SANDIA REPORT

SAND2003-2643
Unlimited Release
Printed July 2003

Algorithms and Analysis for Underwater Vehicle Plume Tracing

Raymond H. Byrne, Steven E. Eskridge, John E. Hurtado and Elizabeth L. Savage

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND2003-2643
Unlimited Release
Printed July 2003

ALGORITHMS AND ANALYSIS FOR UNDERWATER VEHICLE PLUME TRACING

Raymond H. Byrne and Steven E. Eskridge
Sandia National Laboratories
MS 0501, PO Box 5800
Albuquerque, NM 87185-0501
email: rhbyrne@sandia.gov

John E. Hurtado and Elizabeth L. Savage
Department of Aerospace Engineering
Texas A&M University
College Station, TX 77843-3141
email: hurtado@aero.tamu.edu

Abstract

The goal of this research was to develop and demonstrate cooperative 3-D plume tracing algorithms for miniature autonomous underwater vehicles. Applications for this technology include Lost Asset and Survivor Location Systems (L-SALS) and Ship-in-Port Patrol and Protection (SP3). This research was a joint effort that included Nekton Research, LLC, Sandia National Laboratories, and Texas A&M University. Nekton Research developed the miniature autonomous underwater vehicles while Sandia and Texas A&M developed the 3-D plume tracing algorithms. This report describes the plume tracing algorithm and presents test results from successful underwater testing with pseudo-plume sources.

This work was funded by the Defense Advanced Research Projects Agency (DARPA) under contract DE-AC04-94AL85000

Contents

List of Figures

1 Introduction

The goal of this research was to develop and demonstrate cooperative 3-D plume tracing algorithms for miniature autonomous underwater vehicles. Applications for this technology include Lost Asset and Survivor Location Systems (L-SALS) and Ship-in-Port Patrol and Protection (SP3). This research was a joint effort that included Nekton Research, LLC, Sandia National Laboratories, and Texas A&M University. Nekton Research developed the miniature autonomous underwater vehicles while Sandia developed the 3-D plume tracing algorithms. The 3-D plume tracing algorithms are based on a 2-D plume tracing algorithm developed at Sandia for miniature mobile robots [1, 2]. Johnny Hurtado at Texas A&M was involved with the original 2-D work while at Sandia and helped with the 3-D algorithm development. Additional papers describing various aspects of the algorithm have been submitted to several journals [3, 4, 5].

The 2-D plume tracing algorithm was originally designed for locating chemical, nuclear, or biological plumes using teams of miniature mobile robots. The algorithms were successfully demonstrated in 1999 on teams of miniature mobile robots. The robots employed temperature sensors to seek out a temperature source generated with dry ice. A picture of the mobile robots appears in Figure 1. The goal of this project was to develop a 3-D plume tracing algorithm based on the successful 2-D algorithm and then demonstrate the 3-D algorithm on Nekton's autonomous underwater vehicle. A picture of Nekton's autonomous underwater vehicle appears in Figure 2. The 3-D plume tracing algorithms were successfully demonstrated in the Spring of 2003 using pseudo-plume sources. The pseudo-plume was a quadratic function $F(x, y, z) \approx K_1/(x^2+y^2+z^2+K_2)$ that realistically models many $1/r^2$ plume sources.

The plume tracing algorithm is based on a quadratic estimation of the plume data. The robots then use this approximation to calculate a search direction. Each robot, or agent, has a table of plume data which consists of (x, y, z, f) data, where f is the plume amplitude sampled at (x, y, z). The table consists of data gathered by the robot as well as information received from other robots. Each agent may have a drastically different table of data depending on the type of communications (local or global) and the location of the agent. There is no centralized control. Each robot independently calculates a search direction based on its table of sensor data.

A description of the 3-D plume tracing algorithm appears in the next section.

2 3-D Plume Tracing Algorithm

The central idea behind our cooperative localization method is to represent the true function with a local quadratic approximation. The control law for the i^{th} robot is then

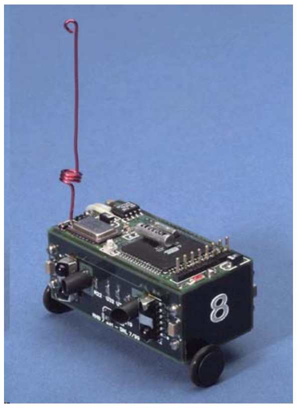


Figure 1: Miniature Robotic Vehicle

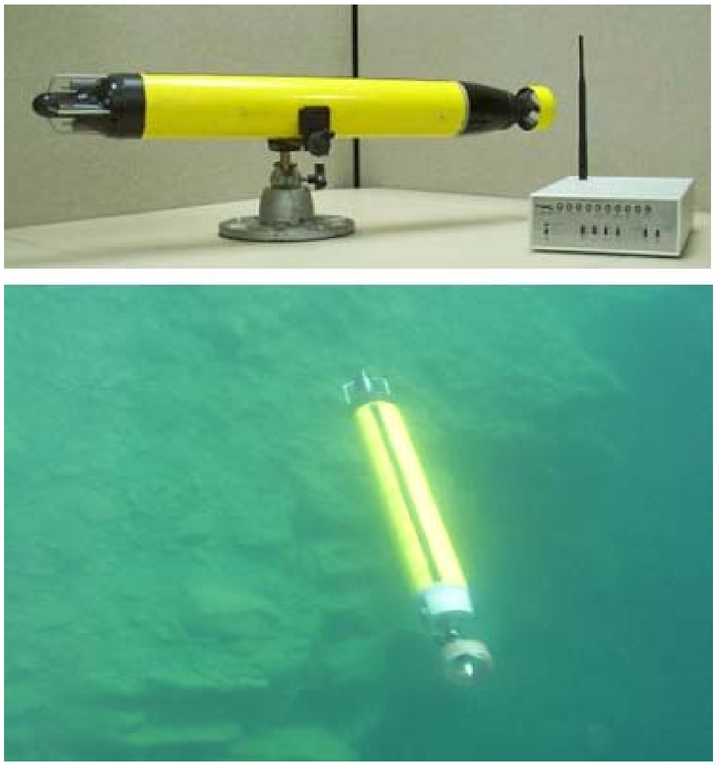


Figure 2: Nekton Research, LLC Miniature Autonomous Underwater Vehicle

based on the quadratic approximation.

In 3-dimensional space, a local quadratic approximation to the true function for the i^{th} robot reads

$$\begin{aligned}F(x,y,z) \approx\ & a_0 + a_1(x-x_i) + a_2(y-y_i) + a_3(z-z_i) \\ & + a_4(x-x_i)(y-y_i) + a_5(x-x_i)(z-z_i) + a_6(y-y_i)(z-z_i) \\ & + \frac{1}{2}a_7(x-x_i)^2 + \frac{1}{2}a_8(y-y_i)^2 + \frac{1}{2}a_9(z-z_i)^2.\end{aligned}$$

In this equation, $F(x,y,z)$ is the true function value as a function of (x,y,z), the triple (x_i,y_i,z_i) represents the location of the i^{th} robot, and a_j are unknown coefficients that define the quadratic approximation. Notice that the unknown coefficients a_j appear linearly in the quadratic approximation. This allows us to write an equation of the above type for every neighboring robot and assemble the equations as a linear system of equations of the form

$$Da = f$$

Here, D is the coefficient matrix, a is a vector of unknown coefficients, and f is the data vector. More specifically, considering the j^{th} neighboring robot, we have

$$\begin{aligned}F_j = F(x_j,y_j,z_j) \approx\ & a_0 + a_1(x_j-x_i) + a_2(y_j-y_i) + a_3(z_j-z_i) \\ & + a_4(x_j-x_i)(y_j-y_i) + a_5(x_j-x_i)(z_j-z_i) + a_6(y_j-y_i)(z_j-z_i) \\ & + \frac{1}{2}a_7(x_j-x_i)^2 + \frac{1}{2}a_8(y_j-y_i)^2 + \frac{1}{2}a_9(z_j-z_i)^2.\end{aligned}$$

So f becomes a column vector with entries $\{F_1, F_2, \ldots, F_p\}$, where p is the number of neighboring robots. The vector a is a column vector with entries $\{a_0, a_1, \ldots, a_9\}$. And the matrix D, dimensioned $p \times 10$, has the relative robot positions as coefficients for each of the p equations. That is, the j^{th} row of the matrix D reads

$$\begin{aligned}D(j,:) = \{ & 1,\ (x_j-x_i),\ (y_j-y_i),\ (z_j-z_i),\ (x_j-x_i)(y_j-y_i), \\ & (x_j-x_i)(z_j-z_i),\ (y_j-y_i)(z_j-z_i),\ (x_j-x_i)^2,\ (y_j-y_i)^2,\ (z_j-z_i)^2\}.\end{aligned}$$

Notice that because there are ten unknown coefficients we require at least ten neighboring robots, or ten valid sensor readings which may come from a single robot provided the robot is able to utilize memory to store previous readings. The unknown coefficients a are determined from the linear system of equations $Da = f$ using the principle of least squares, which a robot's micro controller can perform using QR factorization. This will be discussed further subsequently.

Once the coefficients that define the quadratic model are computed, the update for the i^{th} robot is determined from equating the gradient of the function approximation to zero. This leads to the position update equation

$$\left\{ \begin{array}{ccc} x_{i,\text{new}} & y_{i,\text{new}} & z_{i,\text{new}} \end{array} \right\}^T = \left\{ \begin{array}{ccc} x_{i,\text{old}} & y_{i,\text{old}} & z_{i,\text{old}} \end{array} \right\}^T + \frac{1}{\Delta} \begin{bmatrix} a_8a_9 - a_6^2 & -a_4a_9 + a_5a_6 & a_4a_6 - a_5a_8 \\ -a_4a_9 + a_5a_6 & a_7a_9 - a_5^2 & -a_7a_6 + a_4a_5 \\ a_4a_6 - a_5a_8 & -a_7a_6 + a_4a_5 & a_7a_8 - a_4^2 \end{bmatrix} \left\{ \begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array} \right\},$$

where

$$\Delta = a_7a_8a_9 - a_7a_6^2 - a_4^2a_9 + 2a_4a_5a_6 - a_5^2a_8.$$

2.1 Determining the Unknown Coefficients for Ill-Conditioned Cases - Method I

In the previous section we arrived at the linear equation $Da = f$ which must be solved to determine the unknown coefficients a. The coefficients a define the local quadratic approximation of the true function for the i^{th} robot. A straightforward QR factorization can be used if the matrix D is well-conditioned, which will happen provided that the j^{th} positions are sufficiently separated from the i^{th} positions. Unfortunately, the matrix D can become ill-conditioned under innocent situations. For example, if the j^{th} and i^{th} positions all occur in a constant z plane, then there is no z dependence in the data and the matrix D will have a rank less than 10. This situation should be identified before the coefficients a are determined from $Da = f$. One method to identified this situation is to perform a singular value decomposition on the matrix D. Indeed, the *Total Least Squares* (TLS) method can be used to clearly identify which columns of D are dependent.

The TLS method considers errors in a model (matrix D) as well as errors in the data vector f. Traditional least squares (LS) only considers errors in the data vector f. The basic TLS solution is given by

$$a_{\text{TLS}} = (D^TD - \sigma_{11}^2 I)^{-1} D^T f \tag{1}$$

where σ_{11} is the 11th singular value of the augmented matrix $[D; f]$, which has dimension $p \times 11$. When D is rank-deficient, the nonzero entries of v_{11}, which represents the 11th right singular vector (or the 11th column of the right singular matrix) determine which columns of D are dependent. These columns of D, and appropriate coefficients in a, can be discarded to produce a reduce model of the plume. For example, if the j^{th} and i^{th} positions all occur in a constant z plane, then the 10th entry in v_{11} is nonzero, which corresponds to the z^2 coefficient a_9. Consequently, there is no quadratic z dependence and columns [4,6,7,10] from D are removed, whereas columns [1,2,3,5,8,9] are retained. The reduced plume model is then a quadratic function of x and y only.

The reduced problem is written as $D_r a_r = f$. This system of linear equations can now be solved for the unknown coefficients a_r using a QR factorization, or a singular value decomposition (SVD) method, or a TLS method on a robot's micro controller. The unknown coefficients a_r are then used to compute the position update for the i^{th} robot.

2.2 Determining the Unknown Coefficients for Ill-Conditioned Cases - Method II

Another approach to handling ill-conditioned data is to check the condition of the D matrix when solving the matrix equation

$$Da = f \tag{2}$$

where D is the coefficient matrix, a is the vector of unkown coefficients, and f is the data vector. If the solution is calculated using a singular value decomposition, which is the most robust numerical approach, the condition number $C(D)$ is easily calculated and defined by

$$C(D) = \frac{\sigma_1}{\sigma_N} \tag{3}$$

where σ_1 is the largest singular value and σ_N is the smallest singular value. As the condition number grows larger, the quality of the solution becomes questionable. The most common numerical problem encountered during field testing was a lack of diversity in depth data for the 3-D case. Rather than looking at the depth data, the 3-D algorithm was modified to return the condition number as well as the search direction. By monitoring the condition number, the robot can switch to the 2-D algorithm if the data does not have sufficent depth diversity. Another advantage of monitoring the condition number is that it will also catch other numerical problems (e.g. 3-D data in a plane). The condition number was also returned for the 2-D algorithm to catch potential numerical problems in the 2-D case (e.g. data in a straight line). This approach was implemented on the vehicles to handle numerical problems that can be encountered with typical sensor data.

The next section describes the light plume model.

3 Modeling of Light Data

In our work, we specifically considered underwater vehicles tasked with localizing a light source. We assumed that the light intensity varies from a source as the function $G(x, y, z) = k/R^2$, where $R^2 = (x-x_s)^2+(y-y_s)^2+(z-z_s)^2$, k is some constant, and the triple (x_s, y_s, z_s) is the unknown source location. $G(x, y, z)$ is the value that a vehicle sensor oriented directly toward the source would measure. Because the cooperative localization method is based on a local quadratic approximation, we inverted the sensor reading $F(x, y, z) = 1/G(x, y, z)$ and approximated the inverted reading F by a local quadratic approximation. The steps to determining a robot position update, as outlined in the previous section, were carried out using the inverted sensor reading F.

4 Standard MATLAB Simulation Code

The 3-D localization algorithm was programmed in MATLAB. All MATLAB m-files developed for this project appear in Appendix A. The main simulation filename is `Local3D.m`. Figure 3 shows a typical result from this simulation code. In this example, ten robots were placed in a volume. The source was located at the position $(x_s, y_s, z_s) = (0, 0, 0)$. The upper left plot shows the movement of the ten robots in 3-dimensional space; the upper right plot shows movement in the (x, y) plane; and the bottom plots show movement in the (y, z) and (x, z) planes. The plot shows that after 25 update steps, six of the ten robots have converged upon the source. The simulation was halted after 25 steps because, if not, then *all* ten robots would converge to the solution and the leading matrix D would become singular. In practice, this type of singularity condition is avoided because two robots can not physically occupy the same location.

Simulation Comments:

1. The results are obtained by typing `[x,y,z]=Local3D('r2',25)` at the MATLAB command prompt.
2. The output vectors of this function are the new x, y, z positions of all ten robots.
3. The first input of this function `'r2'` is the filename that provides a robot's sensor reading depending on its position.
4. The second input of this function `25` indicates the number of update steps to be performed.
5. There are MATLAB files that support `Local3D.m`. These include `getneighbors.m`, `getnewpos.m`, and `getupdate.m`.
6. The files to run this simulation are found in the folder `SimuA`

5 Sensor Measurement Dependence on Vehicle Orientation

The measurement that a light sensor records is truly dependent on a vehicle's orientation. This was ignored in the previous discussion and simulation. If the vehicle pitch

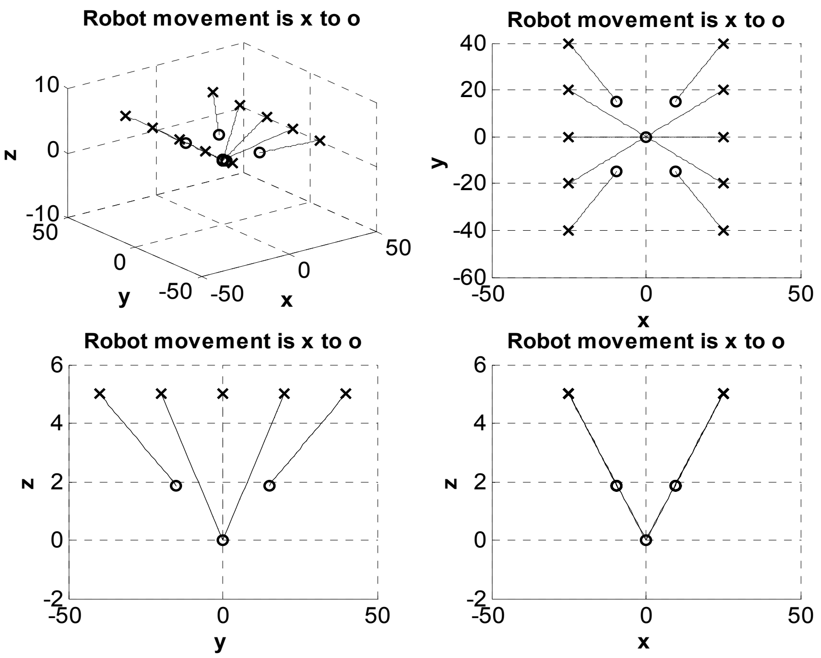


Figure 3: Typical result from `Local3D.m`. The source is located at $(x_s, y_s, z_s) = (0, 0, 0)$. Ten robots cooperate to localize the source. Six of the ten robots have converged upon the source after 25 update steps.

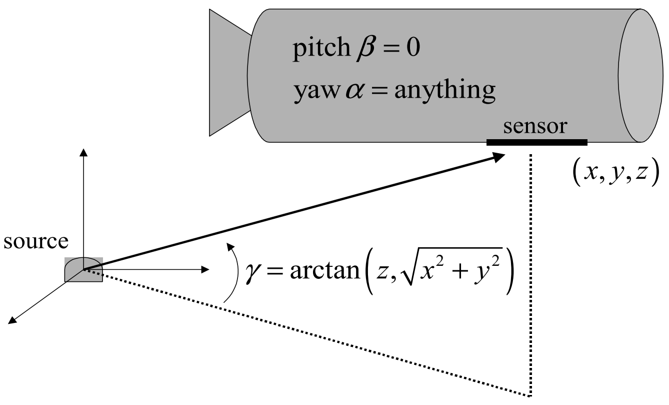


Figure 4: This illustration shows the dependence of a light sensor measurement on vehicle orientation.

angle β is held to equal zero, then the sensor reading is given by $G(x, y, z) = k|\sin\gamma|/R^2$ (see Figure 4). This means that the sensor reading is truly a *vector* quantity, which is different from our previous temperature localization studies that focused on *scalar* quantities only.

To modify the sensor reading so that it appears as a scalar quantity, we note that if a vehicle is close to the source, then $\sin\gamma \approx 1$. This motivated us to perform simulations to study the following:

> Suppose the dependence on vehicle orientation was ignored and the sensor reading was treated as $G = k/R^2$ even though the true sensor reading is given by $G = k|\sin\gamma|/R^2$. Then, how far away can a vehicle be from the source so that the algorithm is not too affected by this modification? Restated, how far away can a vehicle be from an unknown source location such that the localization method based on $F(x, y, z) = 1/G(x, y, z)$, where $G = k/R^2$, still works well even though the true readings are expressed by $G = k|\sin\gamma|/R^2$?

Our simulations showed that the vehicles could be up to 30 to 40 meters away and the localization method still works well. The consequence is that the number of update steps that the robots need to localize the source increases because of the modification.

An example is shown in Figure 5. The first plot shows the initial positions of the ten robots in the (x, y) plane. Some of the vehicles begin more than 30 meters away from the source. In the other three plots, the black traces are the simulation results using $G = k/R^2$ as the sensor readings (i.e., not accounting for vehicle orientation) and using the algorithm based on this. The red traces are the simulation results using $G = k|\sin\gamma|/R^2$ as the sensor readings (i.e., accounting for vehicle orientation) but using the algorithm based on $G = k/R^2$. The black traces show the vehicles immediately heading directly toward the unknown source location; The red traces do not show this. The red traces do show, however, that after the first few position updates, the vehicles indeed begin heading toward the unknown source location. The black traces are shown for 12 position updates, whereas the red traces are shown for 20 position updates. As previously mentioned, the primary consequence of the modification is that the number of update steps that the robots need to localize the source increases.

Simulation Comments:

1. The modified results, wherein the dependence on vehicle orientation is accounted for in the sensor reading but ignored in the localization algorithm, are obtained by typing `[x,y,z]=Local3D('r2depend',20)` at the MATLAB command prompt. Notice that the algorithm is the same (`Local3D.m`).
2. The input file, however, has changed to `'r2depend'`. This input file accounts for the sensor dependence on vehicle orientation.
3. The file `'r2depend.m'` is in the folder `SimuA`

6 Sensor Measurement Noise

We performed numerical experiments to determine the influence of sensor noise on the localization method. We considered a pool measuring $50 \times 50 \times 10$ meters and a team of ten robots. We considered two different robot initial position configurations. In the first configuration, the robots essentially surround the unknown source location. This configuration is referred to as the Surround Configuration. In the second configuration, the robots begin in one corner of an area. This configuration is referred to as the Corner Configuration. The two different initial position configurations are shown in Figure 6.

For the numerical simulations, the robots were given initial conditions like those above but with an additional random initial position perturbation of up to one meter. The sensor noise level was set to equal a percent of the true value. As an example, for a 1 percent noise level, a sensor reading would fall between $(100 \pm 1)\%$ of the actual (true) value. The cooperative algorithm was set to run for 50 position updates, with a maximum distance per update of 0.5 meters in the x, y, or z directions.

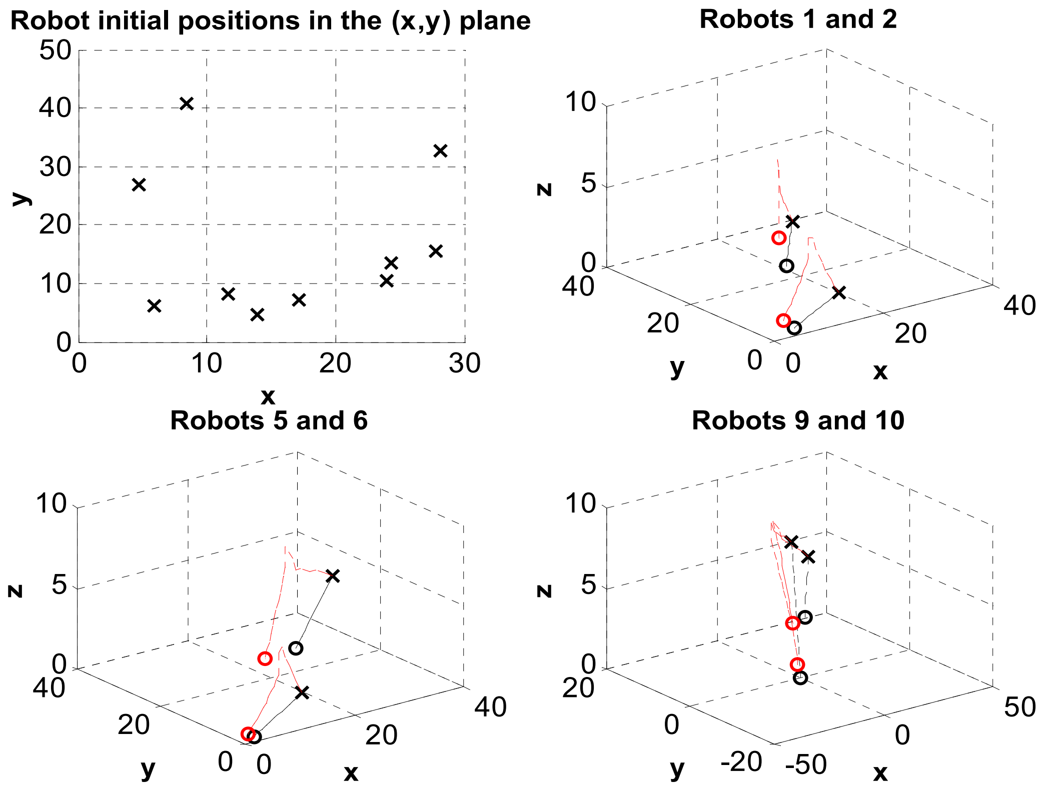
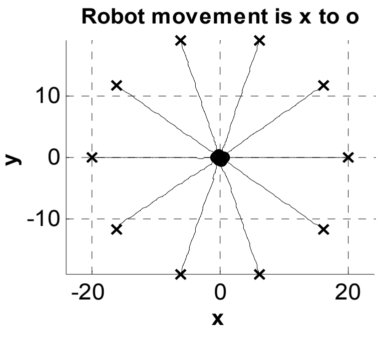


Figure 5: Position updates of ten robots. Black traces correspond to sensor readings and algorithm based on $G = k/R^2$. The red traces correspond to sensor readings based on $G = k|\sin\gamma|/R^2$, but an algorithm based on $G = k/R^2$. This essentially removes (ignores) the dependence on vehicle orientation.



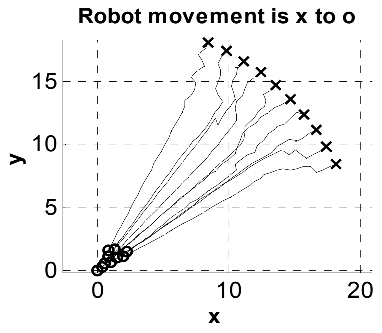


Figure 6: Two initial position configurations of ten robots for sensor noise study (Surround Configuration Left and Corner Configuration right). The initial positions are marked with the **x**. The initial depth (not shown) was set at -2 meters from the surface, or 8 meters from the bottom.

The source was located at the (x, y, z) position $(0, 0, 0)$. To study the results, we calculated and recorded the distance of the center-of-mass (CM) of the robot team from the light source. We calculated this for many simulations for each initial robot position configuration. The position of the CM from the source location represents the mean distance from the source. We also recorded the closest and furthest robot distances from the source.

The results for the Surround Configuration are listed in Table 1 and a typical result is shown in Figure 7. Incidentally, the robots quit updating their positions after approximately 48 update steps. The reason is that the robots were not allowed to collide with one another, consequently, the robots would get "locked" in a configuration after a robot found the source. A locked configuration is then the final configuration, like that shown in Figure 7. The table indicates that the results (CM distance, closest robot distance, and furthest robot distance) did not depend on sensor noise: the robots found the source and ended in very similar final "locked" configurations.

The results for the Corner Configuration are listed in Table 2 and a typical result is shown in Figure 8. For this case, the robots quit updating their positions after approximately 35 update steps because they achieved a "locked" configuration. The table indicates that the results (CM distance, closest robot distance, and furthest robot distance) are only slightly dependent on sensor noise: the robots found the source and the average distance of the furthest robot is slightly more as more sensor noise is added.

Sensor Noise (%)	Avg Distance of CM from source (m)	Avg Distance of closest robot (m)	Avg Distance of furthest robot (m)
0	0.9744	0.1714	1.7439
5	0.9655	0.0934	1.7365
10	0.9643	0.1028	1.7457

Table 1: Effects of sensor noise with initial Surround Configuration. These are average values of the simulations.

Sensor Noise (%)	Avg Distance of CM from source (m)	Avg Distance of closest robot (m)	Avg Distance of furthest robot (m)
0	1.7435	0.2141	3.0771
5	1.8330	0.0500	3.2518
10	1.8974	0.0361	3.5291

Table 2: Effects of sensor noise with initial Corner Configuration. These are average values of the simulations.

Overall, the results indicate that sensor noise does not greatly influence the ability of

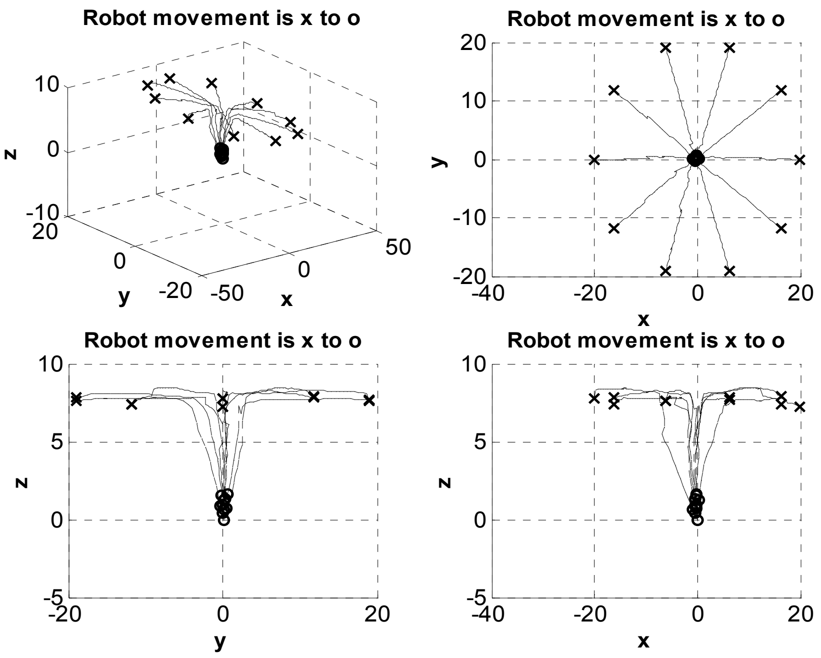


Figure 7: Typical position updates of ten robots in Surround Configuration. (±5% sensor noise added to the true readings.)

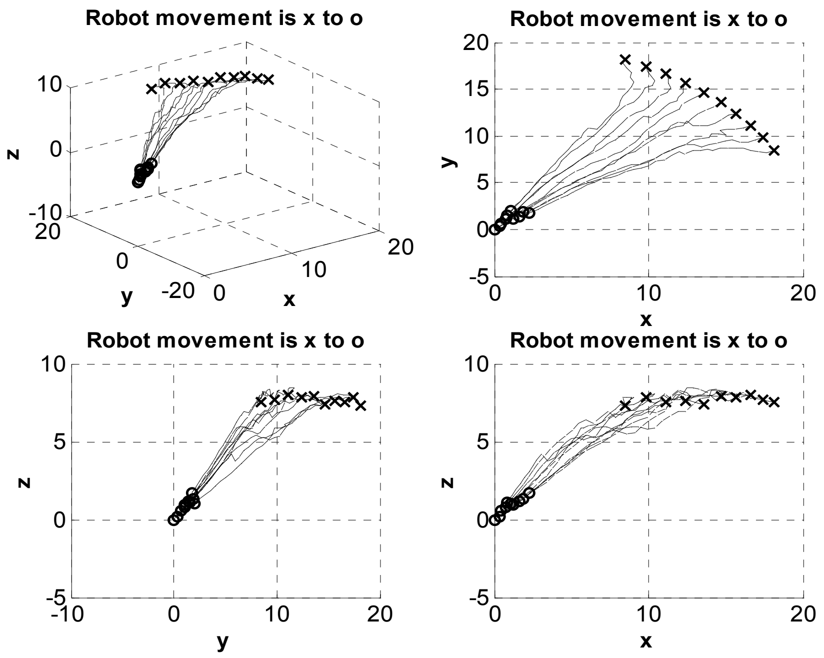


Figure 8: Typical position updates of ten robots in Corner Configuration. (±5% sensor noise added to the true readings.)

the localization method to find the source. The robots did take longer to converge to the source when they started in the Surround Configuration (48 update steps) than when they started in the Corner Configuration (35 update steps). This can be witnessed by the robot trajectories. In the initial Surround Configuration the robots first move in an almost horizontal plane before they begin to dive. In the initial Corner Configuration, the robots begin to dive as they move toward the source.

7 Robot Position Error

We performed numerical experiments to determine the influence of robot position error on the localization method. We considered the same two robot initial position configurations (Surround and Corner).

The robot position error was added to the x and y position only because the z (depth) measurements are typically very accurate. We considered errors ranging from 0 meters in x and y to 1 meter in x and y. The cooperative algorithm was set to run for 80 position updates, with a maximum distance per update of 0.5 meters in the x, y, or z directions.

As before, the source was located at the (x, y, z) position $(0, 0, 0)$ and we calculated and recorded the distance of the center-of-mass (CM) from the source, and the closest and furthest robot distances from the source.

The results for the Surround Configuration are listed in Table 3 and a typical result is shown in Figure 9. The results for the Corner Configuration are listed in Table 4 and a typical result is shown in Figure 10. Overall, the results indicate that the performance of the localization method is dependent on robot position error. This is especially apparent at the 1 meter position error level. We emphasize that it is the performance of the localization method that suffers. That is, from 0 meter error to 0.5 meter error, the average distance of the closest robot indicates that the source has been nearly localized, but the distance of the furthest robot is significant indicating that a robot (or two) is still far away.

8 Navigational Error

We performed numerical experiments to determine the influence of robot navigation error on the localization method. We considered the same two robot initial position

Position Error (m)	Avg Distance of CM from source (m)	Avg Distance of closest robot (m)	Avg Distance of furthest robot (m)
0.0	1.0743	0.0705	2.1043
0.25	2.4574	1.3136	4.1251
0.5	3.5261	1.4422	6.6741
1.0	12.2822	6.1496	20.1739

Table 3: Effects of position noise with initial Surround Configuration. These are average values of the simulations.

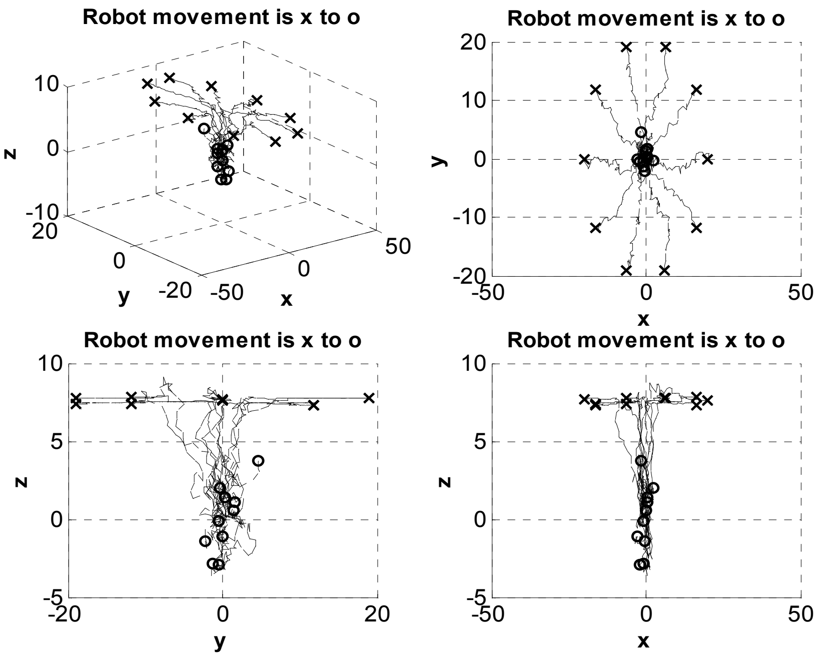


Figure 9: Typical position updates of ten robots in Surround Configuration. (±0.25 meters error added to the true position readings.)

Position Error (m)	Avg Distance of CM from source (m)	Avg Distance of closest robot (m)	Avg Distance of furthest robot (m)
0.0	1.7470	0.0193	3.0423
0.25	2.0529	0.7228	4.0736
0.5	4.5930	1.8751	8.8456
1.0	13.3710	6.2784	25.3038

Table 4: Effects of position error with initial Corner Configuration. These are average values of the simulations.

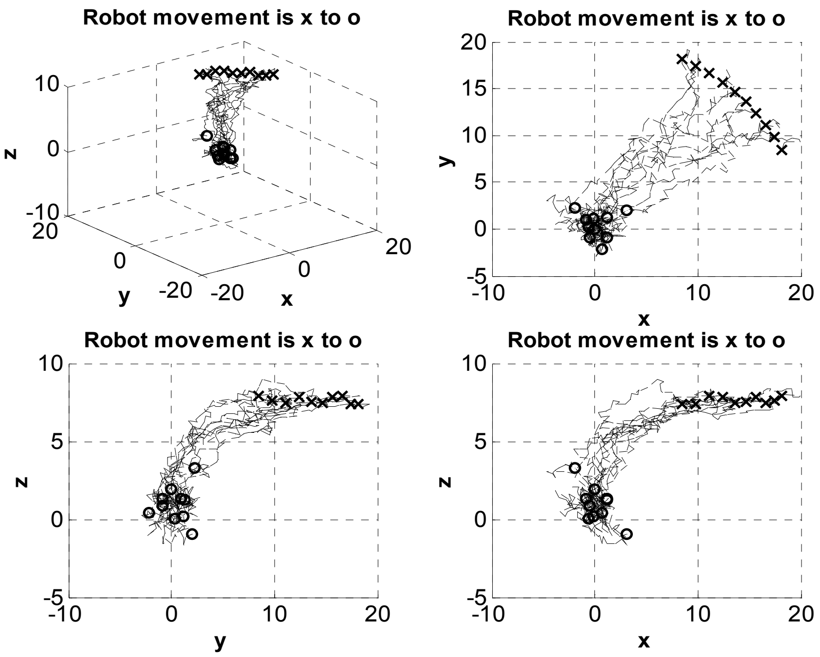


Figure 10: Typical position updates of ten robots in Corner Configuration. (±0.25 meters error added to the true position readings.)

configurations (Surround and Corner).

Heading error can be best visualized by considering a robot commanded to travel in a straight line. The error introduced will cause the robot to deviate from its intended path. The further the robot travels the greater the deviation. A larger error will mean that the robot deviates faster from the path. We have assumed that the system can identify the position of each robot exactly. Because of this assumption, the heading error is harmless. Although the robot will not arrive at the exact location specified by the algorithm update, the position sensors will be able to locate the new (deviated) position and calculate the next update using this information. The path a robot will take to get to the source will be slightly longer than without navigation error.

Figures 11 through 13 show typical robot updates with heading errors ranging from 10° to 30°. The robots begin in a Surround Configuration. The heading error cause the robots to deviate from their intended path, but because the the robots can identify their true positions the robots are still able to locate the source.

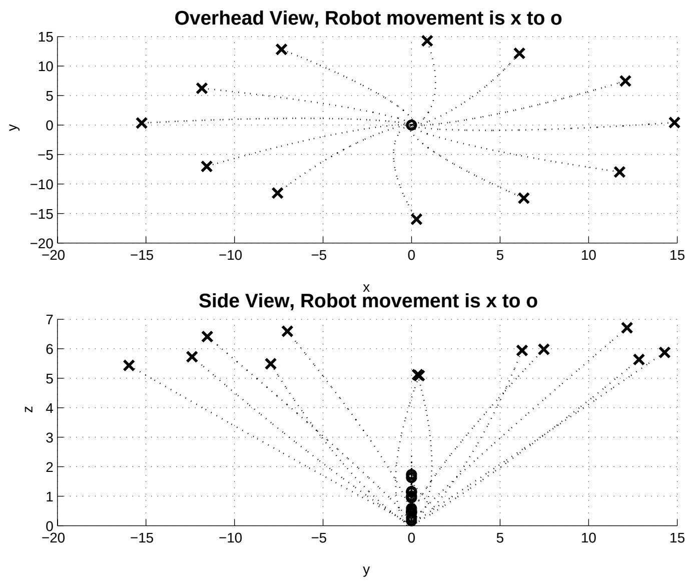


Figure 11: Typical position updates for robots in Surround Configuration with 10° heading error.

Figures 14 through 16 show typical robot updates with heading errors ranging from 10° to 30° when the robots begin in a Corner Configuration. As with the Surround Configuration, the heading error cause the robots to deviate from their intended path, but because the the robots can identify their true positions the robots are still able to locate the source.

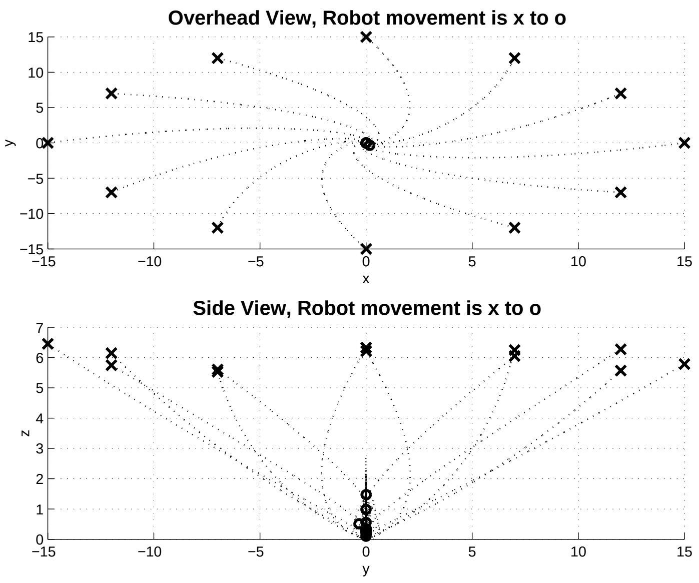


Figure 12: Typical position updates for robots in Surround Configuration with 20° heading error.

9 Software Implementation

The 3-D and 2-D plume tracing algorithms were incorporated as functions in a c-callable libray, library.lib, for the Texas Instruments 671x digital signal processor (DSP) family. The robot stores sensor data in a sorted linked list that is sorted by sensor value. This list includes local sensor data as well as sensor data received from other robots. Every time the algorithm is called, the "best" 20 sensor data points are loaded into a global array that contains $(x, y, z, sensor)$ data. The search algorithm is invoked by calling one of the library functions listed below:

```
double xpUpdate_3D() //3-D search algorithm
double xpUpdate_2D() //2-D search algorithm
```

Both the 2D and 3D algorithms return the condition of the D matrix to check for numerical problems and provide a measure of confidence in the result. The 3D algorithm is usually called first. If the condition number is unacceptable, then the 2D algorithm is called. The 2D algorithm is inherently more robust because the determinant of the 2D system is much larger than the determinant of the 3D system ($det = \prod \lambda_i$ where λ_i are the eigenvalues - the eigenvalues of the plume are usually small and a function of the plume shape so for the same plume the 3D determinant is smaller).

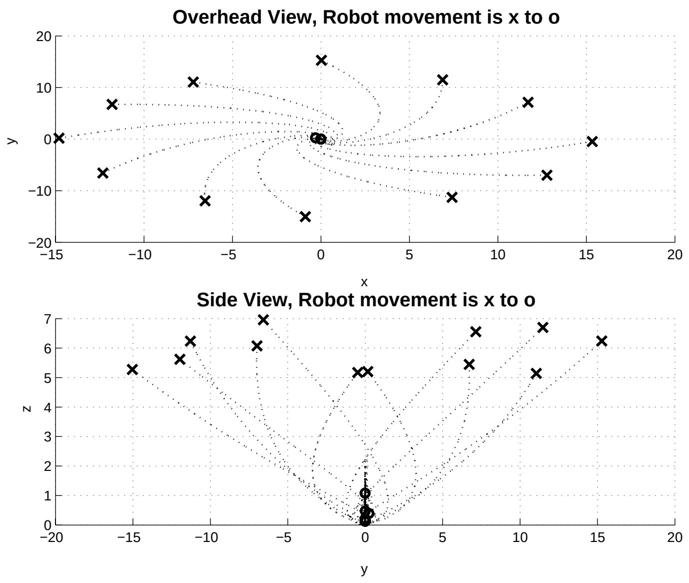


Figure 13: Typical position updates for robots in Surround Configuration with 30° heading error.

The search algorithm returns the search direction via three global variables:

```
xpos_new  //desired x-pos
ypos_new  //desired y-pos
zpos_new  // desired z-pos (depth)
```

The desired search vector is fed into the vehicle control system which steers the vehicle in the proper direction.

10 Test Results

Robot testing was performed by Nekton. All tests were performed with a pseudo plume of the form $F(x, y, z) \approx K_1/(x^2 + y^2 + z^2 + K_2)$. Test data is shown in Figure 17 for a test run with two vehicles in the water. Test results for three vehicles are shown in Figures 18 - 20. For both test cases the robots successfully located the plume source. These results are representative of the test results obtained on numerous test runs. The periodic rise to the surface is intentional - this allows the robots to periodically obtain a GPS position fix and to communicate with the base station. This capability was added to replace the acoustic communications and position location system that Woods Hole Oceanographic Institute was unable to successfully develop. In many respects, the

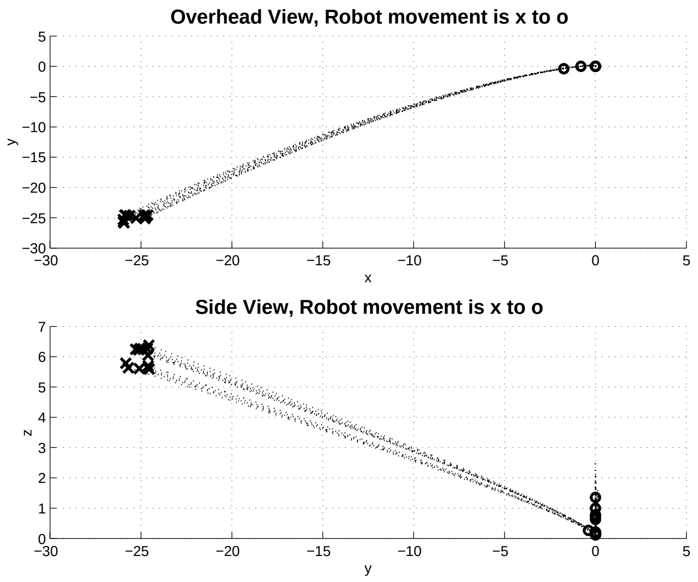


Figure 14: Typical position updates for robots in Corner Configuration with 10° heading error.

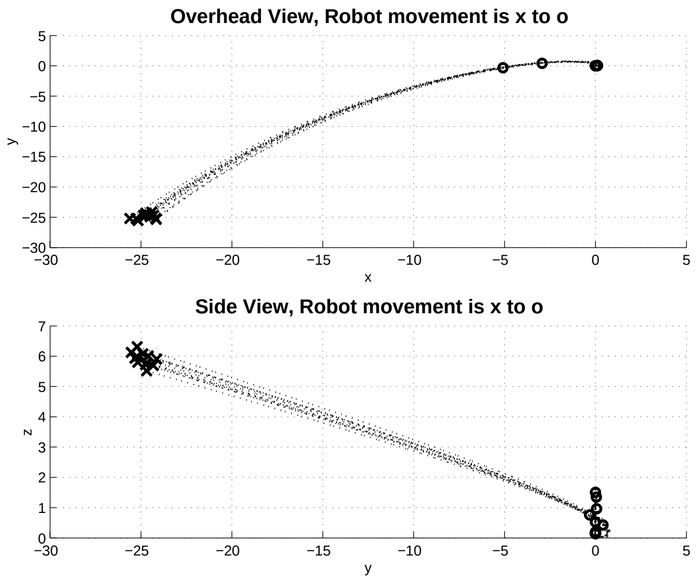


Figure 15: Typical position updates for robots in Corner Configuration with 20° heading error.

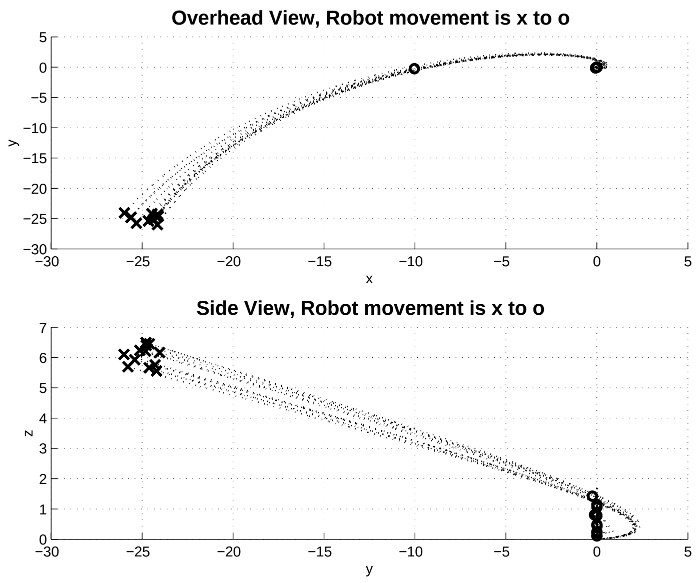


Figure 16: Typical position updates for robots in Corner Configuration with 30° heading error.

GPS-based system is superior to the underwater acoustic system - the GPS system is much more portable and does not required surveyed buoys.

11 Summary

In summary, we successfully developed and demonstrated 3D plume tracing algorithms on a miniature autonomous underwater vehicle. The algorithms were verified with field testing using a pseudo-plume source that represents many $1/r^2$ plumes found in nature. The algorithms were incorporated into a set of c-callable library functions for the Texas Instruments 671x digital signal processor (DSP) family used on the Nekton Ranger vehicle. In addition, we verified the expected performance of the 3D plume tracing algorithm through analysis and simulation.

Using numerical simulations, we found that the source localization algorithms were slightly dependent on sensor noise. The robots were able to locate the source for varying levels of sensor noise, although convergence to the source generally took longer because of the presence of the noise.

Regarding robot position error, the numerical simulations indicate that the performance of the localization method is dependent on robot position error. This was especially apparent for a robot position level of 1 meter. Indeed, it is the performance

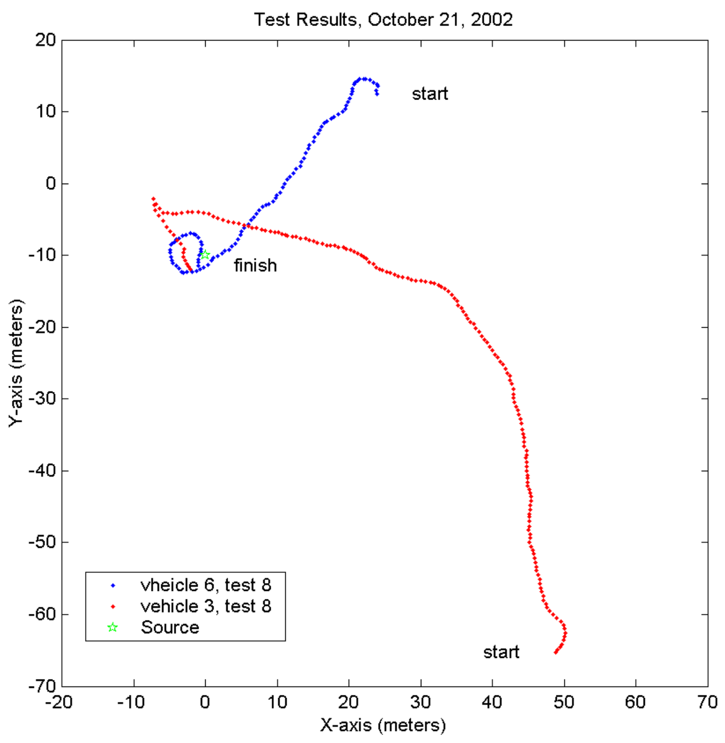


Figure 17: Test Results - Two Vehicles, Pseudo-Plume

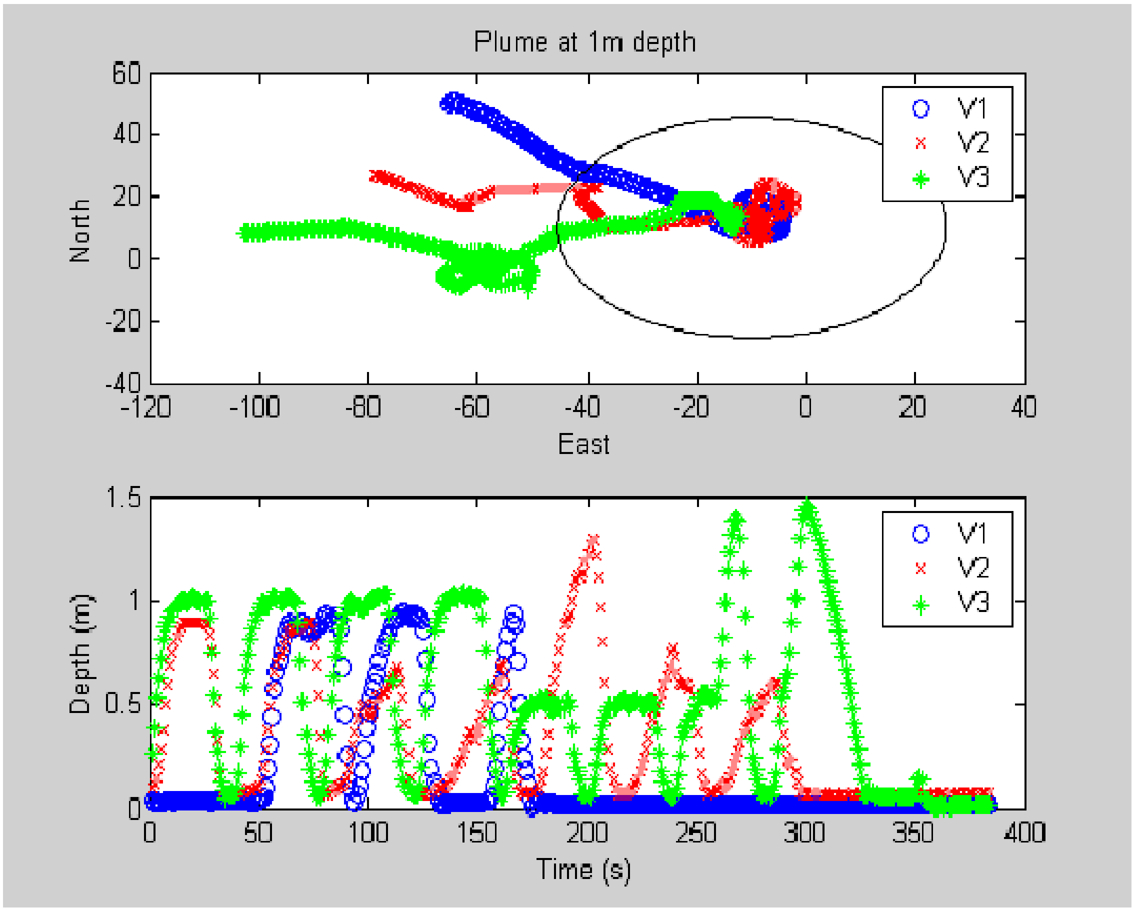


Figure 18: Test Results - Pseudo-plume at depth of 1 meter

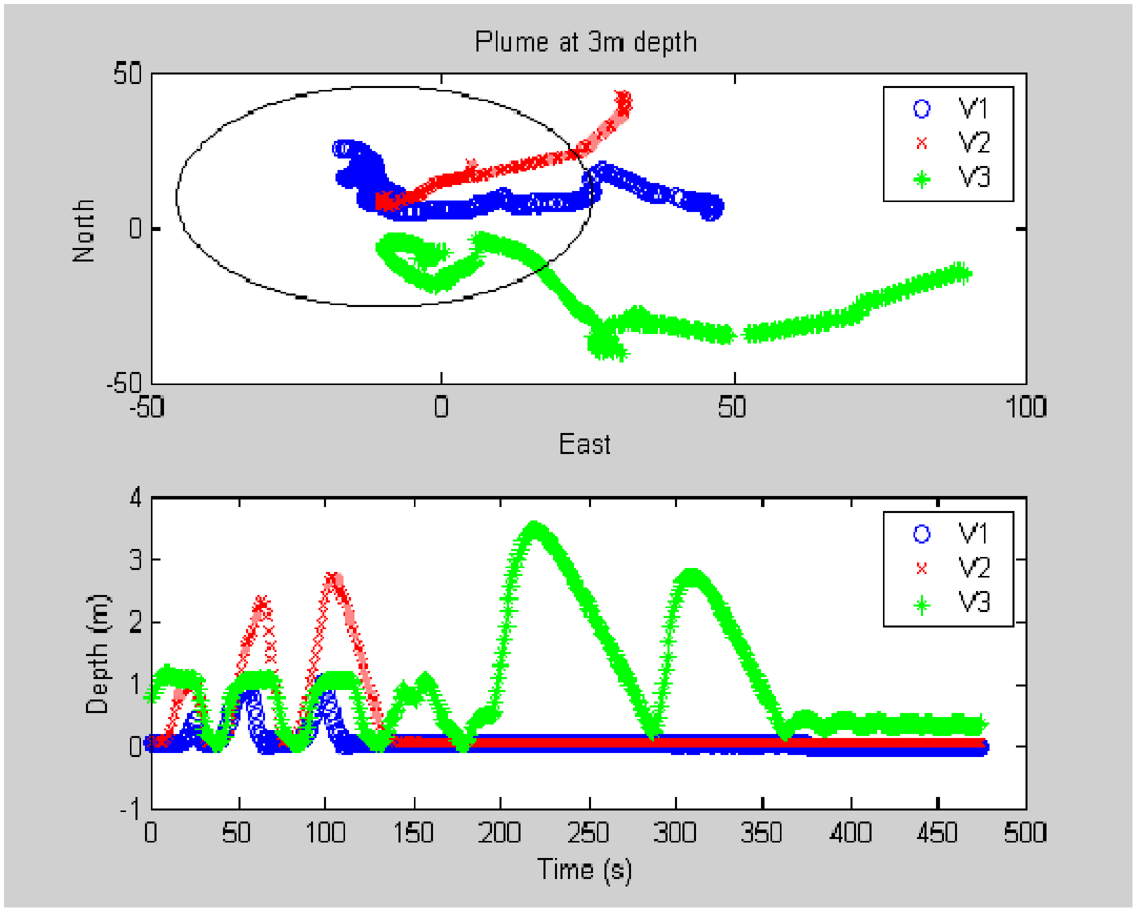


Figure 19: Test Results - Pseudo-plume at depth of 3 meters

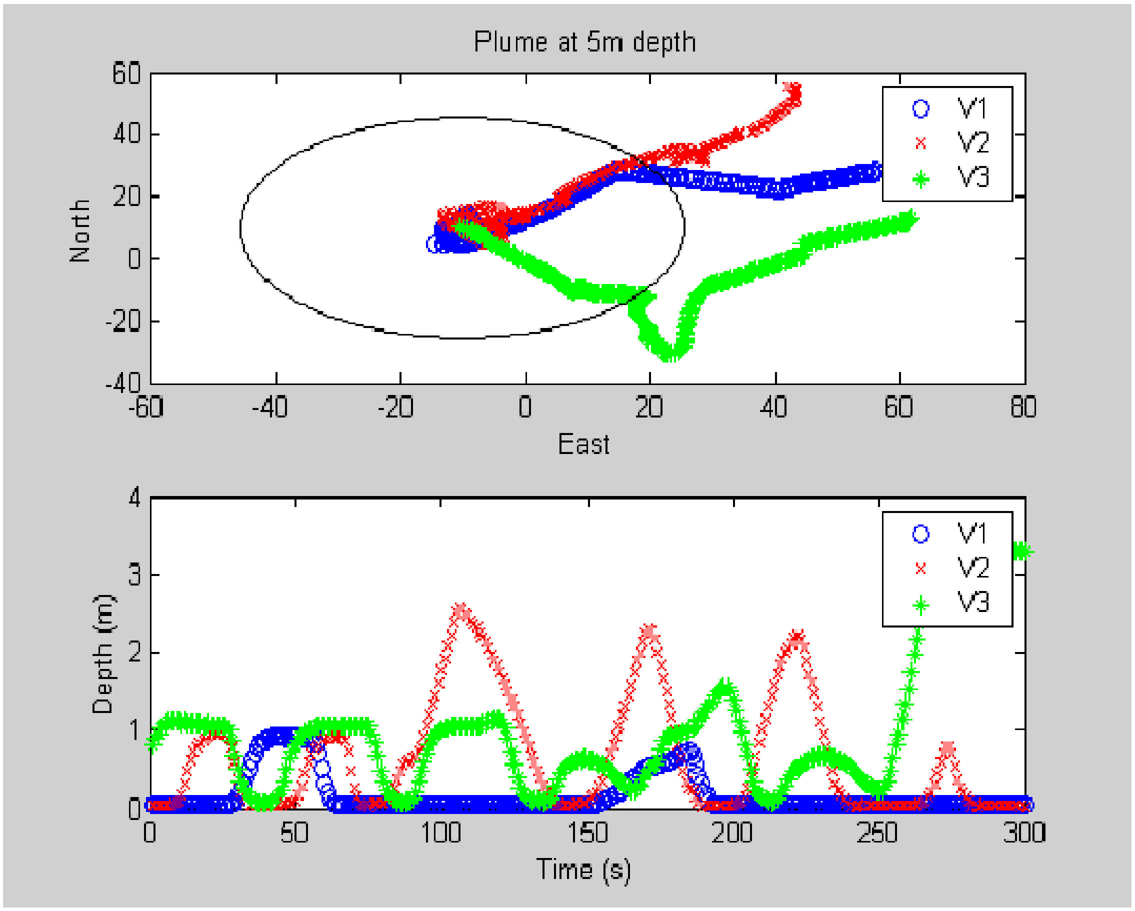


Figure 20: Test Results - Pseudo-plume at depth of 5 meters

of the localization method that suffers, in that the average distance of the closest robot for a typical simulation indicates that the source has been nearly localized, but the distance of the furthest robot is significant indicating that a robot (or two) is still far away.

From numerical simulation, the algorithm was found to be robust with regard to navigation or heading error. Heading error will cause the robot to deviate from its intended path. As long as a robot could correctly identify its position, however, this type of error was not found a problem. The robots were able to localize the source although the eventual path was found to deviate from the original intended path.

12 Acknowledgments

Many people were instrumental in making this project a success. Johnny Hurtado with assistance from Elizabeth Savage developed the 3-D algorithm and performed the simulation studies. Steven Eskridge coded the algorithm for the 671x DSP. Ray Byrne made minor modifications to the algorithm (applied different numerical techniques), completed the DSP coding, and served as project leader and principal investigator. Heather Pinnix and Bryan Schulz at Nekton integrated the Sandia code with the Nekton vehicle and performed the field testing. The Nekton team also included: Brett Hobson, Mathieu Kemp, Jim Meyer, Ryan Moody, Mathew St. Clair, Jeff Bourn, Gordon Caudle, Harriet Crisp, Rick Vosburgh, and Jason Janet. Dr. Elana Ethridge served as the DARPA program manager for this effort.

References

[1] J. E. Hurtado, R. H. Byrne, S. E. Eskridge, and J. J. Harrington, “Miniature mobile robots for plume tracking and source localization research.” Workshop on Mobile Micro-Robots, 2000 IEEE International Conference on Robotics and Automation, April 28, 2000.

[2] R. H. Byrne, D. R. Adkins, S. E. Eskridge, J. J. Harrington, E. J. Heller, and J. E. Hurtado, “Miniature mobile robots for plume tracking and source localization research,” *Journal of Micromechatronics*, vol. 1, no. 3, pp. 253–261, 2002.

[3] J. E. Hurtado and R. D. Robinett, “Convergence of newton’s method via lyapunov analysis.” Submitted to the Journal of Guidance, Control, and Dynamics (October 2002).

[4] J. E. Hurtado and R. D. Robinett, “Control design, stability and performance for robotic collectives.” Submitted to the Journal of Intelligent and Robotic Systems: Theory and Applications (October 2002).

[5] J. E. Hurtado, R. D. Robinett, C. R. Dohrmann, and S. Y. Goldsmith, "Decentralized control for a swarm of vehicles performing source localization." Submitted to the Journal of Intelligent and Robotic Systems: Theory and Applications (October 2002).

A MATLAB M-FILES

A.1 batchsimu.m

```
N=80;
for indx=1:5,
[x,y,z]=local3DSN('r2depend',N);
R=x(N+1,:).^2+y(N+1,:).^2+z(N+1,:).^2;
R=sqrt(R);
CM(indx) = mean(R);
Sm(indx) = min(R);
Lg(indx) = max(R);
[CM;Sm;Lg]
end
```

A.2 getneighbors.m

```
function [iflag,ii]=getneighbors(i,xa,ya,za,scrmin,scrmax)
% find the neighbors near robot i

      iflag=1;
      xd=xa-xa(i); yd=ya-ya(i); zd=za-za(i);
      r=sqrt(xd.*xd + yd.*yd + zd.*zd);
      rcurr=scrmin;
      ii=find(r<scrmin);
      if length(ii)<10
         iflag=0; rcurr=scrmin;
         while iflag==0,
            rcurr=1.5*rcurr;
            ii=find(r<rcurr);
            if length(ii)>=10
               iflag=1;
            end
            if rcurr>scrmax,
               iflag=2;
            end
         end
      end
```

A.3 getnewpos.m

```
function [delx,dely,delz]=getnewpos(i,p,dxmax,dymax,dzmax,xa,ya,za,nxy)
         dx=p(1);  dy=p(2);  dz=p(3);
         sf=max([abs(dx)/dxmax(i),abs(dy)/dymax(i),abs(dz)/dzmax(i)]);
         alpha=min(1,1/sf);

             mx=1;
             delx=alpha*dx;
             dely=alpha*dy;
             delz=alpha*dz;
% return  % turn off the "collision check"

% collision avoidance to remove simulation singularity
         mx=1;
         tooclose=1;
         while tooclose==1;
             tooclose=0;
             delxnew=mx*alpha*dx;
             delynew=mx*alpha*dy;
             delznew=mx*alpha*dz;
             xnew=xa(i)+delxnew;
             ynew=ya(i)+delynew;
             znew=za(i)+delznew;
             xd=xnew-xa; yd=ynew-ya; zd=znew-za;
             r=sqrt(xd.*xd + yd.*yd + zd.*zd);
             ro=.5;
%             ro=.5;
             for L=1:nxy,
                 if L~=i
                     if r(L)<ro,
                         tooclose=1;
                         mx=mx-.1;;
                     end
                 end
             end
         end
         delx=delxnew;
         dely=delynew;
         delz=delznew;
```

A.4 getupdate.m

```
function p=getupdate(ii,i,xa,ya,za,b,nxy)
% flag==1, solve the LS problem
% flag==2, solve the TLS problem

%         xc=(xa(ii) + rangenoise(ii)')';
         xc=(xa(ii))';
         yc=(ya(ii))';
         zc=(za(ii))';
         xcnoise=xc;
```

```
        ycnoise=yc;
        zcnoise=zc;
        xcn=xcnoise-xa(i); ycn=ycnoise-ya(i); zcn=zcnoise-za(i);
        A=zeros(length(ii),10);
        A=[ones(length(ii),1) xcn ycn zcn xcn.*xcn/2 ycn.*ycn/2 zcn.*zcn/2
                     xcn.*ycn xcn.*zcn ycn.*zcn];
        bpart=zeros(length(ii),1);
        bpart(:,1)=b(ii(:),1);

        a=A\bpart(:,1);

        tol=.01;
        H=[a(5) a(8) a(9);
           a(8) a(6) a(10);
           a(9) a(10) a(7)];
        if(min(eig(H)))<tol,
           [v,d]=eig(H);
           H=v*abs(d)*inv(v);
        end
        bp=-[a(2);a(3);a(4)];
        p=H\bp;
```

A.5 input_r2.m

```
% input file
nx=2; ny=5; nz=1; % total number of bugs
nxy=nx*ny*nz;

% close
boun=[-20,20,-20,20,0,10];
bounp=boun;

% far
%boun=[-50,50,-50,50,0,10];
%bounp=boun;

%boun=1.3*[28,22,28,22,1,9];
%bounp=1.3*[-30,30,-30,30,0,10];

aleft=boun(1); aright=boun(2); % left/right boundaries
bleft=boun(3); bright=boun(4); % left/right boundaries
cleft=boun(5); cright=boun(6); % left/right boundaries
apl=bounp(1); apr=bounp(2); % left/right boundaries for plotting
bpl=bounp(3); bpr=bounp(4);
cpl=bounp(5); cpr=bounp(6);

dx=(aright-aleft)/nx; % partition up the grid
dy=(bright-bleft)/ny;
```

```
dz=(cright-cleft)/nz;

dxmax=1; dymax=1; dzmax=1; % max allowable dx,dy step for optimization

dxmax=dxmax*ones(1,nxy);
dymax=dymax*ones(1,nxy);
dzmax=dzmax*ones(1,nxy);

sensornoise=0 * 1/500*(rand(nxy,1)-0.5);
rangenoise= 0 * 1/10*(rand(nxy,1)-0.5);
percentSN = 0;

scrmin = 100;
scrmax = 100;
```

A.6 input_r2depend.m

```
% input file
nx=2; ny=5; nz=1; % total number of bugs
nxy=nx*ny*nz;

% close
boun=[-20,20,-20,20,0,10];
bounp=boun;

% far
%boun=[-50,50,-50,50,0,10];
%bounp=boun;

%boun=1.3*[28,22,28,22,1,9];
%bounp=1.3*[-30,30,-30,30,0,10];

aleft=boun(1); aright=boun(2); % left/right boundaries
bleft=boun(3); bright=boun(4); % left/right boundaries
cleft=boun(5); cright=boun(6); % left/right boundaries
apl=bounp(1); apr=bounp(2); % left/right boundaries for plotting
bpl=bounp(3); bpr=bounp(4);
cpl=bounp(5); cpr=bounp(6);

dx=(aright-aleft)/nx; % partition up the grid
dy=(bright-bleft)/ny;
dz=(cright-cleft)/nz;

dxmax=.5; dymax=.5; dzmax=.5; % max allowable dx,dy step for optimization

dxmax=dxmax*ones(1,nxy);
dymax=dymax*ones(1,nxy);
dzmax=dzmax*ones(1,nxy);
```

```
sensornoise=0 * 1/500*(rand(nxy,1)-0.5);  % not used but keep at zero
rangenoise= 0 * 1/10*(rand(nxy,1)-0.5);   % not used but keep at zero
percentSN = 0;   % sensor noise percent
percentPN = 1;   % position noise in meters

scrmin = 100;
scrmax = 100;
```

A.7 local3D.m

```
function [xLS,yLS,zLS]=dscc3D(fcn,nstep)
% function [xLS,yLS,xTLS,yTLS]=dscc(fcn,cflag,nstep,xa,ya,za)
%
% fcn: character string of function (must also have input fcn name)
% nstep: number of steps (updates) to perform
% xa, ya: if cflag='onn', then starting positions for continuation
%  if cflag='off', then this will be reset
%
%
%---------------------------------------------------------------------------
% 1. Finds which bugs are close.
% 2. Fixes Hessian if not positive definite.
% 3. Robot Avoidance: Takes step length of alpha along search direction.
%    If too close to another robot then backs off until not too close.
% 4. Each updates when ready. Do not wait.

eval(['input_' fcn ]);

cflag='off';

if cflag=='off',
   x=aleft+dx/2:dx:aright-dx/2;
   y=bleft+dy/2:dy:bright-dy/2;
   z=cleft+dz/2:dz:cright-dz/2;
   xa=zeros(1,nxy);
   ya=zeros(1,nxy);
   za=zeros(1,nxy);
   for i=1:ny
     ibeg=(i-1)*nx+1;
     iend=i*nx;
     xa(ibeg:iend)=x;
     ya(ibeg:iend)=y(i)*ones(1,nx);
     za(ibeg:iend)=z(1)*ones(1,nx);
   end
   sp=1/2*.0001;
   xa=xa+2*sp*dx*(rand(1,nxy)-0.5);
   ya=ya+2*sp*dy*(rand(1,nxy)-0.5);
   za=za+2*sp*dz*(rand(1,nxy)-0.5);
end

% specified initial positions ... useful for comparison purposes.
xa = [34.3148   56.2963   11.8889   16.9021   27.8895   55.6815
      48.5589   9.5222  23.1921   47.8658]*.5;
ya = [14.4320   65.1551   12.4608   81.5887   9.3024   30.9941
      26.8817   53.6451  16.3278   21.0988]*.5;
za = [0.6828    1.2523    1.6615    9.1142    1.3626    6.1700
      2.6898    2.2066   7.1290    5.4900];

xLS(1,:)=xa;
yLS(1,:)=ya;
zLS(1,:)=za;

figure(1), clf

subplot(221), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,
subplot(222), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,
subplot(223), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,
subplot(224), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,

for k=1:nstep
   eval(['bLS=' fcn '(xa,ya,za);']);
   for i=1:nxy
%      eval(['bLS=' fcn '(xa,ya,za);']);

      [iflagLS,iiLS]=getneighbors(i,xa,ya,za,scrmin,scrmax);

      if iflagLS==1,
         pLS=getupdate(iiLS,i,xa,ya,za,bLS,nxy);
         [delxLS(i),delyLS(i),delzLS(i)]=getnewpos(i,pLS,dxmax,dymax,dzmax,xa,ya,za,nxy);
      elseif iflagLS==2,
         delxLS(i)=0; delyLS(i)=0; delzLS(i)=0;
      end

% update when ready
%      xa(i)=xa(i)+delxLS(i);
%      ya(i)=ya(i)+delyLS(i);
%      za(i)=za(i)+delzLS(i);
%      if za(i)<0, za(i)=0; end
   end

% everyone update at once
   for i=1:nxy
      xa(i)=xa(i)+delxLS(i);
      ya(i)=ya(i)+delyLS(i);
      za(i)=za(i)+delzLS(i);
   end
   xLS(k+1,:)=xa;
   yLS(k+1,:)=ya;
   zLS(k+1,:)=za;
end

figure(1),
subplot(221),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
%plot(xb,yb,'ro','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
%for i=1:nxy, plot(xTLS(:,i),yTLS(:,i),'r','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off

subplot(222),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off
view(2)

subplot(223),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off
view(90,0)

subplot(224),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off
view(0,0)

% XTRAS
%xcm=zeros(nstep+1,1); ycm=zeros(nstep+1,1);
%xcm(1,1)=sum(xa)/nxy; ycm(1,1)=sum(ya)/nxy;
%plot(xa,ya,'bx',xcm(1,1),ycm(1,1),'mx'); hold on,
%plot(xcm(:,1),ycm(:,1),'m')
%plot(xcm(nstep+1,1),ycm(nstep+1,1),'mo')
```

A.8 Local3DSN.m

```
function [xLS,yLS,zLS]=dscc3D(fcn,nstep)
% function [xLS,yLS,xTLS,yTLS]=dscc(fcn,cflag,nstep,xa,ya,za)
%
% fcn: character string of function (must also have input fcn name)
% nstep: number of steps (updates) to perform
% xa, ya: if cflag='onn', then starting positions for continuation
%  if cflag='off', then this will be reset
%
%
%-----------------------------------------------------------------------
% 1. Finds which bugs are close.
% 2. Fixes Hessian if not positive definite.
% 3. Robot Avoidance: Takes step length of alpha along search direction.
%      If too close to another robot then backs off until not too close.
% 4. Each updates when ready. Do not wait.

eval(['input_' fcn ]);

cflag='off';

if cflag=='off',
   x=aleft+dx/2:dx:aright-dx/2;
   y=bleft+dy/2:dy:bright-dy/2;
   z=cleft+dz/2:dz:cright-dz/2;
   xa=zeros(1,nxy);
   ya=zeros(1,nxy);
   za=zeros(1,nxy);
   for i=1:ny
     ibeg=(i-1)*nx+1;
```

```
      iend=i*nx;
      xa(ibeg:iend)=x;
      ya(ibeg:iend)=y(i)*ones(1,nx);
      za(ibeg:iend)=z(1)*ones(1,nx);
   end
   sp=1/2*.0001;
   xa=xa+2*sp*dx*(rand(1,nxy)-0.5);
   ya=ya+2*sp*dy*(rand(1,nxy)-0.5);
   za=za+2*sp*dz*(rand(1,nxy)-0.5);
end

% specified initial positions ... useful for comparison purposes.
%xa = [34.3148   56.2963   11.8889   16.9021   27.8895
% 55.6815   48.5589   9.5222  23.1921   47.8658]*.5;
%ya = [14.4320   65.1551   12.4608   81.5887   9.3024
% 30.9941   26.8817   53.6451  16.3278   21.0988]*.5;
%za = [0.6828    1.2523    1.6615    9.1142    1.3626
% 6.1700    2.6898    2.2066   7.1290    5.4900];

percentRN = .1;
R = 20*(1 + percentRN/100 * 2*(rand(1,nxy)-0.5));
theta = linspace(0,324,10)*pi/180;   % surround
%theta = linspace(25,65,10)*pi/180;    % corner arc
xa = R.*cos(theta);
ya = R.*sin(theta);
za = 8 * (1 + 10/100 * (rand(1,nxy)-1));

xLS(1,:)=xa;
yLS(1,:)=ya;
zLS(1,:)=za;

figure(1), clf

subplot(221), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,
subplot(222), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,
subplot(223), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,
subplot(224), plot3(xa,ya,za,'kx','markersize',10,'linewidth',2); hold on,

for k=1:nstep
% for update at once
%   eval(['bLS=' fcn '(xa,ya,za);']);
%   bLS = bLS*(1 + percentSN/100 * 2*(rand(1,nxy)-0.5));
   for i=1:nxy
% for update when ready
      eval(['bLS=' fcn '(xa,ya,za);']);
      bLS = bLS*(1 + percentSN/100 * 2*(rand(1,nxy)-0.5));
      xa = xa + percentPN * 2*(rand(1,nxy)-0.5);
      ya = ya + percentPN * 2*(rand(1,nxy)-0.5);

      [iflagLS,iiLS]=getneighbors(i,xa,ya,za,scrmin,scrmax);
      if iflagLS==1,
         pLS=getupdate(iiLS,i,xa,ya,za,bLS,nxy);
         [delxLS(i),delyLS(i),delzLS(i)]=getnewpos(i,pLS,dxmax,dymax,dzmax,xa,ya,za,nxy);
      elseif iflagLS==2,
         delxLS(i)=0; delyLS(i)=0; delzLS(i)=0;
      end

% update when ready
      xa(i)=xa(i)+delxLS(i);
      ya(i)=ya(i)+delyLS(i);
      za(i)=za(i)+delzLS(i);
%      if za(i)<0, za(i)=0; end
   end

% everyone update at once
%   for i=1:nxy
%      xa(i)=xa(i)+delxLS(i);
%      ya(i)=ya(i)+delyLS(i);
%      za(i)=za(i)+delzLS(i);
%   end

   xLS(k+1,:)=xa;
   yLS(k+1,:)=ya;
   zLS(k+1,:)=za;
end

figure(1),
subplot(221),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
%plot(xb,yb,'ro','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
%for i=1:nxy, plot(xTLS(:,i),yTLS(:,i),'r','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off

subplot(222),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
```

```
set(h,'fontsize',14);
grid on
hold off
view(2)

subplot(223),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off
view(90,0)

subplot(224),
plot3(xa,ya,za,'ko','markersize',6,'linewidth',2);
for i=1:nxy, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'),
ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot movement is x to o','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off
view(0,0)

% XTRAS
%xcm=zeros(nstep+1,1); ycm=zeros(nstep+1,1);
%xcm(1,1)=sum(xa)/nxy; ycm(1,1)=sum(ya)/nxy;
%plot(xa,ya,'bx',xcm(1,1),ycm(1,1),'mx'); hold on,
%plot(xcm(:,1),ycm(:,1),'m')
%plot(xcm(nstep+1,1),ycm(nstep+1,1),'mo')
```

A.9 plotcompare.m

```
% a plot function for comparison

figure(2),

N=1:10;
subplot(221), plot3(xLS(1,N),yLS(1,N),zLS(1,N),'kx','markersize',10,'linewidth',2); hold on,
xlabel('x','fontsize',14,'fontweight','bold'), ylabel('y','fontsize',14,'fontweight','bold'),
title('Robot initial positions in the (x,y) plane','fontsize',14,'fontweight','bold');
```

```
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
view(2),
grid on
hold off

N=1:2;
subplot(222), plot3(xLS(1,N),yLS(1,N),zLS(1,N),'kx','markersize',10,'linewidth',2); hold on,
subplot(222), plot3(xLSg(1,N),yLSg(1,N),zLSg(1,N),'kx','markersize',10,'linewidth',2);

subplot(222), plot3(xLS(13,N),yLS(13,N),zLS(13,N),'ko','markersize',6,'linewidth',2); hold on,
subplot(222), plot3(xLSg(21,N),yLSg(21,N),zLSg(21,N),'ro','markersize',6,'linewidth',2);

for i=N, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
for i=N, plot3(xLSg(:,i),yLSg(:,i),zLSg(:,i),'r:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'), ylabel('y','fontsize',14,'fontweight','bold'),
title('Robots 1 and 2','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off

N=5:6;
subplot(223), plot3(xLS(1,N),yLS(1,N),zLS(1,N),'kx','markersize',10,'linewidth',2); hold on,
subplot(223), plot3(xLSg(1,N),yLSg(1,N),zLSg(1,N),'kx','markersize',10,'linewidth',2);

subplot(223), plot3(xLS(13,N),yLS(13,N),zLS(13,N),'ko','markersize',6,'linewidth',2); hold on,
subplot(223), plot3(xLSg(21,N),yLSg(21,N),zLSg(21,N),'ro','markersize',6,'linewidth',2);

for i=N, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
for i=N, plot3(xLSg(:,i),yLSg(:,i),zLSg(:,i),'r:','linewidth',1); end
xlabel('x','fontsize',14,'fontweight','bold'), ylabel('y','fontsize',14,'fontweight','bold'),
title('Robots 5 and 6','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off

N=9:10;
subplot(224), plot3(xLS(1,N),yLS(1,N),zLS(1,N),'kx','markersize',10,'linewidth',2); hold on,
subplot(224), plot3(xLSg(1,N),yLSg(1,N),zLSg(1,N),'kx','markersize',10,'linewidth',2);

subplot(224), plot3(xLS(13,N),yLS(13,N),zLS(13,N),'ko','markersize',6,'linewidth',2); hold on,
subplot(224), plot3(xLSg(21,N),yLSg(21,N),zLSg(21,N),'ro','markersize',6,'linewidth',2);

for i=N, plot3(xLS(:,i),yLS(:,i),zLS(:,i),'k:','linewidth',1); end
for i=N, plot3(xLSg(:,i),yLSg(:,i),zLSg(:,i),'r:','linewidth',1); end
```

```
xlabel('x','fontsize',14,'fontweight','bold'), ylabel('y','fontsize',14,'fontweight','bold'),
title('Robots 9 and 10','fontsize',14,'fontweight','bold');
zlabel('z','fontsize',14,'fontweight','bold'),
h=gca;
set(h,'fontsize',14);
grid on
hold off
```

A.10 r2depend.m

```
function f=r2(xc,yc,zc)
% Nekton 1/r^2 with sin(gamma)


nn=length(xc);
f=zeros(nn,1);

for j=1:nn,
   gamma(j) = atan(zc(j)/sqrt(xc(j)^2+yc(j)^2));
   %  the sensor reading is orientation dependent
%   gamma(j) = pi/2;
   den = xc(j)^2 + yc(j)^2 + zc(j)^2;
   %  the sensor reading varies as 1/r^2
   f(j,1)=abs(sin(gamma(j)))/den;
   %  what the sensor truly reads
   f(j,1)=1.0/f(j,1);
   % We invert the signal for optimization
end
```

A.11 r2.m

```
function f=r2(xc,yc,zc)
% Nekton 1/r^2 with sin(gamma)


nn=length(xc);
f=zeros(nn,1);

for j=1:nn,
%   gamma(j) = atan(zc(j)/sqrt(xc(j)^2+yc(j)^2));
%  the sensor reading is orientation dependent
   gamma(j) = pi/2;
   den = xc(j)^2 + yc(j)^2 + zc(j)^2;
   %  the sensor reading varies as 1/r^2
   f(j,1)=abs(sin(gamma(j)))/den;
   %  what the sensor truly reads
   f(j,1)=1.0/f(j,1);
   % We invert the signal for optimization
end
```

SAND REPORT DISTRIBUTION

Internal:

MS 0501	(2338)	Raymond Byrne (20)
MS 1003	(15211)	John Feddema (5)
MS 1004	(15221)	Raymond Harrigan (2)
MS 1002	(15200)	Stephen Roehrig (2)
MS 9018	(8945-1)	Central Technical Files
MS 0899	(9616)	Technical Library (2)

External:

Johnny Hurtado, Ph.D. (2)
Assistant Professor
Texas A&M University
Department of Aerospace Engineering
727C H.R. Bright Bldg
3141 TAMU
College Station, TX 77843-3141

Steven Eskridge (2)
TEES Assistant Research Engineer
Commercial Space Center for Engineering
Texas A&M University
3118 TAMU
College Station, TX 77843-3118

Elana C. Ethridge, Ph.D. (2)
DARPA Program Manager
Microsystems Technology Office
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Bryan Schulz (5)
Senior Project Engineer
Nekton Research, LLC
4625 Industry Lane
Durham NC, 27713