k-Vector Range Searching Techniques*

Daniele Mortari†, and Beny Neta‡

Abstract

Various k-vector range searching techniques are presented here. These methods accomplish the range search by taking advantage of an n-long vector of integers, called the k-vector, to avoid the search phase and, thus, to minimize the search time. The price is increased memory requirement for the k-vector allocation. However, it is possible to balance the extra memory required and the speed attained by choosing a step parameter h which samples the k-vector. The proposed method is compared with the well known "binary search" technique, and demonstrates a high speed gain rate (from 10 to more than 50 times). The application of the k-vector technique to dynamic databases (when entries can be deleted or inserted), is also presented. Then, the general two-level k-vector technique, for piecewise linear data distributions, is presented. Finally, just to show one of the wide-range possible applications, a two-level k-vector technique is applied to compute the *arcsin* function, by means of a look-up table approach.

Introduction

In Refs. [1, 2], the first author presented the Search Less Algorithm (SLA), a new tool to identify the stars observed by a wide Field-Of-View (FOV) star tracker. This technique was developed within the general problem of spacecraft attitude determination for the *lost-in-space* general case, that is, when the spacecraft attitude is completely unknown. SLA accomplishes the task much faster than other star identification algorithms thanks to the introduction of a new range searching approach, named the *k-vector* technique, which constitutes the heart of SLA. This new range searching method requires the construction of an n-long vector of integers, the vector k, whose entries contain information useful to solve the range searching problem by avoiding the searching phase, which constitutes the heaviest computational load of the star pattern recognition algorithms.

The k-vector technique was then applied in various star pattern recognition approaches: the SP-Search, presented in Ref. [3], and then extended in Ref. [4] for the multiple FOV star trackers StarNav II and III (see Refs. [5, 6, 7]). Lately, the k-vector technique is also adopted within the LISA algorithm, recently presented in Ref. [8], which will be tested and used to accomplish the star identification process for

*This work was supported by Italian Space Agency (ASI), under contract ARS 98-79.

†Dr Mortari is affiliated with Università degli Studi "La Sapienza" di Roma, Via Salaria 851, 00138 Roma, Italy. daniele@psm.uniroma1.it

‡Dr Neta is affiliated with the Naval Postgraduate School, Department of Mathematics, Code MA/Nd, Monterey, CA 93943. bneta@nps.navy.mil

the StarNav I test experiment on board the Space Shuttle (flight STS 107, expected launch date: January 11, 2001).

Since the k-vector technique represents a general mathematical tool to solve the range searching problem, it is presented here in its complete form with many new useful applications. These new k-vector techniques can be generalized to a broader class of problems, of which the star pattern recognition application is only a subset.

The original k-vector searching technique

The range searching problem consists of identifying, within a large n-long ($n \gg 1$) database y, the set of all data elements $y(i)$ falling within a given required range $[y_a,\ y_b]$, where $y_a < y_b$. Up until now, this problem has been solved by using the "Binary Search Technique" (BST), which mainly requires an amount of $2\log_2 n$ comparisons.

Many variants of BSTs have been introduced to be used for particular databases in order to minimize the range searching time with respect to the original BST (see Bentley and Sedgewick [9]). As for the speed gain, a substantial improvement has been obtained by the introduction of a new range searching method, the k-vector technique (Refs. [1, 2]), whose original presentation is hereafter summarized.

Let y be an n-long data vector ($n \gg 1$) and s the same vector but sorted in ascending mode, i.e. $s(i) \le s(i+1),\ i \le (n-1)$. This implies $y(I(i)) = s(i)$, where I is the n-long integer vector associated with the sorting, and $i = 1\text{-}n$. In particular we have $y_{\min} = \min_i y(i) = s(1)$, and $y_{\max} = \max_i y(i) = s(n)$.

The straight line connecting the two extreme points $(1,\ y_{\min})$ and $(n,\ y_{\max})$ has, on average, $E_0 = n/(n-1)$ elements for each $d = (y_{\max} - y_{\min})/(n-1)$ step. However, let us consider a slightly steeper line which connects the point $(1,\ y_{\min} - \xi)$ with the point $(n,\ y_{\max} + \xi)$, where $\xi = \varepsilon \max[\,|y_{\min}|,\, |y_{\max}|\,]$ and ε is the relative machine precision ($\varepsilon \approx 2.22 \times 10^{-16}$ for double precision arithmetic). This slightly steeper line (see slanted line in Fig. 1), as it will be clear later, assures $k(1) = 0$, and $k(n) = n$, thus simplifying the code by avoiding many index checks.

The equation of this second line can be written as

$$z(x) = m\ x + q \tag{1}$$

where

$$m = \frac{y_{\max} - y_{\min} + 2\xi}{n-1} \qquad \text{and} \qquad q = y_{\min} - m - \xi$$

Setting $k(1) = 0$, and $k(n) = n$, the integer vector k is then constructed as follows

$$k(i) = j \quad \text{where} \quad s(j) \le z(i) < s(j+1) \tag{2}$$

where the index i ranges from 2 to $(n-1)$. From a practical point of view, $k(i)$ gives the number of the elements $s(j)$ below the value $z(i)$.

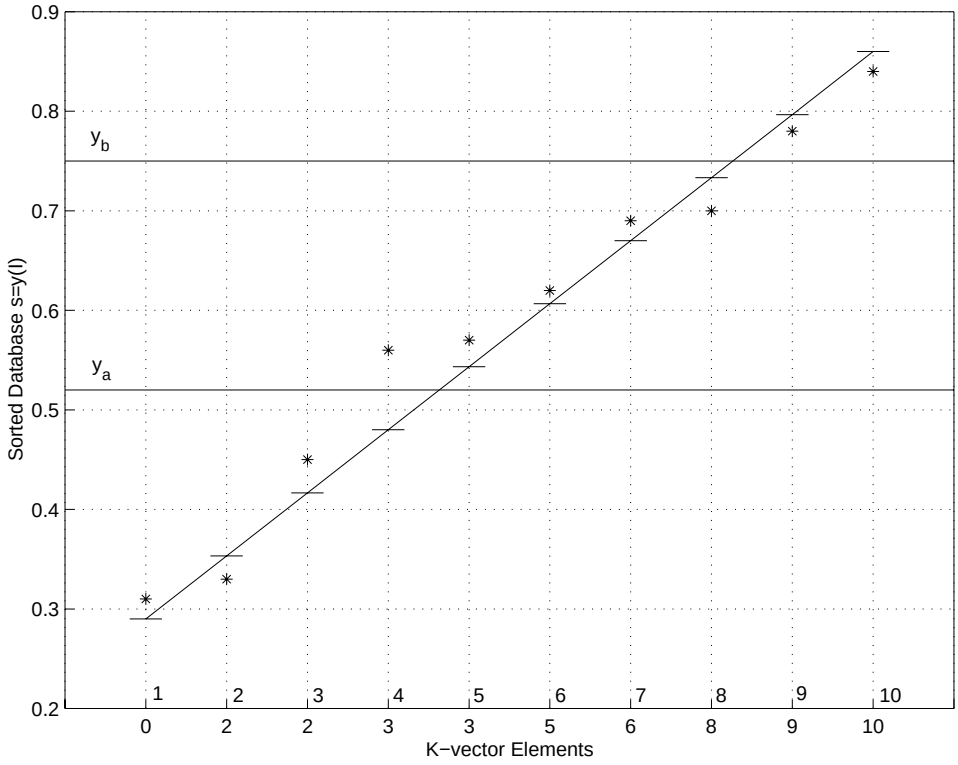

Figure 1: Example of k-vector Construction

For the sake of clarity Fig. 1 shows the construction of the k-vector for a 10-element database. The small horizontal lines are equally spaced at points $z(i)$ and they give the k-vector values 0, 2, 2, 3, 3, 5, 6, 8, 9, 10.

Once the k-vector has been constructed, the evaluation of the two indices identifying, in the s vector, all of the possible data falling within the range $[y_a,\ y_b]$ (see the example in Fig. 1), becomes a straightforward and fast task. In fact, the indices associated with these values in the s vector are simply provided as

$$j_b = \left\lfloor \frac{y_a - q}{m} \right\rfloor \qquad \text{and} \qquad j_t = \left\lceil \frac{y_b - q}{m} \right\rceil \tag{3}$$

where the function $\lfloor x \rfloor$ is the integer number immediately below x, and $\lceil x \rceil$ is the larger integer number next to x. In our example of Fig. 1, we have $j_b = 4$ and $j_t = 9$.

Once the indices j_b and j_t are evaluated, it is possible to compute

$$k_{\text{start}} = k(j_b) + 1 \qquad \text{and} \qquad k_{\text{end}} = k(j_t) \tag{4}$$

The knowledge of k_{start} and k_{end} allows an immediate identification of the searched elements $y(i) \in [y_a,\ y_b]$, which are all the $y(I(k))$ where k ranges from k_{start} to k_{end}.

Note that the k-vector range searching technique, which is based on Eqs. (3-4), does not need the sorted vector s because $s = y(I)$, which requires the use of the sorting integer vector I.

In our example, however, the searched elements should be those identified by the range indices $k_{\text{start}} = 4$ and $k_{\text{end}} = 8$, while the proposed technique outputs $k_{\text{start}} = 4$ and $k_{\text{end}} = 9$. This additional extraneous element comes from the fact that, in average, the k_{start} and the k_{end} elements have 50% each of probability of not belonging to the $[y_a,\ z(j_a+1)]$ and to the $[z(j_b),\ y_b]$ ranges, respectively. Therefore, if E_0 represents the expected number of element in the $[z(j),\ z(j+1)]$ range, the expected number of elements $y(I(k))$ extraneous to any range $[y_a,\ y_b]$ is $2E_0/2 = E_0 = n/(n-1)$.

Hence, if the presence of these E_0 extraneous elements cannot be tolerated, two local searching are needed to eliminate them at the extremes. These local searching are

$$\begin{cases} \text{check if} \quad y(I(k)) < y_a \quad \text{where} \quad k = k_{\text{start}}, \cdots \quad \text{(with positive step)} \\ \text{check if} \quad y(I(k)) > y_b \quad \text{where} \quad k = k_{\text{end}}, \cdots \quad \text{(with negative step)} \end{cases} \tag{5}$$

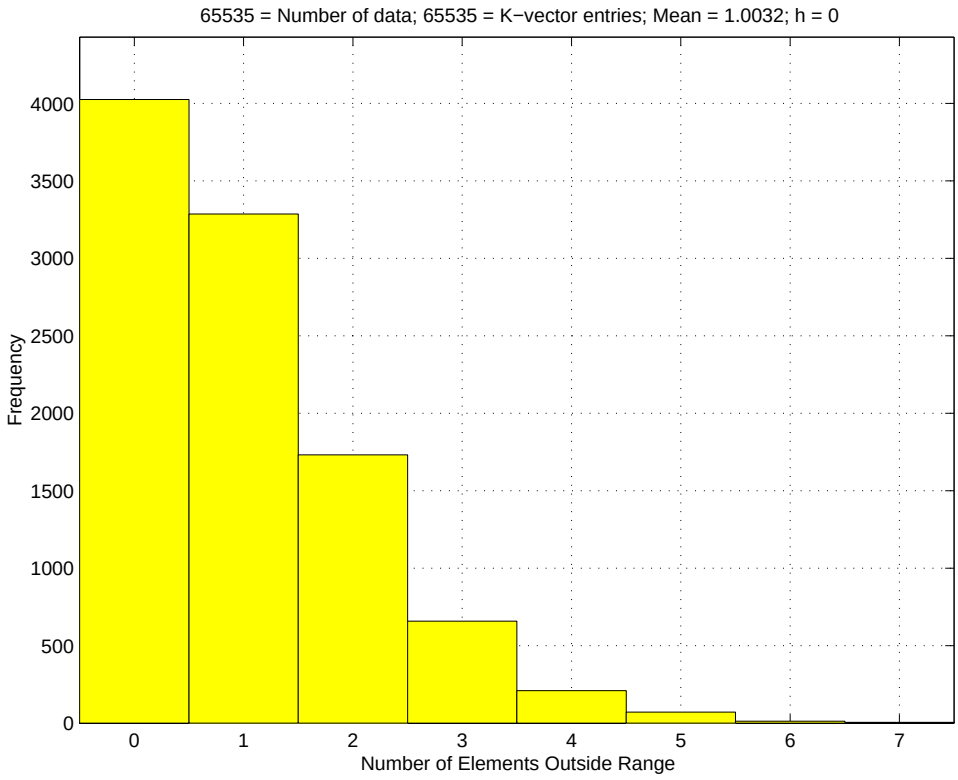

Figure 2: Histogram of E_0

For large databases ($n \gg 1$), since $\lim_{n\to\infty} E_0 = 1$, the number of searched data can be approximated by $n_d = k_{\text{end}} - k_{\text{start}}$, because a number of E_0 extraneous data are expected. Thus, on average, the method provides the solution with only three comparisons[§], a number much less than the $2\log_2 n$ comparisons that the binary searching technique requires. This explains the high speed gain of the k-vector with respect to the BST.

[§]One for each check given in Eq. (5) plus the comparison needed to identify the E_0 extraneous elements.

To demonstrate this, we have randomly created an 65535-long[¶] vectors y and generated the relevant k-vector. In Fig. 2 the histogram of E_0 (the extraneous elements), obtained in 10,000 trials, is given. The mean value obtained is 1.0032, which agrees with the theoretical expected value $E_0 = 65535/(65535-1) = 1.000015$[‖].

Sampled k-vector Technique

The k-vector and the sorting vector I are integer vectors of same length as the real data vector y. Therefore, when y is very large ($n \gg 1$), it may be possible that the storage allocation for k and I become critical. This section shows that it is possible to reduce the length of the k and the I vectors by introducing a technique which samples the k-vector by skipping h elements.

Table 1 depicts the relationship between the indices i and j associated with the element $y(i) = y(I(k(j)))$ for some values of the spacing h.

h	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	i=11	$\cdots$
0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
1	1	-	2	-	3	-	4	-	5	-	6	$\cdots$
2	1	-	-	2	-	-	3	-	-	4	-	$\cdots$
3	1	-	-	-	2	-	-	-	3	-	-	$\cdots$
4	1	-	-	-	-	2	-	-	-	-	3	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Table 1: j Values as a Function of i and the Spacing h for Sampled k-Vectors

This relationship can mathematically be written as

$$i = (j-1)(h+1)+1 \qquad \text{or} \qquad j = \frac{i-1}{h+1} + 1 \tag{6}$$

Thus, the straight line $z(i) = mi + q$ can be written as a function of the index j and the h spacing. In other words, the $z(i)$ function given in Eq. (1), which is the z line associated with $h = 0$, can be substituted by the following more general $z_h(j)$ line

$$z_h(j) = m_h\, j + q_h \qquad \text{where} \left\{ \begin{array}{rcl} m_h & = & m(h+1) \\ q_h & = & q - mh \end{array} \right. \tag{7}$$

and where j ranges from 1 to $\lceil (n+1)/(h+1) \rceil$. Therefore, for a given h value, the associated $z_h(j)$ line can be seen as the $z(i)$ function with increased slope m and decreased q.

[¶]The number of elements is $65535 = 2^{16} - 1$ so that each element index can be stored in 2-bytes.

[‖]The small difference between experimental and theoretical values comes from the fact that, when $k_{\text{end}} = k_{\text{start}} + 1$ occurs, then the number of expected extraneous elements becomes $E_0/2$.

When a sampled k-vector is used ($h > 0$), two *consecutive* indices are identified by $i(j) = (j-1)(h+1)+1$ and $i(j+1) = j(h+1)+1$, which are displaced by a number of $\Delta = i(j+1) - i(j) = h+1$ indices. This implies that the expected number of extraneous elements becomes $E_h = \Delta E_0 = (h+1)n/(n-1)$. For very large database we will have $\lim_{n\to\infty} E_h = h+1$. This means that the expected number of comparisons required to identify these elements is only $E_h + 2 = h+3$.

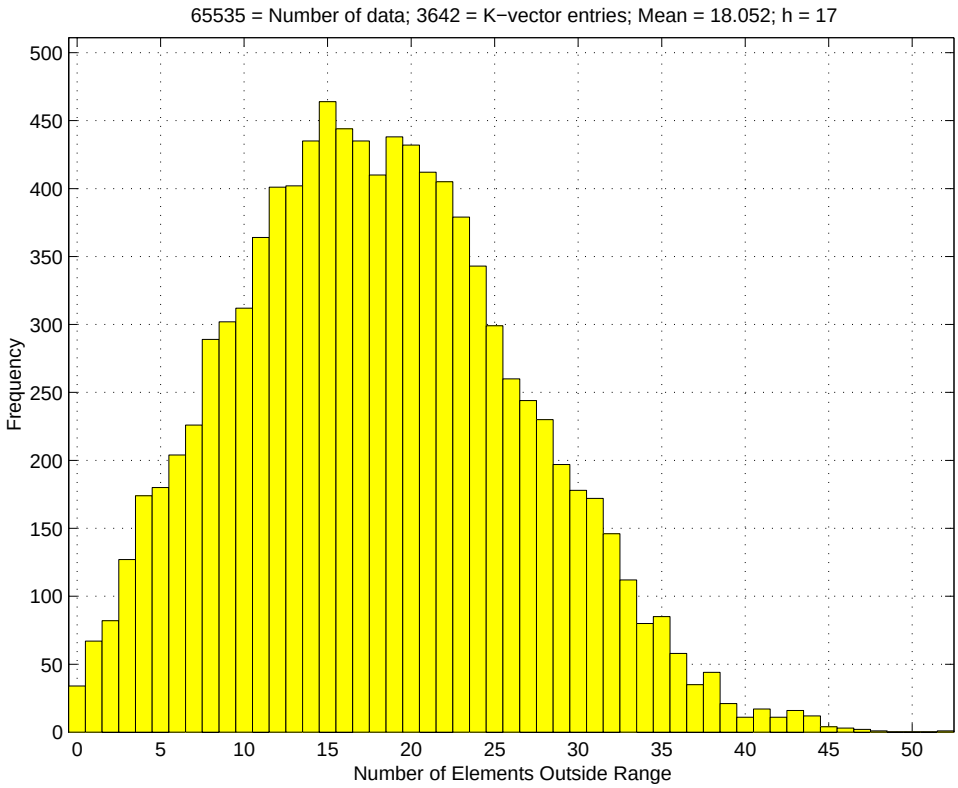

Figure 3: Histogram of E_h with $h = 17$

Figure 3 shows the histogram of E_h for $h = 17$. The mean value obtained (18.052) agrees with the theoretical value of about $E_h = (h+1)n/(n-1) \approx 18.00027$.

It is clear that constructing the k-vector could be time consuming. In Fig. 4, we present the CPU time consumed by a 350Mhz PC-Pentium II (using MATLAB code) to construct k-vectors of various lengths and various h spacings. The time includes sorting the randomly generated y vector **and** constructing the k-vector. Clearly the time increases with the length and decreases with increasing h.

It is important to outline that the construction of the k-vector, that has to be done just once prior the usage, can easily be accomplished via parallel computing approaches. The definition of the optimal size of the sub k-vectors is presently under study by the authors.

Figure 4 shows, for example, that moving from a 15,000 to a 45,000 long database (three times longer) and for $h = 0$, the time required to construct the associated

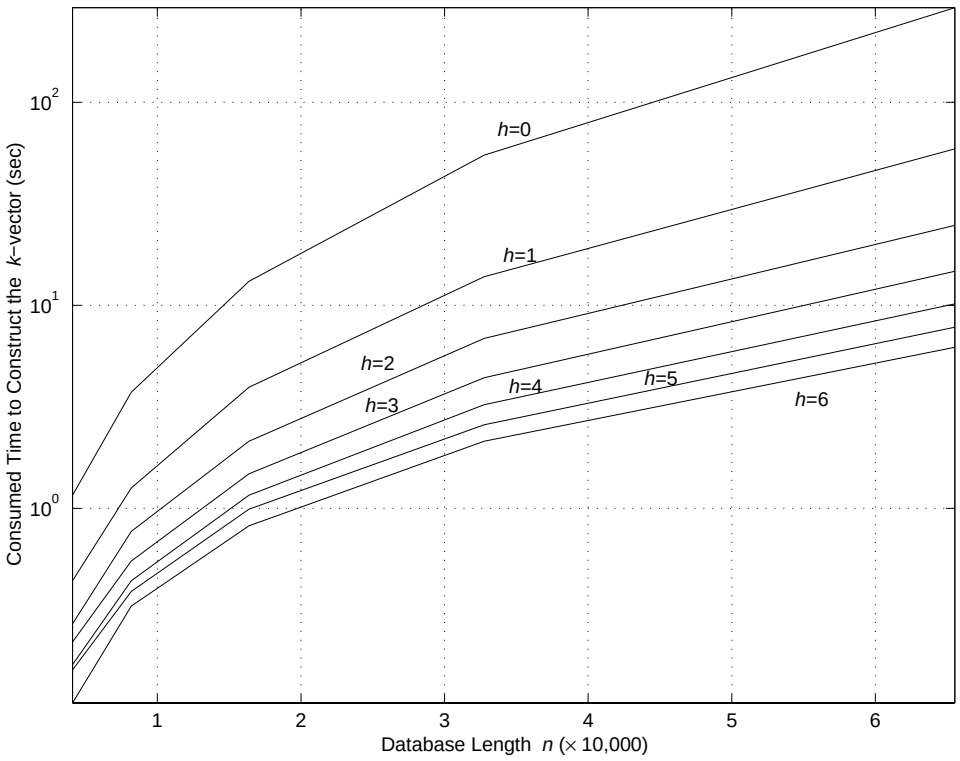

Figure 4: Time to construct the k-vectors of various n and h

k-vectors increases 10 fold. Therefore, it is convenient to split a 45,000 long vector $s = y(I)$ in three pieces and construct three k-vectors instead of only one. As it will be clear later, the time required for the construction and the strategy of when the k-vector should be rebuilt, becomes important for dynamic database, that is when entries have to be added and/or deleted.

Figure 5 shows with marks the CPU times (in sec) consumed to perform 10,000 random searching tests using the k-vector method and for various values of n and h. The continuous line indicates the average values obtained for the corresponding value of h. It is easy to see that the CPU time grows linearly with h and it is independent from the database length n. So one can sacrifice CPU time for storage. For example with $h = 0$, it takes 2.66 seconds, while for $h = 6$ the consumed CPU time for 10,000 tests requires 3.37 seconds. Therefore moving from $h = 0$ to $h = 6$ the saving in terms of storage is about a factor of 6 while the cost rises by about 0.7 second (about 27% increase).

Figure 6 shows, for 50,000 random tests, how the memory required for the I and k-vectors and the associated speed vary as a function of the h spacing. It is clear that the memory decreases as h increases with a faster rate in the beginning. On the other hand the CPU time required increases with h.

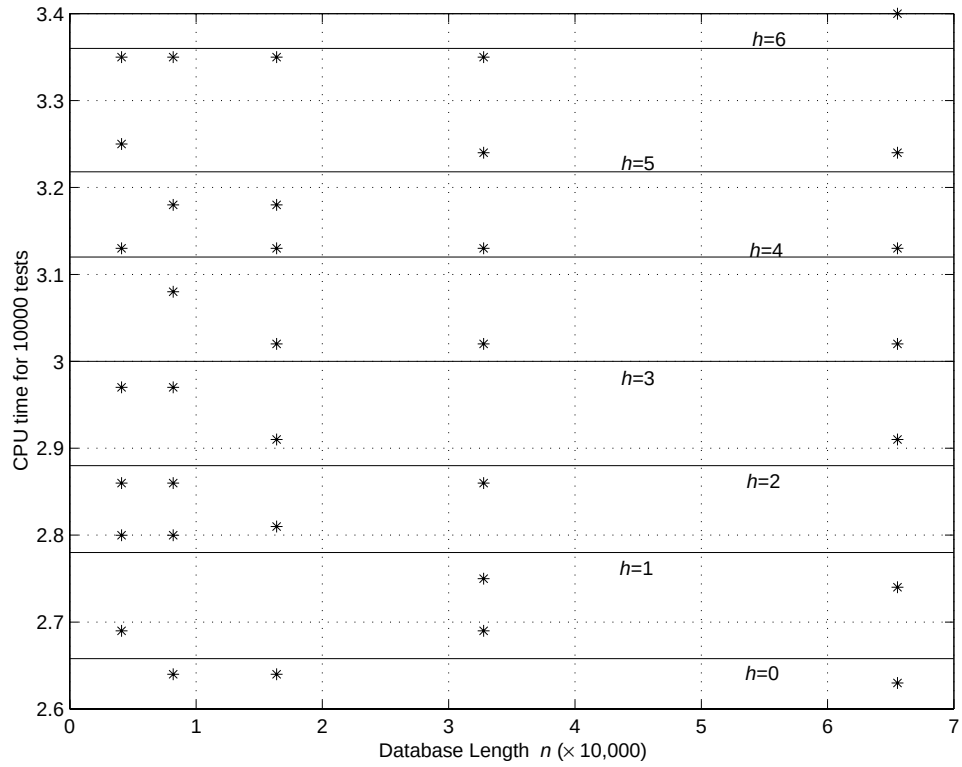

Figure 5: CPU Time for 10,000 searching tests and for various n and h

Speed Comparison with Binary Search Technique

In this section the tests comparing binary search to a sampled k-vector techniques, are discussed. The speed test, whose results obtained using a 500Mhz PC-Pentium III are plotted in Fig. 7, consists to measure the CPU time (in seconds) it takes to use a binary search on arrays of length from $n = 2^{12} - 1$ to $n = 2^{16} - 1$. Two sampled k-vectors (with $h = 0$ and $h = 5$) have been compared vs the binary search. The binary search time to find 10,000 numbers rises from about 18.7 seconds for the $n = 2^{12} - 1$ long database to about 76.7 seconds for those with $n = 2^{12} - 1$ elements. The k-vector technique, on the other hand, requires about the same time (1.43 seconds) with $h = 0$ and about 1.7 seconds for $h = 5$.

Figure 8 shows the time ratios between the binary to the two sampled k-vector considered. A factor of 11 to 13 times more CPU time is required by the binary search for smaller data set ($n = 2^{12} - 1$) for h=5 and h=0, respectively. These gains go up from 42 to 54 times for larger data set (65,535 entries).

Application to a Dynamic Database

In most of the applications requiring range searching, the database changes so that new entries may be inserted and others deleted. Therefore, even though the number

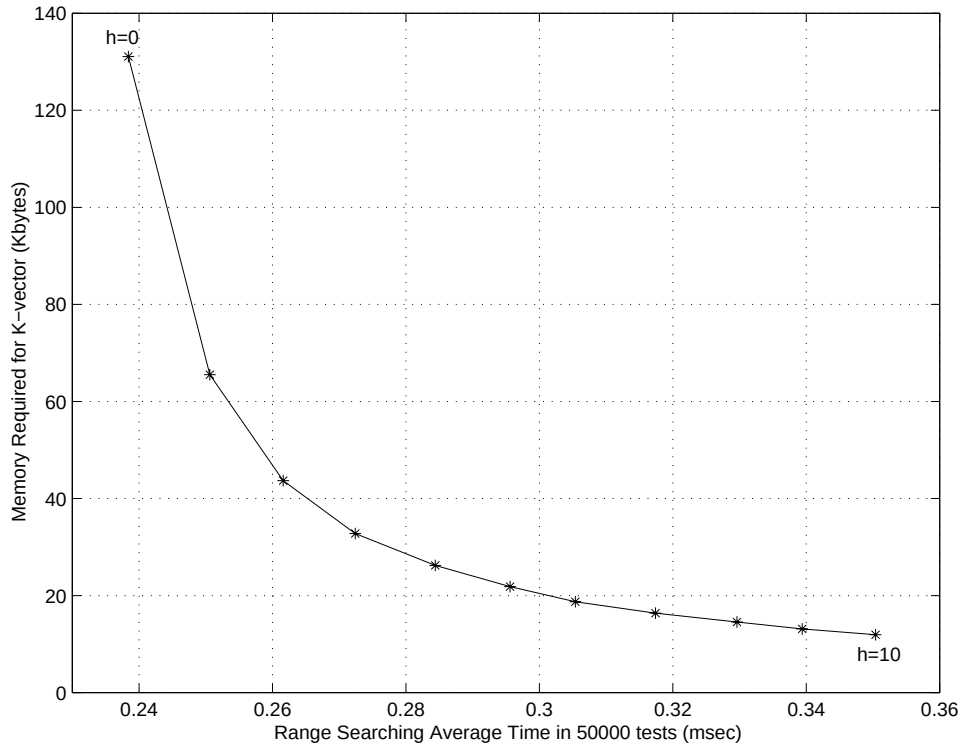

Figure 6: Memory Allocation and Speed

of applications dealing with a static database (as for instance the database listing the interstar angles used in Mortari [1, 2]) is quite large, a range searching algorithm capable of handling this dynamic feature will have a wider range of applications. For instance, the BST allows an easy handling of dynamic database applications and has no limitation on adding and/or deleting elements.

The k-vector technique can also be easily extended and applied to dynamic databases. This is accomplished by modifying the k-vector, without the need to modify the straight line used to build the k-vector elements. This is an important fact since the k-vector structure fully depends on the line considered. Unfortunately, the proposed method has one limitation, that is the incapability of adding elements outside the database range (over the endpoints). When this occurs the k-vector must be re-built.

INSERTING. Let y_{new} be an element to be added into the n-long y database. Using Eq. (4) and Eq. (9), it is possible to evaluate

$$j_b = \left\lceil \frac{(y_{new} - q)/m - 1}{h + 1} + 1 \right\rceil$$

then, Eq. (4) is modified as follows

$$k_{\text{start}} = k(j_b - 1) + 1 \qquad \text{and} \qquad k_{\text{end}} = k(j_b) + 1 \tag{8}$$

then, the y_{new} new entry is inserted between $y_{\ell-1}$ and y_ℓ, where the index ℓ is the first index satisfying $y(\ell) > y_{new}$ where ℓ varies between k_{start} and k_{end}. The k vector is

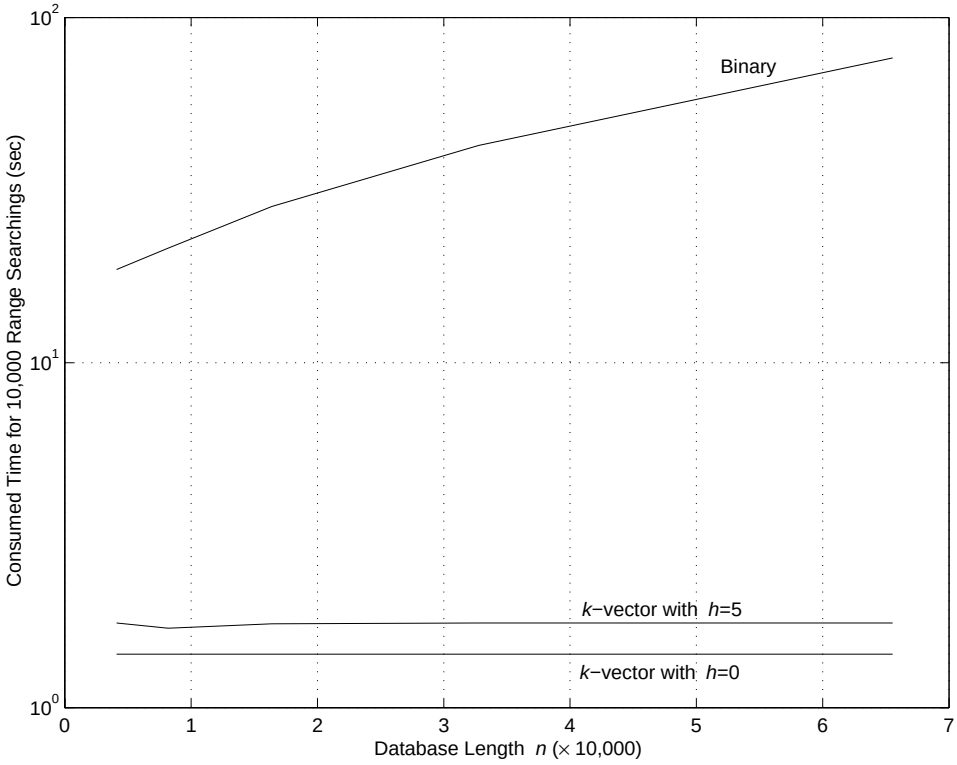

Figure 7: CPU Time Comparison

then updated by adding one to all of the elements with indices greater than or equal to j_b. The remaining procedure is unchanged.

DELETING. Similarly, when the element $y(\ell)$ has to be deleted, then it is necessary to evaluate the index

$$j_b = \left\lceil \frac{(y(\ell) - q)/m - 1}{h + 1} + 1 \right\rceil$$

and then all the elements of the k-vector starting with index j_b will be decreased by one.

In case we need to insert a new first or last point or delete one of the endpoints, the reconstruction of the k-vector is necessary. This is why the minimization of the time required to construct the k-vector is important. This minimization can be accomplished via parallel computing and by splitting the n-long sorted database in smaller sorted databases of *optimal* size.

Two-Level k-vector Technique

Sometimes the data is not linearly distributed, but it is piecewise linear. In such a case, we suggest to have a two level k-vector, consisting of a main k-vector and many sub k-vectors. The higher level will give the endpoints of each linear piece and the lower level is a sub k-vector for each. In Fig. 9, we have an example of data that

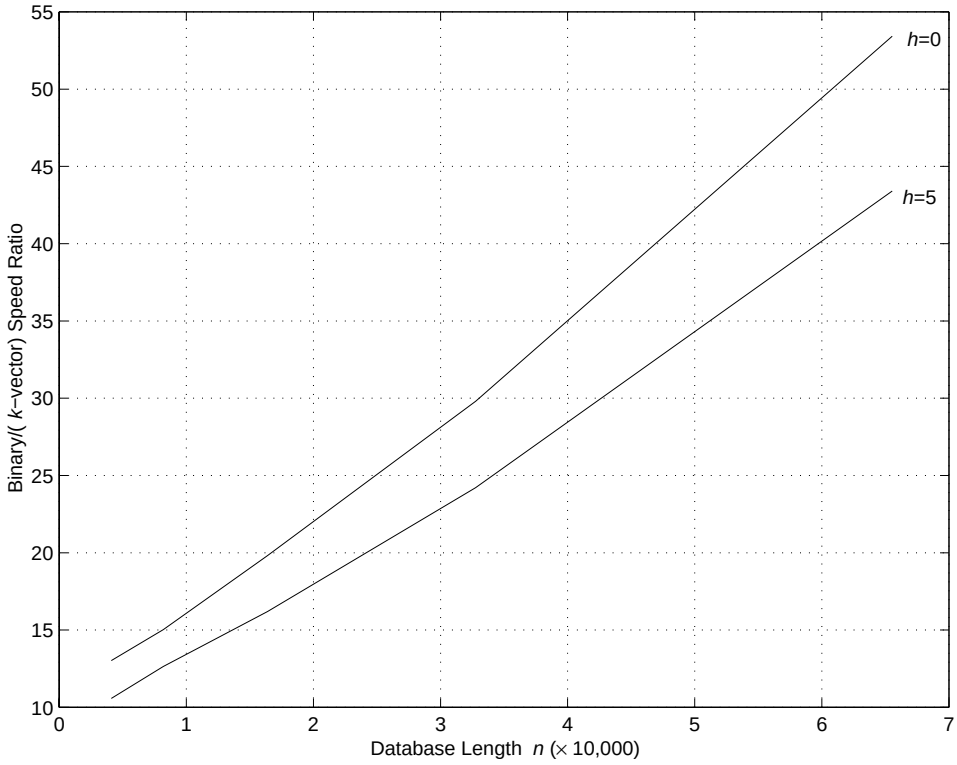

Figure 8: CPU Time Comparison

lends itself to a two-level k-vector. The main level has the four entries 50, 100, 800, and 900, while the second (the sub) level consists of three k-vectors, one for each piece. The first sub k-vector starts with 1, the following sub k-vectors start with one more than the last entry of the previous sub k-vector. Note that the break points are included in two sub k-vectors.

Even though the idea can be generalized to multi-level, the two-level approach seems to be the most general approach. In fact, since the sub k-vector length can be chosen as small as it needs, then the two-level approach includes all the possible data distributions.

A Matlab software, to decide how many linear pieces are required and to create the k-vectors for each, has been developed. The method is based on total least squares (see Golub and van Loan [10]). Finally, just to show one of the possible applications, in the next section a two-level k-vector is applied to evaluate the *arcsin* trigonometric function via lookup table.

Application to function evaluation

A two-level k-vector technique is applied to approximate the value of the trigonometric function $\arcsin x$. The main level k-vector has four entries which have been identified by the total least square method using $\sin x$ for $x \in [0, \pi/2]$ and a tolerance of

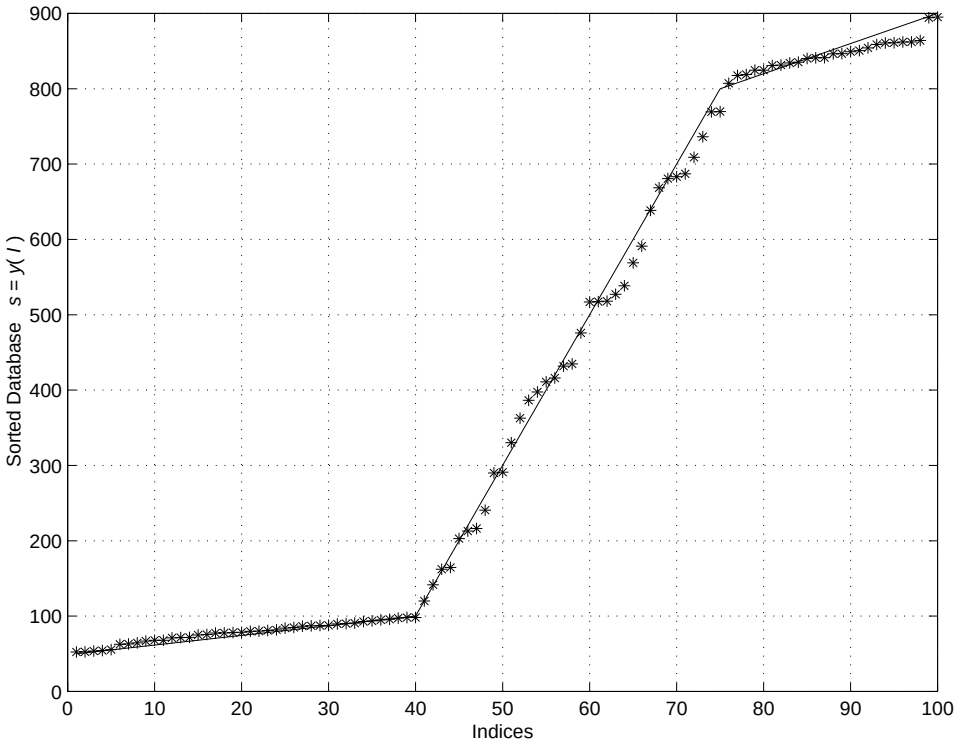

Figure 9: Sorted y vector requiring two level k-vector

$\xi = 0.005$. Clearly this number of entries, which has been chosen just for example, is a function of the tolerance ξ. It is obvious that, for an operative software, the value of ξ has to be chosen much lower.

Figure 10 plots the function $\sin x$ on the interval $[0, \pi/2]$ and the four straight lines approximating it (obtained by total least squares).

The Matlab software developed demonstrates that, using the k-vector technique, it is possible to obtain a value for $\arcsin x$ for any random choice of $x \in [0, \pi/2]$, with approximation which depends on the choice of the tolerance ξ. Just for example, in Fig. 11 one of the test case runs, is given. The stars (marks) are the endpoints of each of the second level of k-vectors.

Conclusions

This paper presents various k-vector range searching techniques. These methods, which accomplish the range search by taking advantage of a vector of integers (the k-vector), allows one to accomplish the search by minimizing the time required. This result is obtained because the k-vector avoids the search phase. It is shown that it is possible to balance the speed attained and the extra memory required for the k-vector allocation by the choice of a step parameter h which samples the k-vector.

A speed comparison of the k-vector method with the "binary search" technique

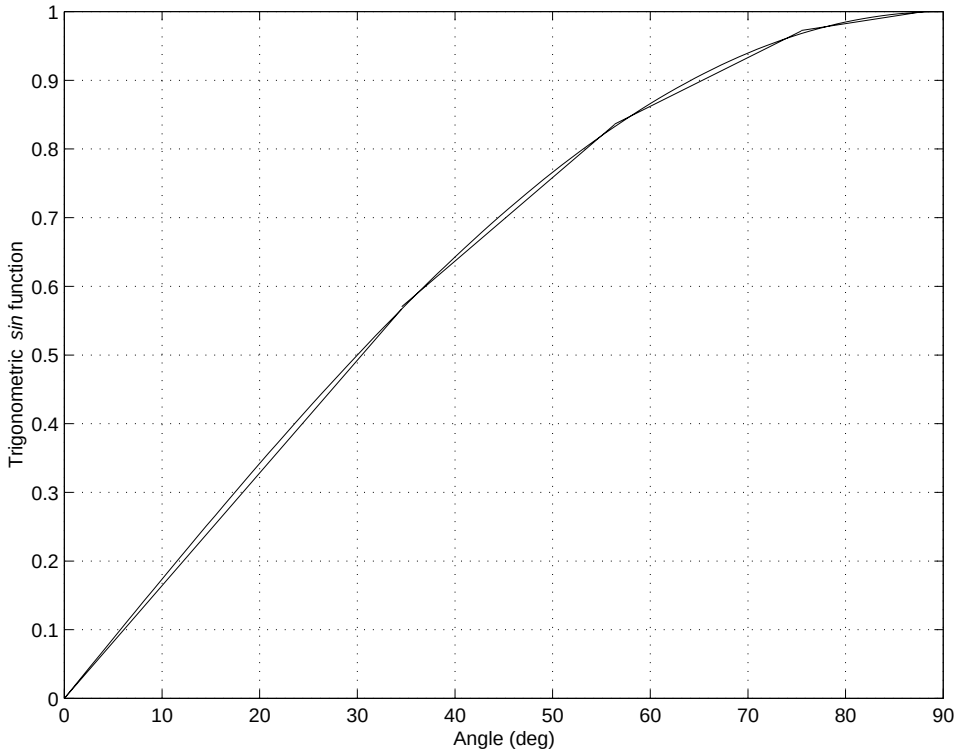

Figure 10: The *sin* Function and Four Linear Pieces

demonstrates that the proposed technique presents a speed gain rate which ranges from 10 to more than 50 times, depending on the database length n and the chosen sampling parameter h.

It is possible to adapt the proposed methods with dynamic database, for which entries can be deleted or inserted.

Finally, the general two-level k-vector technique for piecewise linear data distributions, is presented. This technique, which is applied to evaluation of the trigonometric function *arcsin* via a lookup table, can be used to any data distribution.

The application of the method for multi-dimensional data and the algorithm to optimally define the sub-level k-vectors, are under study by the authors.

The Space Shuttle flight 107, whose lift off is expected on January 2001, will test the StarNav experiment, a new star tracker, whose star pattern identification process will be accomplished, in real time and for the *lost-in-space* general case, by the k-vector technique.

Acknowledgements

This modest work is dedicated to the memory of Carlo Arduini, deeply mist scientist, mentor, and friend.

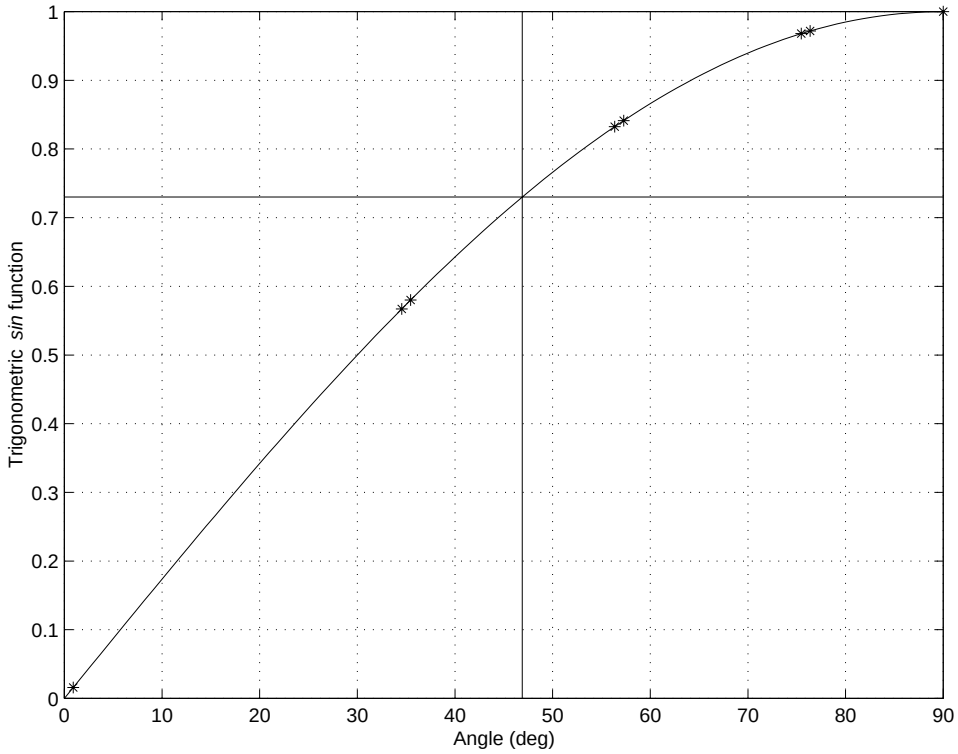

Figure 11: Using k Vector to Find a Function Value

References

[1] Mortari, D. "A Fast On-Board Autonomous Attitude Determination System based on a new Star-ID Technique for a Wide FOV Star Tracker," Advances in the Astronautical Sciences, Vol. 93, Pt. II, pp. 893-903. Paper 96-158 of the Sixth Annual AIAA/AAS Space Flight Mechanics Meeting, Austin, TX, Feb. 11-15, 1996.

[2] Mortari, D. "Search-Less Algorithm for Star Pattern Recognition," *Journal of the Astronautical Sciences*, Vol. 45, No. 2, April-June 1997, pp. 179-194.

[3] Mortari, D. "SP-Search: A New Algorithm for Star Pattern Recognition," Advances in the Astronautical Sciences, Vol. 102, Pt. II, pp. 1165-1174. Paper AAS 99-181 of the 9th Annual AAS/AIAA Space Flight Mechanics Meeting, Breckenridge, CO, Feb. 7-10, 1999.

[4] Mortari, D., and Junkins, L.J. "SP-Search Star Pattern Recognition for Multiple Fields of View Star Trackers," Advances in the Astronautical Sciences, to appear. Paper 99-437 of the AAS/AIAA Astrodynamics Specialist Conference, Girdwood, AK, August 15-19, 1999.

[5] Mortari, D., Pollock, T.C., and Junkins, J.L. "Towards the Most Accurate Attitude Determination System Using Star Trackers," Advances in the Astronau-

tical Sciences, Vol. 99, Pt. II, pp. 839-850. Paper AAS 98-159 of the 8th Annual AIAA/AAS Space Flight Mechanics Meeting, Monterey, CA, Feb. 9-11, 1998.

[6] Mortari, D., and Angelucci, M. "Star Pattern Recognition and Mirror Assembly Misalignment for DIGISTAR II and III Star Sensors," Advances in the Astronautical Sciences, Vol. 102, Pt. II, pp. 1175-1184. Paper AAS 99-182 of the 9th Annual AAS/AIAA Space Flight Mechanics Meeting, Breckenridge, CO, Feb. 7-10, 1999.

[7] Angelucci, M. "Sensori Stellari a Campi di Vista Multipli: Identificazione Stellare e Disallineamento," Thesys on Aerospace Engineering, University of Rome, July 14, 1999.

[8] Ju, G., Kim, Y.H., Pollock, C.T., Junkins, L.J., Juang, N.J., and Mortari, D. "Lost-In-Space: A Star Pattern Recognition and Attitude Estimation Approach for the Case of No A Priori Attitude Information," Paper AAS 00-004 of the 2000 AAS Guidance & Control Conference, Breckenridge, CO, Feb. 2-6, 2000.

[9] Bentley, L.J., and Sedgewick, R. "Fast Algorithms for Sorting and Searching Strings," preprint.

[10] Golub, G. H., and van Loan, C. F., "An Analysis of the Total Least Squares Problem," SIAM J. Numer. Anal., Vol, 17, 1980, 883-893.