \) as the validation set, i.e.,

\[
V = \{ 2k \mid k = 1, \ldots, \lfloor N/2 \rfloor \}.
\]

Ideally, if we choose a cross-validation scheme, we can reverse the role of the training data and the validation data and redo the computation. For simplicity, in our simulation, we simply devote the data with indices in \( V \) to validation, and report the result directly after one round of training and validation. As a result, the size of the training set is \( N \frac{N}{2} = 50 \).

Simulation Results

With the simulation settings described above, we implemented the algorithm described in Section 4.6, and obtained the results shown in Fig. 4.13 and Fig. 4.14.

70
In Fig. 4.13, when the value of $r$ is underestimated, the trajectory looks rather rugged. This is due to the fact that to make each observation fall within the disk with radius $r$, which is smaller than the actual object, the fitted trajectory of the center must move more to accommodate to the observations on the surface. On the other hand, when the value of $r$ is overestimated, the trajectory looks very smooth, often taking "shortcuts" at the sharp corners, because with a larger value of the radius, the center does not have to move much in order to cover all the observations. Therefore, the movement of the center looks quite sluggish. For the correct value of $r$, the algorithm reconstructs the trajectory almost perfectly, except for the beginning and the end.

Fig. 4.13 shows that with the correct value of $r$, our algorithm can reconstruct the trajectory very well. However, we may ask whether our system is capable of selecting the correct value of $r$ based on the validation data set. The answer is affirmative, as shown in Fig. 4.14. The right hand bottom figure shows the average value of $D_V(r)$ for different values of $r$. The function reaches its minimum at the true value of radius $r = 0.8$. Moreover, note that $D_V(r)$ is a convex function of parabolic shape with respect to $r$; therefore we may employ the surrogate method to speed up the search for the optimal value of $r$.

The other three graphs in Fig. 4.14 depict the relative positions of observations in the validation set with respect to the reconstructed center of the disk. When $r$ is underestimated, most of the observations in the validation set fall outside the disk; while for the case in which $r$ is overestimated, most of the observations fall inside the disk. Only when $r$ is close to the true model do the observa-
Figure 4.14: The relative position of observations in the validation set with respect to the reconstructed center of the rigid body.

...observations in the validation set approximately fall onto the circumference of the disk, which generates the smallest $D_V(r)$. These properties demonstrate that $D_V(r)$ is a reasonable measure for determining the value of $r$.

4.8 Discussion and Future Work

4.8.1 A more efficient solution for the rod and ellipse problems

As described in Section 4.4 and Section 4.5, the optimization problems for the rod and ellipse cases are solved using simulated annealing due to the non-convexity of the constraints. However, the simulated annealing algorithm is computationally intensive, especially when the shape parameters are unknown and it has to be run multiple times to search for the optimal shape parameters. It is desirable to develop an algorithm that can solve the energy minimization problems more efficiently so that the trajectory of the rod/ellipse can be found quickly even when the shape parameters are unknown.
4.8.2 Introducing Noise

The model described above assumes that the collected observations on the surface of the rigid body are not contaminated by noise. This assumption is unrealistic for practical use. If we introduce additive Gaussian noise to each observation $p_j$, then the geometric constraint (4.4) needs to be revised. A possible choice is to minimize a weighted sum of the energy dissipation and the geometric constraint. For example, in the disk trajectory recovery case, a possible objective could be

$$\min_{c_i, i \in I} \sum_{i \in I} ||c_{i+1} - c_i||^2 + \lambda \sum_{j \in J} \frac{1}{\sigma_j^2} (||p_j - c_jM|| - r)^2. \quad (4.56)$$

where $\sigma_j^2$ is the variance of Gaussian noise added to the $j$th observation. The potential problem arising with this new objective function is the choice of $\lambda$ and its non-convexity, though $\lambda$ can be selected by cross-validation.

4.8.3 Sequential Algorithms

The algorithms proposed in this chapter are based on batch processing. In other words, they need the entire set of observations to be collected before being processed. Of course, many practical tracking algorithms require sequential approaches so that the estimates can be updated as observations are taken. Thus it is of interest to find sequential versions of our schemes to solve the trajectory reconstruction problem.

4.9 Conclusion

In this chapter, we have proposed a new method for reconstructing the trajectory of a rigid body with known or unknown shape parameters that moves in a damped environment from distributively collected, asynchronous observations. We have discussed the trajectory recovery problem for a disk, a rod and an ellipse with known/unknown shape parameters. We have modeled the problem as an energy minimization problem, subject to geometric constrains. The optimization problem has been solved by employing convex relaxation, cross-validation or numerical methods, i.e., simulated annealing. The efficacy of the algorithms has been demonstrated by extensive simulation results.
Chapter 5

Improved Distributed Spring Model Algorithms for Sensor Localization

5.1 Review of Previous Methods

As an essential tool of many sensor network applications, localization has a rich literature. Over the years, various algorithms have been developed to solve the sensor localization problem from different perspectives [18] [39] [32]. Franceschini et al. characterized them according to four criteria in their survey paper [18]: anchor-based vs. anchor free, incremental vs. concurrent, fine-grained vs. coarse-grained, and centralized vs. distributed.

In this chapter, we propose several techniques to improve an anchor based, concurrent, fine-grained, and distributed spring-model-based sensor localization algorithm, in terms of its robustness and computational efficiency. We start our discussion by reviewing several representative fine-grained, distributed sensor localization algorithms.

Savarese et al. [46] proposed an incremental and anchor-based localization algorithm that uses a two-phase approach, including an initial position estimation stage and a second stage for position refinement using trilateration. It is accurate compared to traditional incremental methods, which are prone to error propagation. However, the method can produce dramatically incorrect node displacement when measurement noise is present.
To increase accuracy in a noisy environment, Moore et al. [37] argue that measurement noise can cause flip ambiguities during trilateration. They proposed a robust incremental algorithm, dramatically reducing the amount of error propagation. However, this algorithm has relatively high computational complexity and its third stage can hardly be handled without a centralized node [18]. Furthermore, Franceschini et al. [18] points out that, under large measurement noise, the algorithm may fail to localize enough number of nodes.

Ihler et al. [26] proposed a distributed localization algorithm from a different perspective. It is based on graph models and requires a prior distribution for the sensor locations. It is accurate and fully distributed. However, prior knowledge of the sensor deployment might not be available in many applications.

Algorithms based on machine learning techniques have also been proposed [38][50]. These algorithms typically assume that a large number of anchor points are present, which is unrealistic and impractical in many cases.

Many methods also try to take advantage of additional/alternative information beyond range/distance measurement, including sector/bearing information [12], connectivity among sensor nodes [47], connectivity to anchor nodes [7] and measurement errors (vertex errors and edge errors) [35]. These methods use the additional/alternative information above to improve accuracy, accelerate convergence and reduce computational complexity.

Priyantha et al. [41] proposed a two stage anchor-free distributed algorithm (AFL). The first stage, the initialization stage, produces a qualitative network node graph, while the second stage refines the result by using a spring model, which is the most relevant to our algorithm.

In the spring model method, any pair of sensors with knowledge of mutual distance are considered to be nodes connected by a spring of the same length. The force each spring applies to the nodes depends on the difference between inter-node estimated distances and actual distances. Given sufficient connections amongst the sensors, the system has a unique zero-energy state with all the sensors in correct relative position, up to global transition, rotation and mirror transformation. Furthermore, if enough anchor nodes (sensors with knowledge of their true coordinates) are introduced, the absolute coordinates of the sensors can also be estimated.

One of the advantages of the spring model is the ease with which it can be implemented as a fully distributed algorithm, i.e., each sensor needs to communicate only with its neighbors, repeatedly updating its estimated location until convergence. However, there are several drawbacks to the spring model.

The most prominent problem is the “folding” phenomenon, in which the spring system falls into
an energy local minimum and cannot unfold itself. To tackle this problem, many pre-processing methods have been proposed, such as the first stage of the AFL algorithm, which may fail due to an insufficient number of nodes or to measurement noise. Gotsman et al. [20] improved the first stage of the AFL algorithm, which solves an eigenvector problem. It can be made distributed, but requires many iterations, thus incurring significant communication cost.

Another problem with the spring model (with Hooke’s law) is that the potential energy, measured by the squared error between the true distance and the estimated distance, is sensitive to measurement error, and hence is not robust enough for many applications.

Moreover, in [41] and [20], all scenarios are established on relative coordinates with no anchor. In the case of the presence of anchors, or if absolute coordinates are required, another round of information propagation is needed to transform the coordinates of each sensor to the corresponding absolute coordinates.

In this chapter, we propose a new spring-model algorithm to resolve the problems mentioned above with the presence of anchors. First, we solve a two-dimensional (2-D) sensor localization problem in a three-dimensional (3-D) space, which significantly reduces the occurrence of “folding”. This algorithm requires no preprocessing, and operates in a distributed manner, which converges with competitive rate compared to algorithms in [41] and [20]. Moreover, to improve the noise resistance of the algorithm to range measurement noise, we investigate the estimation error of different types of spring potential functions under different noise distributions. Furthermore, we design a customized spring force so that the spring has a larger strength coefficient once the estimated distance between two sensors is close to the measured distance between them. This customized spring can effectively accelerate the convergence speed of the system and also reduce the estimation error.

The rest of this chapter is organized as follows. In Section 5.2, the spring model sensor localization problem is defined. In Section 5.2.2, a distributed algorithm is proposed to solve the spring model sensor network localization problem. The idea of dimension reduction is explained and simulated in Section 5.3. In Section 5.4, the optimal spring potential function under different noise distributions is explored via extensive simulations. In Section 5.5, a customized spring force is proposed to increase convergence speed as well as to reduce estimation error, and Section 5.6 concludes this chapter.
5.2 Spring Model with Anchors

5.2.1 Basic Model

Suppose we have $N$ sensors, and each sensor has a communication radius $r$. We denote the set of sensors within the communication radius $r$ of the $j$th sensor by $N_j(r)$, and the measured distance between sensor $i$ and $j$ by $d_{ij}$ (assuming sensor $i$ and sensor $j$ have reached consensus, i.e., $d_{ij} = d_{ji}$). If the distance between the estimated locations of sensor $i$, $x_i$, and sensor $j$, $x_j$, is defined as $\|x_i - x_j\|$, and the potential energy $E_{ij}$ caused by the difference between the measured distance and the estimated distance is defined by the potential function

$$E_{ij} = P (\|x_i - x_j\| - d_{ij}), \quad (5.1)$$

then the localization problem can be described as

$$\arg\min_{\mathbf{x}_i, i = 1, \ldots, N} \sum_{i=1}^{N} \sum_{j \in N_i(r)} P (\|x_i - x_j\| - d_{ij}). \quad (5.2)$$

Assume that $A$ is the set of indices of the anchor sensors; then their coordinates should be excluded from the variables of the optimization problem. Therefore, (5.2) can be specialized as

$$\arg\min_{\mathbf{x}_i, i \in A} \sum_{i=1}^{N} \sum_{j \in N_i(r)} P (\|x_i - x_j\| - d_{ij}), \quad (5.3)$$

given $\{x_i, i \in A\}$ known and fixed.

Next, we will introduce an iterative, distributed algorithm that solves (5.2), and further specialize this algorithm to solve its derived problems for different network configurations and noise types.

5.2.2 Distributed Algorithm Solving the Spring Model for Sensor Localization

The basic idea of the spring model is to run the system in a physical world with energy dissipation until it reaches its lowest potential state. Each sensor is deemed to be a ball of mass $m$ and negligible size, and the entire system is immersed in a damping liquid of viscosity $\beta$. At time $t$, each sensor is pushed/pulled by the force exerted by the tension of the spring connected to its neighbors, and also dragged by the damping force opposite to its velocity. Under these assumptions, a distributed, iterative algorithm solving (5.2) can be summarized as follows:
**Algorithm 12:** Distributed Algorithm for Spring Model

```plaintext
while Termination Condition is false do
  /* Local Communication Stage */
  for i ∈ {1, ..., N} do
    Sensor i collects \( \{x_j, j ∈ \mathcal{N}_i(r)\} \) from its neighbors
  end

  /* Local Computation Stage */
  for i ∈ {1, ..., N} do
    \( a_i ← \sum_{j ∈ \mathcal{N}_i(r)} \frac{p'(\|x_i - x_j\| - d_{ij})}{m} \frac{x_j - x_i}{\|x_j - x_i\|} - \frac{\bar{m}v_i}{m} \)
  end

  for i ∈ {1, ..., N} do
    \( v_i ← v_i + a_i \Delta t \)
    \( x_i ← x_i + v_i \Delta t \)
  end
end
```

The basic idea of Algorithm 12 is to set up a physical environment with appropriate damping for the system to run freely until the kinetic and elastic potential energy dissipates. The steady state of the system is a local minimum of the objective function, which is likely to be the desirable solution of the localization problem.

Moreover, Algorithm 12 is purely distributed. There are two stages in the algorithm:

1. **Local communication stage**: Each sensor collects the current estimates of the coordinates from its neighbors. This process is purely local within the radius \( r \).

2. **Local computation stage**: Each sensor updates the estimate of its coordinates (and corresponding virtual speed and acceleration) based on the information collected locally.

Most importantly, this model is highly configurable by setting the mass, viscosity, spring strength, and spring force type of the system. Next, we will discuss a special formulation that introduces anchor nodes into the system.

### 5.2.3 Introducing Anchors: Long-range Anchor Scenario

Intuitively, when the coordinates of the anchors are known accurately, they can be deemed as ordinary sensor nodes, except with infinite mass, so that they will not update their coordinates. When the anchors’ coordinates are known up to a confidence level, they can be modeled as ordinary sensors, connected to an imaginary anchor (with infinite mass and set at the believed location) by a hard spring, with strength \( k \) much higher than others, reflecting the confidence level of the anchor in its coordinates. In our simulations, we assume that the anchors have accurate absolute coordinates, and hence do not update their own coordinates at all.
Moreover, since the anchors do not need to update their coordinates, they are not part of the iterative algorithm. Their only task is to broadcast their coordinates and distance measurements to their neighbors once. This gives rise to an interesting scenario, discussed next, in which the anchors have longer broadcasting range than ordinary sensors.

Since we are interested in finding the absolute coordinates of all the sensors in a distributed manner, the speed that the information provided by the anchors spreads in the network is essential to the convergence speed of our algorithm. In principle, only when each sensor knows, directly or indirectly, the coordinates of at least three anchors, and the relative distance to them, can it estimate its absolute coordinates. Therefore, it is desirable for more sensors to know their distances to at least one anchor, and this can greatly accelerate the spread speed of anchor information throughout the network.

Following the arguments described previously, we introduce a scenario shown in Figure 5.1, in which the four anchors have longer communication ranges than ordinary sensors, and does not enter into the iterations of the algorithm.

In Figure 5.1, the four anchors are located at the four vertices of the unit square, with communication radius approximately \( r_{\text{anchor}} = 0.35 \). All the other sensors have a smaller communication radius of \( r = 0.1 \). The red region is completely out of the range of anchors. Simulations in Section 5.5, in which we combine all techniques to improve the spring model algorithm for sensor localization, employ this configuration.
5.3 Dimension Expansion

5.3.1 “Folding” Phenomenon and Dimension Expansion

The most prominent problem when running Algorithm 12 is the “folding” phenomenon. In a two-dimensional space, two parts of the system can fold together, with a few springs strained, becoming stuck in an energy local minimum. Figure 5.2 illustrates a typical example of the folding phenomenon.

![Figure 5.2: Illustration of the folding phenomenon. Left: correct displacement of the sensors, global minimum; middle: folding phenomenon, system reaches local minimum - forces within the plane cannot unfold the system; right: the system unfolds easily with the introduction of the third dimension.](image)

For example, in Figure 5.2, the left graph represents the global minimum, yet the middle is a “folding” scenario, with four springs strained. Unfortunately, once the spring system falls into a folding local minimum in a 2-D space, it is rather difficult for the system to resolve on its own, without external perturbation, since all forces are within the same plane. However, if the 2-D spring system is allowed to evolve in a 3-D space, a small displacement in the third dimension can effectively unfold the system that falls into a “folding” local minimum, as shown in the right part of Figure 5.2. We refer to this technique as dimension expansion.

By initializing the third dimension of the sensor coordinates according to a uniform distribution, the probability of the “folded” sensors lying exactly on the same 2-D plane is zero. As a result, dimension expansion can resolve many folding scenarios and hence circumvent local minima. Moreover, since the anchors are located on the same 2-D plane, the global minimum of the relaxed system remains the same as the original problem (given that all the distance measurements are accurate), with all the sensors lying in the same 2-D plane as the anchors.

However, the benefits come with a price. First, computationally, dimension expansion adds one more coordinate that the sensors need to maintain and communicate. Second, with measurement noise, this may cause the third dimension to overfit to the noise.

Nonetheless, our simulations demonstrate that the dimension expansion technique has almost the
same convergence rate as its 2-D counterpart while significantly reducing the chances of “folding”, and performs well for large noise scenarios with a sufficient number of anchor points. These properties will be demonstrated via simulations next and in Section 5.5.

5.3.2 Efficacy of Dimension Expansion

We first compare the performance of the spring system with and without dimension expansion. The springs obey Hooke force in the experiment, i.e., the potential function is \( p(x) = kx^2 \). \( N = 500 \) sensors are scattered in a \( 1 \times 1 \) 2-D square. On each vertex, the midpoint of each side, and the center of the square, there is an anchor (a total of 9). Assuming all measurements of distance within communication radius \( r \) are accurate, we run a series of simulations for different values of \( r \), with \( m = 1 \) for all sensors, \( k = 5 \), \( \Delta t = 0.01 \) and \( \beta = 1 \), where \( m \) is the mass of the sensor, and \( \beta \) is the viscosity of the liquid, as described in Section 5.2. The simulation results are shown in Figure 5.3.

![Figure 5.3: Average localization errors (of 100 trials) for 2-D and 3-D algorithms.](image)

The initial location of each sensor is a uniformly distributed random point in the unit square. For the 3-D algorithm, the coordinate for the third dimension is initialized as a uniformly distributed value on \([−0.1, 0.1]\). Assuming that \( x_i^0 \) is the true location of sensor \( i \), the localization error \( E \) is measured by

\[
E = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \|x_i - x_i^0\|^2}.
\]

For small values of \( r \), some sensors do not have sufficient constraints to specify their absolute coordinates. Hence, both 2-D and 3-D algorithms have large estimation errors. However, when \( r \) grows larger, the 3-D algorithm performs very well, consistently converging to the desired global
minimum. However, as the links among the sensors increase, the 2-D algorithm has many chances to fall into local minima and hence incurs a large estimation error. This phenomenon is the reason for the efficacy of dimension expansion.

Another important issue is the convergence speed. Figure 5.4 compares the convergence rates for the case of $N = 1000$ and $r = 0.15$, for both 2-D and 3-D algorithms. In these and other examples, we find that the 3-D algorithm, though having one more dimension, does not have an observable disadvantage in terms of the rate of reducing the localization estimation error.

5.4 Optimal Spring Potential

Besides changing the dimension of the space in which we run the algorithm, the distributed spring model algorithm also has the advantage that it can easily adapt to different types of noise, with the mere change of the force/potential (use non-Hooke force). Various types of potentials can be designed to adapt to different types of noise.

To simplify the problem, in this section, we analyze the group of potentials of the form

$$E_{ij} = P (\|x_i - x_j\| - d_{ij}) = \|x_i - x_j\|^{p} - d_{ij}, \quad (5.4)$$

where $p \geq 1$, to identify the optimal spring potential, i.e., we seek optimal values of $p$, for different types of noise. For simplicity, we henceforth call the potential function represented in equation (5.4) the $L_p$ potential.
Let us first consider a simplified model with \(N\) anchors and only one sensor to be localized, with noisy distance/range measurements. Let \(A_i, i = 1, ..., N\), be anchors located uniformly on a unit circle in two dimensional space, with locations denoted as \(x_{A_i}, \|x_{A_i}\| = 1\). Suppose the only sensor to be localized, \(S\), is located at the center of the circle. The distance measurements between \(S\) and the anchors \(A_i, i = 1, ..., N\), are corrupted by different types of noise with mean 0 and standard deviation \(\sigma\).

Given the range measurements and \(x_{A_i}, i = 1, ..., N\), sensor \(S\) can easily estimate its location, \(x_S\), using the distributed spring model algorithm described in Section 5.2, with the desired force/potential. For each specific type of noise and potential, sensor \(S\) solves the following optimization problem:

\[
\min_{x_S} \sum_{i=1}^{N} \|\|x_{A_i} - x_S\| - (1 + \xi_i)\|^p,
\]

(5.5)

where \(\xi_i, i = 1, ..., N\), are independent and identically distributed (i.i.d.) random variables with means 0 and standard deviations \(\sigma\).

Given a particular \(p\) and the joint distribution of \(\xi_i, i = 1, ..., N\), the optimization result of (5.5) forms a distribution \(\text{dist}_p(x_S)\). In order to investigate \(L_p\) potentials with different values of \(p\), we compare the group of distributions \(\text{dist}_p(x_S), p \geq 1\), to find the optimal potential, in terms of accuracy, for different types of noise.

Intuitively, we desire the distribution \(\text{dist}_p(x_S)\) to concentrate around the origin as closely as possible. Specifically, we desire a potential function, as specified by parameter \(p\), that has a spiky \(\text{dist}_p(x_S)\) centering around the true location \((0, 0)\), since this demonstrates the robustness of the spring potential to resist noise. Therefore a natural measure of the performance would be the \(q\)-quantile of the distributions of the distance from our estimation to the true location, and deem the potential with smallest \(q\)-quantile distribution the \(q\)-optimal \(L_p\) potential for the particular noise.

To be more specific, the distance from our estimate to the true location \((0, 0)\) is simply \(\|x_S\|\); so from the distribution function of \(x_S\), i.e., \(\text{dist}_p(x_S)\), we can derive the distribution of \(\|x_S\|\), i.e.

\[
\Phi_p(r) = \mathbb{P}\{\|x_S\| \leq r\}.
\]

(5.6)

Given that the set up of the system and the distribution of noise being the same, the distribution function \(\Phi_p(r)\) is a function of the parameter \(p\) only. Therefore, the \(q\)-optimal value of \(p\) is defined as

\[
p^* = \arg\min_p \Phi_p^{-1}(q),
\]

(5.7)
where $\Phi^{-1}_p(q)$ is the inverse function of the distribution function of $\|x_S\|$.

Since the optimization problem (5.5) is non-convex, it is difficult to find the analytical form of $\text{dist}_p(x_S)$ and thus the $q$-optimal spring potential. Therefore we turn to numerical methods to investigate the $q$-optimal potential for different types of noise.

In the numerical experiments, $N$ anchors are fixed uniformly on the unit circle, with coordinates $(\cos \frac{2\pi i}{N}, \sin \frac{2\pi i}{N})$. Then, a large number of rounds of simulations are run to generate the distribution $\text{dist}_p(x_S)$ for a specific $p$. At each round, the distance measurements between $S$ and the anchors $A_i$, $i = 1, ..., N$, are corrupted by the desired additive noise. In order to find the localization result of sensor $S$, i.e., the global minimum of the optimization problem (5.5), we search in fine grids within the square circumscribed about the unit circle. The total number of times each grid point becomes the global minimum in all rounds are counted, and the distribution $\text{dist}_p(x_S)$ is then represented by the frequency of “hits” received at each grid point.

Figure 5.5 shows the setting and resulting $\text{dist}_p(x_S)$ of a particular run of the experiment, where $p = 2$. In this experiment, the range measurements are corrupted by additive Gaussian noise, with mean 0 and standard deviation 0.2. Ten anchors are used and ten thousand rounds of simulations are run to generate the distribution $\text{dist}_2(x_S)$, in which, at each round, a 100-by-100 grid is searched for the global minimum of (5.5).

On the left hand graph of Figure 5.5, ten dots on the unit circle shows the position of the anchors, while color represents the number of “hits” received by each grid point. The right hand graph is a 3-D mesh plot of the left hand graph, which shows $\text{dist}_2(x_S)$ under Gaussian noise.

Given $\text{dist}_p(x_S)$ represented by the frequency of “hits” received at each grid point, the distribution function $\Phi_p(r)$ can be naturally interpreted as the frequency of “hits” received at the grid points locate within radius $r$ from the origin.

Figure 5.6 shows a group of $\Phi_p(r)$ for distributions $\text{dist}_p(x_S)$ with various $p$ values, where the x-axis represents the radius $r$. In this experiment, the range measurements are corrupted with additive Gaussian noise (left graph) and Laplacian noise (right graph), respectively, with mean 0 and standard deviation 0.2. The anchor locations are the same as the previous simulation, and 10,000 rounds of simulations are run to generate $\Phi_p(r)$ for each $p$ value under each type of noise.

A distribution with $\Phi_p(r)$ that is close to the upper-left corner of the graph is desired, since it is spikier and demonstrates the robustness of the corresponding spring potential with respect to the specific type of noise. Therefore, as shown in Figure 5.6, the spring potential with $p = 2$ works the best under Gaussian noise, while the potential with $p$ around 1.3 performs well with Laplacian noise.

We further compare the $q$-quantile of the distributions, to find the $q$-optimal $L_p$ potential as
Figure 5.5: Setting and resulting \( \text{dist}_2(x_S) \) of a particular run of experiment, where the range measurements are corrupted by additive Gaussian noise: \( \xi_i \sim \mathcal{N}(0, 0.2), i = 1, \ldots, 10 \). Ten anchors are used and 10,000 rounds of simulations are run to generate \( \text{dist}_2(x_S) \). Left: anchor positions and color plot of “hits” received by each grid point; Right: 3-D mesh plot of the left hand graph, representing \( \text{dist}_2(x_S) \).

Figure 5.6: Comparison of \( \Phi_p(r) \) for distributions \( \text{dist}_p(x_S) \) with various \( p \) values, where the x-axis represents the radius \( r \). Left: \( \Phi_p(r) \) under Gaussian noise; Right: \( \Phi_p(r) \) under Laplacian noise.
described above, under Gaussian and Laplacian noise. The simulation results are shown in Figure 5.7, where the x-axis represents the $p$ values and the y-axis shows the radius $r$. The left and right hand graphs show the quantile comparison under Gaussian and Laplacian noise respectively. Same simulation settings are used as the previous experiment, and 20%, 50%, 80% and 90% quantiles of distributions with various $p$ values are compared.

It is easy to read the $q$-optimal spring potential from Figure 5.7. For example, $p = 2$ is the 90%-optimal spring potential under Gaussian noise in the group of $L_p$ potentials, while $p = 1.3$ is the 80%-optimal spring potential under Laplacian noise.

![Figure 5.7: Comparison of quantiles for distributions dist$_p(x_S)$, where the x-axis represents the $p$ values and the y-axis shows the radius $r$ of the circle. 20%, 50%, 80% and 90% quantiles of distributions with various $p$ values are compared. Left: quantile comparison under Gaussian noise; Right: quantile comparison under Laplacian noise.](image)

To conclude, the distributed spring model algorithm has the advantage that it can easily adapt to different types of noise, and the appropriate spring potential can be chosen according to the numerical scheme described in this section.

### 5.5 Customizing the Potential for Faster Convergence

In the previous section, all the potential functions that we have discussed are generic $L_p$ norms. Different types of potential functions work for different types of noises. However, in practical scenarios, situations can get more convoluted. Searching for the optimal value of $p$ may not be enough. Moreover, as the system grows large, with more sensors to localize, convergence speed becomes a more important measure of the performance of our algorithm.
5.5.1 Spring with “Lock-in” Mode

A commonly observed phenomenon in the spring model algorithm is that even when some part of the system has approximately reached the correct configuration, due to the soft spring and under-damped environment, the system continues to oscillate before eventually converging to a low-estimation-error state. This can significantly prolong the convergence time of the system. Increasing spring strength or the viscous coefficient of the environment may not be effective because these may cause the system to fall into local minima and reach premature convergence.

To cure this problem, one simple idea is to design the springs so that they can “lock in” when the actual distance between two sensors is approximately the same as the length of the spring. To be more specific, we would like the strength coefficient to be significantly higher when the spring approaches its natural state. This way, in the situation where the sensors have approximately reached the correct configuration, the system will “harden-up” to keep the current configuration so as to avoid being disturbed into disorder again. If handled properly, the newly designed spring system can reach the correct configuration faster than the vanilla spring model algorithm.

This idea can be viewed from another perspective. The incremental methods [46] and [37] are actually the case in which all springs are rigid (with infinite strength), so that all locally resolved configurations are kept unchangeable. The concurrent method (e.g., the vanilla spring model) is another extreme in which nothing is kept fixed until the very end. This new spring model with “lock-in” mode is something in between. Specifically, we make well-configured parts of the entire system more rigid than the ill-configured parts, so that we can take potential benefits from both methods.

To make things simple and concrete, we assume the new spring to be a Hooke spring (linear force), with different strengths for different $\Delta d = \|x_i - x_j\| - d_{ij}$ values. For $\Delta d$ greater than a critical distance $\rho$, we assume that the spring is softer, with strength coefficient $k_0$; and for $\Delta d$ less than $\rho$, we assume that the spring is harder, with strength coefficient $k_1$, where $k_1 > k_0$. Therefore, the force function is as shown below:

$$f(x) = \begin{cases} -k_1x & |x| \leq \rho \\ -k_0x & |x| > \rho \end{cases}$$

(5.8)

A typical potential function for the customized spring potential is shown in Figure 5.8. Note that there is a “potential well” with radius $\rho$. Therefore, when $\Delta d$ is no more than $\rho$, the two sensors will be locked in with a strong bond.
5.5.2 Numerical Caveats

Ideally, the “lock-in” mode strength coefficient $k_1$ should be very high compared to the normal strength so that the length is effectively fixed. However, since we need to numerically evolve the spring model with a fixed time step $\Delta t$, a very high spring strength can result in unstable numerical results. Therefore, in our simulations, the lock-in mode spring strength is chosen to be 3 to 5 times that of the normal spring strength. Despite the moderate $k_1$ value, the customized spring potential is proven to be effective by the simulation result next.

5.5.3 Simulation Results

To demonstrate the efficacy of the spring model algorithm with “lock-in” mode, we compare the convergence speed of the following three cases:

1. Hooke spring with soft strength $k_0$;

2. Hooke spring with hard strength $k_1$;

3. Customized spring potential with $k_1$ within $\rho$ and $k_0$ beyond that range.

It is worth noting that dimension expansion is applied in all three cases.

The setup with long-range anchors discussed in Section 5.2 (shown in Figure 5.1) is employed, which includes 800 normal sensors with communication radius 0.1, and four anchors with broadcast radius $\frac{1}{2\sqrt{2}}$. We choose viscosity $\gamma = 1$, critical radius $\rho = 0.05$, and $k_1 = 5k_0 = 250$. The aforementioned three cases are compared under two scenarios: range measurements with no noises.
and range measurements corrupted with additive Gaussian noise with mean 0 and standard deviation 0.1.

Figure 5.9: Comparison of convergence rates. Left: range measurements with no noise; Right: range measurements corrupted with Gaussian noise with zero mean and \( \sigma = 0.1 \).

Figure 5.9 illustrates the comparison for one particular run of the three cases. It is clear from the figure that the customized spring case converges with a much faster rate, with or without noise. In both situations, the case with customized spring potential reaches the same level of estimation error with about 150 fewer iterations than its counterparts.

More detailed simulation results are shown in Table 5.1. It records the average error rate of 100 independent simulations, at iterations of multiples of 100. The results in Table 5.1 demonstrates that the algorithm with customized spring potential works consistently better than the other two cases.

<table>
<thead>
<tr>
<th>Noise Level</th>
<th>Spring Type</th>
<th>Error Rate at Iterations of Multiples of 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma = 0.00 )</td>
<td>Soft spring 0.2467 0.1468 0.1514 0.1402 0.0702 0.0419 0.0256 0.0174</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strong spring 0.2501 0.1568 0.1796 0.1077 0.0750 0.0437 0.0277 0.0189</td>
<td></td>
</tr>
<tr>
<td></td>
<td>“Lock-in” mode 0.1347 0.1651 0.1013 0.0527 0.0325 0.0213 0.0144 0.0100</td>
<td></td>
</tr>
<tr>
<td>( \sigma = 0.05 )</td>
<td>Soft spring 0.2486 0.1458 0.1499 0.1406 0.0693 0.0419 0.0259 0.0181</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strong spring 0.2508 0.1566 0.1784 0.1085 0.0745 0.0437 0.0285 0.0190</td>
<td></td>
</tr>
<tr>
<td></td>
<td>“Lock-in” mode 0.1340 0.1651 0.1004 0.0519 0.0323 0.0216 0.0151 0.0128</td>
<td></td>
</tr>
<tr>
<td>( \sigma = 0.10 )</td>
<td>Soft spring 0.2485 0.1478 0.1528 0.1389 0.0710 0.0427 0.0270 0.0204</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strong spring 0.2507 0.1569 0.1791 0.1082 0.0742 0.0438 0.0288 0.0215</td>
<td></td>
</tr>
<tr>
<td></td>
<td>“Lock-in” mode 0.1363 0.1640 0.1016 0.0528 0.0334 0.0236 0.0199 0.0214</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of average convergence rates under different noise levels

Note that there are rises and falls in the aggregated localization error in Table 5.1, because of the oscillations in the spring model. To better quantify the difference in convergence speed, we compute
the integrated localization error of the entire process of the algorithm, i.e., the ratio between the total area under the convergence curve and the total number of iterations (800 in this experiment), and depict the results in Figure 5.10. It is obvious that under different noise levels, the customized spring model outperforms the other two cases significantly. The average localization error throughout the process of convergence is merely about 60% of its counterparts.

It is worth mentioning that both extreme cases, with soft and strong springs, underperform the customized spring case, which is a compromise of two types of springs. It is the combination of the two that creates effective improvement in convergence speed, without compromising the final localization error.

![Figure 5.10: Comparison of integrated convergence rate under Gaussian noise with standard deviation 0.00, 0.05 and 0.10.](image)

**5.6 Conclusion**

In this chapter, we have discussed several techniques to improve a spring-model-based algorithm that solves sensor localization problems in a fully distributed manner. Our prototype algorithm, with no prior knowledge of the coordinates of the sensors except a few anchor points, iteratively localizes the sensors using only short-range communication among neighboring sensors.

The first technique to improve the prototype algorithm is based on expanding a two-dimension problem into three dimensions. The dimension expansion technique can greatly reduce the occurrence of the “folding” phenomenon and hence prevent the algorithm from falling into local minima.

Second, we have investigated the optimality of different types of spring potential functions under different types of noises. Using a simplified model, our simulation demonstrates that $L_p$ potential
works best for Gaussian noise when $p \approx 2$, while $p \approx 1.3$ works best for Laplacian noise. This simulation based conclusion sheds some light on spring-model selection under different noise circumstances.

Third, we discussed the customized design of a spring model beyond $L_p$ potential functions to accelerate convergence speed. The customized spring model has higher strength when the distance between sensors is close to the true length of the spring. Extensive simulations demonstrate that this customized spring model can effectively “lock in” correctly localized sensors and hence make the iterative algorithm more efficient. From another perspective, this customized spring model can be deemed as a compromise between incremental and concurrent sensor localization algorithms.
Bibliography


