Robust Algorithms for Object Localization

Aaron S. Wallack [*]
Computer Science Division
University of California
Berkeley, CA 94720
wallack@robotics.eecs.berkeley.edu

Dinesh Manocha [†]
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
manocha@cs.unc.edu

Abstract

Object localization using sensed data features and corresponding model features is a fundamental problem in machine vision. We reformulate object localization as a least squares problem: the optimal pose estimate minimizes the squared error (discrepancy) between the sensed and predicted data. The resulting problem is non-linear and previous attempts to estimate the optimal pose using local methods such as gradient descent suffer from local minima and, at times, return incorrect results. In this paper, we describe an exact, accurate and efficient algorithm based on resultants, linear algebra, and numerical analysis, for solving the nonlinear least squares problem associated with localizing two-dimensional objects given two-dimensional data. This work is aimed at tasks where the sensor features and the model features are of different types and where either the sensor features or model features are points. It is applicable to localizing modeled objects from image data, and estimates the pose using all of the pixels in the detected edges. The algorithm's running time depends mainly on the type of non-point features, and it also depends to a small extent on the number of features. On a SPARC 10, the algorithm takes a few microseconds for rectilinear features, a few milliseconds for linear features, and a few seconds for circular features.

[*]Supported by Fannie and John Hertz Fellowship

[†]Supported in part by Junior Faculty Award, University Research Award, NSF Grant CCR-9319957 and ARPA Contract DABT63-93-C-0048.

1 Introduction

Model-based recognition and localization are fundamental tasks in analyzing and understanding two-dimensional and three-dimensional scenes [12, 10]. There are many applications for vision systems: inspection and quality control, mobile robot navigation, monitoring and surveillance. The input to such systems typically consists of two-dimensional image data or range data. The model-based recognition problem is to determine which object O from a set of candidate objects $\{O_1, O_2, \ldots\}$ best explains the input data. Since the object O can have any orientation or location, the localization problem is to compute the pose p for a given object O which best fits the sensed data.

The problems of recognition and localization have been extensively studied in the vision literature [12, 10]. The majority of the object recognition algorithms utilize edge-based data, where edges are extracted from image data using filter-based methods. Edge data is either comprised of pixels which have exceeded some threshold, or edge segments formed by straight line approximations of those pixels. On the other hand, objects are usually modeled as a set of non-point features, such as line segments, polygons, or parameterized curves. For this reason, localization algorithms must handle matching sensor features and model features which are of different types. We present an exact and robust model-based object localization algorithm for features of different types, where either the sensed features or the model features are points (refer Figure 1). Unlike Kalman filtering based approaches to the same problem, this algorithm is immune to problems of local minima, and unlike approaches which reduce the problem to matching point features which can be solved exactly [16, 26, 5], this algorithm does not introduce error by sampling points from model features.

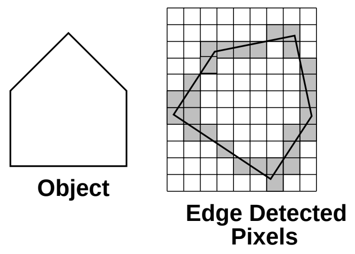


Figure 1: Given image data, this localization algorithm estimates the object's pose directly from the edge detected pixels and the model features.

Our approach involves reducing the model-based object localization problem between dissimilar features to a *nonlinear* least squares error problem, where the error of a pose is defined to be the discrepancy between the predicted sensor readings and the actual sensor readings. In this paper, we concentrate on the object localization problem, *i.e.*, we assume that the features have already been interpreted using a correspondence algorithm. We use *global* methods to solve the problem of optimal pose estimation. In particular these methods compute the pose that minimizes the sum of the squared errors between the sensed data and the predicted data. Rather than using local methods such as gradient descent, the problem is formulated algebraically and all of the local extrema of the sum of squared error function, including the global minimum, are computed. For the case of matching points to circular features, we use an approximation of the squared error because the error cannot be formulated algebraically without introducing an additional variable for each circular feature.

The algorithm's running time mainly depends upon the algebraic complexity of the resulting system, which is a function the type of non-point features. Since the algorithm takes all of the sensor features and model features into account, the running time is linear in the size of the data set. It has been implemented and works well in practice. In particular, objects with rectilinear boundaries are localized in *microseconds* on a SPARC 10 workstation. The running time increases with the complexity of the non-point features: the algorithm takes a few *tens of milliseconds* to localize polygonal models, and one to two seconds to localize models with circular arcs.

1.1 Previous Work

The localization techniques presented in this paper are applicable to the general field of model-based machine vision. In machine vision applications, localization is usually reformulated as a least squares error problem. Some methods have focused on interesting sensor features such as vertices, and ignored the remainder of the image data [11, 5, 3, 14, 6, 7], thereby reducing the problem to a problem of matching similar features. These approaches are likely to be suboptimal because they utilize a subset of the data [23]. In order to utilize all of the edge data, modeled objects must be compared directly to edge pixel data. In the model-based machine vision literature, there are three main approaches for localizing objects where the model features and the sensed features are of different types: sampling points from the non-point features to reduce it to the problem of localizing point sets; constructing *virtual features* from the point feature data in order to reduce it to the problem of matching like features; estimating the object's pose by computing the error function between the dissimilar feature types and finding the minimum using gradient descent techniques, or

Kalman filtering techniques.

One approach for object localization involves sampling points from non-point features in order to reduce it to the problem of matching points to points. The main advantage of reducing the problem of matching points to non-point features to the problem of matching point sets is that the problem is exactly solvable in constant time. For two-dimensional rigid transformations, there are only three degrees of freedom: x, y, θ [16, 26]. By decoupling these degrees of freedom, the resultant problems become linear, and are exactly solvable in constant time. The crucial observation was that the relative position, (x, y), of the point sets could be determined exactly irrespective of the orientation, θ since the average of the x-coordinates and y-coordinates was invariant to rotation. This decoupling technique is valid for three dimensions as well [5]. To reduce the problem of matching points to non-point features to the solvable problem of matching points to points, points are sampled from the non-point features. This approach is optimal when objects are modeled as point-sets, but can be suboptimal if objects are modeled as non-point features, because sampling introduces errors when the sampled model points are *out of phase* with the actual sensed points.

Another approach to object localization for dissimilar features involves constructing *virtual features* from sensed feature data and then computing the object's pose by matching the like model features and *virtual features.* Such approaches solve an artificial problem, that of minimizing the error between *virtual* features and model features rather than minimizing the error between the sensed data and the modeled objects. Faugeras and Hebert [5] presented an algorithm for localizing three-dimensional objects using pointwise positional information. In their approach, virtual features were constructed from sensed data and then, objects were identified and localized from the features, rather than the sensed data. Forsyth *et al.* described a three-dimensional localization technique which represented the data by parameterized conic sections [7]. Objects were recognized and localized using the algebraic parameters characterizing the conic sections. Recently, Ponce, Hoogs, and Kriegman have presented algorithms to compute the pose of curved three-dimensional objects [21, 22]. They relate image observables to models, reformulate the problem geometrically, and utilize algebraic elimination theory to solve a least squares error problem. One drawback of their approach is that they minimize the resultant equation, not the actual error function. Ponce *et al.* use the Levenberg-Marquardt algorithm for the least square solution, which may return a local minimum [21].

Another approach to object localization involves reducing it to the correct least squares problem, but then solving that least squares problem using gradient descent techniques. The least squares approach involves formulating an equation to be minimized; consequently, the error function between a point and a feature segment cannot take into account feature boundaries. Gradient descent methods require an initial search point and suffer from problems of local minima. Huttenlocher and Ullman used gradient descent techniques to improve the pose estimates developed from comparing sets of three model point features and three image point features [14]. Many localization algorithms have relied on Kalman filters to match points to non-point features. Kalman filters suffer from local minima problems as well. Given a set of points and corresponding non-point features, Ayache and Faugeras [3] attempted to compute the least squares error pose by iteration and by Kalman filters. Kalman filter-based approaches have demonstrated success in various machine-vision minimization applications [17, 28, 4]. Kalman filter-based approaches suffer from two problems: sensitivity to local minima problems, and the requirement of multiple data points.

A great deal of work has been done on matching features (points to points, lines to lines, points to features etc.) in the context of interpreting *structure from motion* [6, 13]. The problem involves matching sensor features from two distinct two-dimensional views of an object; the problem of motion computation is reduced to solving systems of nonlinear algebraic equations. In practice, the resulting system of equations is overconstrained and the coefficients of the equations are inaccurate due to noise. As a result, the optimal solution is usually found using gradient descent methods on related least squares problems. Most of the local methods known in the literature do not work well in practice [15] and an algorithm based on global minimization is greatly desired.

Sparse sensors have recently demonstrated success in recognition and localization applications [27, 20]. The need for accurate localization algorithms has arisen with the emergence of these sparse probing sensor techniques. This technique has been applied to recognition and localization using beam sensor *probe data* and results in pose estimation with positional accuracy of $\frac{1}{1000}$" and orientational accuracy of 0.3^o [27, ?].

The algorithm is applicable to both sparse and dense sensing paradigms; in sparse sensing applications, such as probing, a small set of data is collected in each experiment, whereas in dense sensing applications, such as machine vision, an inordinate amount of data is collected in each experiment. Sparse sensing applications necessitated development of this localization algorithm because sparse sensing strategies require an exact optimal pose strategy. First, a small data set cannot accommodate sampling points from the data set in order to transform the problem of matching features of dissimilar types to matching features of similar types, and second, a small data set is more susceptible to problems of local minima which can arise with local methods such as gradient descent or Kalman filtering.

1.2 Organization

The rest of the paper is organized in the following manner. In Section 2, we review the algebraic methods used in this paper and explain how probabilistic analysis provides a basis for using a least square error model to solve the localization problem. We describe the localization problem and show how algebraic methods can be applied for optimal pose determination in Section 3. The algorithm for localizing polygonal objects given point data is described in Section 4 and the algorithm for localizing generalized polygonal objects, *i.e.*, objects with linear and circular features is described in Section 5. In Section 6, we present various examples, and use the localization technique to compare the relative localization accuracy of utilizing all available data compared to using only an *interesting* subset of the data. Finally, we conclude by highlighting the contributions of this work.

2 Background

2.1 Error Model

In this section, we describe the error model used for localization. Probability theory provides the basis or reducing the object localization to a least squares error problem [25]. Assuming imperfect sensing and imperfect models, our goal is to tolerate these discrepancies and provide the best estimate of the object's pose. Real data presents constraints which are inconsistent with perfect models, and for this reason, we reformulate the problem into a related solvable problem; minimizing the squared error is a common method for *solving* problems with inconsistent constraints, and it provides to the maximum likelihood estimate, assuming that the errors (noise) approximate independent normal (Gaussian) distributions.

The reason we assume a normal (Gaussian) error distribution is the composition of error from many different sources. Even though we may be using high precision, high accuracy sensors to probe a two-dimensional object, such sensors have many intrinsic sources of error. Furthermore, real objects can never be perfectly modeled, and this accounts for an additional source of error. These error distributions define a conditional probability distribution on the sensed data values given the boundary position (which depends on the object's pose, equation (1)). The conditional probability can be rewritten in terms of the sum of the squared discrepancies (equation (2)). The conditional probability of the pose given the sensor data is related to the conditional probability of the sensor data given the pose by Bayes' theorem (equations (3,4)).

The minimum sum squared error pose corresponds to the maximum probability (maximum likelihood estimate), since minimizing the negated exponent maximizes the result of the whole expression. Thereby, computing the least squares error pose is analogous to performing maximum likelihood estimation.

$$\begin{aligned}
\Pr(sensor = S|pose = \Theta) &\propto e^{-\frac{Discrepancy(s_1,s_{1,predicted})^2}{2\sigma^2}-\frac{Discrepancy(s_2,s_{2,predicted})^2}{2\sigma^2}-\cdots} && (1)\\
\Pr(sensor = S|pose = \Theta) &\propto e^{-\frac{\sum_i Discrepancy(s_i,s_{i,predicted})^2}{2\sigma^2}} && (2)\\
\Pr(pose = \Theta|sensor = S) &= \frac{\Pr(sensor = S|pose = \Theta)\Pr(pose = \Theta)}{\Pr(sensor = S)} && (3)\\
\Pr(pose = \Theta|sensor = S) &\propto e^{-\frac{\sum_i Discrepancy(s_i,s_{i,predicted})^2}{2\sigma^2}}\frac{\Pr(pose = \Theta)}{\Pr(sensor = S)} \\
\Pr(pose = \Theta|sensor = S) &\propto e^{-\frac{\sum_i Discrepancy(s_i,s_{i,predicted})^2}{2\sigma^2}}\frac{1}{2\pi} && (4)
\end{aligned}$$

2.2 Directly Comparing Features

The sensor data point features should be directly compared to the model features, even when the sensor features and model features are of different types. One measure of discrepancy between points and nonpoint features is the Euclidean distance between a sensor data point and a specific predicted point on the model feature. Measuring the discrepancy in this way relies on the mistaken assumption that the sensed point feature corresponds to a known point on the model feature. The correspondence information specifies a model feature relating to each sensor feature point, it does not indicate particular points on the model features. A more reasonable measure of the discrepancy is the minimum distance between the sensor feature point and the corresponding model feature, such as the minimum distance between a point and a line, or the minimum distance between a point and a circle. For machine vision applications, measuring the error in this way allows one to utilize all of the edge detected pixels without introducing any artificial error (by sampling the model features, for example). It turns out that the discrepancy (error), or minimum distance, between a sensor data point and a linear model feature or a circular arc model is a nonlinear computation. In particular, the minimum distance between a point and a circular arc cannot be formulated exactly without introducing an additional variable; in order to achieve constant time performance, we use a approximation to that minimum distance function which does not require introduction of an additional variable.

2.3 Solving Algebraic Systems

The key to this algorithm is that we reduce the problem of finding the global minimum of an algebraic function to solving a system of nonlinear algebraic equations. Finding the global minimum of a function is a classic problem and there many algorithms known in the literature. The algorithms can be categorized into local methods and global methods. Local techniques such as Newton's method require good initial estimates in order to find all the solutions in the given domain, and this is difficult in our applications. Global methods such as resultants and Gröbner bases are based on symbolic reduction and computing roots of univariate polynomials. They are rather slow in practice and in the context of floating point computations suffer from numerical problems. Other global techniques include homotopy methods. However, they are rather slow for practical applications [13]. In this paper, we use some recently developed algorithms based on multipolynomial resultants and matrix computations described by Manocha [19]. Resultants are well known in classical algebraic geometry literature [24, 18] and have recently been effectively used in many problems in robotics, computer graphics and geometric modeling [19].

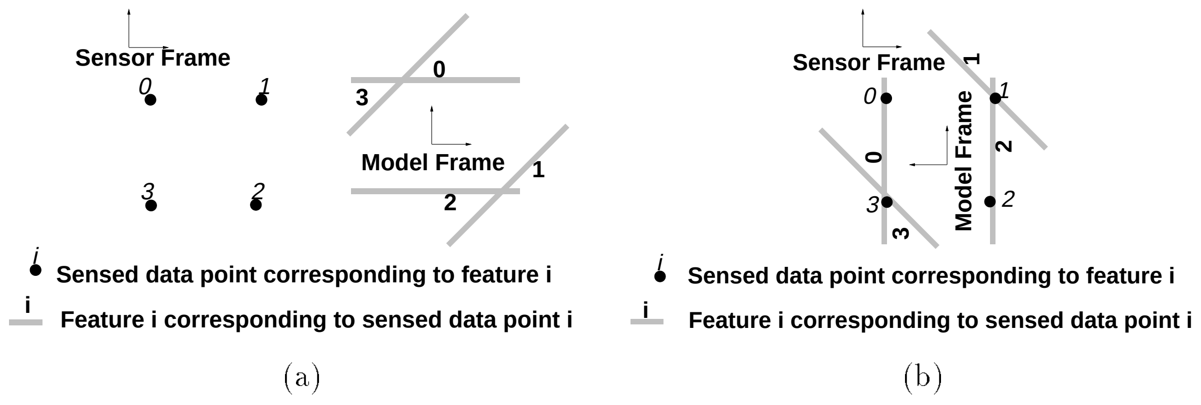

Sensor Frame
0
1
3
2
0
3
Model Frame
1
2
Sensed data point corresponding to feature i
Feature i corresponding to sensed data point i
(a)
Sensor Frame
1
0
1
2
0
Model Frame
3
2
3
Sensed data point corresponding to feature i
Feature i corresponding to sensed data point i
(b)

Figure 2: (a): Given a set of sensed data points in the sensor reference frame, and non-point model features in a model reference frame, the task is to determine the model position (in the sensor reference frame). (b): The model's position can be expressed as the transformation between the sensor frame and the model frame.

The algorithm for solving non-linear equations linearizes the problem using resultants and uses matrix computations such as eigendecomposition, singular value decomposition and Gaussian elimination [9]. Routines for these matrix computations are available as part of standard numerical linear algebra libraries. In particular, we used LAPACK [2] and EISPACK [8] routines in our implementation of the algorithm.

2.4 Problem Specification

Given a set of point features in one frame of reference and non-point features defined in another frame of reference, we determine the transformation which optimally maps the features onto each other. Assume without loss of generality that sensor features are data points (in the sensor reference frame) and correspond to non-point model features (in the model frame). The task is to localize the model (object) in the sensor reference frame; i.e., to determine the transformation between the sensor frame and the model frame consistent with the data (see Figure 2(a)). Figure 2(b) shows the most likely pose of a particular model/object for the points shown in Figure 2(a).

There are two approaches we can take: transform the non-point model features, and then match them to points, or transform the points, and then match them to the non-point model features. We have chosen to transform the points because it enables us to use a uniform approach to solving the mathematical problems corresponding to different feature types. Due to the duality of these two related transformations, the algorithm for solving for the optimal transformation of the points can also be used to solve for the optimal transformation of the non-point model features by computing the inverse transformation.

3 Theoretical Framework

3.1 Transformation Matrices

Two-dimensional rigid transformations can be parameterized by X, Y, θ (refer equation (5)). In order to arrive at algebraic equations, we utilize the trigonometric substitution ($t = \tan(\frac{\theta}{2})$: $\sin\theta = \frac{2t}{1+t^2}$ and $\cos\theta = \frac{1-t^2}{1+t^2}$). The algebraic matrix $Mat(X, Y, t)$, which is used throughout this paper, is given in equation (5). It is important to remember that $Mat(X, Y, t)$ is not exactly $\mathcal{T}(X, Y, \theta)$ but rather $(1 + t^2)\mathcal{T}(X, Y, \theta)$.

$$\mathcal{T}(X,Y,\theta) = \begin{bmatrix} \cos(\theta) & \Leftrightarrow\sin(\theta) & X \\ \sin(\theta) & \cos(\theta) & Y \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow Mat(X,Y,t) \overset{t=\tan(\frac{\theta}{2})}{=} \begin{bmatrix} 1 \Leftrightarrow t^2 & \Leftrightarrow 2t & X(1+t^2) \\ 2t & 1 \Leftrightarrow t^2 & Y(1+t^2) \\ 0 & 0 & 1+t^2 \end{bmatrix} \tag{5}$$

3.2 Determining Local Extrema

One way to extract the global minimum of a function is to compute all of the local extrema (by solving for $\vec{x}$ in $\nabla \cdot F(x_1, x_2, \ldots, x_n) = (0, 0, \overset{\rightarrow}{\ldots}, 0)$) and find the minimum function value at all the local extrema. This approach succeeds in the non-degenerate case, when the global minimum may correspond to one or more points (a zero dimensional set), and when there are a finite number of local extrema, such as in the case of object localization. We utilize this basic approach in order to find the minimum squared error pose. Let $Error(X, Y, t)$ describe the total squared error as a function of the pose parameters. First, we find all the local extremum of $Error(X, Y, t)$ by solving $\nabla \cdot Error(X, Y, t) = (0, \vec{0}, 0)$. Then, the global minimum is found by comparing the error function at all of the local extrema poses.

Previous attempts to compute the global minimum of the error function using gradient descent methods suffer from problems of local minima because the partial derivatives of the error function are nonlinear functions. In this algorithm, we utilize elimination techniques to solve for all of the roots of the multivariate system of equations and thus avoid the problems of local methods.

3.3 Symbolic Representation

Elimination theory deals with solving for coordinates of simultaneous solutions of multivariable systems [18]. Resultant methods are used for projection and elimination. We use resultants to solve for one particular coordinate at a time; these methods involve constructing a *resultant equation* in each coordinate, where the resultant construction depends on the structure of the underlying system of equations.

The resulting computation involves a considerable amount of symbolic preprocessing. As opposed to constructing the resultant formulation for each data set, we use a generic formulation. The algebraic formulation of the squared error between the transformed points and the lines allows us to use the algebraic equation as a generic representation for the squared error. The total error function between points and linear features can be expressed as a rational polynomial, whose numerator consists of 24 monomials terms. Likewise, the algebraic formulation of the squared approximate error between transformed points and circles allows us to use the algebraic equation as a generic representation for the squared error. The approximate total error function between points and linear features and circular features has a symbolic representation as a rational polynomial with a numerator consisting of 50 monomial terms. The algebraic formulation of the total error functions serves as the intermediate description. After reparameterizing the data sets in terms of algebraic coefficients, only one resultant formulation is necessary.

For each set of points and features, the sum of squared error between the set of points and corresponding features is described by a rational equation in terms of the pose parameters X, Y, t. The numerator of this rational equation is describable by the algebraic coefficients $X^i Y^j t^k$, where our nomenclature is such that $\mathtt{ax_i y_j t_k}$ refers to the coefficient for the monomial $X^i Y^j t^k$.

The multipolynomial resultant is used to eliminate variables from the system of partial derivative equations, and thus to produce a function in one variable. This univariate function is necessarily zero for all values of that variable which are common roots to the system of equations. In the context of this application, the orientation parameter t is the most difficult pose parameter to determine. The first step in both of these techniques involves *eliminating* X, Y to produce a resultant polynomial in t for which the partial derivatives of the error function are simultaneously zero. For each such t, we can then compute the two remaining pose parameters X_t and Y_t. In the case of polygonal models, X_t and Y_t are determined by solving a system of two linear equations in X_t and Y_t. For models with circular arc features, computing $X_{Y_t,t}$ and Y_t involves setting up another resultant to compute Y_t by eliminating X_t, and then computing $X_{Y_t,t}$.

4 Optimal Pose Determination for Models With Linear Features

In this section, we describe the algebraic localization technique for polygonal models given the boundary data points. We define the problem as follows: compute the two-dimensional rigid transformation $\mathcal{T}(X, Y, \theta)$ which best transforms the linear model features $\vec{a}$ onto the data points $\vec{x}$ (refer section 2.4). This is accomplished by finding the two-dimensional rigid transformation $\mathcal{T}(X, Y, \theta)$ which best transforms the data points $\vec{x}$ onto the linear model features $\vec{a}$ and then compute the inverse transformation. Pose estimation is reduced to a least squares error problem, which is accomplished by computing all of the local extrema of the error function, and then checking the error function at all of the local extrema.

The main distinction between our approach and the related work in the literature lies in our use of algebraic elimination theory to solve the simultaneous system of partial derivative equations. We believe that gradient descent methods are unsuitable for solving this system of equations. Algebraic elimination methods are used for the principal step of solving the system of equations to enumerate the poses with local extrema of the error function.

The rest of the section is organized in the following manner. We first present an overview of the algorithm. Then, we present an algebraic formulation of the error between an arbitrary transformed point and an arbitrary line, followed by a generic description of the total error function, in terms of symbolic algebraic coefficients. Then, we detail the symbolic resultant matrix for solving for one of the pose parameters (t) of the generic error function.

The symbolic coefficients are recomputed each time the algorithm is run. In section 4.3, we describe numerical methods for computing all the orientations t corresponding to the local extrema poses. In section 4.4, we describe how the remaining pose parameters X_t, Y_t are computed for each orientation t.

Throughout this section, we will try to elucidate the main concepts using a simple example with four points corresponding to three lines. We will graphically depict the error function between each point and each corresponding linear feature, and we will depict the sum squared error function between the points and linear features. Unfortunately though, since the problem has three degrees of freedom, x, y, θ, and we want to use an additional degree of freedom to describe the error function, we cannot graphically depict the entire problem completely since we do not want to confuse the situation by trying to describe a four dimensional scenario. For these reasons, the graphics depict the three dimensional slice along the hyperplane $x = 0$.

We run through an example in section 4.5, and in section 4.7 we present a data set for which gradient descent methods may fail to find the optimal pose estimate. Finally, we present a faster localization technique for rectilinear objects which exploits a symbolic factoring of the resultant matrix.

4.1 Algorithmic Overview

Algebraic methods are used to compute the orientation, θ, of polygonal objects. This involves online and offline computations: the symbolic resultant matrix for θ is constructed from the symbolic partial derivatives offline; online, the symbolic coefficients are computed given the data set, thereby constructing the symbolic resultant matrix. Then, the resultant matrix is *solved* to enumerate all of the *orientations*, θ, of poses with local extrema of the error function. Then, the remaining pose parameters (X, Y) are determined for each orientation θ. Finally, the global minimum is found by comparing the error of all of the local extrema poses.

The localization algorithm involves four offline steps:

1. Determine the structure of a generic error function by examining an algebraic expression of the error between an arbitrary transformed point and an arbitrary linear feature.

2. Express the error function $Error(X, Y, \theta)$ as a generic algebraic function in terms of symbolic coefficients.

3. Formulate the partial derivatives of the generic error function $Error(X, Y, \theta)$ with respect to the coordinates: X, Y, and θ. The motivation is that each zero-dimensional pose with local extremum error satisfies $\nabla \cdot Error(X, Y, \theta) = \vec{0}$.

4. Eliminate X and Y from the system of partial derivatives $\nabla \cdot Error(X, Y, \theta) = \vec{0}$ in order to generate a expression *solely in* θ. The result of this step is a symbolic resultant matrix which will be used to solve for all of the θ of poses with local extrema error function. Given the orientation, θ, the remaining pose parameters X_θ, Y_θ can be determined by solving a system of linear equations. .

In addition to the offline steps, the localization algorithm involves three online steps (given a data set):

1. Instantiate the symbolic coefficients of the error function using the data set of points and associated linear features.

2. Compute all of the candidate orientation parameters θ by solving the resultant matrix polynomial using eigenvalue techniques (refer section 4.3)

3. $\forall\ \hat{\theta}$, if $\hat{\theta}$ is a candidate orientation ($\mathrm{Im}(\hat{\theta}) \approx 0$) (where $Im(x)$ refers to the imaginary component of a complex number x):

(a) Compute the remaining pose parameters X_θ, Y_θ for each θ by solving the linear system of equations $\frac{\partial Error(X,Y,\theta)}{\partial X}|_{\hat{\theta}} = 0$, $\frac{\partial Error(X,Y,\theta)}{\partial Y}|_{\hat{\theta}} = 0$

(b) Compute $Error(X_{\hat{\theta}}, Y_{\hat{\theta}}, \hat{\theta})$ at each local extremum in order to find the global minimum error pose.

4.2 Points and Linear Features

In this section we highlight the squared error function between points and linear features. Given a point (`X,Y`) and an associated linear feature described by $\vec{a} = (\mathtt{a}, \mathtt{b}, \mathtt{c})$ where $\mathtt{a}X + \mathtt{b}Y = \mathtt{c}$, the error between the point and the feature is: `aX + bY - c`, or $(\mathtt{a}, \mathtt{b}, \Leftrightarrow\mathtt{c}) \cdot (\mathtt{X}, \mathtt{Y}, \mathtt{1})$. Let $\vec{a}$ be $(\mathtt{a}, \mathtt{b}, \Leftrightarrow\mathtt{c})^{\mathsf{T}}$ and $\vec{x}$ be $(\mathtt{X}, \mathtt{Y}, \mathtt{1})$. The squared error between the point $\vec{x}$ and the feature $\vec{a}$, as a function of the transformation $\mathcal{T}(X, Y, \theta)$ is given in equation (6). The total squared error for all of the points and associated features is given in equation (7). In order to realize rational functions, we use the trigonometric substitution $t = \tan(\frac{\theta}{2})$ to arrive at equation (7).

To circumvent the problems related to numerical inaccuracy, the points and features in the data set should be centered at the origin before invoking the algorithm, and the optimal transformation should be complementarily transformed afterwards. This reduces the symbolic coefficients and thus improves the numerical precision.

$$\|(\mathtt{a}, \mathtt{b}, \Leftrightarrow\mathtt{c})^{\mathsf{T}} \cdot (\mathcal{T}(X, Y, \theta)(\mathtt{X}, \mathtt{Y}, \mathtt{1})^{\mathsf{T}})\|^2 = \frac{\|(\mathtt{a}, \mathtt{b}, \Leftrightarrow\mathtt{c})^{\mathsf{T}} \cdot (Mat(X, Y, t)(\mathtt{X}, \mathtt{Y}, \mathtt{1})^{\mathsf{T}})\|^2}{(1+t^2)^2} \tag{6}$$

$$Error(X, Y, \theta, \vec{x}, \vec{a}) = \|\vec{a}^{\mathsf{T}} \cdot (\mathcal{T}(X, Y, \theta)\vec{x}^{\mathsf{T}})\|^2 = \frac{\|\vec{a}^{\mathsf{T}} \cdot (Mat(X, Y, t)\vec{x}^{\mathsf{T}})\|^2}{(1+t^2)^2}$$

$$Error(X, Y, \theta) = Error(X, Y, t) = \sum_i Error(X, Y, \theta, \vec{x}_i, \vec{a}_i) =$$

$$\sum_i \|\vec{a_i}^{\mathsf{T}} \cdot (\mathcal{T}(X, Y, \theta)\vec{x_i}^{\mathsf{T}})\|^2 = \frac{\sum_i \|\vec{a_i}^{\mathsf{T}} \cdot (Mat(X, Y, t)\vec{x_i}^{\mathsf{T}})\|^2}{(1+t^2)^2} \tag{7}$$

For example, consider the case of the four points and corresponding three lines depicted in Figure 3. One pose which exactly maps the four points onto their corresponding features is given in Figure 4. The four points are: $(\frac{7}{2}, \frac{-1}{2}), (\frac{9}{2}, \frac{-1}{2}), (4 + \frac{\sqrt{3}}{2}, \frac{1}{2}), (4 \Leftrightarrow \frac{\sqrt{3}}{2}, \frac{1}{2})$, and the three associated lines are $x + 1 = 0$, $\frac{1}{2}x \Leftrightarrow \frac{\sqrt{3}}{2}y \Leftrightarrow 1 = 0$, $\frac{1}{2}x + \frac{\sqrt{3}}{2}y \Leftrightarrow 1 = 0$; the first two points both correspond to the first line.

In order to illustrate the concept of an error function, consider the pose $\theta = 0, y = 0$ (refer Figure 5). In this case, the error between the first point and first corresponding linear

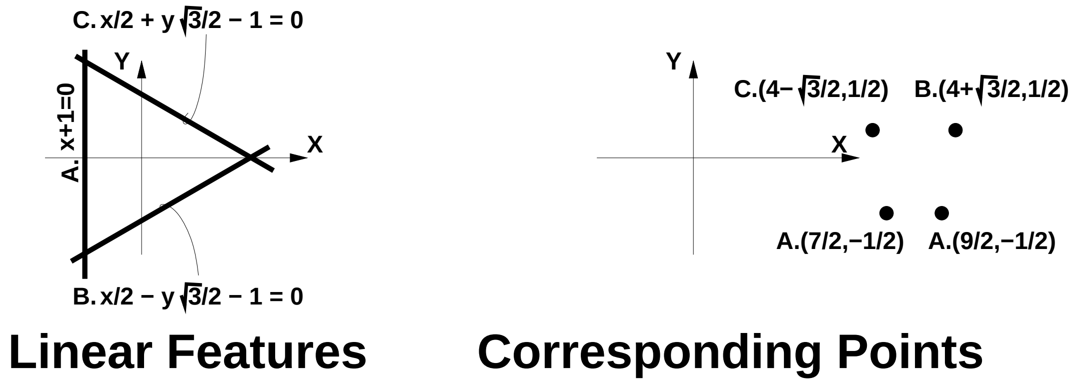


Figure 3: The simple example used to illustrate the main concepts

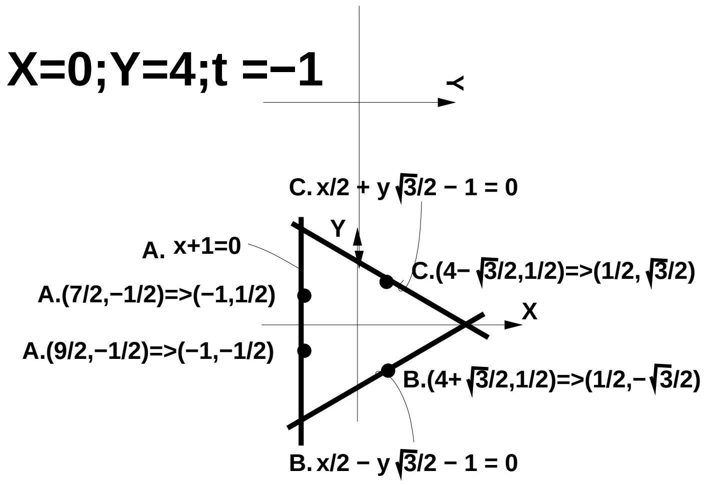


Figure 4: One solution transformation which maps the four points onto the corresponding linear features.

feature is $\frac{9}{2}$, and the error of the second point is $\frac{11}{2}$, and 1 for the both the third and fourth points.

To further illustrate the concept of the error function, in Figure 6, we depict the squared error function between the transformed points and the associated linear features as functions of y and t. The total error function $Error(Y, \theta)$, which is the sum of the four individual squared error functions is given in Figure 7. The task, which we use resultant techniques to accomplish, is to efficiently find the global minimum of this error function; notice that the global minimum of the error function is zero at $\theta = \frac{3\pi}{2}$, $Y = 4$, which corresponds to mapping the vertices directly onto the corresponding linear features.

The sum of squared error between a set of points and corresponding lines is an algebraic function of the translation and rotation matrix $Mat(X, Y, t)$ (multiplied by $(1+t^2)^2$). The

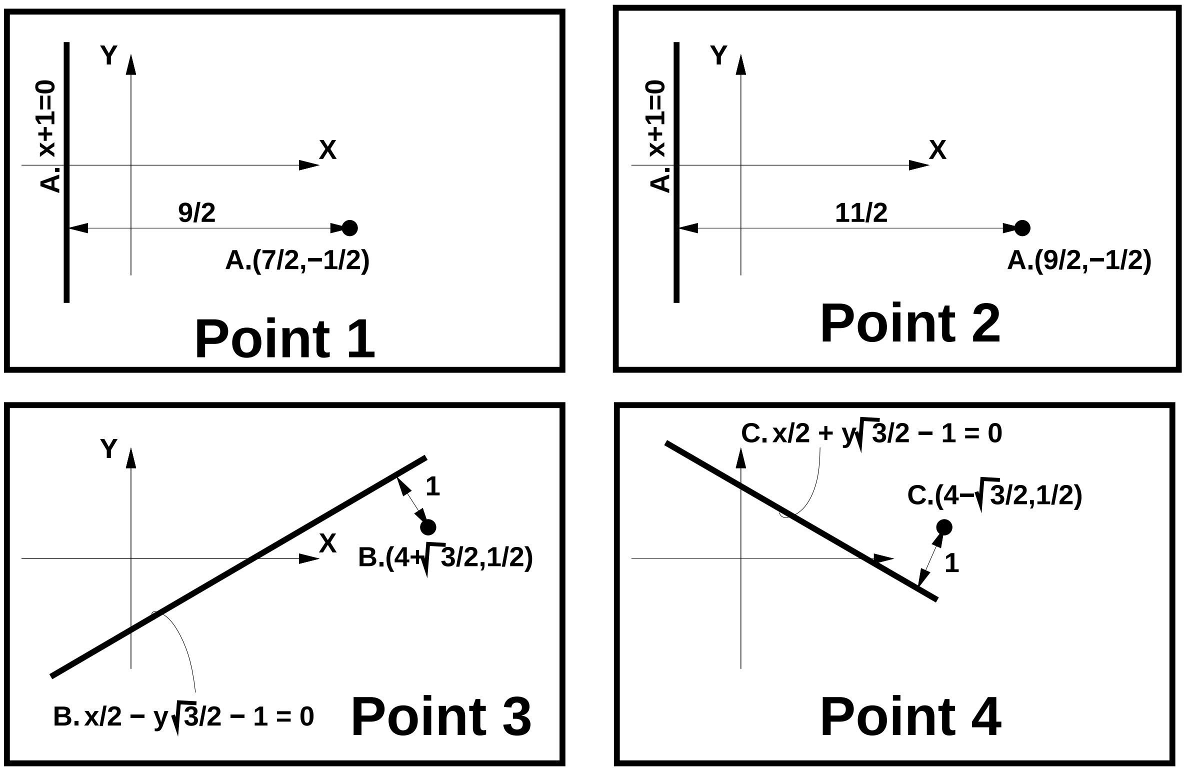


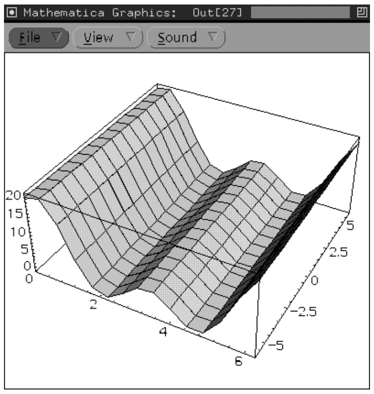
Figure 5: The distance between the points and the lines following the identity transformation $X = Y = t = 0$.

coefficients $\mathtt{ax_iy_jt_k}$ are computable using the data set of points and associated linear features. The total sum squared error function (equation (7)) is multiplied by $(1+t^2)^2$ to arrive at a polynomial $FF(X,Y,t)$ in X,Y and t (equations (8),(9)).

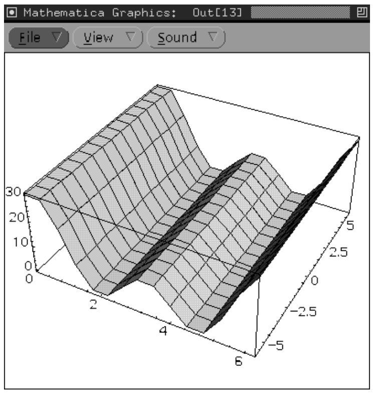
For each point $\vec{x}$ and associated linear feature $\vec{a}$, the function $FF(X,Y,t,\vec{x},\vec{a})$ is a polynomial expression in X,Y and t. The total function $FF(X,Y,t)$ is the sum of all the individual $FF(X,Y,t,\vec{x},\vec{a})$ error functions, and we can keep a running tally of all of the the symbolic coefficients (`a, ax1, ax1y1, ...`).

$$Error(X,Y,t) = \frac{FF(X,Y,t)}{(1+t^2)^2} \tag{8}$$

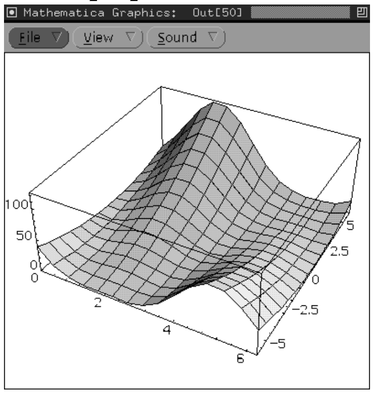
$$\begin{aligned} FF(X,Y,t) = \sum_i \|\vec{a_i}^{\mathsf{T}} \cdot (Mat(X,Y,t)\vec{x_i}^{\mathsf{T}})\|^2 = \mathtt{a} + X\mathtt{ax1} + XY\mathtt{ax1y1} + X^2\mathtt{ax2} + Y\mathtt{ay1} + \\ Y^2\mathtt{ay2} + t\mathtt{at1} + Xt\mathtt{ax1t1} + Yt\mathtt{ay1t1} + t^2\mathtt{at2} + Xt^2\mathtt{ax1t2} + XYt^2\mathtt{ax1y1t2} + \\ X^2t^2\mathtt{ax2t2} + Yt^2\mathtt{ay1t2} + Y^2t^2\mathtt{ay2t2} + t^3\mathtt{at3} + Xt^3\mathtt{ax1t3} + Yt^3\mathtt{ay1t3} + \\ t^4\mathtt{at4} + Xt^4\mathtt{ax1t4} + XYt^4\mathtt{ax1y1t4} + X^2t^4\mathtt{ax2t4} + Yt^4\mathtt{ay1t4} + Y^2t^4\mathtt{ay2t4} \end{aligned} \tag{9}$$



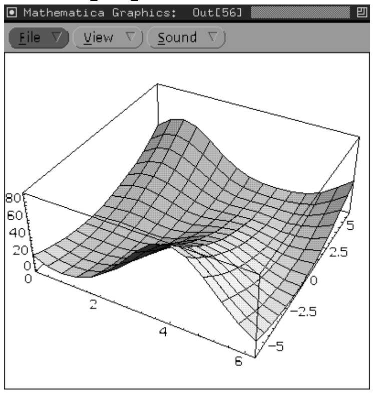
$\vec{x}_1 = (\frac{7}{2}, \frac{-1}{2}), \vec{a}_1 = (1, 0, \Leftrightarrow 1)$



$\vec{x}_2 = (\frac{9}{2}, \frac{-1}{2}), \vec{a}_2 = (1, 0, \Leftrightarrow 1)$



$\vec{x}_3 = (4 + \frac{\sqrt{3}}{2}, \frac{1}{2}), \vec{a}_3 = (\frac{1}{2}, \frac{-\sqrt{3}}{2}, 1)$



$\vec{x}_4 = (4 \Leftrightarrow \frac{\sqrt{3}}{2}, \frac{1}{2}), \vec{a}_4 = (\frac{1}{2}, \frac{\sqrt{3}}{2}, 1)$

Figure 6: The three dimensional plot of the error function $Error(Y, \theta, \vec{x}_i, \vec{a}_i)$ for each of the four points and corresponding linear features.

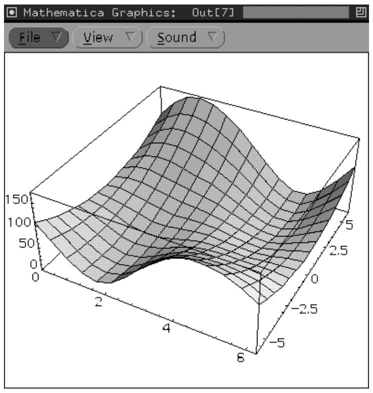


Figure 7: The total error function $Error(Y, \theta)$. The task, which we use resultant techniques to accomplish, is to efficiently find the global minimum of this error function.

In the case of polygonal objects, some of the algebraic coefficients are redundant because the generic algebraic expression includes coefficients which are dependent. We note these redundancies because they can be exploited in the resultant construction.

$$\begin{array}{cccc} \texttt{ax1y1t2} = 2\ \texttt{ax1y1} & \texttt{ay2t2} = 2\ \texttt{ay2} & \texttt{ax2t2} = 2\ \texttt{ax2} & \texttt{ax2t4} = \texttt{ax2} \\ \texttt{ax1y1t4} = \texttt{ax1y1} & \texttt{ay2t4} = \texttt{ay2} & \texttt{ax1t3} = \texttt{ax1t1} & \texttt{ay1t3} = \texttt{ay1t1} \end{array}$$

All of the minima of $Error(X,Y,t)$, both local and global, have partial derivatives (with respect to X,Y,t) equal to zero. To solve for the global minimum of $Error(X,Y,t)$, we want to find the configurations X,Y,t for which the partial derivatives are zero (refer equation (10)).

$$\nabla \cdot Error(X,Y,\theta) = (0,\vec{0},0) \Rightarrow \nabla \cdot Error(X,Y,t) = (0,\vec{0},0) \tag{10}$$

Resultant techniques require algebraic (not rational) functions. Therefore, we replace the *rational* partial derivatives with *equivalent algebraic* counterparts. Since we are only finding the common roots of equation (10), we are at liberty to use any function $G_X(X,Y,\theta)$ for $\frac{\partial Error(X,Y,t)}{\partial X}$ where $\frac{\partial Error(X,Y,t)}{\partial X} = 0 \Rightarrow G_X(X,Y,\theta) = 0$ (and G_y for $\frac{\partial Error(X,Y,t)}{\partial Y}$ and G_t for $\frac{\partial Error(X,Y,t)}{\partial t}$) .

For $\frac{\partial Error(X,Y,t)}{\partial X}$ and $\frac{\partial Error(X,Y,t)}{\partial Y}$, $G_X(X,Y,t)$ and $G_Y(X,Y,t)$ are equal to the partial derivatives $\frac{\partial FF(X,Y,t)}{\partial X}$, $\frac{\partial FF(X,Y,t)}{\partial Y}$. $G_t(X,Y,t)$ is slightly more complicated because the denominator of $\frac{\partial Error(X,Y,t)}{\partial t}$ is a function of t, and that must be taken into account.

$$\begin{aligned} \frac{\partial Error(X,Y,t)}{\partial X} &\propto \frac{\partial FF(X,Y,t)}{\partial X} \\ \Rightarrow G_X(X,Y,t) &= \frac{\partial FF(X,Y,t)}{\partial X} \\ &= 2X\texttt{ax2} + 4X\texttt{ax2}t^2 + 2X\texttt{ax2}t^4 \qquad (11) \\ &\quad +Y\texttt{ax1y1} + 2Y\texttt{ax1y1}t^2 + Y\texttt{ax1y1}t^4 \\ &\quad +\texttt{ax1} + \texttt{ax1t1}t + \texttt{ax1t2}t^2 + \texttt{ax1t3}t^3 + \texttt{ax1t4}t^4 \\ \frac{\partial Error(X,Y,t)}{\partial Y} &\propto \frac{\partial FF(X,Y,t)}{\partial Y} \\ \Rightarrow G_Y(X,Y,t) &= \frac{\partial FF(X,Y,t)}{\partial Y} \\ &= X\texttt{ax1y1} + 2X\texttt{ax1y1}t^2 + X\texttt{ax1y1}t^4 \\ &\quad 2Y\texttt{ay2} + 4Y\texttt{ay2}t^2 + 2Y\texttt{ay2}t^4 \\ &\quad \texttt{ay1} + \texttt{ay1t1}t + \texttt{ay1t2}t^2 + \texttt{ay1t3}t^3 + +\texttt{ay1t4}t^4 \qquad (12) \\ \frac{\partial Error(X,Y,t)}{\partial t} &\propto \frac{\partial \frac{FF(X,Y,t)}{(1+t^2)^2}}{\partial t} = \frac{((1+t^2)\frac{\partial FF(X,Y,t)}{\partial t} \Leftrightarrow 4tFF(X,Y,t))}{(1+t^2)^3} \qquad (13) \end{aligned}$$

$$\begin{aligned}
\Rightarrow G_t(X,Y,t) &= ((1+t^2)\frac{\partial FF(X,Y,t)}{\partial t} \Leftrightarrow 4tFF(X,Y,t)) && (14)\\
\frac{\partial FF(X,Y,t)}{\partial t} &= \texttt{at1} + X\texttt{ax1t1} + Y\texttt{ay1t1} + 2\texttt{at2}t + 2X\texttt{ax1t2}t + 4XY\texttt{ax1y1}t && (15)\\
&+4X^2\texttt{ax2}t + 2Y\texttt{ay1t2}t + 4Y^2\texttt{ay2}t + 3\texttt{at3}t^2 + 3X\texttt{ax1t3}t^2 + 3Y\texttt{ay1t3}t^2\\
&+4\texttt{at4}t^3 + 4X\texttt{ax1t4}t^3 + 4XY\texttt{ax1y1}t^3 + 4X^2\texttt{ax2}t^3\\
&+4Y\texttt{ay1t4}t^3 + 4Y^2\texttt{ay2}t^3\\
G_t(X,Y,t) &= \texttt{ax1t1}X \Leftrightarrow 4\texttt{ax1}tX + 2\texttt{ax1t2}tX \Leftrightarrow 3\texttt{ax1t1}t^2X + 3\texttt{ax1t3}t^2X \Leftrightarrow 2\texttt{ax1t2}t^3X\\
&+4\texttt{ax1t4}t^3X \Leftrightarrow \texttt{ax1t3}t^4X\\
&+\texttt{ay1t1}Y \Leftrightarrow 4\texttt{ay1}tY + 2\texttt{ay1t2}tY \Leftrightarrow 3\texttt{ay1t1}t^2Y + 3\texttt{ay1t3}t^2Y \Leftrightarrow 2\texttt{ay1t2}t^3Y\\
&+4\texttt{ay1t4}t^3Y \Leftrightarrow \texttt{ay1t3}t^4Y\\
&+\texttt{at1} \Leftrightarrow 4\texttt{a}t + 2\texttt{at2}t \Leftrightarrow 3\texttt{at1}t^2 + 3\texttt{at3}t^2 \Leftrightarrow 2\texttt{at2}t^3 + 4\texttt{at4}t^3 \Leftrightarrow \texttt{at3}t^4 && (16)
\end{aligned}$$

4.2.1 Example and Explanation of Resultant Methods

Elimination of X, Y from this system of equations (equation (11)=0, equation (12)=0, equation (16)=0) expresses the resultant as the determinant of a matrix polynomial in t. We now show how these systems are solved using elimination theory. Consider the three functions $G_X(X,Y,t)$, $G_Y(X,Y,t)$, $G_t(X,Y,t)$ which are proportional to the partial derivatives: $\frac{\partial Error(X,Y,t)}{\partial X}$, $\frac{\partial Error(X,Y,t)}{\partial Y}$, $\frac{\partial Error(X,Y,t)}{\partial t}$. Notice that $G_X(X,Y,t)$, $G_Y(X,Y,t)$, $G_t(X,Y,t)$ are all quartic in t, *but linear in X and Y*. Resultants can be constructed for any algebraic system of equations, but they are simplest for cases in which the equations are linear in all but a single variable. We can rewrite the three equations as linear functions of X,Y, and t:

$$\begin{aligned}
G_X(X,Y,t) &= f_{11}(t)X + f_{12}(t)Y + f_{13}(t) = 0\\
G_Y(X,Y,t) &= f_{21}(t)X + f_{22}(t)Y + f_{23}(t) = 0\\
G_t(X,Y,t) &= f_{31}(t)X + f_{32}(t)Y + f_{33}(t) = 0
\end{aligned}$$

Consider the matrix equation (equation 17); in order for the partial derivatives to be simultaneously zero, $(X,Y,1)^{\mathsf{T}}$ must be in the null space of the *resultant* matrix (refer equation (17)). The resultant matrix has a null space if and only if the determinant of the resultant matrix is zero. Furthermore, any solution to the system of equations must lie in the null space of the resultant matrix. We have therefore derived a relationship between roots of the system of equations, and a single expression in a single variable. If t is a coordinate of a simultaneous solution of this system of equations, then the determinant of the resultant matrix as a function of t must be zero. We can solve for all values of t using the resultant matrix, by finding all t such that $|ResultantMatrix(t)| = 0$ (where $|ResultantMatrix(t)|$ signifies the determinant of $ResultantMatrix(t)$) and then we can solve for X_t and Y_t for each t.

$$\begin{bmatrix} G_X(X,Y,t) \\ G_Y(X,Y,t) \\ G_t(X,Y,t) \end{bmatrix} = \underbrace{\begin{bmatrix} f_{11}(t) & f_{12}(t) & f_{13}(t) \\ f_{21}(t) & f_{22}(t) & f_{23}(t) \\ f_{31}(t) & f_{32}(t) & f_{33}(t) \end{bmatrix}}_{ResultantMatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{17}$$

Solving the system of equations is reduced to finding roots of the determinant of a matrix polynomial, where $M(t)$ denotes the resultant matrix.

$$\begin{aligned} M(t) &= Resultant(\{\frac{\partial Error(X,Y,t)}{\partial X}, \frac{\partial Error(X,Y,t)}{\partial Y}, \frac{\partial Error(X,Y,t)}{\partial t}\}, \{X,Y\}) \\ &= Resultant(\{G_X(X,Y,t), G_Y(X,Y,t), G_t(X,Y,t)\}, \{X,Y\}) \qquad (18) \\ &= \begin{bmatrix} \begin{matrix} 2\mathtt{ax2} + 4\mathtt{ax2}t^2 \\ +2\mathtt{ax2}t^4 \end{matrix} & \begin{matrix} \mathtt{ax1y1} + 2\mathtt{ax1y1}t^2 \\ +\mathtt{ax1y1}t^4 \end{matrix} & \begin{matrix} \mathtt{ax1} + \mathtt{ax1t1}t + \mathtt{ax1t2}t^2 \\ \mathtt{ax1t3}t^3 + \mathtt{ax1t4}t^4 \end{matrix} \\ \\ \begin{matrix} \mathtt{ax1y1} + 2\mathtt{ax1y1}t^2 \\ +\mathtt{ax1y1}t^4 \end{matrix} & \begin{matrix} 2\mathtt{ay2} + 4\mathtt{ay2}t^2 \\ +2\mathtt{ay2}t^4 \end{matrix} & \begin{matrix} \mathtt{ay1} + \mathtt{ay1t1}t + \mathtt{ay1t2}t^2 \\ \mathtt{ay1t3}t^3 + \mathtt{ay1t4}t^4 \end{matrix} \\ \\ \begin{matrix} \mathtt{ax1t1} \Leftrightarrow 4\mathtt{ax1t1}t \\ +2\mathtt{ax1t2}t \Leftrightarrow 3\mathtt{ax1t1}t^2 \\ +3\mathtt{ax1t3}t^2 \Leftrightarrow 2\mathtt{ax1t2}t^3 \\ +4\mathtt{ax1t4}t^3 \Leftrightarrow \mathtt{ax1t3}t^4 \end{matrix} & \begin{matrix} \mathtt{ay1t1} \Leftrightarrow 4\mathtt{ay1t1}t \\ +2\mathtt{ay1t2}t \Leftrightarrow 3\mathtt{ay1t1}t^2 \\ +3\mathtt{ay1t3}t^2 \Leftrightarrow 2\mathtt{ay1t2}t^3 \\ +4\mathtt{ay1t4}t^3 \Leftrightarrow \mathtt{ay1t3}t^4 \end{matrix} & \begin{matrix} \mathtt{at1} \Leftrightarrow 4\mathtt{a}t + 2\mathtt{at2}t \\ \Leftrightarrow 3\mathtt{at1}t^2 + 3\mathtt{at3}t^2 \Leftrightarrow 2\mathtt{at2}t^3 \\ +4\mathtt{at4}t^3 \Leftrightarrow \mathtt{at3}t^4 \\ \end{matrix} \end{bmatrix} \end{aligned}$$

4.3 Solving the Matrix Polynomial

$M(t)$ was obtained (equation (18)) by eliminating X and Y from the equations corresponding to the partial derivatives of $Error(X,Y,t)$. In this section, we highlight a numerically accurate and efficient algorithm to compute the roots of $|M(t)| = 0$. All of the t coordinates of the common solutions of the algebraic equations are roots of $|M(t)| = 0$.

A simple procedure for root finding involves expanding the symbolic determinant and computing the roots of the resulting univariate polynomial. This procedure involves a significant amount of symbolic computation (in determinant expansion), which makes it relatively slow.

Moreover, the problem of computing roots of polynomials of degree greater than 10 are can be numerically ill-conditioned. As a result, the solutions obtained using IEEE floating point arithmetic (or any other model of finite precision arithmetic) are likely to be numerically inaccurate. To circumvent these problems, we use matrix computations.

Solving $|M(t)| = 0$ is reduced to determining the eigenvalues of the block companion matrix E. The construction of the block companion matrix E is highlighted in equation (19). It involves matrix inversion and matrix products. In this section we assume that the leading matrix, M_4, is numerically well-conditioned. The eigenvalues of E are exactly the roots of the equations $|M(t)| = 0$ [19] computed using routines from LAPACK [2] and EISPACK [8]. Solving for the roots of $M(t)$ in this manner is numerically better conditioned than solving the resultant polynomial formed by expanding the resultant matrix.

$$M(t) = M_4t^4 + M_3t^3 + M_2t^2 + M_1t + M_0 \tag{19}$$

$$E = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \\ M_4^{-1}M_0 & M_4^{-1}M_1 & M_4^{-1}M_2 & M_4^{-1}M_3 \end{bmatrix} \tag{20}$$

4.3.1 Identifying Possibly Valid Solutions

Resultant formulations can generate extraneous solutions; i.e., roots of the resultant which have no corresponding solutions in the original system of equations. The normal approach is to consider each root of the resultant a candidate solution. The disadvantage of treating all solutions as candidates is that the resultant formalism generates complex solutions even when only real solutions are desired. In this case of matching points to linear features, the orientation t is expected to be real. Furthermore, the resultant construction introduces multiple extraneous roots $i, i, \Leftrightarrow i, \Leftrightarrow i$, and this is seen by the fact that the symbolic resultant's characteristic polynomial is divisible by $(1 + t^2)^2$. We ignore complex solutions by thresholding the imaginary components.

4.3.2 Improving the Accuracy

The eigenvalue decomposition depends upon the invertibility of the largest exponent matrix (in this case, M_4 is well conditioned and the inverse computation does not lead to ill-conditioned matrices). Sometimes, the highest degree matrix M_4 is symbolically rank deficient. In this case, we use an algebraic substitution ($s = f(t)$), and generate a matrix polynomial M' such that $M'(s) = 0 \Leftrightarrow M(t) = 0$ with an invertible largest exponent matrix M'_4 ($M'_4 = Coefficient[M', s^4]$) (where $Coefficient[\mathcal{E}, t^i]$ describes the coefficient of the t^i term in the expression $\mathcal{E}$). Two simple transformations are the geometric inverse ($t = \frac{1}{s}$) and a random generic rational transformation ($t = \frac{\alpha s+\beta}{\gamma s+\delta}$). We first try the geometric inverse substitution $t = \frac{1}{s}$ which corresponds to checking whether M_0 is numerically well-conditioned.

Let $\mathbf{d}_{\mathrm{M}}$ be M's polynomial degree. If $M_0(t)$ is not well conditioned, we instead try a number of generic rational transformations: $t = \frac{\alpha s+\beta}{\gamma s+\delta}$, with random $\alpha, \beta, \gamma, \delta$. If any of these transformations produces a well conditioned matrix M'_4 signified by its condition number, we use the random transformation corresponding to the best conditioned matrix. We compute the eigenvalues s of the related large matrix E' (the companion matrix of the M' matrix), to find all of the s for which $|M'(s)| = 0$. The t values for which $|M(t)| = 0$ are computed by applying the inverse function f^{-1} to the s values.

$$\begin{aligned} t = \frac{1}{s} \quad &\rightarrow \quad M_{t=1/s}(s) = s^{\mathbf{d}_{\mathrm{M}}} M(t) \\ &\rightarrow \quad Coefficient[M_{t=1/s}, s^d] = Coefficient[M, t^{\mathbf{d}_{\mathrm{M}}-d}] \\ t = \frac{\alpha s+\beta}{\gamma s+\delta} \quad &\rightarrow \quad M_{t=\frac{\alpha s+\beta}{\gamma s+\delta}}(s) = M(t)(\gamma s+\delta)^{\mathbf{d}_{\mathrm{M}}} \end{aligned} \tag{21}$$

If none of the rational transformations lead to numerically invertible largest exponent matrices, we can use the generalized eigenvalue formulation.

4.3.3 Generalized Eigenvalue Formulation

In case M_4 is singular or numerically ill-conditioned and none of the transformations produces a well conditioned matrix, we reduce root finding to a generalized eigenvalue problem of a matrix pencil [9]. The matrix pencil for the generalized eigenvalue problem is constructed from the matrix polynomial in the following manner. Let

$$C_1 = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & M_4 \end{bmatrix} \qquad C_2 = \begin{bmatrix} 0 & \Leftrightarrow I & 0 & 0 \\ 0 & 0 & \Leftrightarrow I & 0 \\ 0 & 0 & 0 & \Leftrightarrow I \\ M_0 & M_1 & M_2 & M_3 \end{bmatrix} \tag{22}$$

and the eigenvalues of $C_1 t + C_2$ correspond exactly to the roots of $|M(t)| = 0$.

4.4 Computing X_t and Y_t

Given the value of t computed using eigenvalue routines, the corresponding X_t and Y_t coordinates of the roots of the algebraic equations are found in constant time by solving $\frac{\partial Error(X,Y,t)}{\partial X}|_t = 0$ and $\frac{\partial Error(X,Y,t)}{\partial Y}|_t = 0$. The two linear equations in X and Y can be solved using Cramer's rule.

4.4.1 Verifying That (X_t, Y_t, t) is a Local Extremum

The method highlighted above is necessary *but not sufficient* and can produce extraneous solutions. Therefore, we need to verify that each (X_t, Y_t, t) configuration is actually a local extremum. This is accomplished by computing the absolute values of the partial derivatives of the error function and comparing them to some ϵ value, where the ϵ should be non-zero to accommodate sensor noise and numerical error.

4.5 Example

In this section, we step through an example in order to illustrate how we use resultants to solve the non-linear least squares problem. Notice that we do not recompute $G_x(X,Y,t)$, $G_Y(X,Y,t)$, or $G_t(X,Y,t)$ online. $G_x(X,Y,t)$, $G_Y(X,Y,t)$, or $G_t(X,Y,t)$ were intermediate offline results used to construct the symbolic resultant matrix. In other words, the symbolic resultant matrix is the *compiled* result of the elimination. For brevity, we present the intermediary numeric values to four decimal places, though we use double precision arithmetic.

The resultant computation is comprised of seven steps:

1. Compute the symbolic coefficients using the data set
2. Construct the symbolic resultant using the symbolic coefficients
3. If necessary, apply transformations in order to find a well conditioned invertible highest exponent matrix
4. Construct the companion matrix using the resultant matrix
5. Compute the eigenvalues of the companion matrix
6. If the algorithm applied a transformation in step 3, find the roots of t by applying an inverse function to the eigenvalues
7. Compute the remaining pose parameters, X_t,Y_t for each candidate, t

Data point $(x_{i,1}, x_{i,2})$	Linear Feature Algebraic Parameters $(a_{i,1}, a_{i,2}, a_{i,3})$
(-7.91, -7.91)	(-0.007534555543, 0.999971614834, -9.004401406730)
(7.91, 7.91)	(-0.007534555543, 0.999971614834, 6.805099825207)
(-7.91, 7.91)	(0.700109199157, 0.714035789899, -12.166817390266)
(7.91, -7.91)	(0.700109199157, 0.714035789899, 10.050656124962)
(-7.91, -7.91)	(-0.710861891474, 0.703331622529, -11.545580060073)
(7.91, 7.91)	(-0.710861891474, 0.703331622529, 10.561258166615)

Table 1: Example data set: data points and corresponding algebraically parameterized linear features. The task is to compute the transformation which optimally transforms the set of points onto the corresponding linear features.

4.5.1 Evaluating the Symbolic Coefficients

The data set of points and associated linear features is given in Table 1. The symbolic coefficients' values shown in Table 2 are computed from the points and features. They describe the coefficients of the sum of the algebraic error functions (refer equation(23)). We present $FF(X,Y,t,(\Leftrightarrow 7.91,\vec{\Leftrightarrow 7}.91,0.0),(0.0075,1.0\vec{0}0,\Leftrightarrow 9.0044))$ in equation (24) and Figure 8.

$$\begin{aligned} FF(X,Y,t) &= FF(X,Y,t,(\Leftrightarrow 7.91,\vec{\Leftrightarrow 7}.91,0.0),(0.0075,1.0\vec{0}0,\Leftrightarrow 9.0044)) \\ &\quad +FF(X,Y,t,(7.91,\vec{7}.91,0.0),(0.0075,1.0\vec{0}0,6.8041)) \\ &\quad +FF(X,Y,t,(\Leftrightarrow 7.91,\vec{7}.91,0.0),(0.7001,0.71\vec{4}0,\Leftrightarrow 12.1668))+\ldots \end{aligned} \tag{23}$$

$$\begin{aligned} &FF(X,Y,t,(\Leftrightarrow 7.91,\vec{\Leftrightarrow 7}.91,0.0),(0.0075,1.0\vec{0}0,\Leftrightarrow 9.0044)) = \\ &\quad 1.3322 \Leftrightarrow 36.7938t + 292.9516t^2 \Leftrightarrow 537.2817t^3 + 284.0768t^4 \\ &\quad +2.3084Y \Leftrightarrow 31.8766Yt + 36.0166Yt^2 \Leftrightarrow 31.8766Yt^3 + 33.7082Yt^4 \\ &\quad +0.9999(1+t^2)^2Y^2 + 0.0001(1+t^2)^2X^2 \Leftrightarrow 0.0151(1+t^2)^2XY \\ &\quad \Leftrightarrow 0.0174X + 0.2402Xt \Leftrightarrow 0.2714Xt^2 + 0.2402Xt^3 \Leftrightarrow 0.2540Xt^4 \end{aligned} \tag{24}$$

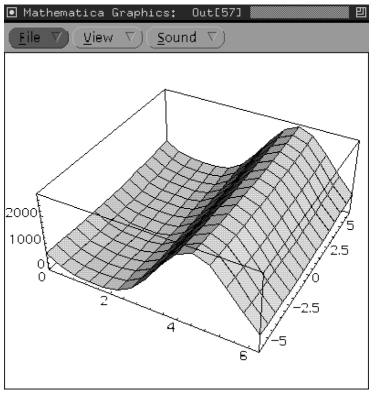


Figure 8: The total error function $Error(Y, \theta)$ for this example

a	503.8720	$\mathtt{at}_4$	985.2446	$\mathtt{ax}_1\mathtt{t}_4$	1.5305	$\mathtt{ay}_1\mathtt{t}_1$	0
$\mathtt{at}_1$	-2001.9285	$\mathtt{ax}_1$	1.5305	$\mathtt{ax}_2$	1.9911	$\mathtt{ay}_1\mathtt{t}_2$	17.6102
$\mathtt{at}_2$	3506.1299	$\mathtt{ax}_1\mathtt{t}_1$	0	$\mathtt{ax}_1\mathtt{y}_1$	-0.0304	$\mathtt{ay}_1\mathtt{t}_4$	8.8051
$\mathtt{at}_3$	-2972.5279	$\mathtt{ax}_1\mathtt{t}_2$	3.0610	$\mathtt{ay}_1$	8.8051	$\mathtt{ay}_2$	4.0089

Table 2: The values of the symbolic coefficients which are used in the resultant formulation: $FF(X, Y, t) = \mathtt{a} + \mathtt{at1}t + \ldots$

4.5.2 Constructing the Matrix Resultant M Using the Symbolic Coefficients

The resultant $M(t)$ in equation (25) is generated from equation (18).

$$M(t) = \begin{bmatrix} 3.9821\mathrm{t}^4 + 7.9643\mathrm{t}^2 + 3.9821 & \Leftrightarrow 0.0304t^4 \Leftrightarrow 0.0608t^2 \Leftrightarrow 0.0304, & 0 \\ \Leftrightarrow 0.0304t^4 \Leftrightarrow 0.0608t^2 \Leftrightarrow 0.0304 & 8.0179t^4 + 16.0357t^2 + 8.0179 & 0 \\ 1.5305t^4 + 3.0610t^2 + 1.5305 & 8.8051t^4 + 17.6102t^2 + 8.8051 & \psi \end{bmatrix} \tag{25}$$

$$\psi = 2972.5279t^4 \Leftrightarrow 3071.2816t^3 \Leftrightarrow 2911.7982t^2 + 4996.7719t \Leftrightarrow 2001.9285$$

4.5.3 Constructing the Companion Matrix E

At this point, we need to determine whether the highest exponent matrix M_4 is numerically invertible. We check its invertibility by computing the condition number for a particular random value of t (in this case, 0.396465). $M_4(t = 0.396465)$ (equation (26)) is obviously not uniformly degenerate since the matrix is well conditioned for a random value of t.

$$M_4(t = 0.396465) = \begin{bmatrix} 5.3324 & \Leftrightarrow 0.0407 & 0 \\ \Leftrightarrow 0.0407 & 10.7365 & 0 \\ 2.0495 & 11.7907 & \Leftrightarrow 596.5278 \end{bmatrix} \tag{26}$$

Computing the companion matrix E of M involves inverting the highest degree matrix (in this case M_4) and then left-multiplying all of the matrixes by M_4^{-1}. We compute $M_4^{-1}M_0 \ldots M_4^{-1}M_3$ and combine them with an identity matrix to construct E.

$$M_4^{-1} = \begin{bmatrix} 0.2511 & 0.0010 & 0 \\ 0.0010 & 0.1247 & 0 \\ \Leftrightarrow 0.0001 & \Leftrightarrow 0.0004 & 0.0003 \end{bmatrix} ; M_4^{-1}M_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \Leftrightarrow 0.6735 \end{bmatrix}$$

$$M_4^{-1}M_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1.6810 \end{bmatrix} ; M_4^{-1}M_2 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & \Leftrightarrow 0.9796 \end{bmatrix} ; M_4^{-1}M_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Leftrightarrow 1.0332 \end{bmatrix}$$

$$E = \begin{bmatrix} \multicolumn{3}{c}{\mathbf{0}^{3\times 9}} & & & & \multicolumn{3}{c}{\mathbf{I}^{9\times 9}} & & & \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \Leftrightarrow 1.0332 & 0 & 0 & \Leftrightarrow 0.9796 & 0 & 0 & 1.6810 & 0 & 0 & \Leftrightarrow 0.6735 \end{bmatrix}$$

4.5.4 Determining Roots of the Matrix Polynomial by Computing the Eigenvalues of Companion Matrix E

The eigenvalues of E are given in equation (27). Some of the 12 eigenvalues of E (E is a square 12×12 matrix) are complex (with imaginary components $> \epsilon$). This is due to the fact that the algebraic formulation of the transformation $Mat(X, Y, t)$ defines a function for complex values of t as well.

$$Eigenvalues(E) = \left\{ \begin{array}{cc} 0.000000009726 + 1.000000009241\,\mathrm{i} & 0.000000009726 + \text{-}1.000000009241\,\mathrm{i} \\ \text{-}0.000000009726 + 0.999999990759\,\mathrm{i} & \text{-}0.000000009726 + \text{-}0.999999990759\,\mathrm{i} \\ \text{-}1.231429745741 + 0.000000000000\,\mathrm{i} & \text{-}0.000000005427 + 1.000000002712\,\mathrm{i} \\ \text{-}0.000000005427 + \text{-}1.000000002712\,\mathrm{i} & 0.000000005427 + 0.999999997288\,\mathrm{i} \\ 0.000000005427 + \text{-}0.999999997288\,\mathrm{i} & 1.008279326594 + 0.000000000000\,\mathrm{i} \\ 0.628186277667 + 0.384444450582\,\mathrm{i} & 0.628186277667 + \text{-}0.384444450582\,\mathrm{i} \end{array} \right\} \quad (27)$$

4.5.5 Computing X_t and Y_t for Candidate Solutions

We compute the remaining pose parameters (X_t, Y_t) for each candidate t, the real eigenvalues, $\{\Leftrightarrow 1.231429745741,\ 1.008279326594\}$. The values of X_t, Y_t, and $Error(X_t, Y_t, t)$ for

Type	t	X_t	Y_t	$Error(X_t, Y_t, t)$
Local Maximum	-1.231429745741	-0.392742825743	-1.099677271638	2537.708141328489
Local Minimum	1.008279326594	-0.392742825743	-1.099677271638	0.047461161151

Table 3: Local extrema found by solving $\nabla \cdot Error(X, Y, t) = 0$

$t = \Leftrightarrow 1.231429745741$ and $t = 1.008279326594$ are given in Table 3. The error function $Error(X_t, Y_t, t)$ at $t = 1.008279326594$ is extremely small, and we observe that it is a local minimum from the signs of the second order partial derivatives of $Error(X_t, Y_t, t)$. The remaining candidates are disregarded because they are complex. Since the global minimum is guaranteed to be one of the local extrema, the pose corresponding to $t = 1.008279326594$ is in fact the global minimum.

The pose is completed by transforming the t value into radians using equation (28). In this case, $\theta(t = 1.008279326594) = 1.5790415$ radians.

$$\theta(t) = 2\tan^{-1}(t) \tag{28}$$

4.6 Effect of Incorrect Correspondence

In this section, we show that the algorithm computes an optimum pose estimate even when the correspondence information is incorrect. The algorithm computes the transformation which maps the point set onto the feature set, regardless of the quality of the fit between the transformed point set and the feature set. To demonstrate its robustness, we applied the localization algorithm on almost the same data set with one exception: the third point now corresponds to the same linear feature as the first two points (refer Table 4). A slice of the error function is depicted in Figure 9, and the poses with extremal squared error are given in Table 5.

Therefore, a high error residual at the local minima would indicate a wrong correspondence.

4.7 Example Which Demonstrates Problems of Local Minima

The main advantage of using an algebraic approach to solve the non-linear least squares problem is that it always returns the global minimum, not a local minimum. Gradient descent techniques can fail by returning a local minimum. Table 6 shows a set of points and linear feature parameters which induce a local minimum of the error function. The error function is depicted in Figure 10, and the local minima are given in Table (7). In the case

Data point $(x_{i,1}, x_{i,2})$	Linear Feature Algebraic Parameters $(a_{i,1}, a_{i,2}, a_{i,3})$
(-7.91, -7.91)	(-0.007534555543, 0.999971614834, -9.004401406730)
(7.91, 7.91)	(-0.007534555543, 0.999971614834, 6.805099825207)
(-7.91, 7.91)	**(-0.007534555543, 0.999971614834, 6.805099825207)**
(7.91, -7.91)	(0.700109199157, 0.714035789899, 10.050656124962)
(-7.91, -7.91)	(-0.710861891474, 0.703331622529, -11.545580060073)
(7.91, 7.91)	(-0.710861891474, 0.703331622529, 10.561258166615)

Table 4: Incorrect data set: an incorrect correspondence is included to demonstrate the localization technique's robustness.

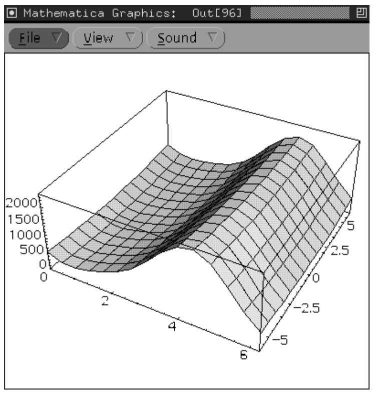


Figure 9: The total error function $Error(Y, \theta)$ given the wrong correspondence information. Notice that the minimum sum squared error is on the order of 82, inferring incorrect correspondence.

of linear features, it is impossible for a single value of t to correspond to two local minima because $\frac{\partial Error(X,Y,t)}{\partial X}|_t = 0$ and $\frac{\partial Error(X,Y,t)}{\partial Y}|_t = 0$ are linear equations in X and Y (allowing only a single generic solution).

In the example given in section 4.5, notice that there are only four non-degenerate values of t (refer Table 7); in the example given in section 4.7, there are also only four non-degenerate

Type	t	X_t	Y_t	$Error(X_t, Y_t, t)$
Local Maximum	-1.318194808282	11.674225332740	4.323041075059	1778.848128685287
Local Minimum	0.391742823602	2.115870897468	1.402893370181	82.262413290594

Table 5: Extremal error poses found for the data set with the incorrect correspondences.

Model point $(x_{i,1}, x_{i,2})$	Algebraic Parameters $(a_{i,1}, a_{i,2}, a_{i,3})$
(1.0, 0.0)	(1.0, 0.0, 1.0)
(0.0, 1.0)	(0.0, 1.0, 1.0)
(-1.0, 0.0)	(1.0, 0.0, -1.0)
(0.0, -1.0)	(0.0, 1.0, -1.0)
(0.6, 0.87)	(3.06, 3.52, 4.09)

Table 6: An example of points and corresponding linear features which induce a local minima in the error function.

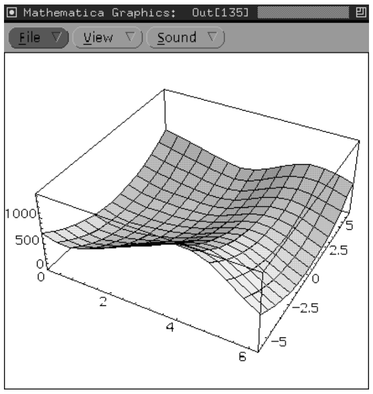


Figure 10: The total error function $Error(Y, \theta)$ for the example with local minima values of t.

4.8 Points and Rectilinear Features

Having gone through the examples for localizing generic points and linear features, we now describe a localization algorithm optimized for points and rectilinear features. This tailored algorithm is much faster than the general case algorithm because the symbolic resultant can be simplified to a fourth order equation when all of the linear features are rectilinear, i.e.

Type	t	X_t	Y_t	$Error(X_t, Y_t, t)$
Local Minimum	0.163456292485	-0.048729446432	-0.056054788052	0.022882658439
Local Minimum	-0.159961402498	-0.094761199366	-0.109006346984	0.055519581104
Local Maxima	-56.167085506603	1.160012442320	1.33439339770	22.817164552496
Local Maxima	-0.061241409474	-0.108071632540	-0.124317694947	0.059481427413

Table 7: Local extrema found by solving $\nabla \cdot Error(X, Y, t) = 0$

either parallel or normal to each other. By redesigning the parts, or concentrating on only the rectilinear edges, this special case can be exploited to achieve real-time localization with cheaper computer hardware.

Although parallel edges can be represented in any manner, the description for which they are all parallel either to the x axis or the y axis is particularly beneficial because some of the symbolic coefficients become redundant, and this redundancy allows us to simplify the symbolic resultant. For example the `ax1t2` coefficient (for the Xt^2 term in $FF(X,Y,t)$) is exactly the sum of the `ax1` coefficient (for the X term in $FF(X,Y,t)$) and the `ax1t4` coefficient (for the Xt^4 term in $FF(X,Y,t)$); although these redundancies are unintuitive, they are observed by expanding the error between points and horizontal lines (refer equation (6)), and recognizing them is beneficial. The redundant formulation of $FF(X,Y,t)$ is given in equation (29).

$$\mathtt{ax1t2} = \mathtt{ax1} + \mathtt{ax1t4}; \quad \mathtt{ay1t2} = \mathtt{ay1} + \mathtt{ay1t4}; \quad \mathtt{ax1y1} = \mathtt{ax1y1t2} = \mathtt{ax1y1t4} = 0$$

$$\begin{aligned} FF(X,Y,t) &= \mathtt{a} + \mathtt{at1}t + \mathtt{at2}t^2 + \mathtt{at3}t^3 + \mathtt{at4}t^4 + \mathtt{ax1}X + \mathtt{ax1t1}Xt \qquad (29) \\ &\quad + \mathtt{ax1}Xt^2 + \mathtt{ax1t4}Xt^2 + \mathtt{ax1t1}Xt^3 + \mathtt{ax1t4}Xt^4 + \mathtt{ax2}X^2 \\ &\quad + 2\mathtt{ax2}X^2t^2 + \mathtt{ax2}X^2t^4 + \mathtt{ay1}Y + \mathtt{ay1t1}Yt + \mathtt{ay1}Yt^2 \\ &\quad + \mathtt{ay1t4}Yt^2 + \mathtt{ay1t1}Yt^3 + \mathtt{ay1t4}Yt^4 + \mathtt{ay2}Y^2 + 2\mathtt{ay2}Y^2t^2 + \mathtt{ay2}Y^2t^4 \end{aligned}$$

In the case of rectilinear features, the resultant polynomial is a twelfth degree polynomial in t, but it can be factored into a fourth order polynomial and eighth order polynomial (with eight extraneous solutions, refer equation (30)). Thereby rectilinear features can be localized faster because the fourth order polynomial in t can be solved exactly in constant time [1].

$$\begin{aligned} \frac{|M(t)|}{(1+t^2)^4} &= a_0 + a_1t + a_2t^2 + a_3t^3 + a_4t^4 \qquad (30) \\ a_0 &= \Leftrightarrow 2 * \mathtt{ax2} * \mathtt{ay1} * \mathtt{ay1t1} \Leftrightarrow 2 * \mathtt{ax1} * \mathtt{ax1t1} * \mathtt{ay2} + 4 * \mathtt{at1} * \mathtt{ax2} * \mathtt{ay2} \\ a_1 &= 4 * \mathtt{ax2} * \mathtt{ay1}^2 \Leftrightarrow 2 * \mathtt{ax2} * \mathtt{ay1t1}^2 \Leftrightarrow 4 * \mathtt{ax2} * \mathtt{ay1} * \mathtt{ay1t4} + 4 * \mathtt{ax1}^2 * \mathtt{ay2} \\ &\quad \Leftrightarrow 2 * \mathtt{ax1t1}^2 * \mathtt{ay2} \Leftrightarrow 4 * \mathtt{ax1} * \mathtt{ax1t4} * \mathtt{ay2} \Leftrightarrow 16 * \mathtt{a} * \mathtt{ax2} * \mathtt{ay2} \\ &\quad + 8 * \mathtt{at2} * \mathtt{ax2} * \mathtt{ay2} \\ a_2 &= 6 * \mathtt{ax2} * \mathtt{ay1} * \mathtt{ay1t1} \Leftrightarrow 6 * \mathtt{ax2} * \mathtt{ay1t1} * \mathtt{ay1t4} + 6 * \mathtt{ax1} * \mathtt{ax1t1} * \mathtt{ay2} \\ &\quad \Leftrightarrow 6 * \mathtt{ax1t1} * \mathtt{ax1t4} * \mathtt{ay2} \Leftrightarrow 12 * \mathtt{at1} * \mathtt{ax2} * \mathtt{ay2} + 12 * \mathtt{at3} * \mathtt{ax2} * \mathtt{ay2} \\ a_3 &= 2 * \mathtt{ax2} * \mathtt{ay1t1}^2 + 4 * \mathtt{ax2} * \mathtt{ay1} * \mathtt{ay1t4} \Leftrightarrow 4 * \mathtt{ax2} * \mathtt{ay1t4}^2 + 2 * \mathtt{ax1t1}^2 * \mathtt{ay2} \end{aligned}$$

$$\begin{aligned}
& +4 * \texttt{ax1} * \texttt{ax1t4} * \texttt{ay2} \Leftrightarrow 4 * \texttt{ax1t4}^2 * \texttt{ay2} \Leftrightarrow 8 * \texttt{at2} * \texttt{ax2} * \texttt{ay2} \\
& +16 * \texttt{at4} * \texttt{ax2} * \texttt{ay2} \\
a_4 \quad = \quad & 2 * \texttt{ax2} * \texttt{ay1t1} * \texttt{ay1t4} + 2 * \texttt{ax1t1} * \texttt{ax1t4} * \texttt{ay2} \Leftrightarrow 4 * \texttt{at3} * \texttt{ax2} * \texttt{ay2} \qquad (31)
\end{aligned}$$

5 Optimal Pose Determination for Models with Circular and Linear Features

In section 4 we described the localization algorithm for polygonal objects and objects with linear features. In this section, we extend the algorithm to the larger class of generalized-polygonal objects: objects with boundaries consisting of linear and circular features. Elimination techniques are again used to solve the system of partial derivative equations. One difference between the two localization algorithms is that resultant methods are used *twice* for the circular features case because the partial derivatives $\frac{\partial Error(X,Y,t)}{\partial X}$ and $\frac{\partial Error(X,Y,t)}{\partial Y}$ are nonlinear in X and Y.

5.1 Algorithm Overview

Resultant methods involve formulating the error function generically in terms of algebraic coefficients, formulating the partial derivatives in terms of the algebraic coefficients, and then constructing the symbolic resultant matrix. Resultant methods are used to eliminate X and Y from the system of partial derivative equations to produce an equation *solely in* θ, which can be solved numerically. Then, resultant methods are used again to eliminate X from a system of 2 partial differential equations to produce an equation solely in Y. Then, the remaining pose parameter X is determined for each orientation θ and y translation component Y.

There are five offline steps (similar to points and linear features):

1. Determine the structure of a generic error function by examining an algebraic expression of the error between an arbitrary transformed point and an arbitrary circular feature.

2. Express the error function $Error(X, Y, \theta)$ as a generic algebraic function in terms of symbolic coefficients.

3. Formulate the partial derivatives of the generic error function $Error(X, Y, \theta)$ with respect to the coordinates: X, Y, and θ. The motivation is that each zero-dimensional pose with local extremum error satisfies $\nabla \cdot Error(X, Y, \theta) = \vec{0}$.

4. Eliminate X and Y from the system of partial derivatives $\nabla \cdot Error(X, Y, \theta) = \vec{0}$ in order to generate a expression *solely in θ*. The result of this step is a symbolic resultant matrix which is used to solve for all of the θ of poses with local extrema error function. Given the orientation, θ, the remaining pose parameters X_θ, Y_θ can be determined by solving a system of linear equations. .

5. Eliminate X from the system of two partial derivative equations $\frac{\partial Error(X,Y,\theta)}{\partial X} = 0$ and $\frac{\partial Error(X,Y,\theta)}{\partial Y} = 0$ to produce an equation *solely in Y_θ*, which can be solved numerically. The remaining pose parameter X is computed numerically using $\frac{\partial Error(X,Y,\theta)}{\partial X} = 0$.

In addition to the five offline steps, there are three online steps:

1. Instantiate the symbolic coefficients of the error function using the data set of points and associated linear features.

2. Compute all of the *interesting* orientations θ by solving the resultant matrix polynomial using eigenvalue techniques (refer section 4.3)

3. $\forall\ \hat{\theta}$, if $\hat{\theta}$ is a candidate orientation ($\text{Im}(\hat{\theta}) \approx 0$):

   (a) Compute all of the *interesting* $\hat{Y_{\hat{\theta}}}$ by solving the resultant matrix polynomial using eigenvalue computations described in section 4.3.

   (b) $\forall\ \hat{Y_{\hat{\theta}}}$, if $\hat{Y_{\hat{\theta}}}$ is a candidate Y translation ($\text{Im}(\hat{Y_{\hat{\theta}}}) \approx 0$):

      i. Compute for $X_{\hat{Y_{\hat{\theta}}},\hat{\theta}}$ using $\frac{\partial Error(X,Y,\theta)}{\partial X}|_{\hat{\theta},\hat{Y_{\hat{\theta}}}} = 0$ which is cubic in $X_{\hat{Y_{\hat{\theta}}},\hat{\theta}}$

      ii. Compute $Error(X_{\hat{Y_{\hat{\theta}}},\hat{\theta}}, \hat{Y_{\hat{\theta}}}, \hat{\theta})$ at each local extremum in order to find the global minimum error pose.

5.2 Approximating the Error Between Points and Circular Features

Unfortunately, resultant techniques cannot effectively minimize the squared error between points and circular features. In this section, we prove that there is no algebraic rational polynomial expression of the point position (x, y) which describes the squared minimum distance between a point and a circle. This is shown by the following example: consider a point $\overrightarrow{(x, 0)}$ on the x-axis and circle C centered at the origin with radius R (see Figure 11(a)). The squared minimum distance between the point $\overrightarrow{(x, 0)}$ and the circle C is shown in Figure 11(b). The slope discontinuity of the squared minimum distance at $x = 0$ proves that no algebraic rational polynomial *in x alone* can describe the square of the minimum distance between a circle and a point.

The minimum distance between a point and a circle can be described by a modified system of algebraic polynomials by introducing an additional variable α denoting a point on the circle C. But introducing a new variable for each circular feature in that manner would prohibit the use of a single generic symbolic resultant matrix. Furthermore, introducing these extra degrees of freedom would increase the complexity (and therefor the running time) of the eigenvalue computations.

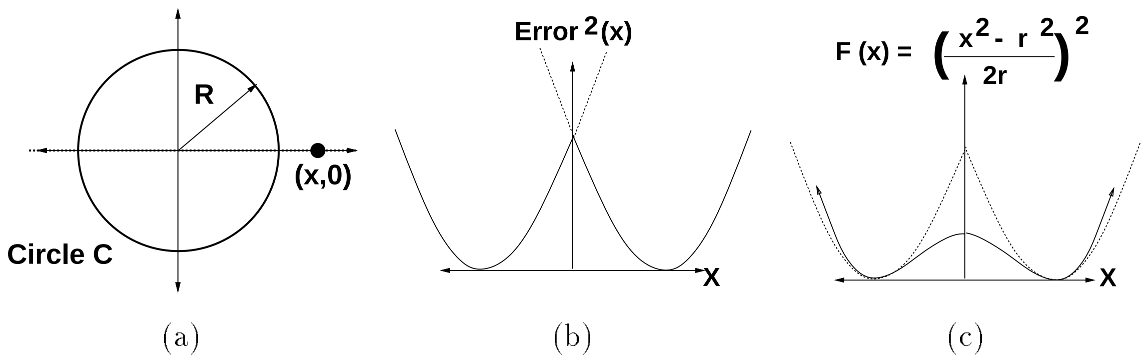


Figure 11: (a): Consider a point $(\vec{x,0})$ on the x-axis and a circle C centered at the origin of radius R. (b): The squared minimum distance between $(\vec{x,0})$ and C as a function of x. Notice the slope discontinuity at $x = 0$ which proves that the squared minimum distance between a point and a circle cannot be written as a algebraic rational polynomial function in the point position (x, y). (c): The function $F(x)$ approximates the squared minimum error for points nearby circular features.

For these reasons, we employ a rational polynomial function $F(x) = (\frac{x^2-r^2}{2r})^2$ to approximate the squared error function between points and circular features (see Figure 11(c)). This approximation should suffice when the data points are nearby the circular features.

5.3 Total Squared Error Function

In this section, we derive the squared error between points and circular features. We show that the squared error between a set of points and a set of linear and circular features can be expressed as a fourth order rational polynomial function in t.

$$Error_{total}(X, Y, \theta) \quad = \quad Error_{linear\ features}(X, Y, \theta) + Error_{circular\ features}(X, Y, \theta) \qquad (32)$$

$$= \underbrace{\sum_i Error_{linear}(X,Y,\theta,\vec{x}_i,\vec{a}_i)}_{linear\ feature\ error} + \underbrace{\sum_i Error_{circular}(X,Y,\theta,\vec{x}_i,\vec{c}_i,r_i)}_{circular\ feature\ error}$$

The error between a transformed point $\mathcal{T}(X,Y,t)\vec{x}$ and a circle C is exactly:

$$\begin{aligned}
Error_{circular}(X,Y,t,\vec{x},\vec{c},r) &= (\|(\mathcal{T}(X,Y,\theta)(\mathtt{X},\mathtt{Y},\mathtt{1})^{\mathsf{T}} \Leftrightarrow (\mathtt{c_x},\mathtt{c_y},\mathtt{1}))\| \Leftrightarrow \mathtt{r})^2 = && (33)\\
(\|\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} \Leftrightarrow \vec{c}\| \Leftrightarrow \mathtt{r})^2 &\approx F(\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} \Leftrightarrow \vec{c}, r) = && (34)\\
\left(\frac{\|\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} \Leftrightarrow \vec{c}\|^2 \Leftrightarrow \mathtt{r}^2}{2r}\right)^2 &= \frac{(\|Mat(X,Y,t)\vec{x}^{\mathsf{T}} \Leftrightarrow (1+t^2)\vec{c}\|^2 \Leftrightarrow (1+t^2)\mathtt{r}^2)^2}{(1+t^2)^2(2r)^2} && (35)
\end{aligned}$$

$$Error_{total}(X,Y,\theta) = \underbrace{\sum_i Error_{linear}(X,Y,\theta,\vec{x}_i,\vec{a}_i)}_{linear\ feature\ error} + \underbrace{\sum_i Error_{circular}(X,Y,\theta,\vec{x}_i,\vec{c}_i,r_i)}_{circular\ feature\ error} \quad (36)$$

$$\approx \sum \|\vec{a}^{\mathsf{T}} \cdot (\mathcal{T}(X,Y,\theta)\vec{x})\|^2 + \sum\left(\frac{\|\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} \Leftrightarrow \vec{c}^{\mathsf{T}}\|^2 \Leftrightarrow \mathtt{r}}{2r}\right)^2 \quad (37)$$

$$= \frac{\sum \|\vec{a}^{\mathsf{T}} \cdot (Mat(X,Y,t)\vec{x})\|^2}{(1+t^2)^2} + \frac{\sum \frac{1}{4r^2}(\|(Mat(X,Y,t)\vec{x} \Leftrightarrow (1+t^2)\vec{c})\|^2 \Leftrightarrow \mathtt{r}(1+t^2)^2)^2}{(1+t^2)^4} \quad (38)$$

It turns out that $\|(Mat(X,Y,t)\vec{x} \Leftrightarrow (1+t^2)\vec{c})\|^2 \Leftrightarrow \mathtt{r}(1+t^2)^2)^2$ is divisible by $(1+t^2)^2$ due to the fact that $\cos^2\theta + \sin^2\theta = 1$. By factoring $(1+t^2)^2$ out of the numerator and denominator of $Error_{circular}(X,Y,t)$, we arrive at a quartic polynomial expression. The linear feature error function and the factored circular feature error function have equal denominators ($(1+t^2)^2$) which enables the error functions to be easily added by adding the numerators of the error functions. The squared error between points and linear features was derived in section 4. We end up with an error function $Error_{total}(X,Y,t)$ which is a quartic function in t divided by $(1+t^2)^2$.

$$\begin{aligned}
Error_{total}(X,Y,t) &= \frac{\sum \|\vec{a} \cdot Mat(X,Y,t)\vec{x}\|^2}{(1+t^2)^2} + \frac{\sum \frac{1}{4r^2}\left(\frac{\|Mat(X,Y,t)\vec{x} - (1+t^2)\vec{c}\|^2}{(1+t^2)^2} \Leftrightarrow (1+t^2)r\right)^2}{(1+t^2)^2} && (39)\\
FF_{total}(X,Y,t) &= Error_{total}(X,Y,t)(1+t^2)^2 && (40)
\end{aligned}$$

Once again, we use an algebraic function $FF_{total}(X,Y,t)$ to describe the sum squared error between points and linear and circular features, where $FF_{total}(X,Y,t) = Error_{total}(X,Y,t)$ (equation (41)). The partials $\frac{\partial Error_{total}(X,Y,t)}{\partial X}$, $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}$, $\frac{\partial Error_{total}(X,Y,t)}{\partial t}$, specify a system of equations defining the local extrema of the error function (including the global minimum).

$$FF_{total}(X,Y,t) = \mathtt{a} + X\mathtt{ax1} + XY\mathtt{ax1y1} + XY^2\mathtt{ax1y2} + X^2\mathtt{ax2} + X^2Y\mathtt{ax2y1} + X^3\mathtt{ax3} \quad (41)$$

$$+Y\texttt{ay1}+Y^2\texttt{ay2}+Y^3\texttt{ay3}+t\texttt{at1}+Xt\texttt{ax1t1}+XYt\texttt{ax1y1t1}+XY^2t\texttt{ax1y2t1}+X^2t\texttt{ax2t1}$$
$$+X^2Yt\texttt{ax2y1t1}+X^3t\texttt{ax3t1}+Yt\texttt{ay1t1}+Y^2t\texttt{ay2t1}+Y^3t\texttt{ay3t1}+t^2\texttt{at2}$$
$$+Xt^2\texttt{ax1t2}+XYt^2\texttt{ax1y1t2}+XY^2t^2\texttt{ax1y2t2}+X^2t^2\texttt{ax2t2}+X^2Yt^2\texttt{ax2y1t2}$$
$$+X^3t^2\texttt{ax3t2}+Yt^2\texttt{ay1t2}+Y^2t^2\texttt{ay2t2}+Y^3t^2\texttt{ay3t2}+t^3\texttt{at3}+Xt^3\texttt{ax1t3}$$
$$+XYt^3\texttt{ax1y1t3}+XY^2t^3\texttt{ax1y2t3}+X^2t^3\texttt{ax2t3}+X^2Yt^3\texttt{ax2y1t3}+X^3t^3\texttt{ax3t3}$$
$$+Yt^3\texttt{ay1t3}+Y^2t^3\texttt{ay2t3}+Y^3t^3\texttt{ay3t3}+t^4\texttt{at4}+Xt^4\texttt{ax1t4}+XYt^4\texttt{ax1y1t4}$$
$$+XY^2t^4\texttt{ax1y2t4}+X^2t^4\texttt{ax2t4}+X^2Yt^4\texttt{ax2y1t4}+X^3t^4\texttt{ax3t4}$$
$$+Yt^4\texttt{ay1t4}+Y^2t^4\texttt{ay2t4}+Y^3t^4\texttt{ay3t4}+(1+t^2)^2(X^2+Y^2)^2\texttt{ax4y4}$$

In the case of generalized-polygonal objects, some of the algebraic coefficients are redundant. We note these redundancies because they led to symbolic simplification.

ax3t2 = 2 ax3 - ay3t1	ax3t3 = ax3t1	ax3t4 = ax3 - ay3t1	ax1y2t1 = ax3t1
ay3t2 = 2 ay3 + ax3t1	ax1y2t3 = ax3t3	ax1y2t4 = ax3t4	ax1y2t2 = ax3t2
ay3t4 = ay3 + ax3t1	ay3t3 = ay3t1	ax2y1t1 = ay3t1	ax2y1t2 = ay3t2
ax2y1t3 = ay3t3	ax2y1t4 = ay3t4		

Again, the global minimum is found by finding all of the local extrema and finding the local extrema with minimum error. All of the local extrema are found by solving for simultaneous solutions to the zeros of the partial derivatives of the error function (refer equation 42).

$$\nabla \cdot Error_{total}(X,Y,\theta) = (0,0,0) \Rightarrow \nabla \cdot Error_{total}(X,Y,t) = (0,0,0) \qquad (42)$$

Again, since we are only finding the common roots of equation (42), we are at liberty to use substitute any function $H_X(X,Y,\theta)$ for $\frac{\partial Error_{total}(X,Y,t)}{\partial X}$ with the proviso that $\frac{\partial Error_{total}(X,Y,t)}{\partial X} = 0 \rightarrow H_X(X,Y,\theta) = 0$. Such a substitution is necessary because we will be using resultant techniques which require algebraic (not rational) functions.

For $\frac{\partial Error_{total}(X,Y,t)}{\partial X}$ and $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}$, $H_X(X,Y,t)$ and $H_Y(X,Y,t)$ are equal to the partial derivatives $\frac{\partial FF_{total}(X,Y,t)}{\partial X}$, $\frac{\partial FF_{total}(X,Y,t)}{\partial Y}$. $H_t(X,Y,t)$ is slightly more complicated because the denominator of $\frac{\partial Error_{total}(X,Y,t)}{\partial t}$ is a function of t, and this must be taken into account.

$$\begin{aligned}
\frac{\partial Error_{total}(X,Y,t)}{\partial X} &\propto \frac{\partial FF_{total}(X,Y,t)}{\partial X} \\
H_X(X,Y,t) &= \frac{\partial FF_{total}(X,Y,t)}{\partial X} \\
\frac{\partial Error_{total}(X,Y,t)}{\partial Y} &\propto \frac{\partial FF_{total}(X,Y,t)}{\partial Y}
\end{aligned}$$

$$\begin{aligned}
H_Y(X,Y,t) &= \frac{\partial FF_{total}(X,Y,t)}{\partial Y} \\
\frac{\partial Error_{total}(X,Y,t)}{\partial t} &\propto \frac{\partial \frac{FF_{total}(X,Y,t)}{(1+t^2)^2}}{\partial t} = \frac{((1+t^2)\frac{\partial FF_{total}(X,Y,t)}{\partial t} \Leftrightarrow 4tFF_{total}(X,Y,t))}{(1+t^2)^3} \qquad (43) \\
H_t(X,Y,t) &= ((1+t^2)\frac{\partial FF_{total}(X,Y,t)}{\partial t} \Leftrightarrow 4tFF_{total}(X,Y,t)) \qquad (44)
\end{aligned}$$

The resultant of this system of equations is obtained by eliminating X and Y using the Macaulay construction [18].

$$\begin{aligned}
M(t) &= Resultant(\{\frac{\partial Error_{total}(X,Y,t)}{\partial X}, \frac{\partial Error_{total}(X,Y,t)}{\partial Y}, \frac{\partial Error_{total}(X,Y,t)}{\partial t}\}, \{X,Y\}) \\
&= Resultant(\{H_X(X,Y,t), H_Y(X,Y,t), H_t(X,Y,t)\}, \{X,Y\}) \qquad (45)
\end{aligned}$$

5.4 Solving for Orientation, t, of Local Extrema Poses

The resultant of the system of equations: $\nabla \cdot Error_{total}(X,Y,t) = 0$ is a 36×36 matrix with rank 34. Furthermore, we can eliminate two rank-deficient rows and many redundant singleton rows (rows containing only one non-zero element). Symbolic manipulation produces a 26×26 matrix with three singleton rows. The roots of the 26×26 matrix polynomial are computed using the eigendecomposition algorithms described in section 4.3.

The numerical precision of the resultant approach depends on the resultant matrix having full rank. Rank deficiency can result from *redundant* coefficients. The 26×26 construction we describe is designed for combinations of linear and circular features. If there are only circular features, we need to reexamine the symbolic resultant to eliminate any remaining rank deficiencies. For eigenvalue computations, rank deficiency results in numerical imprecision. For robustness, we generate both resultant formulations, one assuming linear features are included, and one assuming linear features are not included, and use the appropriate matrix to compute the orientation t.

5.5 Solving for Y Position, Y_t, of Local Extrema Poses

To compute Y_t after solving for t, we again use resultant methods since the remaining equations are nonlinear in X and Y. The resultant is constructed using the equations: $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ and $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t = 0$. At a particular orientation t, $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ can be written as a function $U(X_t,Y_t) = 0$, and $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t = 0$ can be written as a function $V(X_t,Y_t) = 0$. $U(X_t,Y_t) = 0$ is cubic in X_t and $V(X_t,Y_t) = 0$ is quadratic in X_t.

$$U(X_t,Y_t) = 0 \Leftrightarrow u_3(Y_t)X_t^3 + u_2(Y_t)X_t^2 + u_1(Y_t)X_t + u_0(Y_t) = 0 \qquad (46)$$

$$V(X_t,Y_t) = 0 \Leftrightarrow v_2(Y_t)X_t^2 + v_1(Y_t)X_t + v_0(Y_t) = 0 \qquad (47)$$

We use the Macaulay resultant to eliminate X_t and produce a function in Y_t (refer equation (48)) [18]. This determinant of the resultant matrix is a third-order matrix polynomial in Y_t. This construction is more complex than the construction for the case of linear equations (refer section 4.2.1). We compute the roots of Y_t at simultaneous solutions using the eigenvalue decomposition method we previously outlined (see section 4.3).

$$Resultant(\{U(X_t,Y_t),V(X_t,Y_t)\},X_t) = \begin{bmatrix} u_0(Y_t) & 0 & v_0(Y_t) & 0 & 0 \\ u_1(Y_t) & u_0(Y_t) & v_1(Y_t) & v_0(Y_t) & 0 \\ u_2(Y_t) & u_1(Y_t) & v_2(Y_t) & v_1(Y_t) & v_0(Y_t) \\ u_3(Y_t) & u_2(Y_t) & 0 & v_2(Y_t) & v_1(Y_t) \\ 0 & u_3(Y_t) & 0 & 0 & v_2(Y_t) \end{bmatrix} \tag{48}$$

5.6 Solving for X Position, $X_{Y_t,t}$,of Local Extrema Poses

Given Y_t and t, $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ is a cubic equation in $X_{Y_t,t}$ which can be solved numerically. $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ is used, rather than $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t = 0$ (which is quadratic in $X_{Y_t,t}$), in case $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t$ is independent of $X_{Y_t,t}$.

5.7 Verifying (X_{Y_t}, Y_t, t) is a Local Extremum

Resultant methods can produce extraneous solutions because they are necessary but not sufficient condition. This particular method can generate $26 \times 4 \times 5 \times 3$ configurations. Once again, we need to verify that each (X_{Y_t}, Y_t, t) configuration is a local extremum by computing the absolute values of the partial derivatives and comparing them to some ϵ threshold which accommodates sensor noise and numerical error.

5.8 Example

In this example, consider the case of the four points corresponding to two circles as shown in Figure 12. One pose which exactly maps the four points onto their corresponding features is given in Figure 13. The four points are: $(\Leftrightarrow 2, 3), (\Leftrightarrow 1, 2), (\Leftrightarrow 2, 1), (\Leftrightarrow 2, \Leftrightarrow 3)$, and the two associated circles are $(x+2)^2 + y^2 = 1, (x \Leftrightarrow 2)^2 + y^2 = 1$; the first four points all correspond to the first circle.

The total error function between the transformed points and the associated circular features is given in Figure 14 as a function of y and t. The global minimum of the error function is zero at $\theta = \frac{\pi}{2}$, $Y = 2$, which corresponds to mapping the vertices directly onto the corresponding circular features. The algorithm found two distinct local minima, the global minima and also $X = \Leftrightarrow 2.1629, Y = 0.0915, \theta = \pi$).

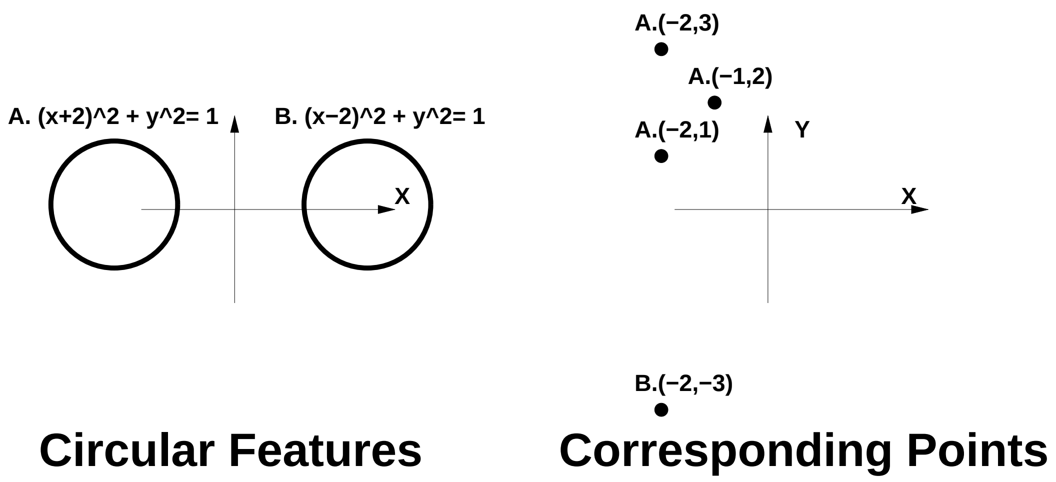


Figure 12: A simple example used to illustrate the algorithm for points and circular features

6 Implementation and Performance

In this section, we describe the performance of the localization algorithm on many cases. We have applied the algorithm on randomly generated data for large data sets as well as investigated the relationship between camera resolution and accuracy using simulated data. We used the localization algorithm to compare the localization performance of using all available boundary information as compared to just utilizing isolated boundary points, and we experimentally tested the technique for small data sets using real data.

In the first case, we generated random sensed data, consisting of random features and then random feature points. To simulate the random positions, all of the feature points were transformed by a random rigid two-dimensional transformation. Finally, we ran the localization on the random features and transformed feature points and compared the estimated transformation with the actual transformation.

For the second suite of experiments, we investigated the relationship between pixel resolution and localization accuracy by localizing objects in known poses at different resolutions. We began with simple polygonal models representing the objects' boundaries, and moved the models into random positions: x, y, θ. For each pose and resolution, we enumerated all of the pixel squares covered by the boundary features and listed these corresponding feature(s) with each pixel. Finally, we compared the estimated transformations with the actual transformations for different pixel resolutions to understand the implications of camera resolution.

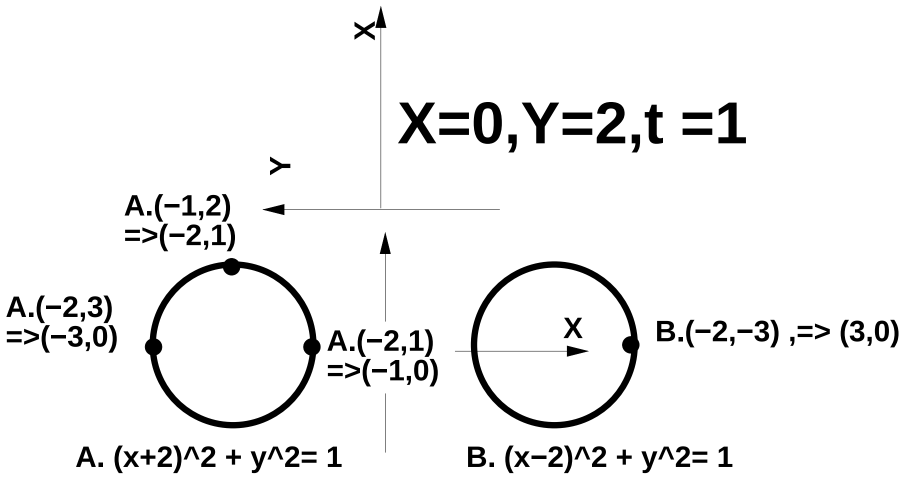


Figure 13: One solution transformation which maps the four points onto the corresponding circular features.

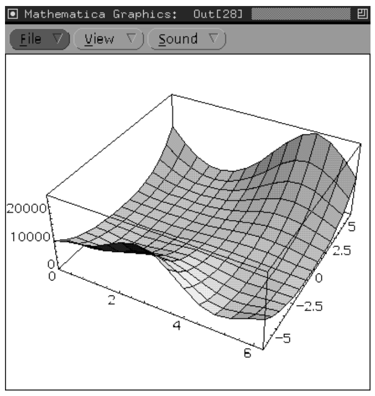


Figure 14: The total error function $Error(Y, \theta)$ for the four points and associated circular features.

6.1 Verifying the Technique With Random Features

6.1.1 Generating Random Point and Linear Feature Data

Point and corresponding linear feature data (edges) were synthesized as shown in Figure 15:

1. Random Linear Feature (Edge) Generation:

   (a) Randomly choose the edge's direction from a uniform distribution $[0 \ldots 2\pi]$.

   (b) Randomly choose the edge's distance from the origin from a uniform distribution $[min_{distance} \ldots max_{distance}]$.

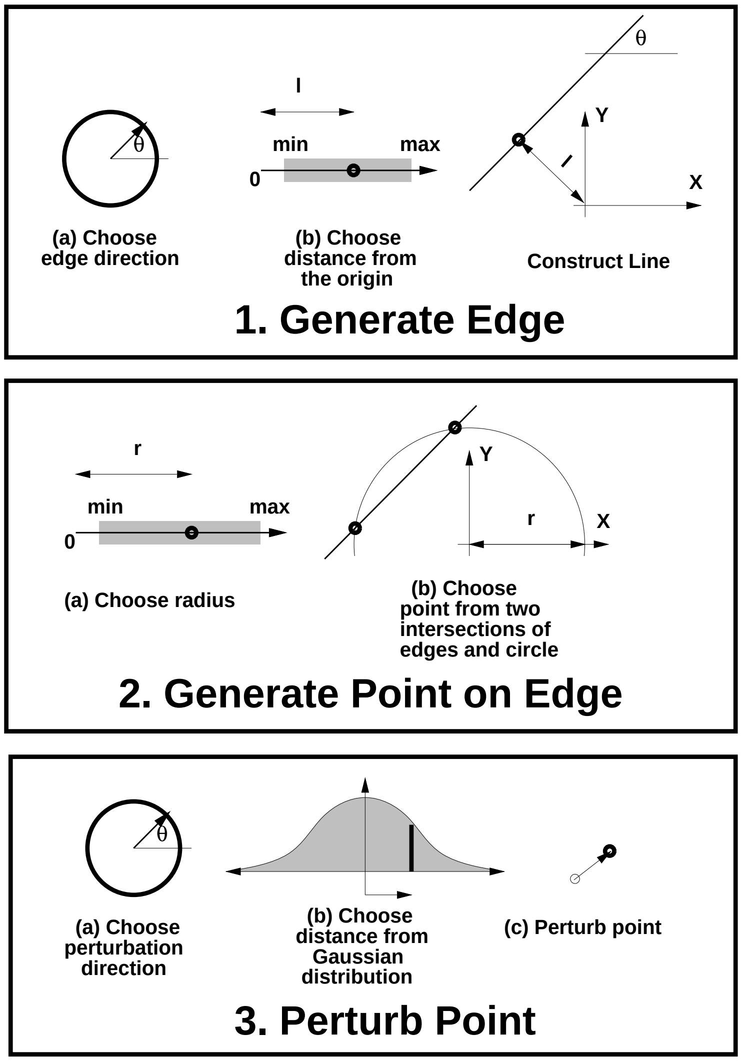

θ
l
min
max
0
Y
X
(a) Choose edge direction
(b) Choose distance from the origin
Construct Line
1. Generate Edge
r
min
max
0
Y
r
X
(a) Choose radius
(b) Choose point from two intersections of edges and circle
2. Generate Point on Edge
θ
(a) Choose perturbation direction
(b) Choose distance from Gaussian distribution
(c) Perturb point
3. Perturb Point

Figure 15: Method for constructing random point and linear feature data

2. Random Point Sampling from Linear Feature (Edge):

   (a) Construct a circle centered at the origin of random radius by randomly choosing a radius r from a uniform distribution $[min_{radius} \ldots max_{radius}]$.

   (b) Find the point on the edge by intersecting the edge with the circle formed in step (a).

3. Random Gaussian Perturbation of Data Point:

   (a) Randomly choose a perturbation to the point, by choosing the orientation and length of the perturbation: orientation from a uniform distribution $[0 \ldots 2\pi]$, length from normal distribution with mean 0 and standard deviation σ.

min_{length} was 0.0, max_{length} was 10.0, min_{radius} was 10.0, max_{radius} was 15.0, and σ was 0.1.

6.1.2 Generating Random Point, Circular Feature Data

Point and corresponding linear feature data and corresponding circular feature data were synthesized by generating point and linear feature data as described in section 6.1.1, and generating point and circular feature data as shown in Figure 16:

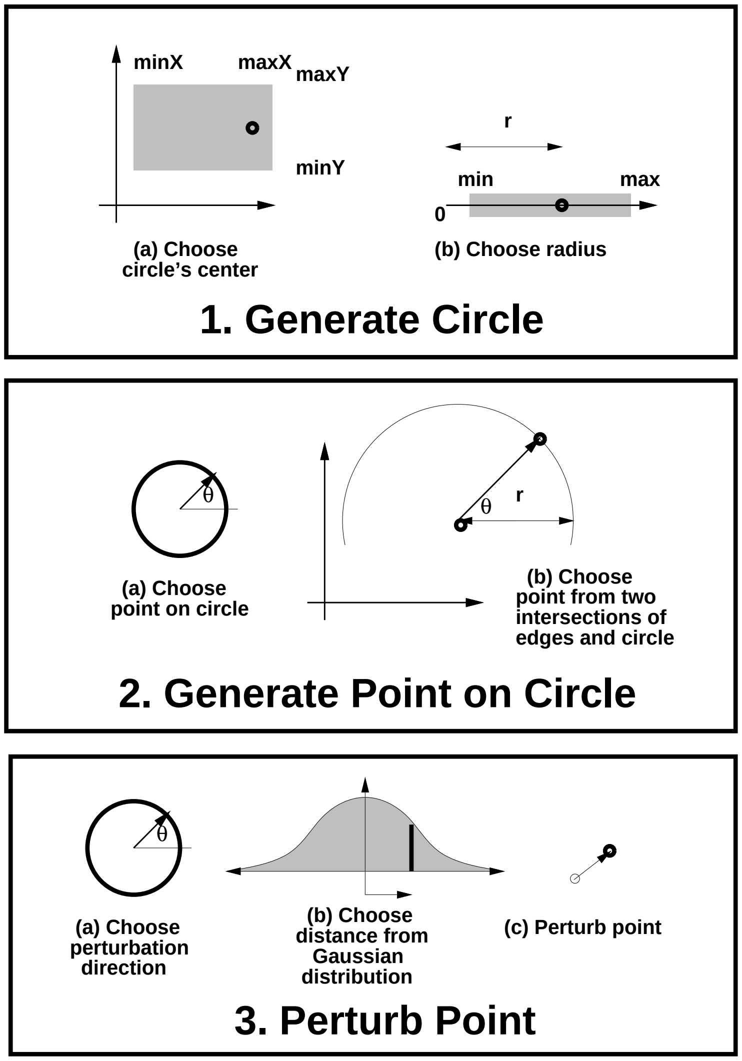

minX
maxX
maxY
minY
(a) Choose circle's center
r
min
max
0
(b) Choose radius
1. Generate Circle
θ
(a) Choose point on circle
θ
r
(b) Choose point from two intersections of edges and circle
2. Generate Point on Circle
θ
(a) Choose perturbation direction
(b) Choose distance from Gaussian distribution
(c) Perturb point
3. Perturb Point

Figure 16: Method for constructing random point and circular feature data

1. Random Circular Feature Generation:

   (a) Randomly choose the center of the circle (from a square) by choosing random x and y positions from uniform distributions: $[min_{coordinate} \ldots max_{coordinate}]$.

   (b) Randomly choose the circle's radius r from a uniform distribution: $[min_{radius} \ldots max_{radius}]$.

2. Random Point Sampling from Circular Feature:

   (a) Randomly choose the point on the circle perimeter from a uniform distribution $[0 \ldots 2\pi]$.

3. Random Gaussian Perturbation of Data Point:

   (a) Randomly choose a perturbation to the point by choosing the orientation and length of the perturbation: orientation from a uniform distribution $[0 \ldots 2\pi]$, length from normal distribution with mean 0 and standard deviation σ.

$min_{coordinate}$ was -10.0, $max_{coordinatetest}$ was 10.0, min_{radius} was 3.0, max_{radius} was 15.0, and σ was 0.1.

6.1.3 Results

Table 8 compares the estimated poses with the actual poses for the randomly generated data sets of points and linear features and circular with perfect sensing ($\sigma = 0.0$). Table 9 compares the estimated poses with the actual poses for the randomly generated data sets of points and linear features with $\sigma = 0.1$. Table 10 compares the estimated poses with the actual poses for the randomly generated data sets of points and linear and circular features with $\sigma = 0.1$.

6.2 Relationship Between Pixel Resolution and Localization Accuracy

Most of the work in machine vision is camera-based and involves high precision, high data bandwidth sensors. This localization algorithm is useful for edge-detection based approaches because it enables the user to utilize all the edge data. In regards to edge-detection based approaches, we wanted to investigate the relationship between camera resolution and localization accuracy.

There are many factors which affect the quality of localization estimates, such as lens distortion, lighting, surface reflectance, etc. In these experiments, we concentrated only on the relationship between pixel resolution and localization accuracy in order to gain a clear

Random Point and Linear and Circular Feature Data			
# linear features	# circular features	Actual pose	Estimated pose
8	0	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.0, Y = 2.0, \theta = 0.7)$
8	0	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.0, Y = 2.0, \theta = 0.8)$
8	0	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.0, Y = 2.0, \theta = 0.9)$
8	0	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.0, Y = 6.0, \theta = 1.0)$
4	4	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.0, Y = 2.0, \theta = 0.7)$
4	4	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.0, Y = 2.0, \theta = 0.8)$
4	4	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.0, Y = 2.0, \theta = 0.9)$
4	4	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.0, Y = 6.0, \theta = 1.0)$
0	8	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.0, Y = 2.0, \theta = 0.7)$
0	8	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.0, Y = 2.0, \theta = 0.8)$
0	8	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.0, Y = 2.0, \theta = 0.9)$
0	8	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.0, Y = 6.0, \theta = 1.0)$

Table 8: Performance of (point,linear and circular feature) localization technique for randomly generated data with $\sigma = 0$.

Random Point and Only Linear Feature Data		
# of random data points	Actual pose	Estimated pose
100000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 1.998597, Y = 2.003240, \theta = 0.700061)$
100000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.001527, Y = 2.002663, \theta = 0.800075)$
100000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.001533, Y = 2.002073, \theta = 0.900087)$
100000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 3.998578, Y = 6.001493, \theta = 1.000099)$
1000000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 1.999183, Y = 2.000571, \theta = 0.699966)$
1000000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.000826, Y = 2.000516, \theta = 0.799959)$
1000000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.000823, Y = 2.000461, \theta = 0.899951)$
1000000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 3.999190, Y = 6.000408, \theta = 0.999945)$

Table 9: Performance of (point,linear feature) localization technique for randomly generated data for $\sigma = 0.1$.

Random Point and Linear and Circular Feature Data			
# linear features	# circular features	Actual pose	Estimated pose
100000	100000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.000344, Y = 2.001496, \theta = 0.699958)$
100000	100000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 2.999460, Y = 2.001097, \theta = 0.799954)$
100000	100000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 0.999193, Y = 2.000738, \theta = 0.899951)$
100000	100000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.001135, Y = 6.00432, \theta = 0.999948)$
1000000	1000000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.000105, Y = 1.999681, \theta = 0.699999)$
1000000	1000000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 2.999905, Y = 1.999641, \theta = 0.799999)$
1000000	1000000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 0.999903, Y = 1.999603, \theta = 0.900003)$
1000000	1000000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.000110, Y = 5.999568, \theta = 1.000005)$

Table 10: Performance of (point,linear and circular features) localization technique for randomly generated data $\sigma = 0.1$.

understanding. The term pixel refers to an individual sensor in the sensor matrix. We assume that the resolution of the data produced by the camera is limited by the pixel size. The term pixel resolution refers to the area in the scene (in the neighborhood of a particular object) characterized by a single pixel.

We believe that many systems rely on comparing isolated points on boundary curves, such as vertices and inflection points, because there are easily reproducible published algorithms for efficiently solving this problem. The purpose of these experiments is to determine the how significantly the localization precision improves by utilizing all of the available data, rather than only isolated boundary points. We wanted to determine whether utilizing all of the edge data was significantly better compared to using only isolated points on boundary curves; it was intuitive to us that using more of the available data always improves performance. We found a distinct advantage to using all of the available information in that our localization technique provided not insignificantly better estimates.

The experiments consisted of localizing a known object from a library of four modeled objects in a known pose, and recording the variation between the actual and estimated pose parameters. These experiments involved generating simulated data for polygonal models in multiple poses for multiple camera resolutions. We used simulated data in order to focus on the relationship between pixel resolution and localization accuracy. We assumed a model of square pixels. The data points were of the form: *¡pixel center, associated linear feature¿*, and the data was synthesized by enumerated the centers of all pixel squares which overlapped any of the modeled boundary features (refer Figure 17). A picture of the pixelization of the hex-model object is shown in Figure 19.

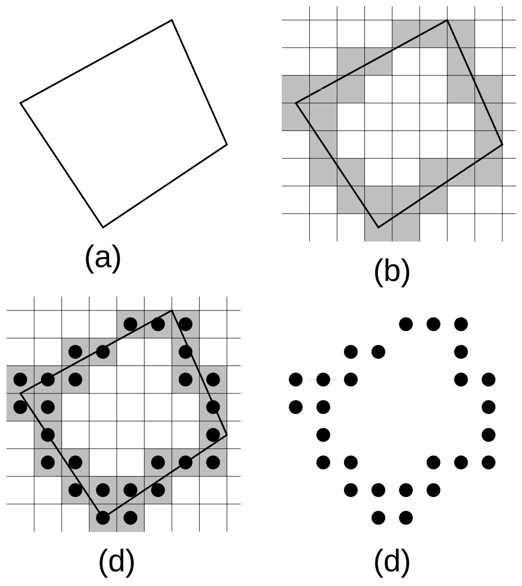


Figure 17: Simulated data was synthesized by enumerating the centers of all of the square pixels crossed by the boundary features

Four different polygonal models were used, and each polygonal model was localized in ten different poses at five different pixel resolutions. In order to characterize the results, we measured the errors in x, y, θ in a manner similar to standard deviations. In other words, we summed up the squares of the discrepancies between the actual and estimated pose parameters for x, y, θ, and computed the square root of the average. To characterize the average error, we simply averaged the errors returned by the optimal fit returned by the localization technique. These error measurements are reported in Table 11.

Table 11

The results from using only data points corresponding to the vertices are shown in Table 12. The most important variable (for polygons) is θ because the x and y pose parameters are linear functions and can be easily solved given θ. Notice that the pose parameter estimates appear to shrink by a factor between three and four each time the resolution is doubled. The x, y, θ values do stop converging at very high resolutions, *i.e.*, for most of the objects, the pose parameters x, y acquiesce to within 0.01 units of translation (0.02% of the object

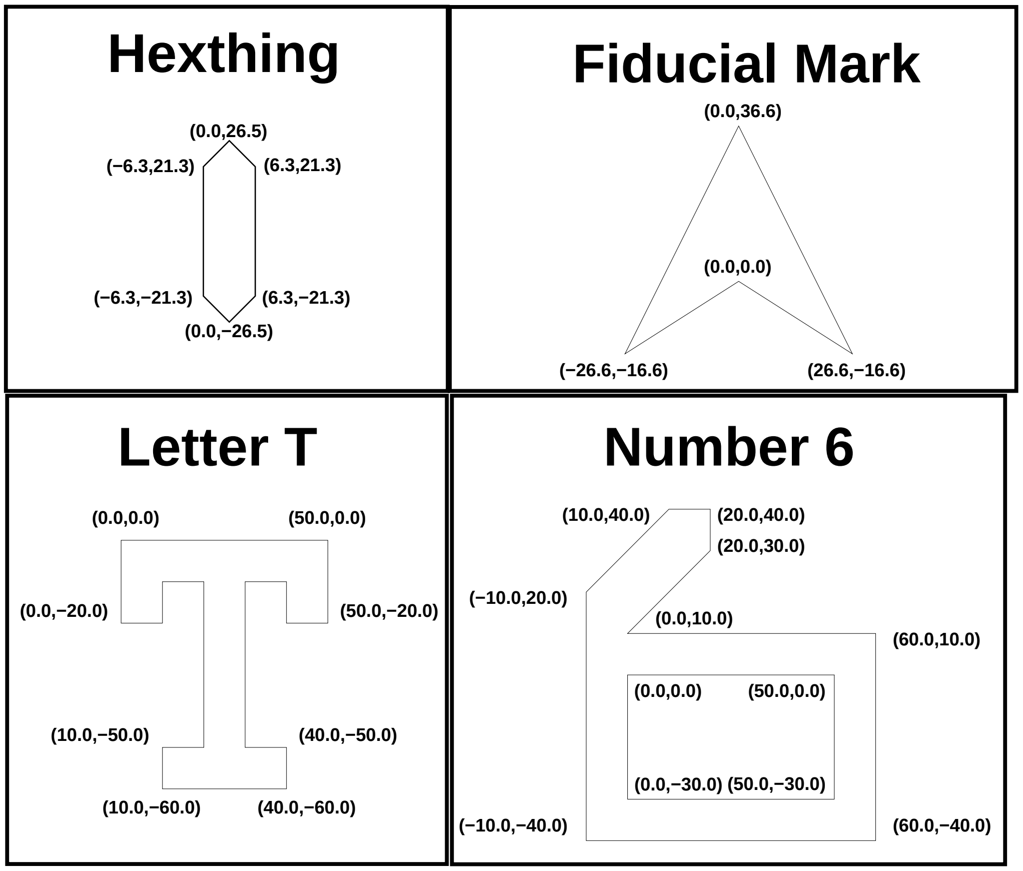


Figure 18: Four objects: Hex-model, Fiducial, Letter T, Number 6. Almost all of the vertex positions are given, but including the vertex positions for the letter T would clutter the figure without adding any additional information

length), and the orientational parameter θ appears to acquiesce to a precision of 0.0001 radians. Also notice that the sum squared error shrinks by a factor of two each time the resolution is doubled, implying that the squared error per datapoint shrinks by a factor of four each time the resolution is doubled.

Next, we investigated the performance degradation related to only utilizing isolated boundary points, rather than all of the available boundary information. To accomplish this, we generated data points We did not utilize the published algorithms for estimating pose from point sets, but instead used the localization technique with only the data points from vertices. All such data points are easily found by modifying the object model, and then synthesizing the data in exactly the same manner. The model was modified by replacing each edge by extremely small edges at the endpoints. In this way, the data synthesizer only generates

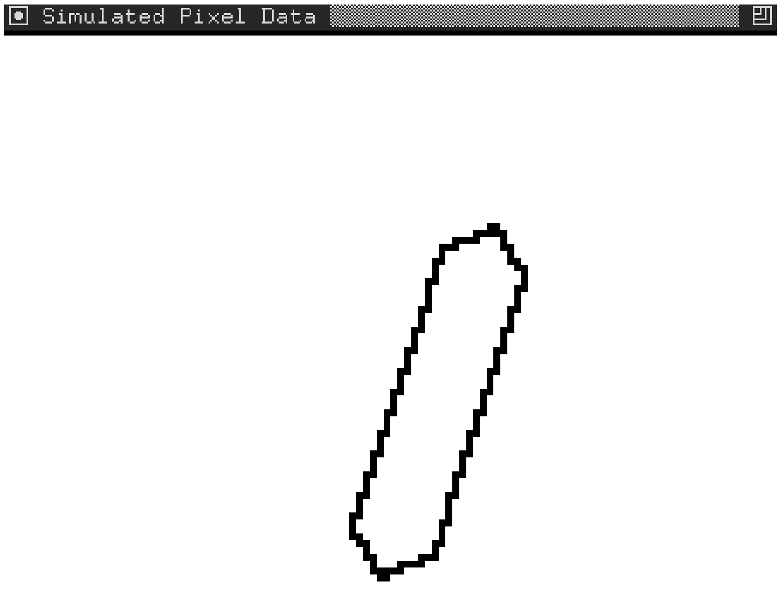


Figure 19: The simulated pixel data for the hex-model object at position (7, ⇔3) at orientation -0.32 radians at resolution 1 pixel unit

data points for the vertices. This approach does not give the exact same answer as using the commonly used point set algorithm because the error function of the localization technique is not the squared distance between the transformed model vertex and the actual vertex, but the sum of the two squared errors between the transformed actual vertices and the two linear model features. The statistics for using only the vertices for the hex-model object are given in Table 12.

We interpret these results from Table 12 in comparison to localization experiments using only the corner data. The most important variable (for polygons) is θ because the x and y pose parameters are linear functions and can be easily solved given θ. At the sharpest resolution, the θ estimates for the localization technique are 20 times more accurate than the vertex-based localization technique. Furthermore, errors in x, y are larger for the vertex case at every resolution save one.

The only error statistic which does not differ significantly between the two approaches is the squared error. On a per-data point basis, the squared error of the localization technique at the sharpest resolution $\frac{5.42}{614.6}$ (approximately 0.01) is very similar to the squared error per data point of the vertex based method, $\frac{0.09}{17}$. We found that the pose estimates were more accurate using more data points, which is an intuitive result. It is confusing to note that the sum squared errors are relatively equivalent; one explanation is that since there are fewer data points, a vertex-based approach more easily fits the data, even though it does not produce a more precise result.

7 Conclusion

7.1 Conclusion

In this paper, we described an object localization technique for sensor and model features of different types, such as polygonal models, and probe points. The approach involved reducing the pose estimation problem to a least squares error problem. We utilized algebraic elimination techniques to exactly solve the least squares error problems. The main advantages of this localization algorithm are its applicability both to sparse and dense sensing strategies, its immunity to local minima problems, its exact computation of the global minimum, and its high speed. We used this technique to compare the relative performance of using only a subset of the data (interesting features such as vertices), with using the entire data set, and we found the intuitive result that utilizing more data provides a more precise position estimate. This algorithm has been successfully applied it for recognition and localization applications using very precise optical sensors.

7.2 Future Work

This approach is extendible to three-dimensional localization from three-dimensional range data. This would involve computing generic error functions between points and features of co-dimension one (planes, surfaces) as a function of the six degrees of freedom. After the generic error functions have been determined, all we need to do is construct larger resultant matrices to solve for the minima. The resultant solving may take a slightly longer time owing to the greater number of degrees of freedom and algebraic complexity. This approach may also be extendible to three-dimensional localization from two-dimensional perspective image data. In this case, we can consider the image points to be rays going through the eye, and compute the disparity between these rays and model edges in three dimensions. The crucial step involves succinctly formulating the generic error between pairs algebraically. Again, resultant solving may take a longer time, owing to the greater number of degrees of freedom and algebraic complexity.

Acknowledgements

The authors wish to acknowledge John Canny for helpful discussion, and Ruth Rosenholtz, Dieter Koller, Jeff Wendlendt, and Clark Olson for their comments.

References

[1] *Standard Mathematical Tables.* Chemical Rubber Company, 1973.

[2] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., Sorensen, and D. *LAPACK User's Guide, Release 1.0.* SIAM, Philadelphia, 1992.

[3] N. Ayache and O. D. Faugeras. Hyper: A new approach for the recognition and positioning of two-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):44–54, 1986.

[4] R. Deriche and O. Faugeras. Tracking line segments. *Image and Vision Computing*, 8:261–270, 1990.

[5] O. D. Faugeras and M. Hebert. The representation, recognition, and locating of 3-d objects. *International Journal of Robotics Research*, 5(3):27–52, 1986.

[6] O.D. Faugeras and S. Maybank. Motion from point matches: Multiplicity of solutions. *International Journal of Computer Vision*, 4:225–246, 1990.

[7] D. Forsyth, L. Mundy, A. Zisserman, A. Heller, and C. Rothwell. Invariant descriptors fo 3-d object recognition and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:971–991, Oct 1991.

[8] B.S. Garbow, J.M. Boyle, J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1977.

[9] G.H. Golub and C.F. Van Loan. *Matrix Computations.* John Hopkins Press, Baltimore, 1989.

[10] W. Eric L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints.* MIT Press, Cambridge, MA, 1990.

[11] W. Eric L. Grimson and Tomas Lozano-Perez. Model-based recognition and localization from sparse range or tactile data. *International Journal of Robotics Research*, 3(3):3–35, 1984.

[12] B. K. P. Horn. *Robot Vision.* McGraw-Hill, seventh edition, 1989.

[13] B. K. P. Horn. Relative orientation revisited. *Journal of Optical Society of America*, 8(10):1630–1638, 1991.

[14] D. P. Huttenlocher and S. Ullman. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2):195–212, 1990.

[15] C. Jerian and R. Jain. Polynomial methods for structure from motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12):1150–1166, 1990.

[16] A. Kalvin, E. Schonberg, J. T. Schwartz, and M. Sharir. Two-dimensional model-based boundary matching using footprints. *International Journal of Robotics Research*, 5(4):38–55, 1986.

[17] D. Koller, K. Daniilidis, and H.-H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10(3):257–281, 1993.

[18] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.

[19] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics.* PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1992.

[20] E. Paulos and J. Canny. Informed peg-in-hole insertion using optical sensors. In *SPIE Conference on Sensor Fusion VI*, 1993. Boston Massachusetts.

[21] J. Ponce, A. Hoogs, and D. J. Kriegman. On using cad models to compute the pose of curved 3d objects. *CVGIP: Image Understanding*, 55(2):184–197, 1992.

[22] J. Ponce and D. J. Kriegman. Elimination theory and computer vision: Recognition and positioning of curved 3d objects from range, intensity, or contours. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 123–146, 1992.

[23] D. Rosen. Errors in digital area measurement of regular 2d figures. In *Vision Geometry II*, pages 26–32, September 1993.

[24] G. Salmon. *Lessons Introductory to the Modern Higher Algebra*. G.E. Stechert & Co., New York, 1885.

[25] H. Schenck. *Theories of Engineering Experimentation*. McGraw-Hill Book Company, 1979.

[26] J. T. Schwartz and M. Sharir. Identification of partially obscured objects in three dimensions by matching noisy characteristic curves. *International Journal of Robotics Research*, 6(2):29–44, 1987.

[27] A. Wallack, J. Canny, and D. Manocha. Object localization using crossbeam sensing. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 692–699, May 1993.

[28] Z. Zhang and O. Faugeras. Building a 3d world model with a mobile robot: 3d line segment representation and integration. In *International Conference on Pattern Recognition*, pages 38–42, 1990.

Object	Resolution	|Pixels|	$\sqrt{\frac{1}{n}\sum_n(x \Leftrightarrow \hat{x})^2}$	$\sqrt{\frac{1}{n}\sum_n(y \Leftrightarrow \hat{y})^2}$	$\sqrt{\frac{1}{n}\sum_n(\theta \Leftrightarrow \hat{\theta})^2}$	$\overline{Error}$
Hex-model	4.0	44.4	0.33446434	0.23129888	0.013330161	92.927956
Hex-model	2.0	82.4	0.07352662	0.049923003	0.0047521484	44.196507
Hex-model	1.0	157.6	0.041777074	0.0063066096	0.0036297794	21.544956
Hex-model	0.5	309.5	0.0031298357	0.0016034179	4.643562e-4	10.784967
Hex-model	0.25	614.6	0.0042416616	3.8894132e-4	1.0569962e-4	5.4200945
Letter "T"	4.0	49.2	0.5133025	0.60952026	0.021071866	103.38039
Letter "T"	2.0	90.4	0.13007967	0.25665605	0.00581296	48.95305
Letter "T"	1.0	174.3	0.054255202	0.124452725	0.0047051026	24.013884
Letter "T"	0.5	343.2	0.012289527	0.017529113	6.203042e-4	12.062334
Letter "T"	0.25	679.6	0.0040623806	0.0034328252	1.9236103e-4	6.0177937
Number "6"	4.0	106.5	0.21361086	0.19730452	0.010275254	228.42712
Number "6"	2.0	205.8	0.054341987	0.049577143	0.004476396	112.03862
Number "6"	1.0	405.9	0.010724932	0.02340998	0.0010456733	56.430653
Number "6"	0.5	803.6	0.0059675984	0.004836324	1.6956787e-4	28.311207
Number "6"	0.25	1600.9	0.0017914316	0.0019320602	7.470778e-5	14.09613
Fiducial	4.0	61.4	0.21314956	0.14251679	0.007586037	128.73416
Fiducial	2.0	120.0	0.057874706	0.07354973	0.0030514833	66.32908
Fiducial	1.0	237.4	0.029797971	0.02263394	8.284851e-4	33.25131
Fiducial	0.5	468.0	0.004916789	0.004921388	1.2238462e-4	16.175451
Fiducial	0.25	932.4	0.007628871	0.0050956924	1.399027e-4	8.207249

Table 11: Average localization technique performance for various objects at various resolutions

Object	Resolution	|Pixels|	$\sqrt{\frac{1}{n}\sum_n (x \Leftrightarrow \hat{x})^2}$	$\sqrt{\frac{1}{n}\sum_n (y \Leftrightarrow \hat{y})^2}$	$\sqrt{\frac{1}{n}\sum_n (\theta \Leftrightarrow \hat{\theta})^2}$	$\overline{Error}$
Hex-model Corners	4.0	12.2	0.8087669	0.74312437	0.021346115	11.319004
Hex-model Corners	2.0	12.5	0.15569122	0.25277224	0.013486578	3.3304203
Hex-model Corners	1.0	13.1	0.036884286	0.116834834	0.00448842	1.0500076
Hex-model Corners	0.5	14.9	0.024275968	0.024562975	0.005042185	0.26821047
Hex-model Corners	0.25	17.4	0.0063632606	0.004727341	0.0024016364	0.09157471

Table 12: Average localization technique performance using only the corners of the hex-model object at various resolutions