SAND REPORT

SAND2002-2837
Unlimited Release
Printed September 2002

Distributed Autonomous Navigation: An LDRD Final Report

G. Richard Eisler

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 under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2002-2837
Unlimited Release
Printed September 2002

Distributed Autonomous Navigation: An LDRD Final Report

G. Richard Eisler
Satellite Data Processing
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0973

Abstract

This report summarizes the analytical and experimental efforts for the Laboratory Directed Research and Development (LDRD) project entitled "Distributed Autonomous Navigation". The principal project goal was to develop distributed navigation using swarms of vehicles. Multiple vehicles would allow navigation with or without the availability of the Global Positioning System (GPS). Another unique thrust of this research was to apply optimization techniques to determine "best" swarm formations for navigating specific types of terrain. The project was partially successful in achieving gains in non-GPS navigation, multi-vehicle trajectory planning, and the application of traditional flight-based navigation methods to ground vehicles. A planned third year for the project was eliminated due to funding shortfalls.

Acknowledgements

This author wishes to thank the following individuals for their invaluable contributions, time, extensive discussions on the project, and input to this final report:

Denise Padilla, Department 15212, and Johnny Hurtado, currently at Texas A&M University, for developing the original proposal and for its initial studies.

Maritza Muguira, 15211 for researching the accuracy of candidate baro-altimeter sensors for localization use on mobile robots.

Chris Lewis, 15211, for basestation and vehicle code development and support, Swarm RATLER™ hardware support, and for providing the initial idea for the terrain correlation algorithm.

Ralph Peters, 15221, for interpolating the surveyed terrain data to produce a DTED model and extracting a rectangular grid for MATLAB analyses.

The author also wishes to thank Ken Jensen, 15212 and Rush Robinett, 6200, for their support and extensive discussions on this project.

Table of contents

Abstract 3

Acknowledgements 4

Table of contents 5

Table of figures 6

Introduction 7

Introduction 7

The Problem 8

Terrain Modeling 10

Sensor Selection 11

Terrain correlation and localization 12

Optimal Deployments for Multiple Vehicles 17

Navigation 18

Milestone Schedule 19

References 20

Appendix A. Altimeter Search for Distributed Navigation 21

Appendix B: Gradient-based motion planners to insure line-of-sight communication for mobile robot collectives traversing arbitrary terrain[12] 24

Appendix C: Users' Manual for Navigation Covariance Analysis Code (NavCov) 31

Table of figures

Figure 1. Swarm RATLER vehicle on test terrain 7
Figure 2. The Distributed Navigation Problem 8
Figure 3. Development of terrain model (moving clockwise from upper left) 10
Figure 4. Vehicle Template for Terrain Correlation on DTED model 12
Figure 5. DTED model with candidate template locations 13
Figure 6. Terrain model used in slope correlation 14
Figure 7. Terrain correlation simulation summary (proceeding clockwise from upper left) 16
Figure 8. Steepest descent deployment of multiple vehicles using altitude and line-of-sight constraints 17
Figure 9. Closed-loop Kalman Filter INS used for HAGAR 18
Figure 10. Navigation position error reduction history 18

Introduction

The ultimate goal of distributed autonomous navigation is for multiple cooperating robot vehicles to navigate through an unstructured environment in less time and cost, and more reliably than a single vehicle. In past LDRD [1] and DARPA-funded projects, the Intelligent Systems and Robotics Center has demonstrated that a single autonomous vehicle can successfully traverse to a goal location in moderately rough terrain using differential GPS. If, however, the GPS signal is lost or the vehicle gets stuck, the mission is often aborted and the vehicle is stranded. This is an unacceptable ending in many tasks, especially those of military importance such as surveillance, target acquisition, physical security, and logistics support. In these emerging threat situations, the mission must be completed with the highest probability of success possible.

It was proposed to develop terrain-aided distributed autonomous navigation hardware and algorithms that will allow a team of robot vehicles (Figure 1) to traverse an extremely rough terrain without being GPS reliant. The ultimate goal was to demonstrate that this team of robot vehicles could successfully navigate through a dense forest or deep canyon using localized sensing and communication and a partial map of the environment.

Figure 1. Swarm RATLER vehicle on test terrain

This was to be done through the enhancement of several unique capabilities that Sandia has developed in the areas of terrain-aided navigation and swarm robotics. In most terrain-aided navigation systems, altitude measurements from a single vehicle are correlated against a digital elevation map to determine position. These solutions require many measurements over a substantial length of terrain, which is impractical for most land and water vehicles. To compensate, we investigated the application of additional features such as terrain gradients, along with the use of swarms of communicating vehicles. This feature vector, combined with the relative 3D positions of each vehicle, provided a geometric template that was correlated against digital map data.

Another focus of this research was to apply distributed optimization techniques to determine optimal swarm formations for navigating through specific types of terrain.

During pursuit of the aforementioned goals in this project, one major change in research direction was made. Control Subsystems Department 2338 was enlisted to add proven missile navigation techniques to the mix of aforementioned approaches as a means to provide quantitative measures of position accuracy and improvement. During the first year of this cooperative effort, the Robotics Center would explore two areas: a)

techniques for relative positioning between vehicles using altitude measurements, and b) techniques for performing terrain aided navigation given a small swarm of vehicles that are collecting a limited number of these measurements. During the remaining two years, as part of the overall effort, Dept. 2338 would build on these two areas by investigating aided inertial navigation techniques that would provide capability for periods when GPS is denied [3].

The Problem

Vehicles must have information to navigate from one point to another. And in the absence of GPS, the information must come from other sources. Terrain-aided navigation (TAN) is one way to navigate without GPS, but in some applications it may be inadequate. For example, a single vehicle may require many measurements over a large area of terrain to estimate its position. To reiterate, this is impractical for land vehicles, so to compensate we proposed the use of many vehicles and more sensors.

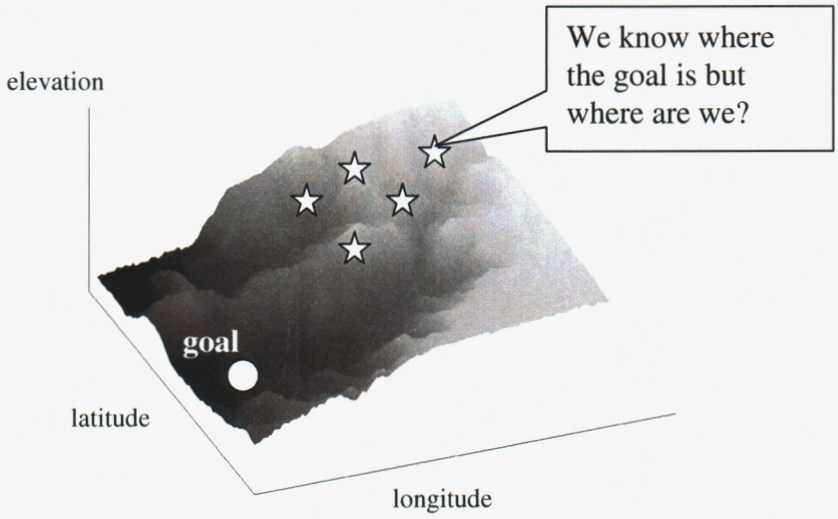


Figure 2. The Distributed Navigation Problem

The primary technical risks associated with distributed autonomous navigation are in the areas of sensing and relative positioning, robust inter-vehicle communication, and terrain map availability. Swarm RATLER™ hardware [2] platforms from the Robotics Center were used to minimize hardware development risks.

The project was organized with approximately equal milestones for analysis and hardware development during its intended 3-year term. These milestones are summarized as follows:

Year 1:

1. Research and select altitude sensor; initial integration into Swarm RATLER™
2. Survey test area
3. Initially demonstrate position estimation using terrain data from multiple vehicles

Year 2:

1. Complete sensor integration
2. Complete terrain correlation algorithm

3. Demonstrate multiple vehicles using terrain-aided navigation to traverse rough terrain at Sandia's Robotic Vehicle Range
4. Develop inertial navigation error analysis tool

Year 3:

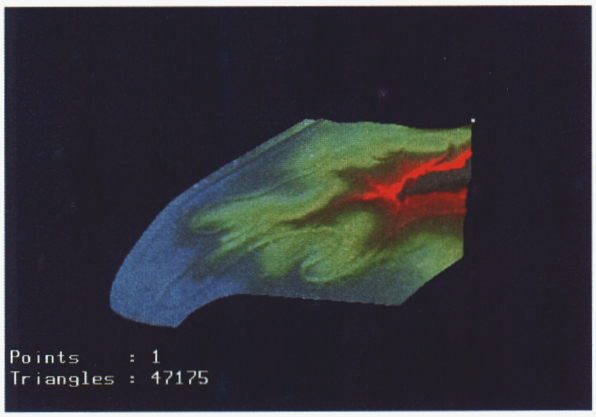
1. Complete optimal formation algorithm (based on terrain, task, etc.)
2. Demonstrate terrain-aided navigation of multiple vehicles in challenging area on Kirtland Air Force Base without GPS.
3. Couple inertial navigation analysis for multiple vehicles

Terrain Modeling

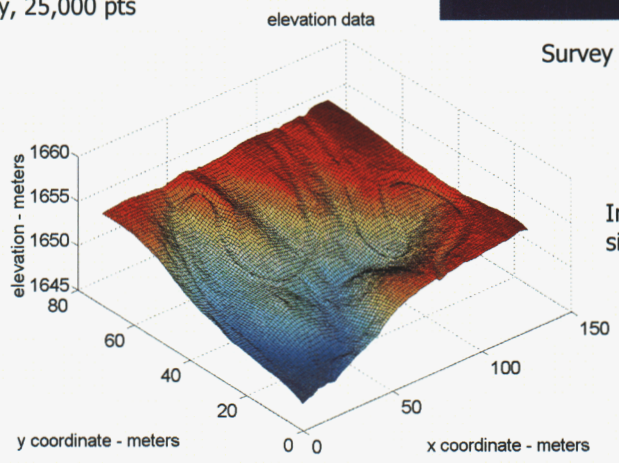
As stated in the original proposal, a single vehicle, even when outfitted with a wide array of sensors, can encounter some ambiguity in its estimated position. Therefore, the premise of this research effort is that a cooperating system of vehicles can minimize some of the ambiguity of the estimated vehicle positions by sensing a unique terrain feature (i.e., altitude) at multiple locations. It was envisioned that a group of Swarm RATLER™ vehicles would be used to record and difference *inaccurate* absolute position measurements to a specified swarm member. Having done this, an *accurate* "template" of *relative* (to-a-base) vehicle positions, would result that could be compared with the terrain model to find possible location matches. Once candidate locations were established, moving the swarm in an expeditious manner would hopefully eliminate the false candidates. A simulation would be developed beforehand to predict numbers of vehicles, allowable sensor errors, and the extent of subsequent vehicle moves in order to determine the vehicles' template location as quickly as possible.

To begin testing this approach, it was decided that a well-characterized terrain testbed be employed for both algorithm verification and sensor calibration in the 2nd and 3rd years of the effort.

GPS-RTK survey, 25,000 pts



Survey interpolated to 1mt resolution



Interpolated data segmented for simulation analysis, 72x127 grid

Figure 3. Development of terrain model (moving clockwise from upper left)

A GPS survey, employing the Real-Time Kinematic (RTK)[1] method, was commissioned with a local Albuquerque firm, Precision Surveys, to map the motocross course adjacent Robotic Vehicle Range at Sandia National Laboratories. Variations in elevation covered a 15-meter range. Approximately 25,000 points were taken and subsequently used to generate a 1-foot interpolated digital terrain elevation data (DTED) map [3]. The transition from terrain to elevation model is shown in Figure 3. A rectangular portion of this surveyed terrain was extracted for analysis in MATLAB [4].

Sensor Selection

As mentioned previously, our goal was to " ... minimize some of the ambiguity of the estimated vehicle positions by sensing a unique terrain feature (i.e., altitude) at multiple locations." After lengthy discussions on possible sensor choices, the selection was narrowed to baro-altimeters and the use of uncorrected GPS, for the sake of expediency. The latter was not felt to contradict the original intent of the project in that the measurement was sufficiently degraded in an absolute sense.

A representative survey of commercial-off-the-shelf baro-altimeters was accomplished [7], including bench testing. Full documentation of these results is provided in *Appendix A: Altimeter Search for Distributed Navigation*. This effort concluded that the deployment of these devices was problematic due to difficulties in "...maintaining a stable reading and sealing the reference port ...", and it was felt that said devices were too sensitive to be used on " ... a rugged vehicle traversing rough terrain".

From these results, hardware sensor usage defaulted to uncorrected GPS.

---

[1] "RTK is currently carrier phase observations processed (corrected) in real-time resulting in position coordinates to a 1-2 centimeter accuracy level. RTK, consists of two or more GPS receivers, three or more radio-modems, a 'fixed-plate initializer', and a handheld survey data collector/computer (TDC1). In RTK, one receiver occupies a known reference station and broadcasts a correction message (Compact Measurement Record or CMR2) to one or more roving receivers. The roving receivers process the information to solve the WGS-84 vectors by solving the integers in real-time within the receiver to produce an accurate position relative to the reference station. Precision of RTK is +/-2 cm + 2ppm, with 1 ppm equating to 1 mm per 1 km (Trimble Navigation, 1993). RTK, as with traditional kinematic GPS procedures, currently requires continuous satellite lock to be maintained. This restriction allows for RTK to be most effective in a non-canopied, no obstructions environment".
(http://wwwsgi.ursus.maine.edu/gisweb/spatdb/acsm/ac94105.html)

Terrain correlation and localization

The method developed in this study correlates estimated multiple vehicle positions to digital terrain data in order to plan the minimum number of individual motions to update location estimates to converge to the true "map" location of all vehicles. This approach will measure the relative positions of all the vehicles to a base, using uncorrected GPS measurements and correlate that "template" to the terrain map to generate candidate map registration values. A slope correlation is then done to assess which vehicles and directions to move. A majority of the following description is provided in Ref. [9]. An interesting aspect to this approach is that the vehicles will be correlated to the map, which may have small differences from reality. Since they have to navigate with respect to the map, this was felt to be the preferred goal.

It was assumed that a swarm of mobile robots was randomly positioned on the terrain and that individual members were able to generate biased, noisy absolute location measurements of the terrain, but accurate relative measurements with respect to one of the vehicles (base). This relative positioning of the vehicles creates a "template" (Figure 4) that can be rather quickly compared to the DTED data to determine various possible locations of the vehicle ensemble.

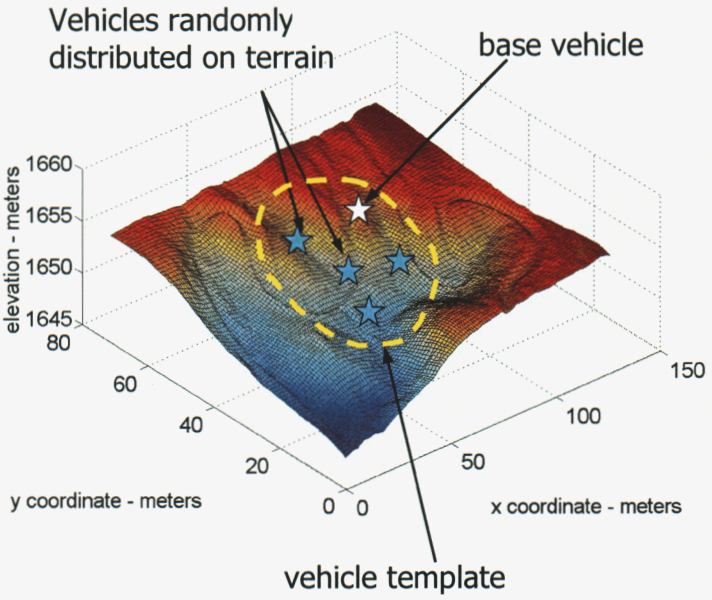


Figure 4. Vehicle Template for Terrain Correlation on DTED model

A root sum squares (RSS) of the differences between the measured and map relative altitude counterparts are computed for all possible locations of the vehicle template on the map. A lower threshold is set on the RSS differences to limit the possible candidate locations.

Figure 5 depicts a starting scenario for five vehicles on a top-down view of the terrain. The solid circles represent the true locations of the vehicles. The stars represent five other possible locations on the map of the "template". The dark stars are possible locations of the reference base vehicle while the lighter ones represent the other members. (Note that one of the candidate locations does correspond to the true location.) The rectangle represents the search area for the base vehicle that maintains all vehicles within the map

for any location in the search area.

A subgrid of four DTED model points around each possible vehicle location is established for local slope data and correlation products. Slope data was acquired through 2nd order curve fits of the map north-south and east-west slopes. Multiplicative products of this data were computed for a given vehicle over all of the subgrid positions amongst only unique candidate pairings (i.e., candidates 1-2,1-3,2-3, etc). (Though four points were used for the subgrids, any size uniform subgrid could be chosen, incurring a proportional penalty in computation time to subgrid size [8]). The vehicles demonstrating the most negatively (i.e., minimum) correlated product sums in the two cardinal directions were chosen.

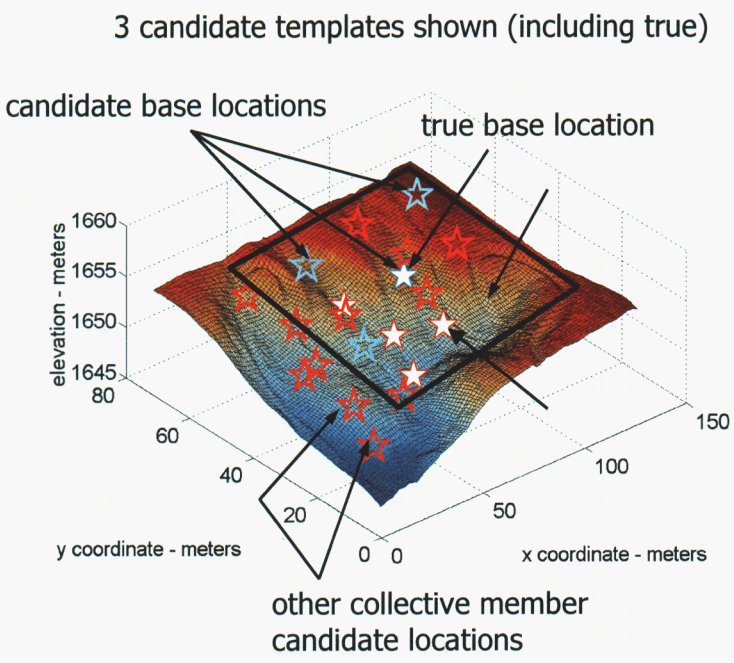


Figure 5. DTED model with candidate template locations

Once they had moved, a new "template" in effect was formed and the relative positions again measured via altitude differences. The goal was to keep moving the vehicles such that the new positions would produce greater RSS measurement errors among the false candidate locations and would henceforth be eliminated. The cycle of vehicle movements, measurement, relative distance RSS-value-thresholding and slope correlation was repeated until a single set of vehicle locations demonstrated a suitably small error.

The algorithm for the position template search and elimination is enumerated as follows:

1. Measure relative positions, form template and scan map for possible collective locations. (Initially, measure relative latitude and longitude as well to establish template).
2. Prune locations based on thresholding relative altitude (h) errors

$$\sum_{i=1}^{N\ vehicles} (h_{relative\ measured} - h_{relative\ from\ map})^2 \Big|_{jth\ candidate\ position} \leq threshold$$

3. Compute slope (ξ) correlation product sums over the 4-point subgrids between unique candidate pairings for the north-south and east-west cardinal directions using the *remaining* candidate locations

4. move the two vehicles that have the smallest respective G_i values a designated number of grid spacings positive in the two cardinal directions (i.e., east, north), where

$$G_i = \sum_{k=1}^{M-1\ candidates} \sum_{l=k+1}^{M\ candidates} \sum_{m=1}^{4\ subgrid\ points} \xi_{ikm}\xi_{ilm} \quad i = 1,...,N\ vehicles$$

The template has now changed shape by the two vehicles that have been moved.

5. Modify the remaining candidates by the moves in step 4, re-measure relative altitudes for the modified remaining candidates, and go to step 2

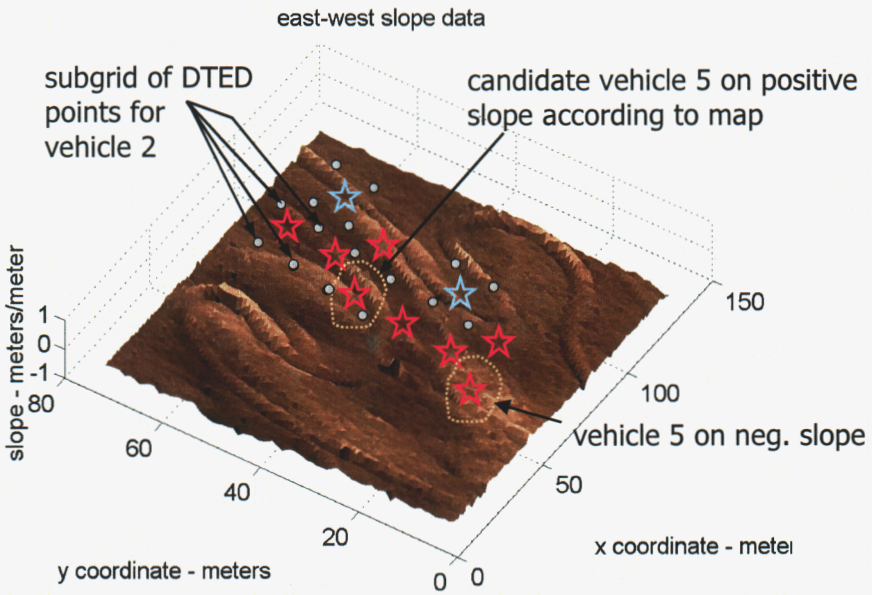


Figure 6. Terrain model used in slope correlation

Figure 7 shows the progress of a typical simulation. The true start locations of the swarm are displayed in the upper left. The initial map search (upper right) reveals 19 possible positions of the swarm to start. The simulated localization converges in 4 moves and the summary of vehicle moves are displayed in the lower right corner.

Extensive testing of this scheme with the actual vehicles on the actual terrain revealed less than satisfactory results (i.e., chattering behavior and lack of convergence) due to the following factors.

1. The actual measurements of relative positions (using uncorrected GPS) appeared to be far noisier than modeled via adding uniformly distributed noise to true locations.
2. Moving the vehicles an accurate number of grid spaces and in the correct directions on actual terrain was problematic and defaulted to the tester's best guess.
3. Test terrain and modeled terrain could differ over a short time.

It is suggested that future work with this algorithm combine previous relative vehicle locations with current locations in the sum-of-squares criterion for pruning false locations to enlarge and enhance the terrain map "signature" that is being compared. The template will increase in size as the product of number of vehicles with the number of moves. This "enlarging template" approach is a way to amass the amount of data sampled in flight weapon, terrain correlation applications.

Terrain correlation simulation:

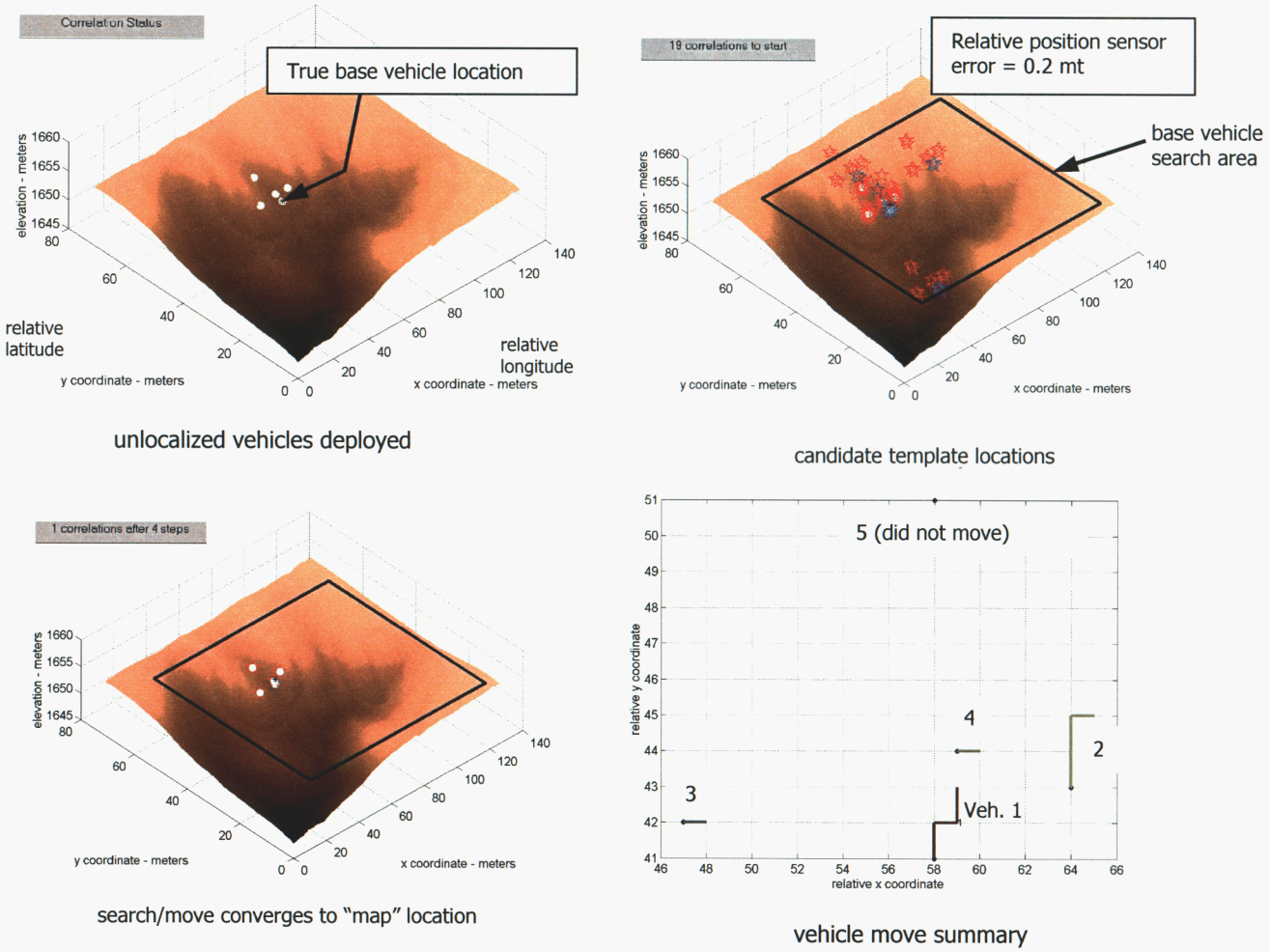


Figure 7. Terrain correlation simulation summary (proceeding clockwise from upper left)

Optimal Deployments for Multiple Vehicles

Optimal deployments for multiple vehicles were investigated via the steepest-descent (i.e. gradient method) to minimize local vehicle position metrics. The example of a large span communications network is a meaningful application of coordinated movements. It is desired to span a distance with a swarm of vehicles such that signals received from a goal point are transmitted back to a start point by maintaining vehicle-to-vehicle line of sight. It is necessary to maintain a line of sight back to the start for said transmissions. For a goal point that is visually obscured from the start point, it may be a lucky happenstance that arbitrary deployments of vehicles may be able to provide this connectivity. To orchestrate the deployment in an optimal sense, we seek to minimize the:

1. fore and aft distances of a given vehicle from its neighbors,
2. altitude of a given vehicle, and
3. positive elevation changes between vehicles (i.e., obstacles).

A simulation was developed to test this algorithm. A grid of hills, which deny line-of-sight between endpoints, was placed on a landscape. A set of vehicles was deployed from random locations close to the start point and moved in a stepwise fashion according to a gradient scheme, satisfying the above-enumerated goals. If only the first of the above-enumerated goals is used, the vehicles tended to spread themselves evenly from the start point to goal in a straight line, violating goals 2 and 3. The second constrained them away from climbing hills (violating 3), and the third insured that they moved where there were no obstacles (i.e., clear line-of-sight) between vehicles. Applying the last two metrics as penalty functions in an optimization scheme provided a deployment pattern that would be ideal for establishing the communication network to relay information over considerable distances.

An example deployment is shown in Figure 4 starting from a random arrangement of six vehicles. The terrain is composed of a grid of 9 hills that obscure the start point from observation of the goal.

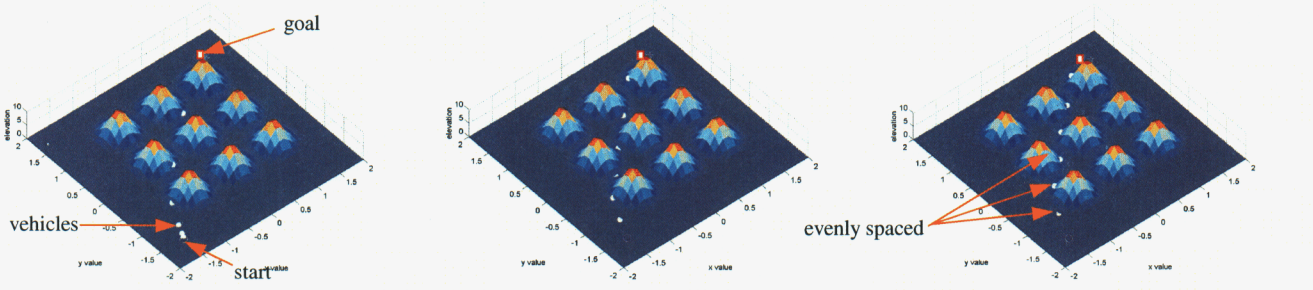


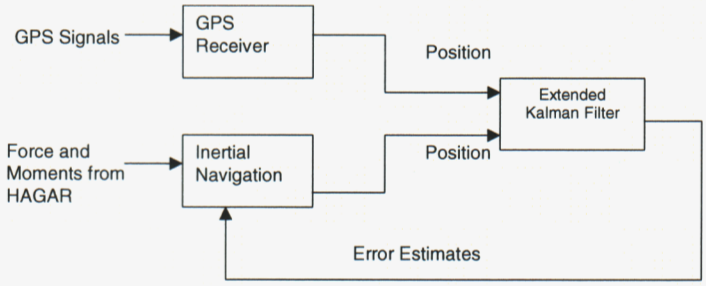
Figure 8. Steepest descent deployment of multiple vehicles using altitude and line-of-sight constraints

Appendix C provides more complete coverage of the gradient-based guidance.

Navigation

Control Subsystems Dept. 2338 supported this effort by investigating aided inertial navigation techniques that would provide capability for periods when GPS is denied. These efforts complimented a 2nd research effort [1] in autonomous navigation and the specific developments are more completely described in this reference. A brief, reiterative description of this work follows:

The navigation effort configured an inertial navigation system (INS) that used a closed-loop Kalman Filter configuration (Figure 9). This arrangement estimated instrument biases and errors using GPS inputs and inertial measurement unit (IMU) computations. These were fed back to correct IMU parameters during vehicle navigation.



Reference: Jordan, J.D, Dept. 2338, personal communication, Feb 2002

Figure 9. Closed-loop Kalman Filter INS used for HAGAR

This configuration could be analyzed and tuned via the *NavCov* program (See Appendix B for a user's manual) developed in Dept. 2338. This code [10] allows one to do tradeoffs on both accuracies and types of position and attitude measurement components used (i.e., IMU's, compasses, odometers, tilt meters, GPS receivers). Borrowing extensively from Sandia missile applications, the code uses detailed operational specifications of candidate hardware navigation components and is directed at allowing one to see isolated (or combined) component characteristic effects on position estimation performance.

Figure 10 displays *NavCov* output of the position error reduction of a representative missile-grade INS with GPS updating applied to a vehicle terrain trajectory. The position error variances in three axes have been used as semi-major axis dimensions of a time dependent error "ellipse". Note that the ellipse initially shows more error in the altitude direction and that the use of Kalman filtering "circularizes" this error during the trajectory. Circular position errors imply that you have reached a point where the error reduction process is proceeding uniformly in all directions indicating equal position estimation accuracy in all directions (the ideal condition).

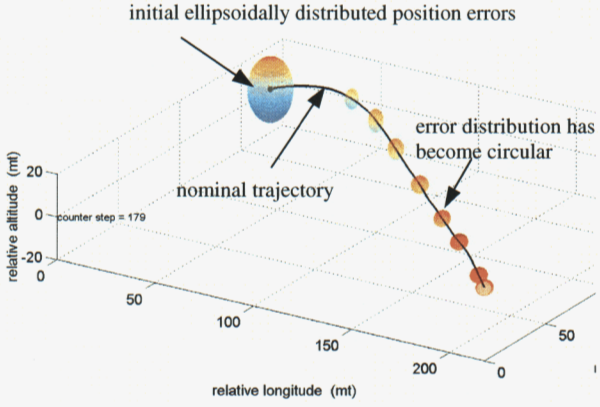


Figure 10. Navigation position error reduction history

Simulated nominal trajectories were developed in MATLAB [4] and are described in detail in [1].

Milestone Schedule

This section summarizes the success of the stated milestones listed in The Problem section.

Year 1: *Sensor research and initial demonstration of terrain-aided navigation*

Altimeter sensors were researched with disappointing results for other than uncorrected GPS. The use of uncorrected GPS still allowed the premise of the correlation of vehicle-measured positions to a map. Vehicle correlation was demonstrated in simulation for arbitrary numbers of vehicles. The simulated algorithm also showed sensitivity of map correlation to sensor noise.

Year 2: *Traversal of terrain at Sandia's Robotic Vehicle Range with chosen sensors and algorithms. Development of inertial navigation analysis tool.*

Use of the TAN algorithm on actual terrain demonstrated the sensitivity of the "search" to sensor noise, map differences, and achieving prescribed vehicle motions on terrain. Additional work is needed on the algorithm to increase terrain signature. The inertial navigation analysis tool, *NavCov*, was completed. NavCov is able to provide navigation error predictions based on a wide assortment of modeled hardware.

Year 3: *Development of optimal formation. Demonstration of terrain-aided navigation of multiple vehicles in challenging area on Kirtland Air Force Base without GPS. Coupling inertial navigation analysis to multiple vehicles.*

Year 3 work cancelled due to funding shortfall.

References

1. Eisler, G.R., "Robust Planning for Autonomous Navigation of Mobile Robots In Unstructured, Dynamic Environments: An LDRD Final Report", August 2002, Sandia National Laboratories
2. Klarer, P.K. [1993], *Recent Developments in the Robotic All Terrain Lunar Exploration Rover (RATLER) Program*, SAND93-1760C, August 93, **Sandia National Laboratories**
3. Peters, R., personal communication, Sandia National Laboratories, Department 15221, Albuquerque, October 2001
4. The MathWorks Inc., **MATLAB – The language of Technical Computing**, Version 6, Nov 2000, Natick, MA, www.mathworks.com
5. Bradley, J.D., "DistAutoNav - 2338 Tasks - Statement of Work", July 2001, Sandia National Laboratories
6. Feddema, J., Lewis, C., Klarer, P. [1999], *Control of Multiple Robotic Sentry Vehicles*, **Proceedings of the SPIE , Unmanned Ground Vehicle Technology**, Orlando, April
7. Muguira, M., personal communication, Sandia National Laboratories, Department 15211, Albuquerque, July 2001
8. Lewis, C., personal communication, Sandia National Laboratories, Department 15211, Albuquerque, June 2002
9. Eisler, R., Lewis, C., "Cooperative Robotic Map Correlation from Relative Position and Terrain Slope Measurements", presented at World Automation Congress 2002, Orlando, June 2002
10. Bradley, J., personal communication, Sandia National Laboratories, Department 2338, Albuquerque, November 2001
11. Jordan, J.D., "Zero Velocity + Position Updating", viewgraph presentation, Department 2338, Sandia National Laboratories, Albuquerque, February 2002
12. Eisler, G.R., "Simple gradient-based motion planners to insure line-of-sight communication for mobile robot collectives traversing arbitrary terrain", memo to J. Feddema, Sandia National Laboratories, July 1999

Appendix A. Altimeter Search for Distributed Navigation

Provided by Maritza Muguira, Intelligent Sensor Systems Department 15212

Background

Surface height (or gradient) is one of the crucial measurements necessary for the distributed navigation concept to pinpoint position without the use of GPS. Accordingly, we are searching for a sensor, which will detect the absolute or relative height of the vehicles in the platoon. Atmospheric pressure sensors were indicated as a possible aid to determine relative or absolute height by comparing relative sensed individual vehicle pressures or by correlating the temperature compensated pressure readings to the expected pressure for a give elevation. For the first year's milestone, we will validate the concept out in the motor cross by Sandia's RVR. In this terrain, there is an approximate maximum of fifty feet elevation range. Thus, a one-foot elevation error, which represents two percent of the total range, is necessary for the distributed navigation simulation to converge upon a reasonable amount of candidates within a reasonable amount of time. The RVR is at an elevation of approximately 5200 feet with an atmospheric pressure of about 12.13 psi (836.6 mbar). Approximately 0.00054 psi (0.037 mbar) represents an elevation difference of one foot.

Off the Shelf Sensors

Off the shelf sensors use various pressure references and fall into one of these categories: gage, differential, sealed gage, and absolute. The gage sensors use the local atmospheric pressure as the reference. Differential sensors measure the pressure difference between two input ports. Sealed gage sensors reference standard atmospheric pressure at sea level. Absolute pressure sensors use a vacuum (zero pressure) as a reference. The desired accuracy plays an important role in the determination of the sensor type, as accuracy is typically a function of the full scale output (FSO). By using a reference pressure close to the expected atmospheric pressure the error can be greatly reduced. For example, a 0 – 15 psi absolute pressure sensor with an error of 0.5% FSO will have a 0.075 psi error while a gage sensor with a 0.036 psi operating range and 0.5% FSO will have a 0.00018 psi error.

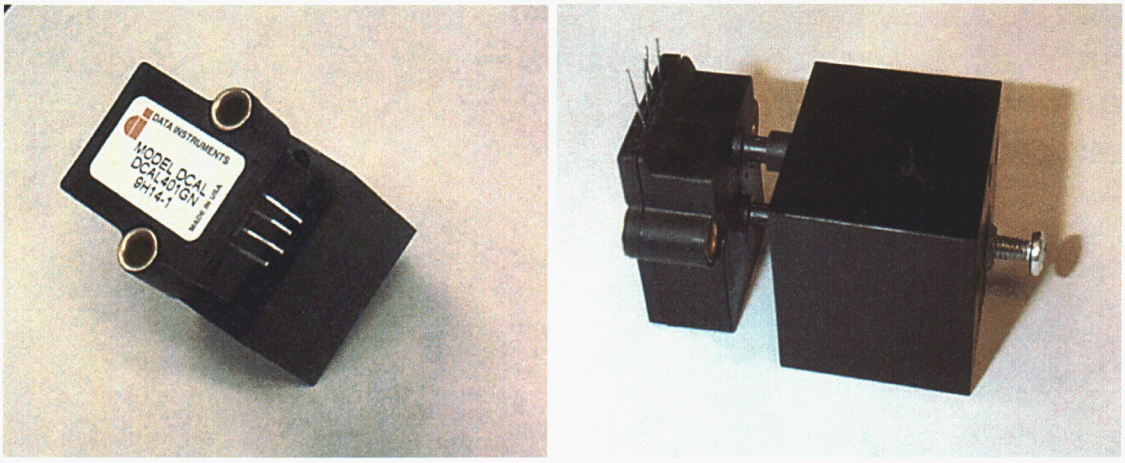
A few pressure sensors with acceptable accuracy, size, weight and electronic output were identified. Vaisala quoted $795 per 800 – 1060 mbar analog barometer with +/- 0.000435 mbar accuracy. Nova Lynx sells a 200 mb span (913 – 1113 mb) analog output barometric pressure sensor with +/-0.00029 psi accuracy for $510. Although that pressure span is too high for this elevation, other ranges are available upon request. Data Harvest offers a barometric pressure sensor with an 800 – 1100 mb range, but without an electronic output. Heise's DXD Series Precision Pressure Transducers have a 0.001 psi accuracy with 5 psi full scale for $825. Weston Aerospace Ltd. quoted us $2640 for the DPM 7885-1B with a range of 0.5 – 19 psi and 0.00185 psi accuracy. Setra's Model 470 barometric pressure sensor reads 800 – 1100 mb with 0.000087 psi accuracy and costs

$1250. Data Instrument's SURESENSE Ultra Low Pressure Sensors exhibit a maximum 0.25% linearity, hysteresis error and cost only $144 a piece. The SURSENSE sensors were out of stock when we called Honeywell, and we were required to place a minimum order of 10. We were able to borrow two 1 inch water operating range differential SURESENSE sensors and two 1 inch water gage SURESENSE sensors in order to conduct some preliminary test to determine the stability of the atmospheric pressure readings versus height. Note that Silicon Microstructures Incorporated makes some sensors (SM5310, SM5350, SM5410, SM5450, SM5501, and SM5502) similar to Data Instrument's SURESENSE sensors. We also identified some wind sensors available from Vaisala should we have difficulties isolating the sensor from the wind and need to compensate the pressure readout. The table below summarizes the candidate sensors and their characteristics.

Company/Model	*Range*	*Accuracy*	*Price*
Vaisala PTB100A	800 - 1060 mbar	+/-0.000435 mbar	$795
Nova Lynx 230-700	913 - 1113 mbar	+/- 0.00029 mbar	$510
Data Harvest	800 – 1100 mbar		
Heise DXD Series	0 – 5 psi (gage)	0.001 psi	$825
Weston Aerospace DPM 7885-1B	0.5 – 19 psi	0.00185 psi	$2640
Setra 470	800 – 1100 mbar	0.000087 psi	$1250
Data Instruments SURSENSE DCAL401DN	0.03613 psi (gage)	0.000090325 psi	$144
Silicon Microstructures SM5350/SM5450	0 – 0.3 psi (gage)	0.003 psi	

Experiments with Data Instrument SURESENSE Ultra Low Pressure Sensors

The gage and differential references provide a small full range centered about the median atmospheric pressure for the median height encountered when the reference is set to the highest possible pressure corresponding to the maximum height. Before we could begin collecting data, we needed to seal the reference port with an appropriate pressure (higher that the atmospheric pressure to be tested). Initially, we placed a plastic tube over the reference port and sealed the end of the tubing with a clip. Although this method sealed the reference port, the change in volume caused by clipping the end was sufficient to saturate the sensor. So, we tried placing a shut off valve at the end of the tube but it also compressed the air and saturated the sensor. Next, we designed and built a custom seal shown in the image below. The volume in the reference chamber could be adjusted with



Custom Seal

a screw that was coated with sealant gel while the testing port (port A) was open to sample ambient air. It was difficult to maintain the seal as we adjusted the volume, and once the seal was maintained the sensor would drift slowly to either saturation limit. We also tried using a very long tube (approximately 40 ft.) such that sealing the end would not cause such a large volume change relative to the entire reference volume, but again we would see the readings drift until the sensor would saturate. We could not understand why this was happening, and we decided to test the sensor with a Sono-Tek syringe pump to verify that the sensor itself was stable. We had to adjust the reference volume with the pump, and we finally stabilized the sensor after three days.

Experiments with Garmin's etrex Barometer

We also tested with Garmin's etrex barometer, and a few samples showed about a 1.5 feet error. However, we needed to collect more samples for conclusive results. Unfortunately, currently Garmin does not provide a method to extract the barometric pressure readings electronically such that testing would be extremely tedious.

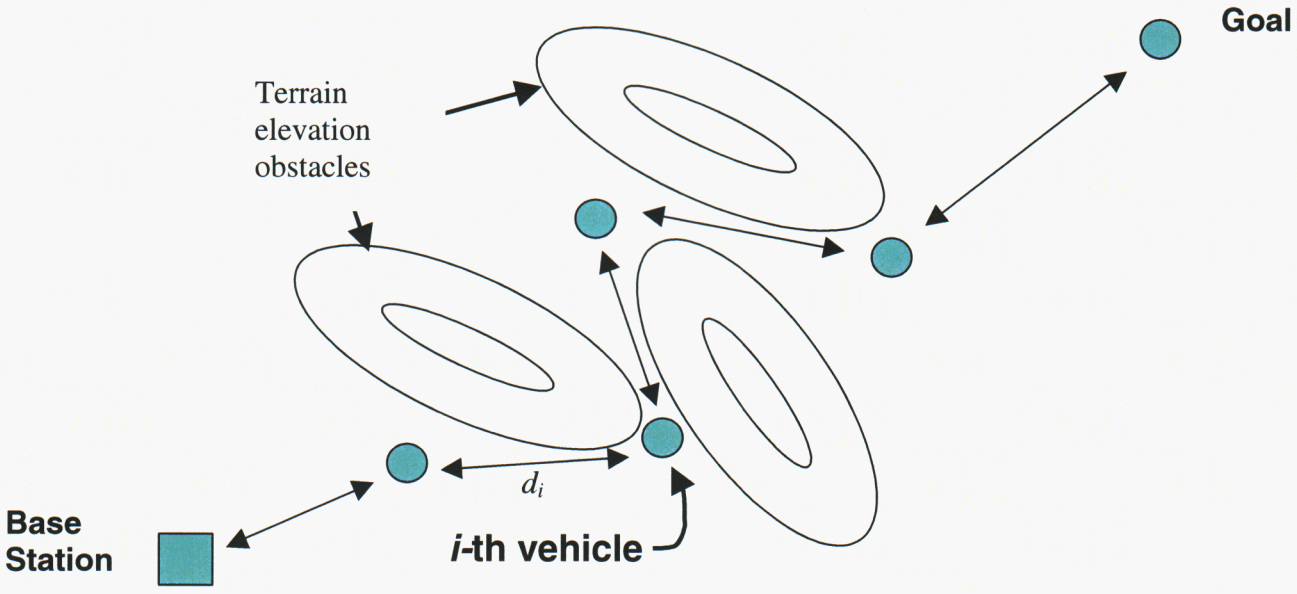
Recommendations

After experiencing so many problems maintaining a stable reading and sealing the reference port, we believe that it would not be wise to use such a sensitive sensor in a rugged vehicle traversing rough terrain. Chris Lewis suggested that we consider using a range finder along with an inclinometer and compass to determine the gradient of the vehicle. Surface gradients could be used rather than elevation. Gradient measurements would be susceptible to false readings from elevation changes in the vehicle due to debris such as rocks or bushes, so we could use the range finder to locate spots where the vehicle is sitting level on the ground regardless of the gradient. The range finder can be rotated about the vehicle in a circle, and all of the readings should be equal (except in the case where an obstacle(s) is located along the scan line) if the vehicle is sitting level.

Appendix B: Gradient-based motion planners to insure line-of-sight communication for mobile robot collectives traversing arbitrary terrain[12]

Introduction

It is desired to use a mobile robot collective to form a communications network between a base station and a goal point, which are separated by arbitrary terrain. Constraining the movements of the individual collective members is the tacit assumption that line-of-sight (LOS) must be maintained between adjacent members and that communication signal strength varies inversely as a power of distance, $1/d_i^n$, between these members (see figure below, courtesy of J. Feddema, Dept. 15211)



Problem Formulation

A suggestion for motion generation by J. Feddema, Dept 15211 would employ a simple gradient scheme to have the vehicles migrate such that each vehicle (i) tries to minimize a power of its *straight-line* distance (regardless of terrain) to both adjacent neighbors. The initial performance metric to be minimized in this effort is

$$V_i : \left(PL_{(i-1)to(i)}\right) + \left(PL_{(i)to(i+1)}\right)$$

$$= \left((x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2\right)^{m/2} + \left((x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2\right)^{m/2}$$

$$for\ i = 1,\ldots,n$$

Vehicles 1 and n will use the base and goal positions as their respective *(i-1)* th and *(i+1)* th neighbors. It is assumed that each vehicle has perfect knowledge of its location (x_i, y_i, z_i). The simple gradient form to update a vehicle position coordinate, ξ_i, based on minimizing metric, V_i, is $\xi_i(k+1) = \xi_i(k) - \alpha \left.\frac{\partial V_i}{\partial \xi_i}\right|_k$ for time step, $k+1$.

Using this form, the gradient expressions for ground coordinate updates (developed by J. Feddema) are:

$$x_i(k+1) = x_i(k) - \alpha \frac{\partial V_i}{\partial x_i}(k) \qquad y_i(k+1) = y_i(k) - \alpha \frac{\partial V_i}{\partial y_i}(k)$$

$$\frac{\partial V_i}{\partial x_i}(k) = m\left(x_i(k) - x_{i-1}(k)\right)\left(\left(x_i(k) - x_{i-1}(k)\right)^2 + \left(y_i(k) - y_{i-1}(k)\right)^2 + \left(z_i(k) - z_{i-1}(k)\right)^2\right)^{(m/2)-1}$$

$$+ m\left(x_i(k) - x_{i+1}(k)\right)\left(\left(x_{i+1}(k) - x_i(k)\right)^2 + \left(y_{i+1}(k) - y_i(k)\right)^2 + \left(z_{i+1}(k) - z_i(k)\right)^2\right)^{(m/2)-1}$$

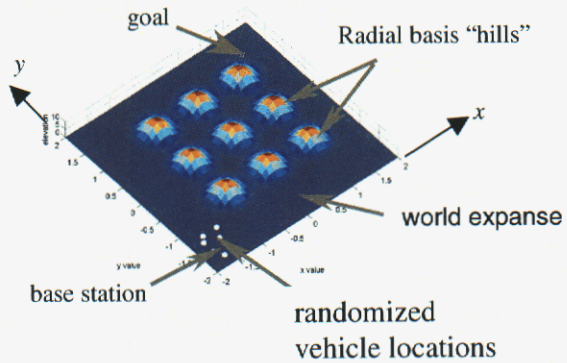
$$\frac{\partial V_i}{\partial y_i}(k) = m\left(y_i(k) - y_{i-1}(k)\right)\left(\left(x_i(k) - x_{i-1}(k)\right)^2 + \left(y_i(k) - y_{i-1}(k)\right)^2 + \left(z_i(k) - z_{i-1}(k)\right)^2\right)^{(m/2)-1}$$

$$+ m\left(y_i(k) - y_{i+1}(k)\right)\left(\left(x_{i+1}(k) - x_i(k)\right)^2 + \left(y_{i+1}(k) - y_i(k)\right)^2 + \left(z_{i+1}(k) - z_i(k)\right)^2\right)^{(m/2)-1}$$

$$for\ i = 1,\ldots,n$$

Values of m between $2 < m < 2.5$ provide stable gradient operations. $m = 2.1$ was used in this study. α was set to 0.1 for similar reasons. Successive updates were generated without regard to vehicle performance or consistent time intervals. Consistent with the gradient definition, the update "steps" are initially large and get progressively smaller as the minimum "network" is achieved.

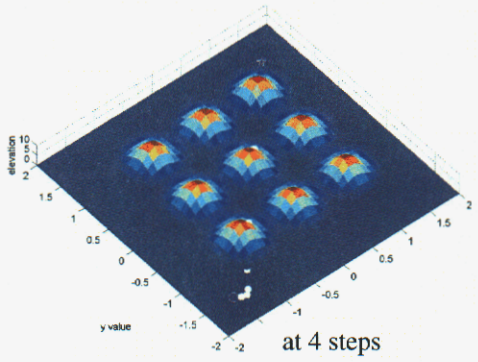
The vehicles "moved" over a *normalized* world expanse of terrain whose dimensions were $-2 < (x, y)_{world} < 2$. A set of vehicle positions, (x_i, y_i) was initialized ($k = 0$) using random, uniformly distributed values contained within a square of 0.8 units on a side centered about a specified *base station* within the world "expanse". Between the base station and a specified goal point, "hilly" terrain, in the form of radial basis functions, was used to block clear LOS. The radial



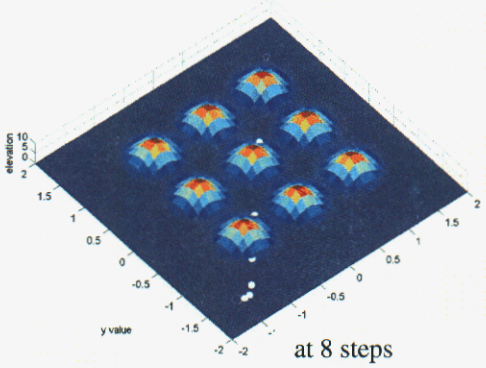
basis function form gives elevation, $z(x, y) = \sum_{i=1}^{l} A_i e^{-b_i\left[(x-rbx_i)^2+(y-rby_i)^2\right]}$ as a function of ground coordinates, x,y. l is the number of radial basis "hills", (rbx_i, rby_i) are the ground coordinates of the i-th hill center, A_i is the "hill" height, and b_i regulates the hill steepness.

Results

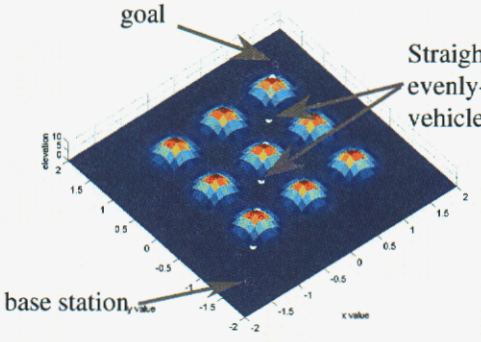
A collective of 6 vehicles was used to demonstrate communication network formation. The base station and goal ground positions were (-1.5, -1.5) and (1.5, 1.5) respectively. A_i = 10, b_i = .25, and nine radial basis hill centers were distributed evenly on a portion of the world grid spanning $-1.25 < (x, y)_{world} < 1.5$. Network completion was assumed when no vehicle changed position by more than 10^{-3} units. The evolution of a typical run follows



at 4 steps



at 8 steps



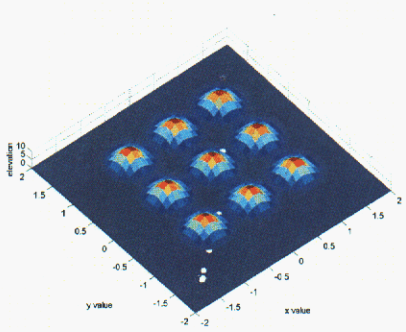
at 135 steps (convergence)

As is plainly evident, the "network" generated from this formulation is a straight line from base station to goal with vehicles approximately evenly spaced. The downside to this strategy is that there is no consideration for elevation or maintaining LOS, which allows neighboring vehicles to be situated on opposite sides of the same hill at varying elevations.

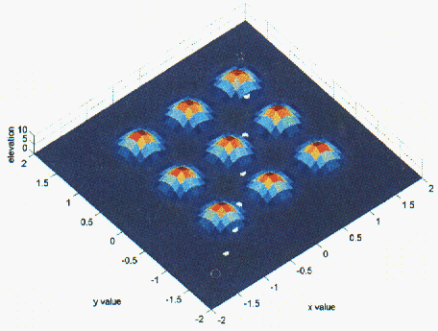
If elevation is an issue, the term $\frac{1}{2} z_i^T W_e z_i$ (where W_e is an arbitrary weight) can be attached to the previous minimum distance metric, V_i, to drive elevation to zero. The gradient *step* form remains the same, but the gradient forms are altered according to

$$\frac{\partial V_i}{\partial x_i}(k) = \left[\frac{\partial V_i}{\partial x_i}(k)\right]_{\min dist} + z_i W_e \frac{\partial z_i}{\partial x_i} \qquad \frac{\partial V_i}{\partial y_i}(k) = \left[\frac{\partial V_i}{\partial y_i}(k)\right]_{\min dist} + z_i W_e \frac{\partial z_i}{\partial y_i}$$

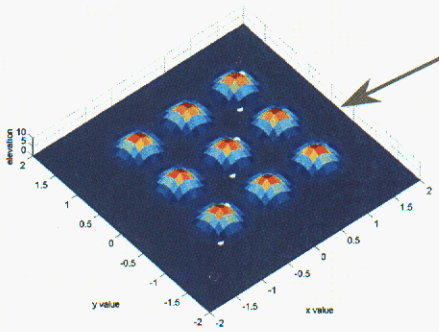
The gradients $\frac{\partial z_i}{\partial x_i}, \frac{\partial z_i}{\partial y_i}$ are computed by differentiating the $z(x,y)$ radial basis function. In field practice, these derivatives would need to be provided from a combination of inclinometer and azimuth sensor readings (unless map data was available). A common weight, W_e = .005, was used for all vehicles. Results from this type of network generation are shown below.



at 10 steps



at 40 steps



at 165 steps

Minimum distance + elevation-penalty network formation

Use of the elevation penalty coerces the vehicles to "skirt" the hills as they form the "comm" link. As the link is finished (165 steps), the vehicles assume positions that are compromises between maintaining equal distances to adjacent neighbors and attaining zero value elevations. The addition of the elevation penalty prevents them from attaining the straight-line configuration seen in the "minimize distance only" solution. A straight-

line solution at zero elevation would position the vehicles in the valleys between hills causing a disparity in intra-vehicle distances, as they progress from valley to hill and beyond. The "comm" link line is bent to satisfy both desired behaviors (to the degree of the relative weightings). Unfortunately, LOS has still not been maintained as adjacent vehicles are still obscured by intervening hills.

LOS can be attained within the same gradient step formulation by adding a final term to be minimized in the performance metric for the *i*th vehicle. *This metric would assume knowledge of terrain elevation on a straight line between adjacent vehicles* at a user-specified number of points. Adding the term, $\frac{1}{2}\sum_{j=1}^{2p} z_{ij}^T W_{los} z_{ij}$, to the total metric will provide a gradient dependence on inter-vehicle terrain. p is the number of equally spaced intermediate points between vehicles i and $i+1$ (as well as between i and i-1), z_{ij} is an intermediate point with vehicle i as the center point of the set, and W_{los} is an arbitrary weight. This term will tend to drive elevation at the $2p$ discrete points to zero. The gradient forms, including this final term are

$$\frac{\partial V_i}{\partial x_i}(k) = \left[\frac{\partial V_i}{\partial x_i}(k)\right]_{\min dist} + z_i W_e \frac{\partial z_i}{\partial x_i} + \sum_{j=1}^{2p} z_{ij}^T W_{los} \frac{\partial z_{ij}}{\partial x_i}$$

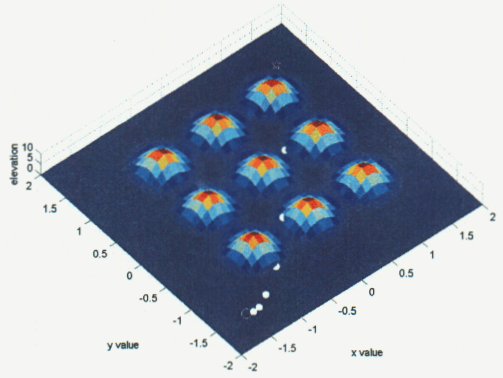
$$\frac{\partial V_i}{\partial y_i}(k) = \left[\frac{\partial V_i}{\partial y_i}(k)\right]_{\min dist} + z_i W_e \frac{\partial z_i}{\partial y_i} + \sum_{j=1}^{2p} z_{ij}^T W_{los} \frac{\partial z_{ij}}{\partial y_i}$$

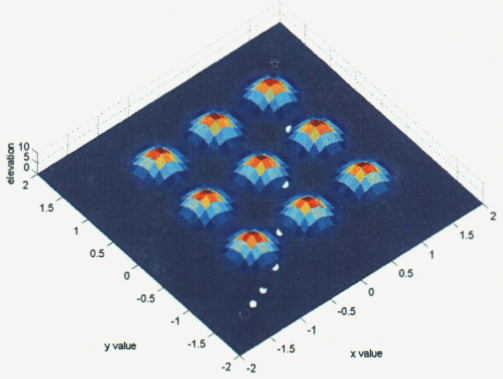
The gradients $\frac{\partial z_{ij}}{\partial x_i}, \frac{\partial z_{ij}}{\partial y_i}$ were computed as -

$$\frac{\partial z_{ij}}{\partial x_i} = \frac{\partial z_{ij}}{\partial x_j} * \frac{\partial x_j}{\partial x_i} = \frac{\partial z_{ij}}{\partial x_j}\left[1 - \frac{j}{p+1}\right] \qquad \frac{\partial z_{ij}}{\partial y_i} = \frac{\partial z_{ij}}{\partial y_j} * \frac{\partial y_j}{\partial y_i} = \frac{\partial z_{ij}}{\partial y_j}\left[1 - \frac{j}{p+1}\right]$$

where the partials $\frac{\partial z_{ij}}{\partial x_j}, \frac{\partial z_{ij}}{\partial y_j}$ are computed as described in the elevation penalty discussion using the radial basis function. Cross derivative terms of the form $\frac{\partial x_j}{\partial y_i}$ were ignored.

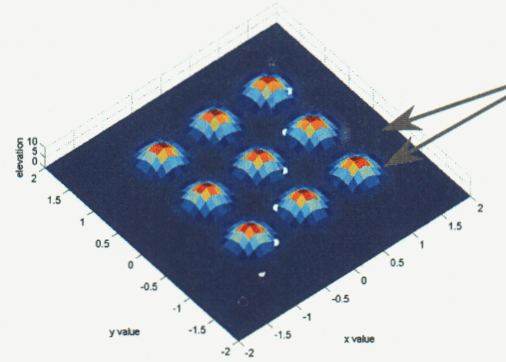
A common weight, W_{los}= .005, and p = 3 intermediate points were used for all vehicles. Results from a network generation based on the three effects (distance, elevation, LOS) are shown below.





At 10 steps

At 20 steps



Vehicles migrate to edge of hills to maintain LOS

At 60 steps

"Minimum distance + elevation penalty + maintaining-LOS" network formation

The addition of the LOS term supplies this dependency in a similar "weighted" fashion as was done for the elevation penalty. The combination of all three effects cause the vehicles to move in the valleys between hills, acquiring just enough elevation to maintain LOS, while attempting to equalize inter-vehicle distances. The final "comm" link (at 60 steps) shows a viable vehicle "collective" configuration to establish a base-to-goal link. As mentioned before, one caveat to the use of this effect is that it presumes you have a way

of obtaining more than just local terrain information. Possibly the possession of accurate on-board topological maps would justify its use.

Summary Observations

The gradient step forms generate low-burden computational forms to effect vehicle movement. Relative weighting between the penalties (distance, elevation, LOS = 1., W_e, W_{los}) in the performance metric, step size control (via α), and the number of inter-vehicle intermediate points, p, for LOS computations are tunable parameters that can provide a myriad of "marching" behaviors. In the field, distance and elevation gradients could be obtained from inter-vehicle communications and local measurements. LOS gradient requirements would necessitate elevation mapping of the intended network terrain.

Appendix C: Users' Manual for Navigation Covariance Analysis Code (NavCov)

USER MANUAL
for the
NavCov Covariance Analysis Code
developed for the
Distributed AutoNav LDRD
by J.D. Bradley, Control Subsystems Department 2338

version 1.0, 5/1/2002

This manual describes the function and use of the NavCov covariance analysis code. This code is a set of liberally commented Matlab scripts and functions derived from the work of previous projects and oriented for use in defining inertial navigation performance and error budgets for land vehicle applications.

INTRODUCTION

The purpose of a covariance analysis code is either to predict the statistical performance of some aided inertial navigation system under consideration without extensive modeling and simulation, or conversely, to determine the required error budget of an aided inertial navigation system to meet some contemplated performance goal. This covariance analysis code utilizes several tools, including:

1) a model of inertial navigation errors, consisting of a set of linear differential equations describing the behavior of both the general errors and those errors associated with the accelerometers and gyroscopes contained in an IMU.

2) models of measurement errors for sensors used to aid or update the inertial navigator. Error models are included for:
   a) a baro-altimeter damping loop, which is used to stabilize the unstable vertical channel of the inertial navigator,
   b) an odometer velocity measurement, which is used to update the along-track velocity estimate of the inertial navigator,
   c) a compass heading measurement, which is used to update the heading estimate of the inertial navigator,
   d) a two axis inclinometer attitude measurement, which is used to update the roll and pitch estimates of the inertial navigator,
   e) a GPS position measurement, which is used to update the latitude, longitude, and height estimates of the inertial navigator, and

f) zero velocity update, which is used to update the north and east tilts, north east, and down velocities, and latitude, longitude, and height estimates of the inertial navigator when the vehicle is not moving.

3) a linearized Kalman filter used for propagating and updating the covariance matrix and for calculating the optimized gains (or weighting factors) used during updating of the covariance matrix. The Kalman filter used by NavCov provides two modes of operation:
   a) a full order mode which uses the full complement of modeled states to produce the optimum (and potentially optimistic) estimation of performance, and
   b) a reduced-order or *suboptimal* mode used to omit the contribution of some states in the calculation of Kalman gains to achieve a potentially more realistic estimation of performance.

The inertial navigator error model was developed by Dr. William Widnall and Mr. Peter Grundy for the U.S. Air Force and is documented in [ref 1]. Additional information regarding the inertial navigation error model is found in [ref 2]. The baro-altimeter measurement error model is also documented in [ref 1]. Kalman filter equations are documented in numerous sources including [ref 3].

A primary input to the covariance code is a trajectory definition about which partial derivatives in the linearized navigation error model and measurement model can be evaluated. For this application, a pseudo-6-degree-of-freedom (6DOF) simulation was developed separately by Dr. Richard Eisler based on point mass translational motion over digital terrain elevation data (DTED). The simulation is driven by throttle and heading commands. Vehicle attitude rates were generated via finite differencing of DTED terrain features as a function of vehicle speed. No vehicle motion was assumed normal to the local surface.

It is important to understand that a covariance analysis computes, propagates, and updates the variances (aka uncertainties) of the inertial navigation error states rather than the actual inertial navigation states or error states. The actual states, error states, and the sensor measurements do not exist in the covariance analysis; these items would be included in a full simulation of an aided inertial navigator.

It is also important to note that the inertial navigator error model assumes that navigation errors are small so that a linearized model is valid, meaning that position errors remain small relative to the earth's radius, velocity errors remain small relative to orbital velocity, and attitude errors remain small relative to 1 radian [ref 1].

LINEARIZED KALMAN FILTER EQUATIONS

This section borrows heavily from [ref 3]. The continuous time general model of an inertial navigator and the measurements used to aid the navigator take the form of:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u},t) + \mathbf{w}(t) \tag{1}$$
$$\mathbf{z} = \mathbf{h}(\mathbf{x},t) + \mathbf{v}(t) \tag{2}$$

where **x** is the navigation state vector, **u** is a forcing function, **w** is an independent white noise process representing navigation process noise, **z** is the measurement vector, and **v** is another independent white noise process representing measurement noise. The function **f** represents the navigator's state dynamics and the function **h** represents the transformation between the navigation states and the measurements.

Assuming that an approximate trajectory **x***(t) may be determined by some means, the actual trajectory can be represented as:

$$\mathbf{x}(t) = \mathbf{x}^*(t) + \mathbf{e}(t) \tag{3}$$

where **e** is the navigation error state vector. Substituting, equations (1) and (2) become:

$$\dot{\mathbf{x}}^* + \dot{\mathbf{e}} = \mathbf{f}(\mathbf{x}^* + \mathbf{e},\mathbf{u},t) + \mathbf{w}(t) \tag{4}$$
$$\mathbf{z} = \mathbf{h}(\mathbf{x}^* + \mathbf{e},t) + \mathbf{v}(t) \tag{5}$$

To linearize these equations, it is assumed that **e** is small and **f** and **h** are approximated with Taylor series expansions truncated to the first order terms only. The result is:

$$\dot{\mathbf{x}}^* + \dot{\mathbf{e}} \approx \mathbf{f}(\mathbf{x}^*,u,t) + \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}^*} \bullet \mathbf{e} + \mathbf{w}(t) \tag{6}$$

$$\mathbf{z} \approx \mathbf{h}(\mathbf{x}^*,t) + \left[\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}^*} \bullet \mathbf{e} + \mathbf{v}(t) \tag{7}$$

The approximate, or nominal, trajectory **x***(t) is chosen to satisfy the differential equation:

$$\dot{\mathbf{x}}^* = \mathbf{f}(\mathbf{x}^*,\mathbf{u},t) \tag{8}$$

Note that for NavCov, **x***(t) is the trajectory data that is generated separately and provided as an input to the code.

Now, equation (6) becomes:

$$\dot{\mathbf{e}} = \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}^*} \bullet \mathbf{e} + \mathbf{w}(t) \tag{9}$$

or more commonly:

$$\dot{\mathbf{e}} = \mathbf{F} \bullet \mathbf{e} + \mathbf{w}(t) \tag{10}$$

where **F** is known as the linearized error dynamics matrix. The derivation and contents of the **F** matrix for the inertial navigator error model can be found in [ref 1] and [ref 2] and so won't be repeated here. There are some sign differences for several of the elements due to use by NavCov of a North-East-Down (NED) coordinate frame rather than East-North-Up (ENU).

Reorganizing equation (7) leads to:

$$[\mathbf{z} - \mathbf{h}(\mathbf{x}^*, t)] = \left[\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}^*} \bullet \mathbf{e} + \mathbf{v}(t) \tag{11}$$

or more commonly:

$$\mathbf{Z} = \mathbf{H} \bullet \mathbf{e} + \mathbf{v}(t) \tag{12}$$

where **Z** is now the measurement *residual*, or the difference between the actual measurement and the measurement predicted based on the nominal trajectory. **H** is known as the linearized observation (or measurement) matrix.

In a real-time implementation of an aided inertial navigator, an extended Kalman filter would be used rather than a linearized Kalman filter. The difference is that the extended Kalman filter evaluates the partial derivatives in the above equations about the estimated, rather than the nominal, trajectory.

The covariances of the navigation error states are contained in the matrix **P**, defined as:

$$\mathbf{P} = \mathrm{E}(\mathbf{e}\mathbf{e}^T) \tag{13}$$

where E is the expected value function. The variances of the navigation error states reside along the diagonal of **P**. The process noise matrix **Q** is defined as:

$$\mathbf{Q} = E(\mathbf{w}\mathbf{w}^T) \tag{14}$$

Q represents the variances of the process noise for the navigation error states. The measurement noise matrix **R** is defined as:

$$\mathbf{R} = E(\mathbf{v}\mathbf{v}^T) \tag{15}$$

R represents the variances of the measurement noise.

NavCov utilizes the discrete-time version of the Bucy-Joseph form of the Kalman filter equations. Once the **P** matrix is initialized, it is propagated using the equation:

$$\mathbf{P}^{-}(k) = \boldsymbol{\varphi}(k)\mathbf{P}^{+}(k-1)\boldsymbol{\varphi}^{T}(k) + \boldsymbol{\Gamma}(k) \tag{16}$$

where $\mathbf{P}^{-}$ denotes the covariance matrix prior to updating with measurements and $\mathbf{P}^{+}$ denotes the covariance matrix after updating. k indexes the current time step, whereas k-1 indexes the previous time step. The state transition matrix $\boldsymbol{\phi}$ in the above equation is computed using a truncated exponential series expansion via the equation:

$$\boldsymbol{\varphi}(k) = \mathbf{I} + \mathbf{F}(k)\Delta t + \tfrac{1}{2}\mathbf{F}^{2}(k)\Delta t^{2} \tag{17}$$

where **I** is the identity matrix, Δt is the time difference between the current and previous time steps, and F is assumed to be time-invariant. Also, the matrix $\boldsymbol{\Gamma}$ in equation (16) is just:

$$\boldsymbol{\Gamma}(k) = \mathbf{Q}(k)\Delta t \tag{18}$$

If measurements are available during the kth time step, the observation matrix **H** is computed and the Kalman gain matrix **K** is computed using the equation:

$$\mathbf{K}(k) = \mathbf{P}^{-}(k)\mathbf{H}^{T}(k)[\mathbf{H}(k)\mathbf{P}^{-}(k)\mathbf{H}^{T}(k) + \mathbf{R}(k)]^{-1} \tag{19}$$

Then the covariance matrix $\mathbf{P}^{-}$ is updated to $\mathbf{P}^{+}$ using the equation:

$$\mathbf{P}^{+}(k) = [\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k)]\mathbf{P}^{-}(k)[\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k)]^{T} + \mathbf{K}(k)\mathbf{R}(k)\mathbf{K}^{T}(k) \tag{20}$$

If no measurements are available during the kth time step, then the updated covariance matrix $\mathbf{P}^{+}$ is set equal to $\mathbf{P}^{-}$.

COORDINATE FRAMES

NavCov uses a number of coordinate frames. The first is the earth frame, which is a polar coordinate frame that is earth fixed (i.e. it rotates with the earth). It is defined by the WGS-84 ellipsoid, though spherical approximations are used in several places throughout the code. Its components are:

Earth Frame

- Latitude defined to be zero at the equator and positive north, negative south, angular units
- Longitude defined to be zero at the Greenwich meridian and positive east, negative west, angular units
- Height defined to be zero on the ellipsoid and positive up, linear units

Note however, that height error in this code is defined to be positive down rather than positive up.

The second frame is the geographic (also called navigation) frame, which is a locally level, or normal to the gravity vector. The geographic frame is also earth fixed. It is a right-handed Cartesian coordinate frame with its origin at the center of navigation (for this code) and components:

Geographic Frame

North	defined to be positive north, linear units
East	defined to be positive east, linear units
Down	defined to be positive down, linear units

Angles and angular rates about these axes are defined to be positive by right-handed convention.

The third frame is the body frame, which is fixed to the body of the vehicle. It is a right-handed Cartesian coordinate frame with its origin at the center of navigation (for this code) and components:

Body Frame

X	longitudinal axis, defined to be positive out the nose, linear units
Y	lateral axis, defined to be positive out the right side, linear units
Z	"down" axis, defined to be positive out the belly, linear units

Again, angles and angular rates about these axes are defined to be positive by right-handed convention.

These sign conventions differ somewhat from those used in [ref 1] and [ref 2] and result is some sign differences in the various equations contained in NavCov, particularly in the computation of the **F** matrix.

MEASUREMENTS INCLUDED

As mentioned before, the aiding measurements included in NavCov are:

1) baro-altimeter damping loop, which is used to stabilize the unstable vertical channel of the inertial navigator. This measurement is actually not used as an aiding, or updating input, but rather, is implemented as an integral part of the inertial navigator.

2) odometer velocity measurement, which is used to update the along-track velocity estimate of the inertial navigator,

3) compass heading measurement, which is used to update the heading estimate of the inertial navigator,

4) two axis inclinometer attitude measurement, which is used to update the roll and pitch estimates of the inertial navigator,

5) GPS position measurement, which is used to update the latitude, longitude, and height estimates of the inertial navigator, and

6) zero velocity update, which is used to update the north and east tilts, north east, and down velocities, and latitude, longitude, and height estimates of the inertial navigator.

Aiding measurements can be enabled or disabled to explore their impact on navigation performance. In addition, these measurements can be scheduled so that they are only applied during specific parts of the vehicle trajectory. This allows, for instance, GPS to be used for a period at the beginning of the trajectory, and then denied for the remaining part of the trajectory. During this period of GPS-denial, other aiding sources such as zero velocity updating can be used.

Further discussion of the measurement error models is included in Appendix A.

STATES INCLUDED

NavCov initializes, propagates, and updates the covariance for a number of states that represent inertial navigator and updating sensor measurement errors. It is possible to include a vast number of states in pursuit the highest possible fidelity; however, depending on the application, many of these states are of little consequence to the actual performance of the inertial system under consideration. For instance, g-sensitive and g^2-sensitive gyro errors are negligible for a slow land vehicle application. The number of states also impacts the speed at which the code runs due to the many matrix multiplys and the matrix inversion that are a part of the Kalman filter equations. Therefore, the states included in NavCov are those that have the greatest potential to affect the performance of an aided inertial navigator used for the slow land application. In addition, most of these states can be enabled or disabled to explore the sensitivity of inertial navigator performance to the inclusion or exclusion of various states. It should be noted that not all sensor measurements have associated error states.

NavCov computes the covariance for the states listed below. The numbering shown is for convenience only. The actual numbering depends on which states are enabled and disabled.

Basic 9 inertial navigator error states, which cannot be disabled:

1. latitude error
2. longitude error

3. height, or altitude, error
4. north velocity error
5. east velocity error
6. down velocity error
7. north tilt error, which is the angular error about the north axis
8. east tilt error, which is the angular error about the east axis
9. heading error, which is the angular error about the down axis

Gyroscope sensor errors (for 3 gyroscopes)

10.-12. gyro bias errors
13.-15. gyro scale factor errors
16.-21. gyro misalignments

Accelerometer sensor errors (for 3 accelerometers)

22.-24. accelerometer bias errors
25.-27. accelerometer scale factor errors
28.-33. accelerometer misalignments

Baro-altimeter damping loop errors

34. error in integral of difference between inertial and baro altitudes
35. baro-altimeter scale factor error
36. baro-altimeter error due to constant-pressure surface altitude variation

Odometer velocity measurement error

37. odometer velocity measurement bias

Compass heading measurement error

38. compass heading measurement bias

Inclinometer attitude measurement errors

39. x-axis attitude (roll) measurement bias
40. y-axis attitude (pitch) measurement bias

A criticism of covariance analysis is that it often produces optimistic results. NavCov addresses this by allowing for use of a suboptimal Kalman filter, where some of the states are omitted for the purpose of computing Kalman gains. The states that might be omitted are those that are felt to be non-estimatable due to highly nonlinear, non-repeatable behavior. Such states might include gyro bias for inexpensive, low quality IMUs.

The states that are automatically omitted in the suboptimal Kalman filter include:

Baro-altimeter damping loop errors
Odometer velocity measurement error

The states that may be omitted in the suboptimal Kalman filter include:

Gyroscope sensor errors (for 3 gyroscopes)
Accelerometer sensor errors (for 3 accelerometers)
Compass heading measurement error
Inclinometer attitude measurement errors

INSTALLING THE CODE

The code is delivered as a zip file. Place the zip file in an empty directory and unzip it to that same directory. Since NavCov is expected to evolve with use on specific applications, the zip file will eventually become obsolete and can be discarded. Periodic backups of current work are therefore a good idea.

STRUCTURE OF THE CODE

NavCov is a fairly simple, serially executed (single thread) code. In almost all cases, scripts are used rather than functions; therefore, information is passed between the various parts of the code using global variables. (We will pause here to allow the more sophisticated programmers to recover after keeling over in disgust.) On the positive side, the code is at least functionally modular and it is hoped that the code is fairly easy to interpret and modify. The primary sources of complexity are the ability to enable, disable, and schedule aiding sources, and the ability to run a suboptimal filter.

The main script is called navcov.m. It calls other scripts to perform the various steps required during a covariance analysis run. Most of the scripts performing computations are in the same directory with navcov.m. Inputs to the code are placed in various files contained in the subdirectory .\input_files. Results from running the code are placed in the subdirectory .\output_files. Plotting parameters are placed in the subdirectory .\plot_limits.

The following flow chart describes the basic functionality of the code.

APPENDIX A to NavCov Users' Manual: MEASUREMENT MODELS INCLUDED IN VERSION 1.0

BARO-ALTIMETER MEASUREMENT

The baro-altimeter measurement is incorporated in a third-order damping loop integrated with the inertial navigator rather than in the Kalman filter. This mechanization is documented in both [ref 1] and [ref 2]. The error model for the baro-altimeter measurement is also documented in [ref 1]. The various gains, constants, and initial error and process noise variances for this model are contained in the Matlab script .\input_files\meas_models\baro_model.m.

NavCov includes error states for 2 of the 4 baro-altimeter error sources shown in [ref 1]. A third error state is also included as a result of the third order damping loop mechanization. Those states are:

1) e_{p_0}, the error due to variation in altitude of a constant pressure surface, and
2) e_{hsf}, the error due to non-standard temperature, modeled as a scale factor error
3) δa, the integral of the difference between the indicated and reference altitudes

The two remaining error sources listed in [ref 1] are considered negligible for the land vehicle application and so are left out of NavCov. The baro altimeter error dynamics equations used are:

$$\dot{e}_{p_0} = \frac{-V_H}{d_{alt}} e_{p_0} + w(t) \tag{A-1}$$

$$\mathrm{E}[w(t) \cdot w(t)] = \frac{2 * V_H}{d_{alt}} \cdot \sigma^2{}_{alt} \tag{A-2}$$

$$\dot{e}_{hsf} = 0 \tag{A-3}$$

$$\dot{\delta a}_D = k_3 \cdot \delta h_D + k_3 \cdot e_{p_0} + k_3 \cdot e_{hsf} \cdot \Delta h \tag{A-4}$$

$$\dot{\delta V}_D = \dot{\delta V}_{D_{orig.terms}} - k_2 \cdot \delta h_D - k_2 \cdot e_{p_0} - k_2 \cdot e_{hsf} \cdot \Delta h - \delta a_D \tag{A-5}$$

$$\dot{\delta h}_D = \dot{\delta h}_{D_{orig.terms}} - k_1 \cdot \delta h_D - k_1 \cdot e_{p_0} - k_1 \cdot e_{hsf} \cdot \Delta h \tag{A-6}$$

where V_H is the horizontal velocity of the vehicle, d_{alt} is a constant representing the correlation distance of weather patterns, $w(t)$ is a white noise process, E is the expected value function, σ_{alt} is the standard deviation of the variation in altitude of a constant pressure surface, k_1, k_2, and k_3 are constant gain coefficients, δh_D is the height error (where the D subscript reminds us that this error is positive down), and Δh is the change in altitude since the baro-altimeter was initialized. The terms with the subscript "*orig. terms*" denote use of the original terms from the **F** matrix for the undamped inertial navigator.

The error source e_{p_0} is modeled as a first order Markov (exponentially correlated) process. The error source e_{hsf} is modeled as a random constant scale factor error. The terms in equations (A-1) and (A-3 through 6) are included in **F** matrix. The term in equation (A-2) is included in the **Q** matrix. There are some sign differences between these equations and those shown in [ref 1] since NavCov uses an NED coordinate frame rather than ENU. Also, the model used in NavCov assumes that the baro-altimeter can be initialized at any altitude rather than sea level using some means such as GPS. This leads to use of Δh rather than h in the above equations.

ODOMETER VELOCITY MEASUREMENT

The odometer is used as a velocity-measuring sensor by NavCov. This velocity measurement is included in the Kalman filter. Its P, Q, and R values are set in .\input_files\meas_models\odomvel_model.m.

The odometer velocity measurement error is modeled as a random constant plus a random walk. The error dynamics equation for this model is:

$$\dot{e}_{vod} = w(t) \tag{A-7}$$

where $w(t)$ is a white noise process. This model is implemented in the **F** and **Q** matrices. One error state results from this model. It is assumed that this error state is poorly estimate-able (due to wheel slippage), so it is not included in the suboptimal Kalman filter.

The measurement model used in NavCov is:

$$z = V_x + e_{vod} + v(t) \tag{A-8}$$

where V_X is the velocity of the vehicle along the longitudinal, or X axis, and $v(t)$ is a white noise process. This model is of the form of equation (2), except that V_X must be expressed in terms of the states V_N, V_E, and V_D (since V_X is not a state used in NavCov). That relationship is defined via the geographic-to-body direction cosine matrix (DCM) C_{gtob} as:

$$\begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} = C_{gtob} \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix}, \quad \text{or} \quad V_X = C_{gtob}(1,1) \cdot V_N + C_{gtob}(1,2) \cdot V_E + C_{gtob}(1,3) \cdot V_D \tag{A-9}$$

The matrix C_{gtob} is the transpose of the matrix C_{btog}, which is provided as an input to NavCov by the separate 6DOF simulation. The function **h**(**x**) in the form of equation (2) is then:

$$\mathbf{h}(\mathbf{x}) = C_{gtob}(1,1) \cdot V_N + C_{gtob}(1,2) \cdot V_E + C_{gtob}(1,3) \cdot V_D + e_{vod} \tag{A-10}$$

Then, remembering per equations (11) and (12) that the observation matrix **H** is the partial derivative $\partial \mathbf{h}(\mathbf{x}) / \partial \mathbf{x}|_{\mathbf{x}=\mathbf{x}^*}$, the following result is obtained:

$$\mathbf{H} = [0 \quad 0 \quad 0 \quad C_{gtob}(1,1) \quad C_{gtob}(1,2) \quad C_{gtob}(1,3) \quad 0 \quad 0 \quad 0 \quad \ldots \quad 1 \quad \ldots] \tag{A-11}$$

with respect to the error state vector:

$$\mathbf{e} = [\delta lat \quad \delta lng \quad \delta hgt \quad \delta V_N \quad \delta V_E \quad \delta V_D \quad \varepsilon_N \quad \varepsilon_E \quad \varepsilon_D \quad \ldots \quad e_{vod} \quad \ldots]^T \tag{A-12}$$

H matrix elements not shown are 0. This result is implemented in NavCov's **H** matrix along with similar results for the other measurements described below.

COMPASS HEADING MEASUREMENT

The compass is used as a heading-measuring sensor by NavCov. This heading measurement is included in the Kalman filter. Its P, Q, and R values are set in .\input_files\meas_models\comphdg_model.m.

The compass heading measurement error is modeled as a random constant plus a random walk. The random walk contribution is increased with increasing heading rate as a way to cause the filter to re-estimate the bias when heading changes. It is assumed that the compass is at least somewhat calibrated. The error dynamics equation for this model is:

$$\dot{e}_{com} = w(t) \tag{A-13}$$

where *w(t)* is a white noise process. This model is implemented in the **F** and **Q** matrices. One error state results from this model. It is assumed that this error state is predictable, so it is included in the suboptimal Kalman filter.

The Q value for this error model, which is the expected value E[*w(t)* •*w(t)*], is scaled between a minimum value when heading rate is zero and a maximum rate when heading rate reaches an assumed maximum angular velocity. The equation defining this relationship is:

$$Q_{ecom} = (1 + K_{com} \cdot \frac{\omega_D}{\omega_{D_{\max}}}) \cdot Q_{ecom_{\min}} \tag{A-14}$$

where K_{com} is a scaling constant, $Q_{ecom_{min}}$ and $\omega_{D_{max}}$ are assumed constants, and ω_D is the vehicle's heading rate. ω_D is computed using the body-to-geographic DCM from the body angular rates contained in the input trajectory data as:

$$\begin{bmatrix} \omega_N \\ \omega_E \\ \omega_D \end{bmatrix} = C_{btog} \begin{bmatrix} \omega_X \\ \omega_Y \\ \omega_Z \end{bmatrix}, \quad \text{or} \quad \omega_D = C_{btog}(3,1) \cdot \omega_X + C_{btog}(3,2) \cdot \omega_Y + C_{btog}(3,3) \cdot \omega_Z \tag{A-15}$$

The measurement model used in NavCov is:

$$z = \varepsilon_D + e_{com} + v(t) \tag{A-16}$$

where ε_D is the heading of the vehicle, and $v(t)$ is a white noise process.

The function $\mathbf{h}(\mathbf{x})$ in the form of equation (2) is then:

$$\mathbf{h}(\mathbf{x}) = \varepsilon_D + e_{com} \tag{A-17}$$

Then the observation matrix $\mathbf{H}$, computed as the partial derivative $\partial\mathbf{h}(\mathbf{x})/\,\partial\mathbf{x}|_{\mathbf{x}=\mathbf{x}^*}$, is:

$$\mathbf{H} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad \ldots \quad 1 \quad \ldots] \tag{A-18}$$

with respect to the error state vector:

$$\mathbf{e} = [\delta lat \quad \delta lng \quad \delta hgt \quad \delta V_N \quad \delta V_E \quad \delta V_D \quad \varepsilon_N \quad \varepsilon_E \quad \varepsilon_D \quad \ldots \quad e_{com} \quad \ldots]^T \tag{A-19}$$

$\mathbf{H}$ matrix elements not shown are 0.

2-AXIS INCLINOMETER ATTITUDE MEASUREMENTS

The 2-axis inclinometer is used as a roll and pitch-measuring sensor by NavCov. These two attitude measurements are included in the Kalman filter. Their P, Q, and R values are set in .\input_files\meas_models\inclatt_model.m.

The inclinometer attitude measurement errors are each modeled as a random constant plus a random walk. The error dynamics equations for this model are:

$$\dot{e}_{incX} = w_X(t) \qquad \text{(roll measurement error)} \tag{A-20}$$

$$\dot{e}_{incY} = w_Y(t) \qquad \text{(pitch measurement error)} \tag{A-21}$$

where $w_X(t)$ and $w_Y(t)$ are white noise processes. This model is implemented in the **F** and **Q** matrices. Two error states results from this model. It is assumed that these error states are predictable, so they are included in the suboptimal Kalman filter.

The measurement model used in NavCov is:

$$\mathbf{z} = \begin{bmatrix} \varepsilon_X + e_{incX} + v_X(t) \\ \varepsilon_Y + e_{incY} + v_Y(t) \end{bmatrix} \tag{A-22}$$

where ε_X is the roll of the vehicle, ε_Y is the pitch of the vehicle, and $v_X(t)$ and $v_Y(t)$ are white noise processes. ε_X and ε_Y need to be expressed in terms of the states ε_N, ε_E, and ε_D through use of the geographic-to-body DCM as:

$$\begin{bmatrix} \varepsilon_X \\ \varepsilon_Y \\ \varepsilon_Z \end{bmatrix} = C_{gtob} \begin{bmatrix} \varepsilon_N \\ \varepsilon_E \\ \varepsilon_D \end{bmatrix} \tag{A-23}$$

Then the function **h**(**x**) in the form of equation (2) is:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} C_{gtob}(1,1)\cdot\varepsilon_N + C_{gtob}(1,2)\cdot\varepsilon_E + C_{gtob}(1,3)\cdot\varepsilon_D + e_{INCX} \\ C_{gtob}(2,1)\cdot\varepsilon_N + C_{gtob}(2,2)\cdot\varepsilon_E + C_{gtob}(2,3)\cdot\varepsilon_D + e_{INCY} \end{bmatrix} \tag{A-24}$$

Then the observation matrix **H**, computed as the partial derivative $\partial\mathbf{h}(\mathbf{x})/\partial\mathbf{x}|_{\mathbf{x}=\mathbf{x}^*}$, is:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & C_{gtob}(1,1) & C_{gtob}(1,2) & C_{gtob}(1,3) & \dots & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & C_{gtob}(2,1) & C_{gtob}(2,2) & C_{gtob}(2,3) & \dots & 0 & 1 & \dots \end{bmatrix} \tag{A-25}$$

with respect to the error state vector:

$$\mathbf{e} = [\delta lat \quad \delta lng \quad \delta hgt \quad \delta V_N \quad \delta V_E \quad \delta V_D \quad \varepsilon_N \quad \varepsilon_E \quad \varepsilon_D \quad \dots \quad e_{incX} \quad e_{incY} \quad \dots]^T \tag{A-26}$$

H matrix elements not shown are 0.

GPS POSITION MEASUREMENTS

GPS is used as a three dimensional position-measuring sensor by NavCov. These position measurements are included in the Kalman filter. No error states are associated with this simple mechanization. The R values for the position measurements are set in .\input_files\meas_models\gpspos_model.m.

The measurement model used in NavCov is:

$$\mathbf{z} = \begin{bmatrix} lat + v_N(t) \\ lng + v_E(t) \\ hgt + v_D(t) \end{bmatrix} \tag{A-27}$$

where $v_N(t)$, $v_E(t)$, and $v_D(t)$ are white noise processes. Since the latitude and longitude error states are in units of radians while the measurement noise is specified in units of meters, rθ-type unit conversions are required.

The function $\mathbf{h}(\mathbf{x})$ in the form of equation (2) is:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} R_E \cdot lat \\ R_E \cdot \cos(lat) \cdot lng \\ hgt \end{bmatrix} \tag{A-28}$$

Then the observation matrix $\mathbf{H}$, computed as the partial derivative $\partial\mathbf{h}(\mathbf{x})/\,\partial\mathbf{x}|_{\mathbf{x}=\mathbf{x}^*}$, is:

$$\mathbf{H} = \begin{bmatrix} R_E & 0 & 0 & \ldots \\ 0 & R_E \cdot \cos(lat) & 0 & \ldots \\ 0 & 0 & 1 & \ldots \end{bmatrix} \tag{A-29}$$

with respect to the error state vector:

$$\mathbf{e} = [\delta lat \quad \delta lng \quad \delta hgt \quad \ldots] \tag{A-30}$$

$\mathbf{H}$ matrix elements not shown are 0. The units of δlat and δlng are radians.

ZERO VELOCITY UPDATING MEASUREMENTS

Zero Velocity Updating (ZVUPT) is a powerful technique that can be used to identify tilt and velocity errors when a vehicle is stopped. The tilt errors are made observable because the IMU's accelerometers provide an accurate measurement of the gravity vector and therefore the vehicle's pitch and roll angles. The velocity errors are made observable because, during a stop, the vehicle's velocity is very accurately known to be zero. Given frequent stops (allowing an assumption that velocity error rate, or acceleration error, is constant), quadratic position errors can also be estimated as the double integration of the constant acceleration error.

NavCov includes ZVUPT to update the 2 vehicle tilts, 3 dimensions of velocity, and 3 dimensions of position. The measurements are incorporated in the Kalman filter. No error

states are associated with this mechanization. The R values for the position measurements are initialized in .\input_files\meas_models\zvu_model.m. and updated as necessary in .\update_R_ZVUPOS.m and .\update_R_ZVUPOS_subopt.m. ZVUPT of tilts, velocity, and position can be enable and disabled separately.

For the tilt measurements, the measurement model used in NavCov is:

$$\mathbf{z} = \begin{bmatrix} \varepsilon_X + v_X(t) \\ \varepsilon_Y + v_Y(t) \end{bmatrix} \tag{A-31}$$

where ε_X is the roll of the vehicle, ε_Y is the pitch of the vehicle, and $v_X(t)$ and $v_Y(t)$ are white noise processes. ε_X and ε_Y need to be expressed in terms of the states ε_N, ε_E, and ε_D through use of the geographic-to-body DCM as:

$$\begin{bmatrix} \varepsilon_X \\ \varepsilon_Y \\ \varepsilon_Z \end{bmatrix} = C_{gtob} \begin{bmatrix} \varepsilon_N \\ \varepsilon_E \\ \varepsilon_D \end{bmatrix} \tag{A-32}$$

Then the function $\mathbf{h}(\mathbf{x})$ in the form of equation (2) is:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} C_{gtob}(1,1)\cdot\varepsilon_N + C_{gtob}(1,2)\cdot\varepsilon_E + C_{gtob}(1,3)\cdot\varepsilon_D \\ C_{gtob}(2,1)\cdot\varepsilon_N + C_{gtob}(2,2)\cdot\varepsilon_E + C_{gtob}(2,3)\cdot\varepsilon_D \end{bmatrix} \tag{A-33}$$

Then the observation matrix $\mathbf{H}$, computed as the partial derivative $\partial\mathbf{h}(\mathbf{x})/\partial\mathbf{x}|_{\mathbf{x}=\mathbf{x}^*}$, is:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & C_{gtob}(1,1) & C_{gtob}(1,2) & C_{gtob}(1,3) & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & C_{gtob}(2,1) & C_{gtob}(2,2) & C_{gtob}(2,3) & \cdots \end{bmatrix} \tag{A-34}$$

$\mathbf{H}$ elements not shown are 0. For the velocity measurements, the measurement model used in NavCov is:

$$\mathbf{z} = \begin{bmatrix} V_N + v_N(t) \\ V_E + v_E(t) \\ V_D + v_D(t) \end{bmatrix} \tag{A-35}$$

where $v_N(t)$, $v_E(t)$, and $v_D(t)$ are white noise processes. The function $\mathbf{h}(\mathbf{x})$ in the form of equation (2) is:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix} \tag{A-36}$$

Then the observation matrix **H**, computed as the partial derivative $\partial\mathbf{h}(\mathbf{x})/\,\partial\mathbf{x}|_{\mathbf{x}=\mathbf{x}^*}$, is:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \ldots \end{bmatrix} \tag{A-37}$$

H elements not shown are 0. Finally for the position estimate, the measurement model used in NavCov is:

$$\mathbf{z} = \begin{bmatrix} lat + v_N(t) \\ lng + v_E(t) \\ hgt + v_D(t) \end{bmatrix} \tag{A-38}$$

where $v_N(t)$, $v_E(t)$, and $v_D(t)$ are white noise processes. Again, since the latitude and longitude error states are in units of radians while the measurement noise is specified in units of meters, rθ-type unit conversions are required.

The function $\mathbf{h}(\mathbf{x})$ in the form of equation (2) is:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} R_E \cdot lat \\ R_E \cdot \cos(lat) \cdot lng \\ hgt \end{bmatrix} \tag{A-39}$$

Then the observation matrix **H**, computed as the partial derivative $\partial\mathbf{h}(\mathbf{x})/\,\partial\mathbf{x}|_{\mathbf{x}=\mathbf{x}^*}$, is:

$$\mathbf{H} = \begin{bmatrix} R_E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & R_E \cdot \cos(lat) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \end{bmatrix} \tag{A-40}$$

H elements not shown are 0. If ZVUPT of position, velocity, and tilts are all enabled, the combined **H** matrix is:

$$\mathbf{H} = \begin{bmatrix} R_E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & R_E \cdot \cos(lat) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & 0 & 0 & C_{gtob}(1,1) & C_{gtob}(1,2) & C_{gtob}(1,3) & \ldots \\ 0 & 0 & 0 & 0 & 0 & 0 & C_{gtob}(2,1) & C_{gtob}(2,2) & C_{gtob}(2,3) & \ldots \end{bmatrix} \quad \text{(A-41)}$$

with respect to the error state vector:

$$\mathbf{e} = [\delta lat \quad \delta lng \quad \delta hgt \quad \delta V_N \quad \delta V_E \quad \delta V_D \quad \varepsilon_N \quad \varepsilon_E \quad \varepsilon_D \quad \ldots]^T \quad \text{(A-42)}$$

Some additional heuristics are added in the mechanization of ZVUPT of position. The purpose of these heuristics is to parallel the operation and performance of a real-time system utilizing ZVUPT. The mechanization logic includes the following:

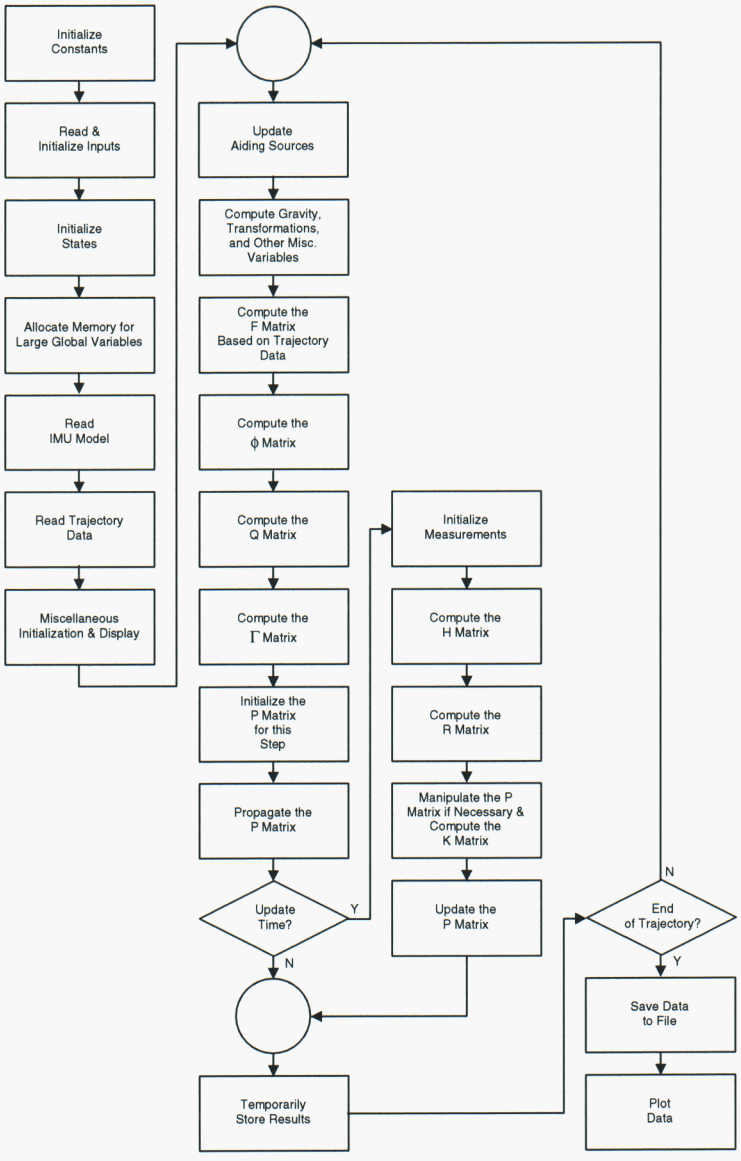
1. Position updating is performed only for the second and subsequent stops after ZVUPT is enabled.

2. The R value for position updating is not expected to be any better than the variance of the position uncertainty at the time when ZVUPT is enabled. Therefore, when ZVUPT is disabled, the R value is matched to the P value for position uncertainty.

3. When the vehicle is stopped and ZVUPT is enabled, the R value for position updating is expected to remain constant. It is therefore held constant during stops.

4. The ZVUPT position update variance for subsequent stops is expected to degrade over time. Therefore, the R value for position updating is incremented after each stop is completed.

This logic is included in .\update_R_ZVUPOS.m and .\update_R_ZVUPOS_subopt.m.



NavCov Flow Chart

INPUTS TO THE CODE

NavCov requires numerous inputs. They are:

1) a Matlab script (.\input_files\inputs.m) containing input definitions. The primary definitions included in this file are:
   a) flag enabling suboptimal filtering
   b) string defining the name of the IMU model
   c) string defining the name of the trajectory
   d) string defining the name of the aiding schedule
   e) variable setting the data rate
   f) variable setting the number of steps between measurements
   g) flags enabling sensor error states
   h) flags enabling measurements

2) a text file containing the initial error and process noise standard deviations for IMU sensor errors (.\input_files\imu_models*.txt). The format of this file follows long-standing convention in Sandia's navigation departments.

3) a Matlab script (.\input_files\imu_models*_pvapq.m) defining the initial position, velocity, and attitude error standard deviations for the particular IMU and application under consideration. These numbers are set to reflect the initialization (i.e. alignment) results expected for the application.

4) a Matlab data file (.\input_files\trajectories*.mat) containing vehicle trajectory data. The frequency, or data rate, of the trajectory data should adequately capture the dynamics of vehicle movement as well as provide some oversampling relative to the frequency of use of the updating sensors. For each time step, the data includes by variable name:
   a) traj_time time (seconds)
   b) traj_p_geo earth frame latitude (rad), longitude (rad), and height (m)
   c) traj_v_geo geographic frame north, east, and down velocities (m/s)
   d) traj_a_body body frame x, y, and z specific forces (accelerations not including gravity, m/s^2)
   e) traj_w_body body frame x, y, and z angular rates (rad/s)
   f) traj_C_btog direction cosine matrix describing the body to geographic transformation
   g) traj_euler euler angles roll, pitch, and yaw (rad)

5) a Matlab script (.\input_files\trajectories*.m) defining the aiding measurement schedule.

6) Matlab scripts (.\input_files\meas_models*.m) defining parameters for each aiding measurement error model.

MANIPULATION OF THE INPUT FILES AND TUNING

Again, the purpose of a covariance analysis code is either to predict the statistical performance of some aided inertial navigation system under consideration without extensive modeling and simulation, or conversely, to determine the required error budget of an aided inertial navigation system to meet some contemplated performance goal. As part of either use, a sensitivity analysis of various parameters can be achieved by making repeated small changes to the parameters of interest and observing the impact these changes have on the variances of the error states. As can be imagined, this is an iterative, possibly tedious, process. The parameters being changed are primarily the initial error state variances (i.e. the P's), the process noise variances (i.e. the Q's), and the measurement noise variances (i.e. the R's). Changes to the trajectory, the aiding schedule, and the measurement frequency can also be contemplated. These changes are all made through the input files listed above.

Before NavCov is used for analysis purposes, various input parameters need to be "tuned". The purpose of tuning is to achieve both reasonable and mathematically stable results from the equations contained in the code. "Reasonable" implies a subjective judgment based on experience - there is some art involved. Tuning guidance is provided in Appendix B. In a nutshell, the process of tuning is iterative and requires repeated manipulation of the input parameters in a systematic fashion, running of the code, evaluation of the results, and repeat until satisfactory results are achieved. The code as delivered has been tuned to achieve reasonable results for the example trajectories and the example sensors included with version 1.0. The tuning can be refined once the analysis "matrix" (i.e. the menu of trajectories, IMU quality, and aiding sensors, schedules, and frequency to be considered) is determined.

After the code is tuned, the analysis matrix can be pursued case-by-case by altering input parameters as necessary and running NavCov for each case. The parameters in the input files are altered using either the Matlab editor or a text editor. The required units for each parameter are documented in the input files.

RUNNING THE CODE

The code is simple to run as follows:

start Matlab
>> cd <directory where code is installed>
>> navcov

OUTPUTS OF THE CODE

Each time NavCov is run, it produces output to Matlab window that appears as follows:

```
>> navcov

   *** COVARIANCE ANALYSIS CODE FOR:  ***
DISTRIBUTED AUTONOMOUS NAVIGATION LDRD

SUBoptimal Filtering
Trajectory          - re2
Aiding Schedule     - re2_sched1
IMU                 - ln200
Data Rate           -  2.0 Hz
Measurement Rate -  1.0 Hz
Aiding Sources:
BARO_ALT ODOM_VEL COMP_HDG INCL_ATT GPS_POS   ZVUPT
      0         0        0        0       1       1

step #  10 completed, time_tag =    4.50
step #  20 completed, time_tag =    9.50
step #  30 completed, time_tag =   14.50
step #  40 completed, time_tag =   19.50
step #  50 completed, time_tag =   24.50
...
step # 170 completed, time_tag =   84.50

Finished
Last step # 179, time_tag =   89.00
```

The purpose of the above output is to provide the user with some feedback showing a few of the input parameters being used (for a sanity check) and also a heartbeat to indicate that the code is indeed running. A 2 second pause occurs before the step counter is displayed. The 0's and 1's following the "Aiding Sources" message indicate which aiding sources are disabled and enabled. In this example, GPS position updates and zero velocity updating are enabled. After the "Last step" message is displayed, the following query is displayed to the user:

Enter a filename for storage of output:

to which the user replies as desired:

Enter a filename for storage of output: **test**

This causes two files to be written to the .\output_files directory. They are test.mat and test-inputs.m. The Matlab data file test.mat contains the results of the covariance analysis run, including by variable name:

SUBOPT	flag indicating whether suboptimal filtering was used
MODEL	string defining the IMU model
TRAJ	string defining the trajectory
meas_rate	constant containing the frequency of updates (Hz)
sim_time	vector containing time (seconds)
traj_data	matrix of vectors containing a copy of the trajectory data input to NavCov, minus time (units same as previously described)
stdev_data	matrix of vectors containing the standard deviations for the states, taken from the diagonal of the full state covariance matrix P (same units as for trajectory data)
stdev_data_subopt	matrix of vectors containing the standard deviations for the states, taken from the diagonal of the reduced order covariance matrix P_subopt (same units as for trajectory data)
aid_data	matrix of flags containing the aiding schedule

The variable stdev_data_subopt is not saved if suboptimal filtering is not used. Also, as a note of explanation, the reduced order covariance matrix, containing fewer states selected as noted earlier, is used to generate the suboptimal Kalman gains when suboptimal filtering is called for. Those suboptimal gains are then used to update the full state covariance matrix (as well as the reduced order covariance matrix).

The Matlab script file test-inputs.m contains a copy of .\input_files\inputs.m for documentation purposes.

The following message then appears in the Matlab window:

Enter a filename for data to be plotted or
press <Enter> if data already loaded:

This message originates in the plot_stuff.m script. The option allows plot_stuff.m to be used independently to read past results from the .\output_files directory and replot them. Since NavCov just produced the results, which still reside in memory, the user can respond by pressing the <Enter> key. Another message from plot_stuff.m is then displayed:

Enter annotation for plots:

to which the user once again replies as desired:

Enter annotation for plots: **test**

A number of plots are then produced. Since suboptimal filtering was used in this example, 4 plots result.

Figure 1 displays the trajectory. The upper left subplot shows a top-down view of the ground track in kilometers relative to the starting position. The ground track is time tagged every 15 seconds in this example. The upper right subplot shows an elevation view of the trajectory vs. time. The elevation units are in kilometers relative to the starting elevation. Again, the trajectory line is time tagged. The lower left subplot shows the north, east, and down velocity profiles vs. time in units of meters/second. The lower right subplot shows the roll, pitch, and heading angles in degrees. The trajectory filename is displayed in the lower left corner.

APPENDIX B to NavCov Users' Manual. - TUNING GUIDELINES

These tuning guidelines assume that the user has been through the user manual and has developed some experience with the code structure and input files. Tuning of the covariance code is a trial-and-error process, involving more art than engineering to achieve stable and realistic results. Here are some helpful guidelines to get started on tuning. The figures mentioned below do not correspond to those in the main body of the user manual; rather, they correspond to the figures generated by running the code.

1. First, the unaided IMU tuning needs to be addressed.

   a. An IMU model defining sensor Ps and Qs is a given (in a text file). Hopefully it is derived from actual test data, but sometimes its fabricated based on manufacturer's specs.

      -> model is contained in text file .\input_files\imu_models\xxx.txt

      xxx is used here and below as a placeholder for a text string or strings. Its meaning is context dependent, but will be obvious from inspection of the input files.

   b. Define the expected initial position, velocity, and attitude (PVA) Ps and Qs. This usually comes from empirical knowledge of how the navigator is initialized. A good starting point for the Qs is to set them to be P/(1000^2).

      -> in the Matlab script .\inputs_files\imu_models\xxx_pvapq.m:
      set values for P0_xxx and Q_xxx

   c. Set up the covariance code to run with all states (also referred to as full state or optimal) and with no aiding.

      -> in the Matlab script .\inputs_files\inputs.m:
      set SUBOPT = NO
      set MEAS_STEP = Inf
      set GYRO_xxx and ACCEL_xxx = ON
      set BARO_ALT and other aiding sources = NO

   d. Run the covariance code to obtain the standard deviation plots (referred to as FIGURE 2 below) for PVA for the given inputs provided for a. and b. plus a given vehicle trajectory. The plots will be in the Matlab Figure No. 2 window, titled UNAIDED. Print this figure for later reference.

   e. Now, define the expected initial PVA Ps and Qs for the suboptimal case (reduced number of states, also referred to as reduced order) case. Start by setting them to be the same as for the full state case.

-> in the Matlab script .\inputs_files\imu_models\xxx_pvapq.m:
set values for P0_xxx_SUBOPT and Q_xxx_SUBOPT equal to P0_xxx and Q_xxx

f. Set up the covariance code to run suboptimal and with no aiding. Usually (i.e. for low-cost IMUs), the IMU sensor error sources will not be included in the reduced states.

-> in the Matlab script .\inputs_files\inputs.m:
set SUBOPT = YES
set MEAS_STEP = Inf
set rGYRO_xxx and rACCEL_xxx = OFF
set BARO_ALT and other aiding sources = NO

g. Run the covariance code again to obtain the standard deviation plots for PVA. There will be two PVA plots generated this time - one that is derived from the reduced order P matrix (referred to as FIGURE 3) and one that is derived from the suboptimal full state P matrix (referred to as FIGURE 2sub). These plots are displayed in two Matlab Figure windows, i.e. No. 2, titled UNAIDED, and Figure No. 3, titled Reduced States Results.

h. Compare the Tilt Errors subplot of FIGURE 3 to its counterpart in FIGURE 2. The goal is to match the Tilt uncertainties in FIGURE 3 to that in FIGURE 2 at some point in time. The point in time chosen would normally be related to the length of time between updates from whatever aiding source is being contemplated. Or in other words, the time that the navigator would be running free inertial. This is where the trial-and-error begins. Start by increasing Q_TILT_SUBOPT in the xxx_pvapq.m script.

-> in the Matlab script .\inputs_files\imu_models\xxx_pvapq.m:
increase value of Q_TILT_SUBOPT

i. Rerun the covariance code to obtain the standard deviation plots for PVA. Again compare the Tilt Errors subplots of FIGURE 3 and FIGURE 2. Continue to increase Q_TILT_SUBOPT, rerun the code, and compare the plotted results until an adequate match is achieved.

Note that for poorer quality IMUs, it is sometimes not possible to obtain a close match. Also note that there will not be much sensitivity to this tuning until the Q value is close to the value being sought. Once the Q value is close, there is a lot of sensitivity.

j. Now perform steps 1h. and 1i. for the Azimuth Error uncertainty. The Q value to manipulate is Q_HDG_SUBOPT. Iterate until an adequate match is achieved.

k. Now perform steps 1h. and 1i. for the Velocity Error uncertainties. The two Q values to manipulate are Q_HVEL_SUBOPT and Q_DVEL_SUBOPT. Iterate until an adequate match is achieved.

l. Now perform steps 1h. and 1i. for the Position Error uncertainties. The two Q values to manipulate are Q_HPOS_SUBOPT and Q_DPOS_SUBOPT. Iterate until an adequate match is achieved.

   Note that the Position Error uncertainties may match fairly closely before tuning the Q_HPOS_SUBOPT and Q_DPOS_SUBOPT values. Go ahead and increase those values until the match begins to diverge.

m. At this point, all subplots of FIGURE 3 and FIGURE 2 should match as closely as possible at the chosen point in time. The above procedure manipulated the Q_xxx_SUBOPT values only. It might be necessary to also manipulate the P_xxx_SUBOPT values to achieve stable results.

2. Now each aiding source needs to be tuned. The required tuning will depend on how the aiding source is modeled. This process is also iterative; however, it is less procedural than above.

   a. For example, GPS position aiding is modeled as a simple position update without any additional error states. The only tuning required is to set the measurement noise uncertainty, R_xxx and R_xxx_SUBOPT. These should be set based on knowledge of the aiding source measurement accuracy. Since there are no error states associated with this type of aiding source, normally R_xxx_SUBOPT is set equal to R_xxx_SUBOPT.

      -> model is contained in a Matlab script .\input_files\meas_models\xxx_model.m
      in this script:
      set values for R_xxx and R_xxx_SUBOPT

   b. Another example is odometer velocity aiding. It is modeled as a measurement with a random bias and a random walk. An error state is included for the bias. This model leads to three tuning parameters - a P_xxx value, a Q_xxx value, and an R_xxx value. As the error state is only included in the optimal filter, there are no suboptimal counterparts for the P and Q values. (Other aiding sources do have error states included in the suboptimal filter.) However, there is an R_xxx_SUBOPT value. To tune...

   c. Set the P_xxx and the R_xxx values equal to the expected measurement noise uncertainty. Set the Q_xxx value as well. A good starting value is P_xxx/(1000^2). This will ensure that the filter is stable and it can be increased from there later.

-> again, model is contained in a Matlab script
.\input_files\meas_models\xxx_model.m
in this script:
set values for P_xxx, Q_xxx, and R_xxx

d. Set up the covariance code to run optimal (full state) and with the desired aiding.

-> in the Matlab script. \inputs_files\inputs.m:
set SUBOPT = NO
set MEAS_STEP = xxx (set to the number of time steps between updates)
set the aiding source, for example ODOM_VEL, = YES

-> in the Matlab script .\input_files\trajectories\xxx_schedx.m
make sure that the aiding schedule turns on the aiding source

e. Run the covariance code to obtain the standard deviation plots (referred to as FIGURE 2) for PVA. The plots will be in the Matlab Figure No. 2 window, titled UNAIDED. Print this figure for later reference.

f. Bring up another Matlab figure and also plot out the variance vs. time for the error state(s) associated with the aiding source (referred to as FIGURE 5).

-> figure(5)

-> plot(sim_time, stdev_data(:, xxx) where xxx is the state number

g. Inspect FIGURE 5 to make sure that the uncertainty for the aiding source error state does not decline too quickly (which indicates an overly optimistic assessment of the filter's ability to estimate the error, and too little Q) or ramp off too quickly (which indicates that the filter is unstable, or too much Q).

h. If necessary, iteratively adjust the Q value, rerun the covariance code, and replot the variance(s) for the aiding source error state(s) until an acceptable variance plot is achieved. "Acceptable" is a subjective judgment on the part of the engineer.

i. Now set the suboptimal tuning parameters. Start with P_xxx_SUBOPT = P_xxx (if included), Q_xxx_SUBOPT = Q_xxx (if included), and R_xxx_SUBOPT = R_xxx.

-> again, model is contained in a Matlab script
.\input_files\meas_models\xxx_model.m
in this script:
set values for P_xxx_SUBOPT, Q_xxx_SUBOPT, and R_xxx_OPT

j. Set up the covariance code to run suboptimal (reduced number of states).

-> in the Matlab script .\inputs_files\inputs.m:
set SUBOPT = YES
set MEAS_STEP = xxx (set to the number of time steps between updates)
set the aiding source, for example ODOM_VEL = YES

-> in the Matlab script .\input_files\trajectories\xxx_schedx.m
make sure that the aiding schedule turns on the aiding source

k. Run the covariance code again to obtain the standard deviation plots for PVA. There will be two PVA plots generated this time - one that is derived from the reduced order P matrix (referred to as FIGURE 3) and one that is derived from the suboptimal full state P matrix (referred to as FIGURE 2sub). These plots are displayed in two Matlab Figure windows, i.e. No. 2, titled UNAIDED, and Figure No. 3, titled Reduced States Results.

l. Compare the subplots of FIGURE 2sub to their counterparts in FIGURE 2. The goal is to match the results in the two figures. This involves more trial-and-error and engineering judgement. Start by increasing R_xxx_SUBOPT in the xxx_model.m script.

-> again, model is contained in a Matlab script
.\input_files\meas_models\xxx_model.m
in this script:
increase value for R_xxx_SUBOPT

m. Rerun the covariance code to obtain the standard deviation plots for PVA. Again compare the results shown in FIGURE 2sub and FIGURE 2. Continue to increase R_xxx_SUBOPT, rerun the code, and compare the plotted results until an adequate match is achieved.

Note that it may be necessary to manipulate the P_xxx_SUBOPT and Q_xxx_SUBOPT values for the aiding source error state as well, if they exist.

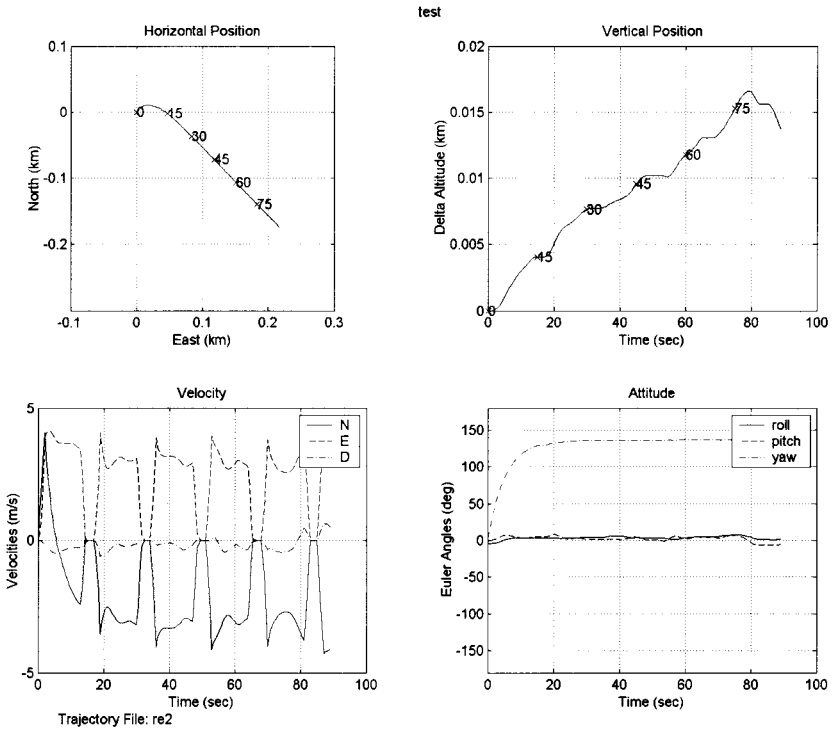


Figure 1 - Trajectory Plot

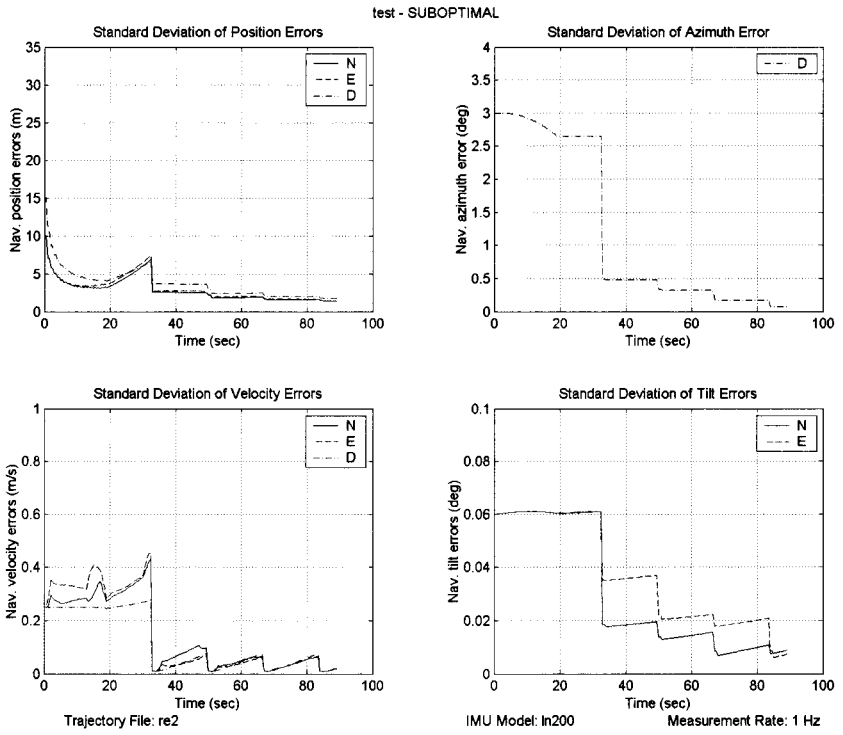


Figure 2 - Standard Deviations of Errors from Full State P Matrix

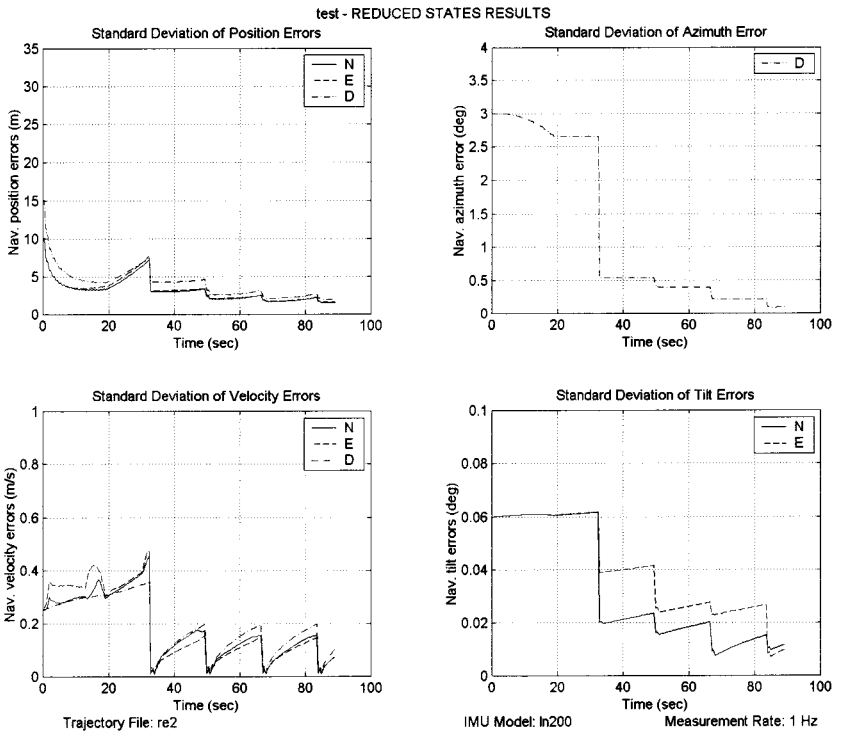


Figure 3 - Standard Deviations of Errors from Reduced Order P Matrix

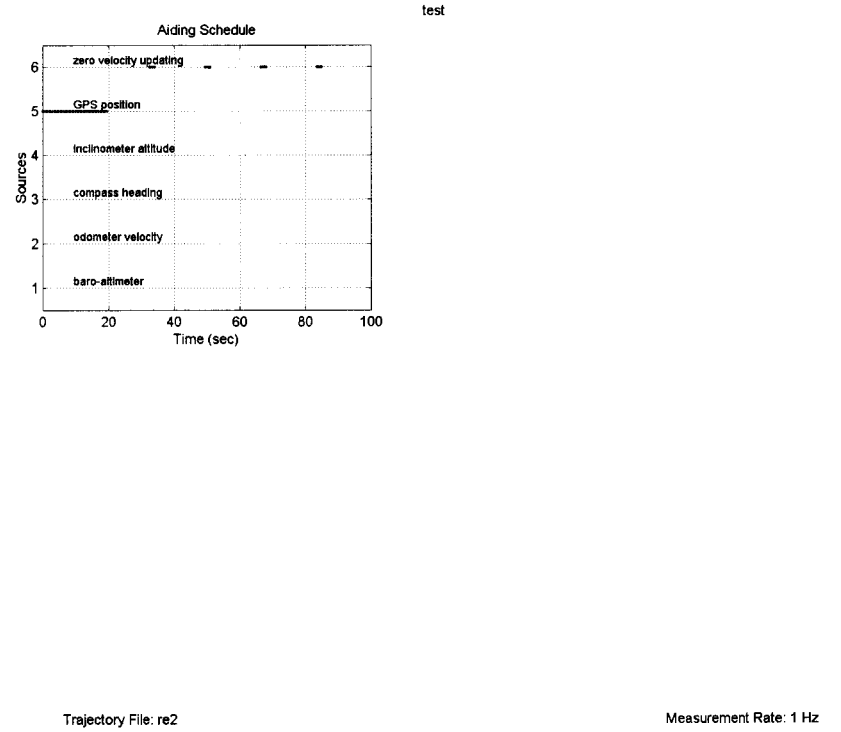


Figure 4 - Aiding Schedule

Figure 2 displays the standard deviations taken from the diagonal of the full state covariance matrix P via a square root function. The upper left subplot shows the standard deviations of north, east and down position errors vs. time in units of meters. The lower left subplot shows the standard deviations of north, east, and down velocities in units of m/s. The upper right subplot shows the standard deviation of heading error in degrees. The lower right subplot shows the standard deviations of north and east tilt errors in degrees. The jaggedness of the traces is a result of the updates being applied periodically. Early in the trajectory, GPS position updating is being used once per second. Later in the trajectory, zero velocity updating is being used once per second, but only when the vehicle is stopped. This occurs approximately every 15 seconds in this example. The trajectory filename, the IMU model, and the measurement rate are shown along the bottom of the plot. The word SUBOPTIMAL in the title of the plot indicates that suboptimal filtering was used to generate these results.

Figure 3 displays results similar to Figure 2, except that they are taken from the reduced order covariance matrix P_subopt instead of from the full state covariance matrix.

Figure 4 displays the aiding schedule, that is, the points where updating of one form or another is applied vs. time.

The axis limits and other properties for these plots are set in a simple manner, depending on the trajectory filename and IMU model. The Matlab script plot_stuff.m tests the filename, and calls the appropriate script from the subdirectory .\plot_limits. The called script tests the IMU model and sets plot limits and properties accordingly. As was mentioned previously, plot_stuff.m can be used to recall results stored by previous NavCov runs in .\output_files.

FINAL COMMENTS

It should be understood that covariance analysis is a first-cut tool for performing navigation system analysis. It is useful for bounding the possibilities of IMU quality and aiding schemes prior to further investment in high-fidelity (aka expensive) modeling and simulation or hardware and software development. Results should always be weighed against experience as a sanity check prior to making the leap to a real system. Once reasonable results are obtained from the covariance analysis, it is often the case that the system simulation and the real-time software are developed in parallel. In that case, the simulation serves as the development environment for the real-time software. Upon completion of the real-time code, the simulation is automatically a high-fidelity representation of that code.

As with many things, the best way to learn covariance analysis is just to jump in and do it. A good start can be made by skimming through the references listed below, followed by skimming through the NavCov code. Then begin using the code to look at a navigation problem of interest. The trial-and-error process will aid in understanding of the

interconnectivity between the various errors. Those connections are often surprising and counterintuitive.

If a bug is found, it will be much appreciated if that bug could be brought to the attention of the author.

REFERENCES to NavCov Users' Manual

[1] W. S. Widnall, P.A. Grundy, "Inertial Navigation System Error Models", TR-03-73, Intermetrics, May 11, 1973.

[2] G. M. Siouris, Aerospace Avionics Systems - A Modern Synthesis, Academic Press, 1993.

[3] R. G. Brown, Introduction to Random Signal Analysis and Kalman Filtering, John Wiley & Sons, 1983.

Distribution

1	MS 0501	Jeff Bradley, 2338
2	MS 0973	G. Richard Eisler, 5742
1	MS 1003	Ken Jensen, 15212
1	MS 1003	Denise Padilla, 15212
1	MS 1003	Maritza Muguira, 15212
1	MS 1003	David Wilson, 15211
1	MS 1125	Paul Klarer, 15252
1	MS 9018	Central Technical Files, 8945-1
2	MS 0899	Technical Library, 9616
1	MS 0612	Review & Approval Desk, 9612
1	MS 0188	LDRD Office, 1030