Recursive Total Least Squares: An Alternative to the Discrete Kalman Filter

Daniel L. Boley and Karen T. Sutherland
Computer Science Department
University of Minnesota
Minneapolis, MN 55455

Abstract

The discrete Kalman filter, which is becoming a common tool for reducing uncertainty in robot navigation, suffers from some basic limitations when used for such applications. In this paper, we describe a recursive total least squares estimator (RTLS) as an alternative to the Kalman filter, and compare their performances in three sets of experiments involving problems in robot navigation. In all cases, the RTLS filter converged faster and to more accuracy than the Kalman filter.

1 Introduction

The discrete Kalman filter [14], commonly used for prediction and detection of signals in communication and control problems, has more recently become a popular method of reducing uncertainty in robot navigation. One of the main advantages of using the filter is that it is recursive, eliminating the necessity for storing large amounts of data. The filter is basically a recursive weighted least squares estimator of the state of a dynamical system using a given transition rule. Suppose we have a discrete dynamical system $\mathbf{x}_i = F_{i-1}\mathbf{x}_{i-1} + e_{i-1}$, where $\mathbf{x}_i$ is the state vector, e_i is the noise vector, and F_{i-1} is the state transition matrix at time step i. We are given a sequence of measurements $\mathbf{b}_i$ obeying the model $\mathbf{b}_i = A_i\mathbf{x}_i + \epsilon_i$, where A_i is the given data matrix and ϵ_i is measurement noise. The Kalman filter is used to find an estimate of the state vector $\mathbf{x}_i$ from the measurement data that minimizes the noise in a least squares sense. The Kalman filter equations and a schematic diagram of the filter are in Appendix I. A complete description of the filter can be found in [9]. It requires an initial estimate of the solution and assumes that noise is weighted white gaussian. The discrete Kalman filter is guaranteed to be optimal in that it is guaranteed to find the best solution in the least squares sense.

Although originally designed as an estimator for dynamical systems, the filter is used in many applications as a static state estimator [19]. In the static problem, the state transition matrix F_{i-1} is the identity matrix I, so the problem is reduced to finding the state vector $\mathbf{x}$ minimizing the weighted Euclidean norm of the measurement noise

$$\|W\epsilon_i\|_2 = \|W(\mathbf{b}_i - A_i\mathbf{x}_i)\|_2,$$

where W is an optional weighting matrix (usually the inverse of the covariance matrix of measurement noise).

Also, due to the fact that functions are frequently non-linear, the extended Kalman filter (EKF) is used [2, 15]. The EKF formalism linearizes the function by taking a first order Taylor expansion around the current estimate of the state vector [9]. Assuming that the function is represented by a set of non-linear equations of the form $f_i(\mathbf{y}_i, \mathbf{x}) = 0$ where $\mathbf{x}$ is the state vector and $\mathbf{y}_i$ represents random parameters of f_i of which estimated measures, $\hat{\mathbf{y}}_i$, are taken, the first order Taylor expansion is given by:

$$f_i(\mathbf{y}_i, \mathbf{x}) = 0 \simeq f_i(\hat{\mathbf{y}}_i, \hat{\mathbf{x}}_{i-1}) + (\mathbf{y}_i - \hat{\mathbf{y}}_i)\frac{\partial \hat{f}_i}{\partial \mathbf{y}} + (\mathbf{x} - \hat{\mathbf{x}}_{i-1})\frac{\partial \hat{f}_i}{\partial \mathbf{x}}$$

where $\hat{\mathbf{x}}_i$ is the i-th estimate of the state vector and the derivatives are estimated at $(\hat{\mathbf{y}}_i, \hat{\mathbf{x}}_{i-1})$. This equation can be rewritten as:

$$\mathbf{b}_i = A_i\mathbf{x} + \epsilon_i$$

where:

$$\mathbf{b}_i = -f_i(\hat{\mathbf{y}}_i, \hat{\mathbf{x}}_{i-1}) + (\hat{\mathbf{x}}_{i-1})\frac{\partial \hat{f}_i}{\partial \mathbf{x}}$$

$$A_i = \frac{\partial \hat{f}_i}{\partial \mathbf{x}}$$

$$\epsilon_i = (\mathbf{y}_i - \hat{\mathbf{y}}_i)\frac{\partial \hat{f}_i}{\partial \mathbf{y}}$$

This linear approximation function is then used as the Kalman filter equation.

There are two basic problems which can occur when using either the Kalman or extended Kalman filter in robot navigation applications:

- The filter was developed for applications such as those in signal processing in which many measurements are taken [14]. Sensing in robot navigation is often done using camera images. The gathering and processing of each image is a time consuming process so a successful method must make do with relatively few readings.

- An underlying assumption in least squares estimation is that the entries in the data matrix are error-free [10]. In many actual applications, the errors in the data matrix can be at least as great as the measurement errors. In such cases, the Kalman filter can give poor results.

Two additional problems occur when using the EKF:

- The linearization process itself has the potential to introduce significant error into the problem.

- The EKF is not guaranteed to be optimal or to even converge [20]. It can easily fall into a local minimum when an initial estimate of the solution is poor, often the type of situation faced by robot navigators.

Our work in outdoor navigation [23], where measurements are expensive to obtain and have very significant error inherent to the system, motivated us to look for another filtering method, preferably one which would not require numerous measurements to converge and was not dependent on an error-free data matrix.

As the interesting recent work by Mintz et al [11, 18] in robust estimation and modeling of sensor noise has demonstrated, the criterion of optimality depends critically on the specific model being used. Given two methods, the first may produce optimality in one sense but not do as well as the second in another sense. When error exists in both the measurement and the data matrix, the best solution in the *least squares* sense is often not as good as the best solution in the *eigenvector* sense, where the sum of the squares of the

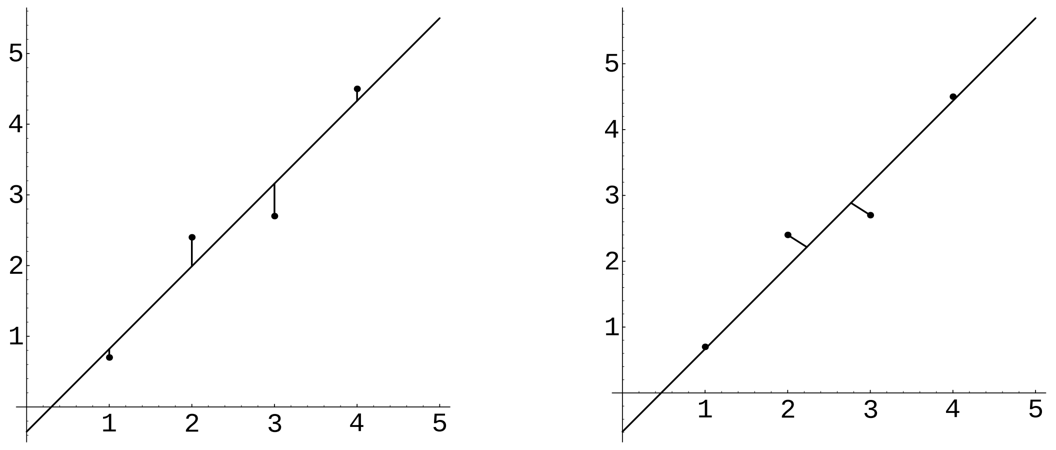

FIG. 1. *(a) In an LS solution, the sum of the squared vertical distances to the line of best fit is minimized. (b) In a TLS solution, the sum of the squared perpendicular distances to the line of best fit is minimized.*

perpendicular distances from the points to the lines are minimized [8] (Fig. 1). This second method is known in the statistical literature as *orthogonal regression* and in numerical analysis as *total least squares* (TLS) [24].

The TLS problem, in its simplest form, is to find a matrix E and vector $\mathbf{f}$ that minimizes $\|(E, \mathbf{f})\|_2$ such that $(A + E)\mathbf{x} = \mathbf{b} + \mathbf{f}$ for some vector $\mathbf{x}$. The vector $\mathbf{x}$ corresponding to the optimal $(E, \mathbf{f})$ is called the *TLS solution.* Figure 1 gives a graphical comparison of the two techniques when fitting a straight line. If $\mathbf{v} = (v_1, \ldots, v_p)^T$ is a right singular vector corresponding to the smallest singular value of $(A, \mathbf{b})$, then it is well known that the TLS solution can be obtained by setting $\mathbf{x} = -(v_1, \ldots, v_{p-1})^T / v_p$. If the smallest singular value is multiple, then there are multiple TLS solutions, in which case one usually seeks the solution of smallest norm. If v_p is too small or zero, then the TLS solution may be too big or nonexistent, in which case an approximate solution of reasonable size can be obtained by using the next smallest singular values(s) [24]. The TLS approach has received a lot of attention in the numerical analysis literature, partly because it arises in so many applications (see e.g. [10, 24]). The most common algorithms to compute the TLS solution are based on the Singular Value Decomposition (SVD), a non-recursive matrix decomposition which is computationally expensive to update.

Recently, some recursive TLS filters have been developed for applications in signal processing [4, 5, 7, 27]. Davila [4] used a Kalman filter to obtain a fast update for the eigenvector corresponding to the smallest eigenvalue of the covariance matrix. This eigenvector was then used to solve a symmetric TLS problem for the filter. It was not explained how the algorithm might be modified for the case where the smallest eigenvalue is multiple (i.e., corresponding to a noise subspace of dimension higher than one), or variable (i.e., of unknown multiplicity). In [27], Yu described a method for the fast update of an approximate eigendecomposition of a covariance matrix. He replaced all the eigenvalues in the noise subspace with their "average", and did the same for the eigenvalues in the signal subspace, obtaining an approximation which would be accurate if the exact eigenvalues could be grouped into two clusters of known dimensions. However, if the eigenproblem for the covariance matrix is replaced by the singular value problem on the signals, the condition numbers involved in the least squares solution can be reduced to their square roots [10, 12], potentially doubling the number of digits of accuracy in the computed solutions. In [5, 7], DeGroat and Dowling used this approach combined with the averaging technique used in [27], again assuming that the singular values could be grouped into two clusters. Recently, Bose et al.[3] applied Davila's algorithm to reconstruct images from noisy, undersampled frames after converting complex-valued image data into equivalent real data. All of these methods made some assumptions that the singular values or eigenvalues could be well approximated by two tight clusters, one big and one small. In this paper, we present a recursive algorithm that makes very few assumptions about the distribution of the singular values.

The paper is organized as follows: In section 2 we describe the ULV Decomposition, a recursive analog to the SVD which can be easily updated as new data arrives. In section 3 we show how the ULV can be used to design a recursive total least squares (RTLS) estimator. In section 4, we present some experiments comparing this estimator with recursive least squares (i.e. the Kalman filter in the static sense). In all the experiments we performed, the RTLS filter converged faster and to greater accuracy than did the Kalman filter.

2 The ULV Decomposition

In this section, we describe the ULV Decomposition, first introduced by Stewart [21, 22]. This is a method which reveals the noise subspace (i.e., the subspace corresponding to the smaller singular values), and which is easily updated when new data arrives without making any a priori assumptions about the overall distribution of the singular values. We describe how this method can be used to vary the cut between large and small singular values, and later show how this method may be applied to a TLS problem. We decompose the algorithm into a new set of primitive operations, making it possible to easily adapt the ULV Decomposition to new applications for which the SVD has been, until now, the only alternative.

The SVD is, in most cases, the matrix decomposition used to isolate and extract the smallest singular values and their associated singular vectors. The SVD is not easily updated when new data arrives; such updates generally require $O(p^3)$ operations (where p is the length of one row). The ULV Decomposition accomplishes the same task, at least approximately, but is designed so that when a new row is appended to the original matrix, it can be updated in $O(p^2)$ operations, much faster than the SVD.

The SVD is typically used to isolate the smallest singular values, and the success of any method based on the SVD depends critically on how that method decides which singular values are "small" enough to be isolated. The decision as to how many singular values to isolate may be based on a threshold value (find those values below the threshold), by a count (find the last k values), or by other considerations depending on the application. However, in extracting singular values one often wants to keep clusters of those values together as a unit. For example, if all values in a cluster are below a given threshold except one, which is slightly above the threshold, it is often preferable to change the threshold than split up the cluster. In the SVD, this extraction is easy. Since all the singular values are "displayed", one can easily traverse the entire sequence of singular values to isolate whichever set is desired. In this paper, we propose a set of primitive procedures to provide these same capabilities with the less computationally expensive ULV Decomposition.

As an example, in the Total Least Squares (TLS) problem it is necessary to deflate and isolate the smaller singular values based not only on the magnitude of the singular values, but also on the size or existence of the computed TLS solution, determined by the entries in the singular *vectors*. By isolating the deflation procedure from the criterion selection procedure, it is possible to use criteria peculiar to the TLS problem while directly integrating the ULV procedures into the TLS computation.

2.1 Data Structure

The ULV Decomposition of a real $n \times p$ matrix A (where $n \geq p$) is a triple of 3 matrices U, L, V plus a rank index r, where $A = ULV^T$, V is $p \times p$ and orthogonal, L is $p \times p$ and lower triangular, U has the same shape as A with orthonormal columns, and the leading $r \times r$ part of L has a Frobenius norm approximately equal to the norm of a vector of the r leading singular values of A. That is, $A = ULV^T$ with

$$L = \begin{pmatrix} C & 0 \\ E & F \end{pmatrix} \tag{1}$$

where $\|C\|_F^2 \approx \sigma_1^2(A) + \cdots + \sigma_r^2(A)$ encapsulates the "large" singular values of L. This implies that (E, F) (the trailing $p - r$ rows of L) approximately encapsulate the $p - r$ smallest singular values, and the last $p - r$ columns of V encapsulate the corresponding trailing right singular vectors.

In the data structure actually used for computation, L is needed to determine the rank index at each stage as new rows are appended, but the U is not needed to obtain the right singular vectors. Therefore, a given ULV Decomposition can be represented just by the triple $[L, V, r]$.

2.2 Primitive Procedures

We have developed five primitive procedures to use in updating the ULV Decomposition. Three of them, the *basic* procedures, do not use any tolerances, fudge factors or approximations of zero that might have to be supplied by the user. All of these are isolated in the two *outer* procedures. None of the basic procedures involve any heuristic operations, except that `Extract_Info` requires the use of a condition number estimator, which is by its nature a heuristic procedure, albeit a classical, well-tested one.

2.2.1 Basic Procedures The basic procedures are designed to allow easy updating of the ULV Decomposition as new rows are appended. Each basic procedure costs $O(p^2)$ operations. The basic procedures consist of a series of simple annihilation operations. Each annihilation operation is accomplished with a sequence of plane (Givens) rotations [10], which are orthogonal matrices of the form

$$Q = \begin{pmatrix} I & 0 & 0 & 0 \\ 0 & c & s & 0 \\ 0 & -s & c & 0 \\ 0 & 0 & 0 & I \end{pmatrix}$$

where $c^2 + s^2 = 1$ and the I's represent identity matrices of appropriate dimensions. If l_{ij} denotes an entry of L to be annihilated, and $l_{i,j-1}$ denotes the neighboring entry in the same row absorbing the length lost, then Q is constructed just so that when applied to L from the right, we expose the zero in the i, j position:

$$(i\text{-th row of } L) \cdot Q = (\ldots, l_{i,j-1}, l_{ij}, \ldots) \cdot Q = (\ldots, \times, 0, \ldots).$$

```
    C . . . .          C + . . .          C . . . .          C . . . .
    C C . . .  rotate  C C + . .  rotate  C C . . .  rotate  C C . . .
    C C C . .  from => C C C + .  from => C C C . .  once => C C C . .
    e e e f .  right   e e e f +  left    E E E F .  from    E E E F .
    e e e f f          e e e f f          e e e f f  left    e e e f f
    R R R R R          R . . . .          R . . . .          . . . . .
chop away row of zeroes and increment rank index:
            C . . . .
            C C . . .
       =>   C C C . .
            C C C C .
            e e e e f
```

FIG. 2. *Sketch of* `Absorb_One` *procedure. Upper case letters denote large entries, lower case letters small entries in the ULV partitioning,* `R` *denotes an entry of the new row,* `+` *a temporary fill, and* `.` *a zero entry.*

Analogously, if the entry absorbing the length lost lies in the same column, then the Q is constructed similarly, but applied from the left. By using a sequence of such rotations in a very special order, we can annihilate desired entries while filling in as few zero entries as possible, and then restoring the few zeroes that are filled in. In the diagrams (figures 2 & 3), upper case letters denote larger values to be treated as part of the leading $r \times r$ part, and lower case letters denote smaller values to be treated as part of the trailing part. We show the operations on L, partitioned as in (1). Each rotation applied from the right is also accumulated in V, to maintain the identity $A = ULV^T$, where the U is not saved.

- `Absorb_One`:

  Absorb a new row. The matrix A is augmented by one row, obtaining

$$\begin{pmatrix} A \\ \mathbf{a}^T \end{pmatrix} = \begin{pmatrix} U & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} L \\ \mathbf{a}^T V \end{pmatrix} V^T.$$

  Then the L, V are updated to restore the ULV structure, and the rank index r is incremented by 1. No determination is made if the rank has really increased by 1; this is done elsewhere.

  The process begins by applying p Givens rotations [10] from the right to rotate all the nonzeroes in the new row all the way to the left. Then p rotations are applied from the left to restore L to lower triangular. Finally, a single rotation from the right is used to completely annihilate the last nonzero remaining in the extra row. Once the extra row is all zero, it can be chopped away. Since each rotation modifies at most $2p$ entries of L plus (for the right rotations) $2p$ entries of V, the total cost is $O(p^2)$. In Figure 2, we illustrate the process on a 5×5 example going from rank 3 to rank 4.

- `Extract_Info`:

  The following information is extracted from the ULV Decomposition: (a) the Frobenius norm of (E, F) (i.e., the last $p - r$ rows of L), (b) an approximation of the last singular value of C (i.e., the leading $r \times r$ part of L), and (c) a left singular vector of C corresponding to this singular value. If the rank index were to be reduced

```
v v v v          . . . v

C . . . .        C + . . .         C . . . .             C . . . .
C C . . .  rotate  C C + . .  rotate  C C . . .  decrement  C C . . .
C C C . .  from => C C C + .  from => C C C . .  rank   =>  C C C . .
C C C C .  left    c c c c .  right   c c c c .  index      e e e f .
e e e e f          e e e e f          e e e e f             e e e f f
```

FIG. 3. *Sketch of* `Deflate_One` *procedure. The lower case* `c`*'s denote entries of C of small norm exposed by the rotations constructed to annihilate the entries* `v` *of the left singular vector* $\mathbf{v}$.

by 1, the singular value (item (b)) would be moved to the trailing part, and item (c) is needed to accomplish that. This operation does not change the ULV Decomposition.

Items (b) and (c) are obtained by using a *condition number estimator* essentially a method designed to estimate the norm of the inverse of a matrix and to yield a vector that exhibits that norm, with a cost comparable to that of back-substitution, $O(p^2)$. There have been many condition number estimators proposed in the literature, differing on their accuracy and on which matrix norm they are based, and a large body of computational experience exists [13]. In our implementation we chose the one of Van Loan [26], which is based on the Euclidean norm and usually yields good accuracy especially for well separated singular values. But one could also use the ones in LINPACK [6], LAPACK [1], or any in Higham's excellent survey [13].

- `Deflate_One`:

  Deflate the ULV Decomposition by one (i.e., apply transformation and decrement the rank index by one so that the smallest singular value in the leading $r \times r$ part of L is "moved" to the trailing rows). Specifically, transformations are applied to isolate the smallest singular value in the leading $r \times r$ part of L into the last row of this leading part. Then the rank index is decremented by 1, effectively moving that smallest singular value from the leading part to the trailing part of L. This operation does not check whether the singular value moved is close to zero or any other singular value.

  The process begins by applying r rotations to C from the left to expose a small row at the bottom. These rotations are constructed by annihilating all but the leftmost entry of an approximate left singular vector corresponding to a small singular value, previously supplied by `Extract_Info`. Then r rotations are applied from the right to restore the lower triangular form of L, and the rank index is decremented. As in `Absorb_One`, each rotation modifies at most $2p$ entries of L and (just for the right rotations) $2p$ entries of V, so the total cost is $O(rp) \leq O(p^2)$. In Figure 3, we illustrate this with a 5×5 example going from rank 4 to rank 3. The first reduction is to transform the C so as to expose a row of small norm at the bottom which can then be "moved" to the trailing part. The `v`'s denote the elements of the trailing left singular vector of C. The rotations (call them Q's) are determined by maintaining the invariance of the expression $\mathbf{v}^T Q Q^T C$, while annihilating the $r-1$ leading entries of $\mathbf{v}^T$.

2.2.2 Outer Procedures

- `Deflate_To_Gap`: This procedure uses a heuristic to decide whether or not the rank boundary, represented by the rank index r is in the middle of a cluster of singular values. That is, is the smallest singular value of the leading part of L well separated from the singular values in the trailing part? If the heuristic says no, this procedure deflates repeatedly until the heuristic finds a gap in the singular values marking the end of the cluster. The cost of this procedure is $O(p^2k)$, where k is the number of deflations needed, which is limited by the dimension of the largest cluster.

  Let s be the smallest singular value of the leading r rows of L, and let f be the Frobenius norm of the trailing part. Then the heuristic is simply that a gap exists if $s > df$, where d is a user chosen `Spread`. In order to allow for round-off or other small noise, we pretend that the trailing part has an extra $p+1$-th singular value equal to a user chosen `Zero_Tolerance` b. Then the heuristic actually used is

  $$s^2 > d^2(f^2 + b^2).$$

  Hence, any singular value that is below b or within a cluster of b will be treated as part of the trailing part. The only two user defined parameters needed for this heuristic are the `Spread` d and the `Zero_Tolerance` b.

- `Update`

  This procedure encompasses the entire process. It takes an old ULV Decomposition and a new row to append, and incorporates the row into the ULV Decomposition. The new row is absorbed, and the rank is deflated if necessary to find a gap among the singular values. The smallest singular value is always isolated, so the trailing part is never empty (i.e. we always have $r < p$). The absorption of the new row implicitly increments the rank index, so this procedure allows the rank to either go up by one, or be deflated to its original value, or be deflated to any smaller value. This process calls `Absorb_One` and `Deflate_To_Gap` once. If the rank index $r = n$ after the new row has been absorbed, it is also necessary to call `Deflate_One` once to isolate at least one singular value. Hence the cost will be bounded by $O(p^2k)$, as in `Deflate_To_Gap`. We remark that one could optimize the method by a constant by incorporating into `Absorb_One` a check to see if the rank has really increased, by supplying `Absorb_One` with the user defined tolerances.

3 The RTLS Algorithm

We can adapt the ULV Decomposition to solve the Total Least Squares (TLS) problem $A\mathbf{x} \approx \mathbf{b}$, where new measurements b are continually being added. The adaptation of the ULV to the TLS problem has also been analyzed independently in great detail in [25], though the recursive updating process was not discussed at length. For our specific purposes, let A be an $n \times (p-1)$ matrix and $\mathbf{b}$ be an n-vector, where p is fixed and n is growing as new measurements arrive. Then given a ULV Decomposition of the matrix $(A, \mathbf{b})$ and an approximate TLS solution to $A\mathbf{x} \approx \mathbf{b}$, our task is to find a TLS solution $\widehat{\mathbf{x}}$ to the augmented system $\widehat{A}\widehat{\mathbf{x}} \approx \widehat{\mathbf{b}}$, where

$$\widehat{A} = \begin{pmatrix} \lambda A \\ \mathbf{a}^T \end{pmatrix} \text{ and } \widehat{\mathbf{b}} = \begin{pmatrix} \lambda \mathbf{b} \\ \beta \end{pmatrix},$$

and λ is an optional exponential forgetting factor [12].

The algorithm presented here is derived from the SVD-based TLS algorithm (Algorithm 3.1 of [24]), but simplified for a single right hand side for the sake of clarity. Multiple right hand sides can be handled in a similar fashion. The main computation cost of that algorithm occurs in the computation of the SVD. That cost is $O(p^3)$ for each update. We replace that with a updating ULV computation, so the total cost is reduced to only $O(p^2)$ per input.

The RTLS Algorithm:

- Start with $[L, V, r]$, the ULV Decomposition of $(A, \mathbf{b})$, and the coefficients $\mathbf{a}^T, \beta$ for the new incoming equation $\mathbf{a}^T\mathbf{x} = \beta$.
- Call `Update` with the old ULV Decomposition $[\lambda L, V, r]$ and the new row $(\mathbf{a}^T, \beta)$ to compute the updated ULV Decomposition for the augmented matrix

$$(\,\widehat{A},\quad \widehat{\mathbf{b}}\,) = \begin{pmatrix} \lambda A & \lambda \mathbf{b} \\ \mathbf{a}^T & \beta \end{pmatrix} = \widehat{U}\widehat{L}\widehat{V},$$

  The new ULV Decomposition is $[\widehat{L}, \widehat{V}, \widehat{r}]$ (note that the new $\widehat{U}$ is thrown away).
- Partition

$$\widehat{V} = \begin{pmatrix} \widehat{V}_{11} & \widehat{V}_{12} \\ \widehat{V}_{21} & \widehat{V}_{22} \end{pmatrix},$$

  where $\widehat{V}_{22}$ is $1 \times (p - \widehat{r})$.

  If $||\widehat{V}_{22}||$ is too close to zero (according to a user supplied tolerance), then call `Deflate_To_Gap` to obtain a new ULV Decomposition with a smaller rank index. Denote the new ULV Decomposition $[\widehat{L}, \widehat{V}, \widehat{r}]$ and repeat this step until $||\widehat{V}_{22}||$ is large enough.
- Find an orthogonal matrix Q such that $\widehat{V}_{22}Q = (0, \ldots, 0, \alpha)$, and let $\mathbf{v}$ be the last column of $\widehat{V}_{12}Q$. Then compute the new approximate TLS solution according to the formula $\widehat{\mathbf{x}} = -\mathbf{v}/\alpha$.

This RTLS Algorithm makes very few assumptions about the underlying system, though the user must supply a zero tolerance for $||\widehat{V}_{22}||$, plus a `Spread` and `Zero_Tolerance` for the deflations by `Deflate_To_Gap`. The method depends on the use of the primitive operations `Update` and `Deflate_To_Gap` in a way very analogous to the use of the SVD in the standard algorithm in [24]. For our application of robot navigation, it sufficed to set both zero tolerances to zero and the `Spread` to 1.5.

4 Experimental Results

In our first set of experiments, we compared performance of the Kalman, EKF, and RTLS using a very simple line fitting problem. We fit straight lines to groups of 20 points taken from lines of varying slopes. Random noise was added to both x and y coordinates of the points. Since all point coordinates were integer, error was introduced with a discretized normal distribution of variance 4. The linear equation used for both the Kalman and RTLS filters was:

$$y = mx + c$$

with m the slope and c the y intercept of the line. We formulated the problem so that the i-th measurement consisted of

$$A_i = \begin{bmatrix} x_i & 1 \end{bmatrix}; \qquad \mathbf{b}_i = y_i$$

where A_i is the data matrix and $\mathbf{b}_i$ is the measurement vector. The estimated state vector

$$\mathbf{x}_i = \begin{bmatrix} m \\ c \end{bmatrix};$$

is updated. The non-linear equation used for the EKF was:

$$f(\mathbf{y}, \mathbf{x}) = y_1 cos\theta + y_2 sin\theta - \rho$$

where $\mathbf{y} = \begin{bmatrix} y_1 & y_2 \end{bmatrix}^T$ is the measurement vector and $\mathbf{x} = \begin{bmatrix} \theta & \rho \end{bmatrix}^T$ is the state vector. This equation was linearized and formulated as outlined in section 1.

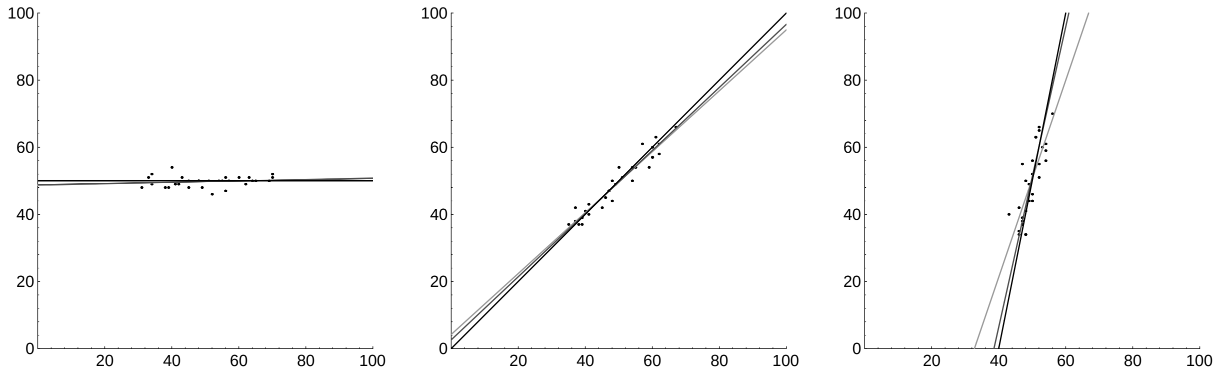

FIG. 4. *Lines fit to noisy points. Solid line is the true line. Dark grey line is the fit with RTLS. Light grey line is the fit with the Kalman filter. Slopes of lines shown are 0, 1, and 5.*

Figure 4 shows the results of three trials with the Kalman and RTLS filters. The EKF often failed to converge to the global minimum, even for this simple problem. Initial estimate for the Kalman filter was the line passing through the first two data points. The RTLS filter does not require an initial estimate. To account for the fact that the rightmost column of the data matrix has no error, we took advantage of the unique flexibility of the RTLS approach to scale that column by a factor of η so that the errors modeled by RTLS were reduced by $1/\eta$. After trying several larger values of η, and comparing with the mixed LS-TLS method [10, 24], a nonrecursive algorithm capable of treating certain columns as exact, we found it sufficed to use $\eta = 100$ for all experiments.

The RTLS filter performs better as the lines become more vertical. This is to be expected since the vertical distance to the line, minimized by the Kalman filter, approaches the perpendicular distance to the line, minimized by RTLS, as the slope of the line approaches zero. Ten trials were run with each algorithm with lines of different slopes. Figure 5 shows the resulting slope and y intercepts of all ten trials for lines of actual slope 0, 1 and 5.

In our second set of experiments, we simulated a simple robot navigation problem typical of that faced by an actual mobile robot [2, 15, 16, 17]. The robot has identified a single landmark in a two-dimensional environment and knows landmark location on a map. It does not know its own position. It moves in a straight line and with a known uniform velocity. Its goal is to estimate its own start position relative to the landmark by measuring

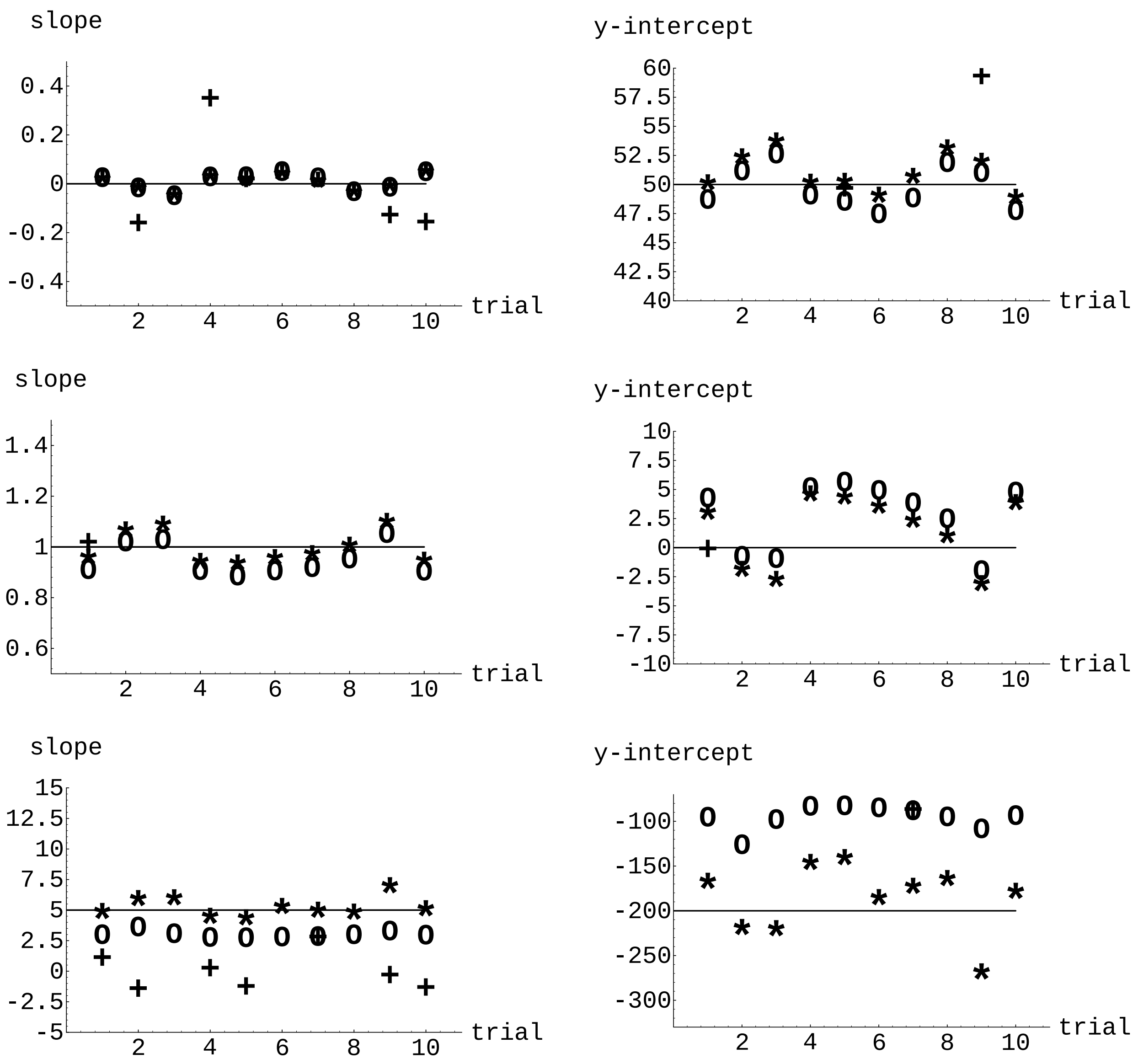

FIG. 5. *Slope and y intercept in all ten trial runs with lines of slopes 0, 1 and 5 and y intercepts of 50, 0 and -200, respectively. The* $*$*'s represent results of the RTLS filter. The* O*'s represent results of the Kalman filter. The* $+$*'s represent results of the EKF. The* $+$*'s are missing at values where the EKF result was off the graph or failed to converge.*

the visual angle α between its direction of heading and the landmark. Measurements are taken periodically as it moves. Figure 6 shows a diagram of the problem. For simplification, it is assumed that the landmark is located at (0,0), that the y coordinate of the robot's start position does not change as the robot moves, and that the robot knows what side of the landmark it is on. Although these assumptions may seem restrictive, it can easily be shown that knowing the actual location of two landmarks on the map and finding robot start position relative to both of them in this way will allow one to uniquely determine actual robot start position on the map.[1]

[1]This process can, of course, be extended to estimate intermediate and current robot position.

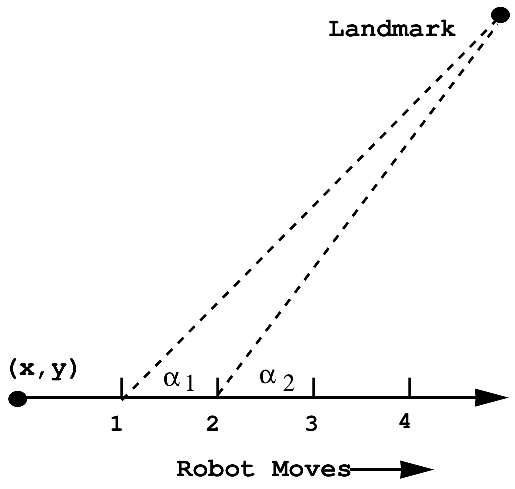

FIG. 6. *Diagram of simulated robot navigation problem. The robot moves along the horizontal line. Landmark location and velocity are known. Angle α_i is the angle from robot heading to the landmark at time i. Goal is to estimate initial robot location (x,y).*

In our experiments, it was assumed that the y coordinate of the robot path was negative (i.e., the path, as shown in Figure 6, was on the side below the landmark), that robot *velocity* was 20 per unit of time and that measurements of α were taken at unit time intervals. At any time t_i:

$$cot(\alpha_i) = \frac{x + t_i * velocity}{y}$$

where (x, y) is robot start position and α_i is the angle from robot heading to the landmark. We are not assuming any particular sensing device for this set of experiments, only that measurements of the angles α_i are made. Random error with a uniform distribution was added to the angle measures and a normally distributed random error was added to the time measurement. We again formulated the problem so that the data matrix, as well as the measurement vector contained error:

$$A_i = \begin{bmatrix} 1 & -cot(\alpha_i) \end{bmatrix}; \quad \mathbf{x}_i = \begin{bmatrix} x \\ y \end{bmatrix}; \quad \mathbf{b}_i = -t_i * velocity$$

where, at time t_i, A_i is the data matrix, $\mathbf{b}_i$ is the measurement vector, and $\mathbf{x}_i$ is the estimated state vector consisting of the coordinates (x, y) of robot start position. The Kalman filter was given an estimated start of (0,0). The RTLS algorithm had no estimated start position provided. As in the previous RTLS experiment, the leading column of the data matrix was scaled by $\eta = 100$ to reduce the allowed errors. Results are summarized in Figure 7. The mean deviations d (of 10 trials) of the estimates from the actual start location of (-460, -455) are compared for six different error amounts. The top three graphs have uniformly distributed error in α of $\pm 2°$ and normally distributed error in t with standard deviation sd=0, .05, and .1. The bottom three graphs have uniformly distributed error in α of $\pm 4°$ and normally distributed error in t with sd=0, .05 and .1. The jump in the RTLS distance at the second measure is due to the fact that RTLS does not require, and is not given, an initial estimate of location. The velocity/time interval used, combined with the error distribution used, produced error on some runs that gave readings of $\alpha_2 < \alpha_1$ (see Figure 6). Since there were only two measurements taken at this point, the system was not yet overdetermined, and the erroneous measures were given significant weight. This demonstrates how quickly the RTLS filter can recover from such errors. Table 1 gives mean deviation from actual location after 15 measurements. For all six groups of experiments, the RTLS filter converged more quickly than the Kalman filter. After 15 measurements, the RTLS estimate was closer to the actual location than was the Kalman in five of the six groups.

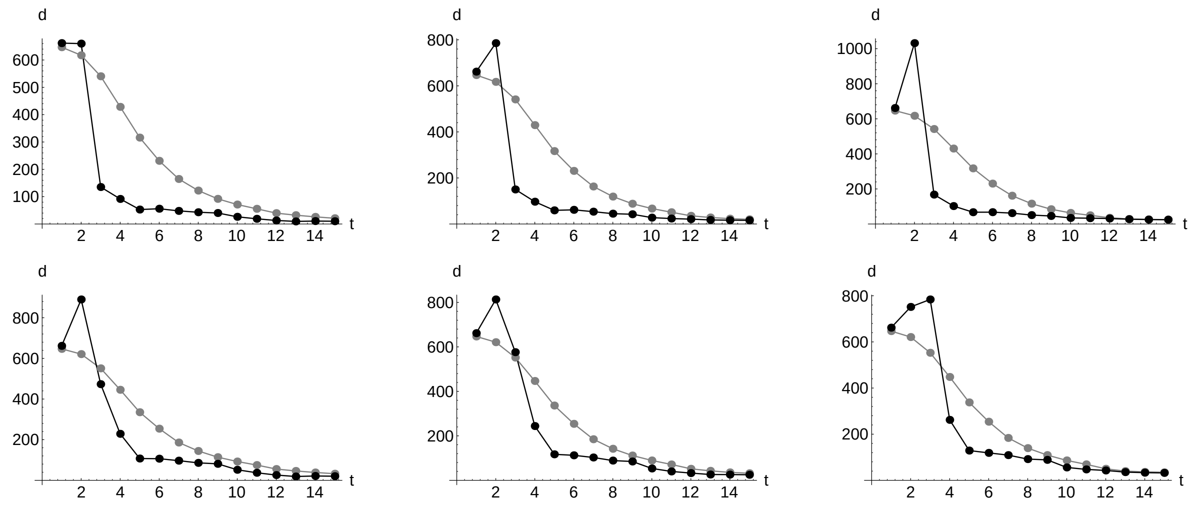

FIG. 7. *Comparison of mean deviations from estimated to actual start position. Measurements were taken at unit time intervals (horizontal axis). Vertical axis gives mean deviation d. Top three graphs have uniformly distributed error in α of $\pm 2°$ and normally distributed error in t with sd=0, .05 and .1. Bottom three graphs have uniformly distributed error in α of $\pm 4°$ and normally distributed error in t with sd=0, .05 and .1. Results using the RTLS algorithm are shown in black. Results using the Kalman filter are shown in grey.*

Error in α	Error in t	0	.05	.1
$\pm 2°$	Kalman	32.47	20.27	24.54
	RTLS	20.24	15.90	24.81
$\pm 4°$	Kalman	21.01	31.80	34.63
	RTLS	10.11	24.97	32.13

TABLE 1
Mean deviation of estimate from actual location after 15 measurements.

The third set of experiments consisted of a sequence of indoor robot runs with our TRC Labmate robot. As in the second set of experiments, the robot did not know its own position on the map, but did know the location of a single landmark. Its task was to take an image, find the landmark in the image, and use the result to determine its start position relative to the landmark.

A Panasonic WV-BL202 camera with a 6mm lens was mounted at an angle of 90° to robot bearing. Horizontal field of view was 56°47′. Images were grabbed on a Sun

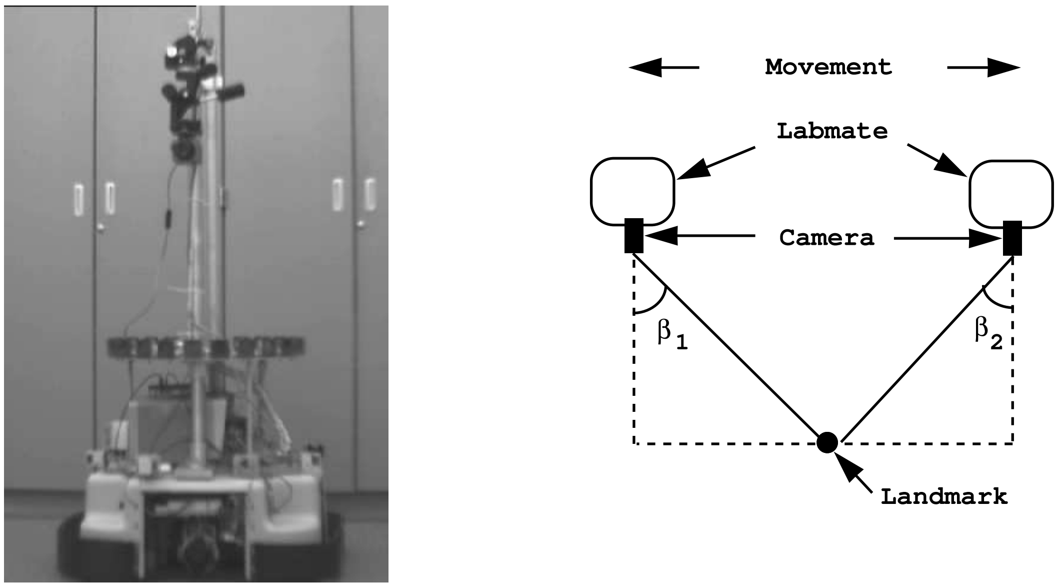

FIG. 8. *TRC Labmate with camera mounted at* 90° *to bearing and diagram showing how angles to landmark are measured. Angle measure is bound by* $\pm 25°22'$ *for the given field of view.*

SPARCstation IPX using Sun's VideoPix. In order to avoid the distortion caused by Sun's conversion of the 480 vertical by 720 horizontal pixels grabbed to the 480 vertical by 640 pixels stored, the initial 480 by 720 image was processed. Forty pixels were cropped off each end of the image, resulting in a 480 by 640 pixel image with a horizontal field of view of $50°44'$. "Landmarks" were mini Maglite high intensity flashlight candles.

The angular position of the landmark was measured in a sequence of images taken while the robot moved across the room at a constant velocity. Not only was there error in angle measure, but error occurred in velocity, robot bearing and in the times at which the images were taken. It is not possible to predict and model these errors. For example, velocity was set at 20mm/second, but average true velocity across runs ranged from 21.4mm/second to 22.5mm/second. In addition, the supposed constant velocity was not constant throughout a single run, varying in an unpredictable manner. It would be unrealistic to assume any of these errors or their combined result to have a gaussian distribution. Thus, it should be noted that the assumption of gaussian distribution of noise cannot be made in this set of experiments. Figure 8 shows the Labmate with camera and a diagram of how the angles are measured. When the landmark is in the left of the camera image, the angle (β_1 in the diagram) is negative. When the landmark is in the right of the camera image, the angle (β_2 in the diagram) is positive. Angle measure is thus bound by $\pm 25°22'$ for the given field of view.

It is again assumed that the landmark is located at (0,0), that the y coordinate of the robot's position does not change as the robot moves, and that the robot knows which side of the landmark it is on. At any step i:

$$tan(\beta_i) = \frac{x + (t_0 + i * interval) * velocity}{y}$$

where (x, y) is robot start position, β_i is the measured angle, t_0 is robot start time, *interval* is the interval at which images are grabbed and *velocity* is robot velocity. The problem

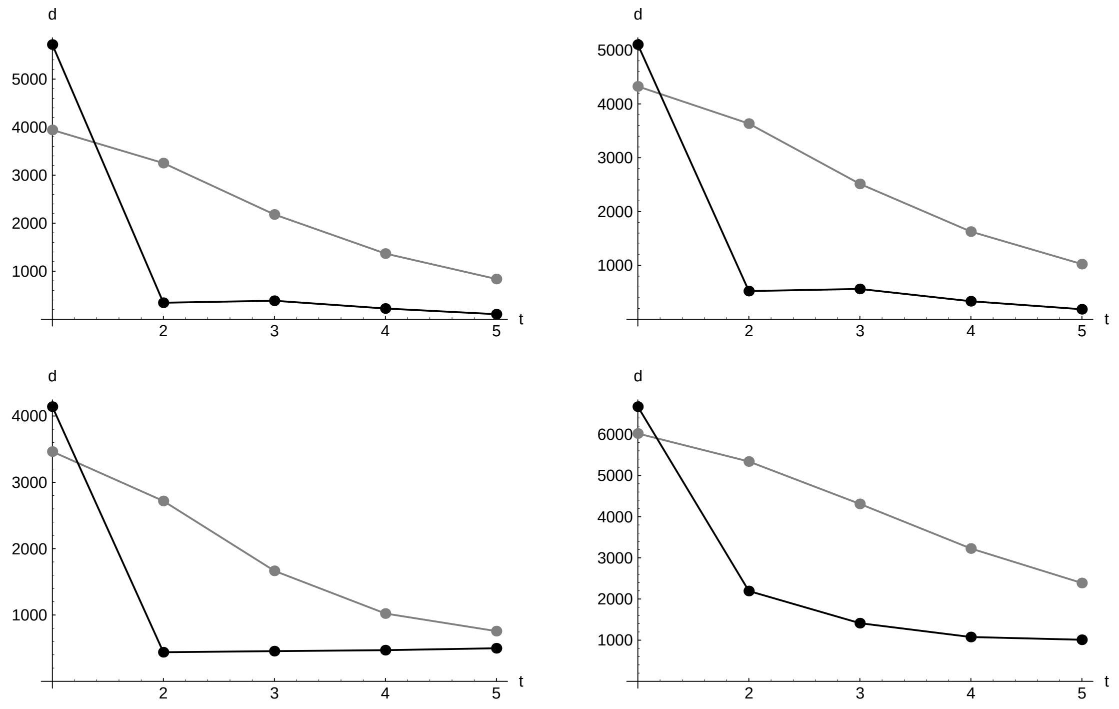

FIG. 9. *Comparison of filters with actual robot runs: Images were grabbed at time intervals t (horizontal axis) 12 seconds apart. Vertical axis gives deviation of estimated start position from actual start position in millimeters. Landmark was placed at a different location for each run. Results using the RTLS algorithm are shown in black. Results using the Kalman filter are shown in grey.*

was expressed as a linear function so that no accuracy was lost by linearizing. However, the data matrix as well as the measurement vector contained error:

$$A_i = \begin{bmatrix} 1 & -tan(\beta_i) \end{bmatrix}; \quad \mathbf{x}_i = \begin{bmatrix} x \\ y \end{bmatrix}; \quad \mathbf{b}_i = -(t_0 + i * interval) * velocity$$

where at any step i, A_i is the data matrix, $\mathbf{b}_i$ is the measurement vector and $\mathbf{x}_i$ is the estimated state vector consisting of the coordinates (x, y) of robot start position. As in the previous set of experiments, the Kalman filter was given an estimated start position of (0,0) and the leading column of the data matrix was weighted by $\eta = 100$.

Figure 9 shows a comparison of four of the robot runs. Robot velocity was set to 20mm/sec. Five images were grabbed 12 seconds apart. Robot start position relative to the landmark used for localization was different in each run. The deviations d of the estimate of start location from actual start location at each 12 second time interval t are compared. As in the simulated runs, the RTLS filter converges faster and to more accuracy than does the Kalman.

5 Conclusion

In this paper, we have presented the Recursive Total Least Squares (RTLS) filter. This filter is easily updated as new data arrives, yet makes very few assumptions about the data or the problem being solved. The method was based on a new reorganization of the ULV Decomposition. We suggested its use as an alternative to the Kalman filter in reducing uncertainty in robot navigation. In this context RTLS does not require an initial state estimate, avoids modeling errors introduced by the extended Kalman filter, does not suffer the traps of local minima, and converges quickly. We have illustrated the method using three different sets of experiments. It has been seen that even on a simple line fitting example, the RTLS can be a much more effective recursive algorithm. It is demonstrated that in the domain of robot navigation the RTLS can easily provide more accurate estimates than the Kalman filter, and in fewer time steps, especially when errors are present in both the measurement vector and the data matrix.

Appendix I
Kalman filter equations:

System Model	$\mathbf{x}_i = F_{i-1}\mathbf{x}_{i-1} + e_{i-1}$
Measurement Model	$\mathbf{b}_i = A_i\mathbf{x}_i + \epsilon_i$
Initial Conditions	$\mathbf{x}_0 = (A_0^T V_0^{-1} A_0)^{-1} A_0^T V_0^{-1}\mathbf{b}_0$
	$P_0 = (A_0^T V_0^{-1} A_0)^{-1}$
State Estimate Extrapolation	$\mathbf{x}_i(-) = F_{i-1}\mathbf{x}_{i-1}(+)$
Error Covariance Extrapolation	$P_i(-) = F_{i-1}P_{i-1}(+)F_{i-1}^T + U_{i-1}$
State Estimate Update	$\mathbf{x}_i(+) = \mathbf{x}_i(-) + K_i[\mathbf{b}_i - A_i\mathbf{x}_i(-)]$
Error Covariance Update	$P_i^{-1}(+) = P_i^{-1}(-) + A_i^T V_i^{-1} A_i$
Kalman Gain Matrix	$K_i = P_i(+)A_i^T V_i^{-1}$

TABLE 2

Discrete Kalman Filter Equations where B is the measurement vector, A is the data matrix, F is the state transition matrix, $\mathbf{x}$ is the state vector, and $e_i \sim N(0, U_i)$, $\epsilon_i \sim N(0, V_i)$

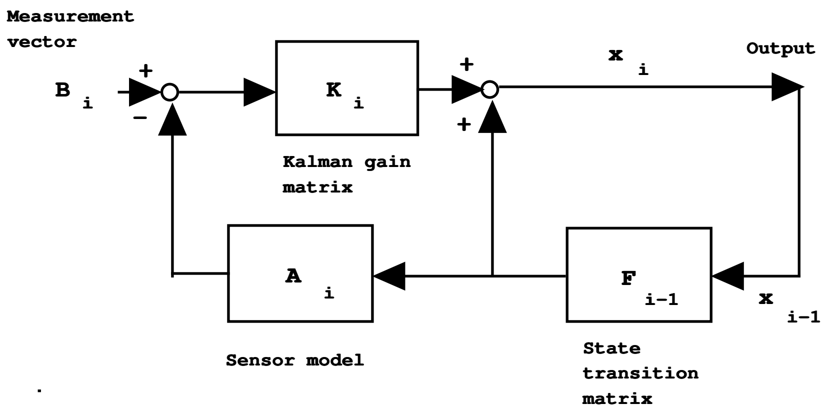

FIG. 10. *Schematic diagram of Kalman filter*

References

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

[2] N. Ayache and O. D. Faugeras, *Maintaining representations of the environment of a mobile robot*, IEEE Transactions on Robotics and Automation, 5 (1989), pp. 804–819.

[3] N. K. Bose, H. C. Kim, and H. M. Valenzuela, *Recursive implementation of total least squares algorithm for image reconstruction from noisy, undersampled multiframes*, in Proceedings of 1993 International Conference on Acoustics, Speech and Signal Processing, IEEE, May 1993, pp. V–269–V–272.

[4] C. E. Davila, *Recursive total least squares algorithms for adaptive filtering*, in Proceedings of 1991 International Conference on Acoustics, Speech and Signal Processing, IEEE, May 1991, pp. 1853–1856.

[5] R. D. DeGroat, *Noniterative subspace tracking*, IEEE Transactions on Signal Processing, 40 (1992), pp. 571–577.

[6] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK User's Guide*, SIAM, Philadelphia, 1979.

[7] E. M. Dowling and R. D. DeGroat, *Recursive total least squares adaptive filtering*, in SPIE Proceedings on Adaptive Signal Processing, vol. 1565, SPIE, July 1991, pp. 35–46.

[8] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, Inc., 1st ed., 1973.

[9] A. Gelb, *Applied Optimal Estimation*, The M. I. T. Press, 1st ed., 1974.

[10] G. H. Golub and C. F. V. Loan, *Matrix Computations*, Johns Hopkins, 2nd ed., 1989.

[11] G. Hager and M. Mintz, *Computational methods for task-directed sensor data fusion and sensor planning*, The International Journal of Robotics Research, 10 (1991), pp. 285–313.

[12] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, 2nd ed., 1991.

[13] N. J. Higham, *A survey of condition number estimators for triangular matrices*, SIAM Rev., 29 (1987), pp. 575–596.

[14] R. E. Kalman, *A new approach to linear filtering and prediction problems*, Journal of Basic Engineering, (1960), pp. 35–45.

[15] A. Kosaka and A. C. Kak, *Fast vision-guided mobile robot navigation using model- based reasoning and prediction of uncertainties*, CVGIP: Image Understanding, 56 (1992), pp. 271–329.

[16] D. J. Kriegman, E. Trendl, and T. O. Binford, *Stereo vision and navigation in buildings for mobile robots*, IEEE Transactions on Robotics and Automation, 5 (1989), pp. 792–803.

[17] L. Matthies and S. A. Shafer, *Error modeling in stereo navigation*, IEEE Journal of Robotics and Automation, RA-3 (1987), pp. 239–248.

[18] R. McKendall and M. Mintz, *Sensor-fusion with statistical decision theory: A prospectus of research in the grasp lab*, Tech. Rep. MS-CIS-90-68, University of Pennsylvania, September 1990.

[19] R. C. Smith and P. Cheeseman, *On the representation and estimation of spatial uncertainty*, The International Journal of Robotics Research, 5 (1986), pp. 56–68.

[20] H. W. Sorenson, *Least-squares estimation: from gauss to kalman*, IEEE Spectrum, (1970), pp. 63–68.

[21] G. W. Stewart, *Updating a rank-revealing ULV decomposition*, Technical Report CS-TR 2627, Department of Computer Science, University of Maryland, 1991.

[22] ——, *An updating algorithm for subspace tracking*, IEEE Trans. Signal Proc., 40 (1992), pp. 1535–1541.

[23] K. T. Sutherland and W. B. Thompson, *Inexact navigation*, in Proceedings 1993 International Conference on Robotics and Automation, IEEE, May 1993.

[24] S. Van Huffel and J. Vandewalle, *The Total Least Squares Problem - Computational Aspects and Analysis*, SIAM, Philadelphia, 1991.

[25] S. Van Huffel and H. Zha, *An efficient total least squares algorithm based on a rank-revealing two-sided orthogonal decomposition*, Numerical Algorithms, 4 (1993), pp. 101–133.

[26] C. F. Van Loan, *On estimating the condition of eigenvalues and eigenvectors*, Lin. Alg. & Appl., 88/89 (1987), pp. 715–732.

[27] K.-B. Yu, *Recursive updating the eigenvalue decomposition of a covariance matrix*, IEEE Transactions on Signal Processing, 39 (1991), pp. 1136–1145.