SAND REPORT

SAND2003-4297
Unlimited Release
Printed December 2003

Adaptive and Mobile Ground Sensor Array

Alexander Maish, Robert Williams, Hung Nguyen and Michael Holzrichter

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

Sandia National Laboratories

SAND2003-4297
Unlimited Release
Printed December 2003

Adaptive and Mobile Ground Sensor Array

Laboratory Directed R&D Project Report

Alexander Maish
Mobile Robotics Department

Robert Williams, Hung Nguyen and Michael Holzrichter
Exploratory Real Time Systems Department

Sandia National Laboratories
PO Box 5800
Albuquerque, NM 87185-1125

Abstract

The goal of this LDRD was to demonstrate the use of robotic vehicles for deploying and autonomously reconfiguring seismic and acoustic sensor arrays with high (centimeter) accuracy to obtain enhancement of our capability to locate and characterize remote targets. The capability to accurately place sensors and then retrieve and reconfigure them allows sensors to be placed in phased arrays in an initial monitoring configuration and then to be reconfigured in an array tuned to the specific frequencies and directions of the selected target. This report reviews the findings and accomplishments achieved during this three-year project. This project successfully demonstrated autonomous deployment and retrieval of a payload package with an accuracy of a few centimeters using differential global positioning system (GPS) signals. It developed an autonomous, multi-sensor, temporally aligned, radio-frequency communication and signal processing capability, and an array optimization algorithm, which was implemented on a digital signal processor (DSP). Additionally, the project converted the existing single-threaded, monolithic robotic vehicle control code into a multi-threaded, *modular* control architecture that enhances the reuse of control code in future projects.

Table of Contents

Table of Figures

Table of Tables

Introduction

The three-year Adaptive and Mobile Ground Sensor Laboratory Directed Research and Development (LDRD) project was initiated to develop and demonstrate the capability to use robotic vehicles to autonomously deploy arrays of seismic/acoustic sensors with high positional accuracy, and then to re-deploy them into optimized configurations based on *in-situ* site measurements. The capabilities developed under this project have specific application for remote targets, but they also have great potential for a much broader range of high-value projects that can attract additional development funding. A number of significant advancements in Sandia's capabilities that provide Sandia with an important technical edge were achieved during the conduct of this LDRD.

Vehicle advancements:

- **Precision vehicle/payload placement control**: Accomplished autonomous precision placement of sensors to *centimeter* accuracies in unstructured (outdoor) environments.
- **Modular vehicle control code architecture**: Converted existing single-thread, monolithic robotic vehicle control code to a multi-threaded, *modular* control architecture. Modular software enables easy reuse of code in future projects allowing Sandia to build upon previous development efforts and rapidly deliver new projects.
- **Mobile manipulation on small autonomous vehicle**: Implemented manipulation on a small-scale, man-portable, semi-autonomous vehicle for the first time.

Sensor system advancements:

- **Autonomous sensor array signal processing**: Achieved near real-time, *autonomous* processing of phased-array sensor data.
- **Sensor array optimization algorithms**: Developed and tested *autonomous reconfiguration algorithms* to determine optimal sensor array layout for a given target and *in-situ* conditions.
- **Sensor array hardware and communication**. Achieved *multi-sensor, temporally aligned, radio-frequency (RF) communication* of seismic array data to a central processor.

These highly significant achievements position Sandia for significant follow-on work with a range of government and civilian organization for such wide-ranging activities as

- Search and Rescue
- Commercial seismic sensor placement
- Military sensor placement for perimeter protection, urban operations, etc.

Project Concept

The original LDRD proposal envisioned a synergistic merging of Sandia's capabilities in the areas of unattended ground sensors and robotic vehicles to achieve a dramatic leap in sensing capability. High-valued, difficult targets are hard to detect and characterize, and site access is limited. Current tactical unattended ground sensor (TUGS) technology developed at Sandia using seismic and acoustic signals can provide extremely valuable information on underground target characteristics and contained equipment, but present performance is limited due to limitations on sensor placement. We know that significant improvements in bearing estimates (localization) can be obtained by deploying a coherent array of sensors spaced and oriented appropriately to the signal frequency. Additionally, using coherent superposition of multiple sensor signals, arrays can be used to increase the signal gain for longer range detection and scanning for specific frequencies. Present sensor emplacement techniques, either hand-emplaced or airdrop, do not provide adequate placement accuracy for these array techniques. Additionally, the signal frequency that determines optimal array spacing can only be determined by *in-situ* measurements, thus requiring a re-deployable system of sensors. Sandia's robotic vehicle capability offers a technology approach to provide mobile, re-deployable sensors enabling on-site, dynamic sensor array configuration in response to site frequencies. This merging of two separate technologies offers a major leap in capability in sensing difficult targets.

The approach proposed in this effort was to integrate TUGS with a small air-deployable robotic vehicle to demonstrate the feasibility of self-adapting sensor arrays exhibiting enhanced performance capabilities. Advantages of this approach include reduced manpower for sensor deployment, since fewer sensors can be optimally positioned automatically for best results, and improved sensor results, since the sensors can be automatically optimized for *in situ* frequencies. With this concept, the vehicle can be deployed with minimal accuracy at a distance (a few kilometers) from its pre-selected observation site, and it can then reposition itself and place sensors within a few tens of centimeters relative to each other to form a sensing array. Once an initial set of measurements is evaluated using an on-board algorithm, the vehicle reconfigures the sensor array in the best arrangement for the site and target conditions. The original proposal envisioned using multiple vehicles each with individual sensors configuring themselves cooperatively, but an analysis performed during the first year identified a single vehicle configuration as preferable. This concept is shown in Fig. 1.

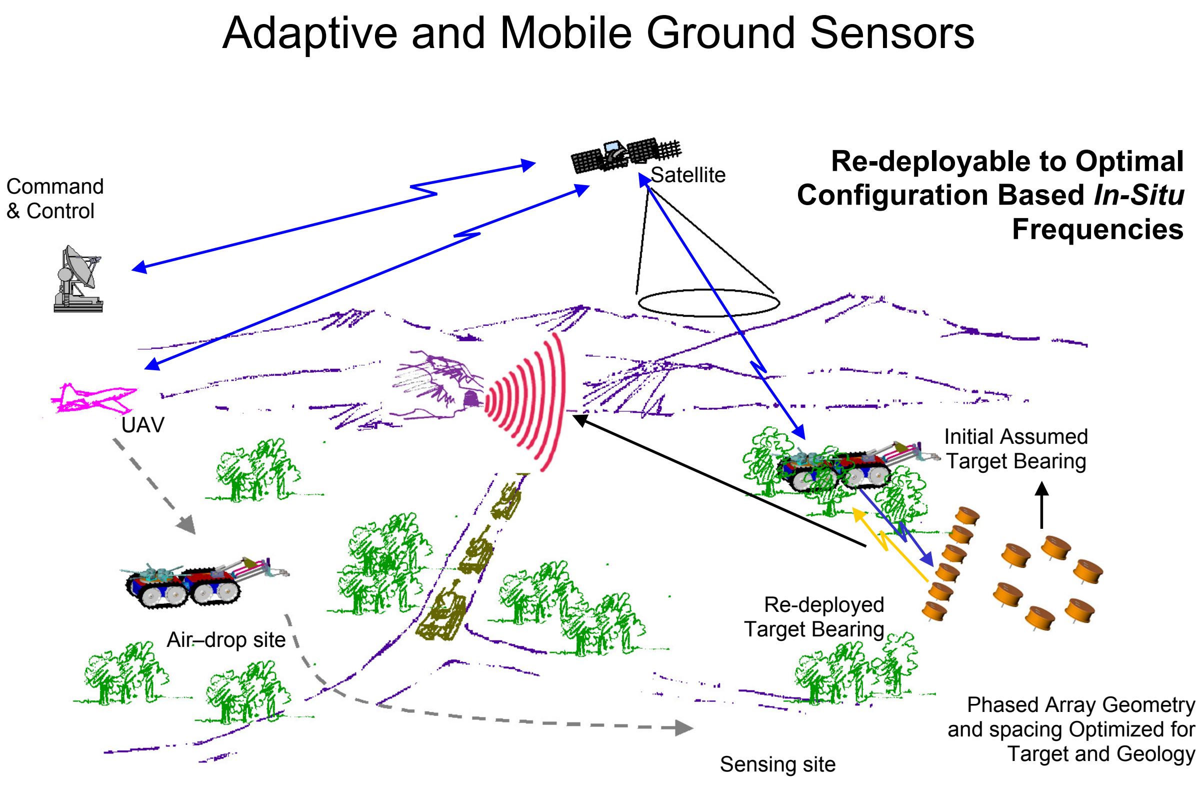

Adaptive and Mobile Ground Sensors
Command & Control
Satellite
Re-deployable to Optimal Configuration Based *In-Situ* Frequencies
UAV
Initial Assumed Target Bearing
Air–drop site
Re-deployed Target Bearing
Phased Array Geometry and spacing Optimized for Target and Geology
Sensing site

Figure 1 - Adaptive and Mobile Ground Sensor Concept

Robotic System Configuration and Seismic Signal Coupling Study

The Adaptive and Mobile Ground Sensor (AMGS) concept was intended to autonomously install a configurable array of seismic and acoustic sensors using one or more robotic vehicles. This left a lot of latitude in system design depending on the specifics of the intended operational environment and scenarios. This section discusses work done during the first year of the LDRD to explore the interrelated considerations and constraints involved in the selection of the sensor configuration, the vehicle configuration, and the seismic sensor coupling to the ground. It identifies a preferred configuration for the vehicle and seismic sensors, and discusses some of the field tests conducted to gather system configuration and algorithm development data.

Operational Assumptions

Although the AMGS system could be used in a variety of operations, some assumptions are made here for the purpose of defining a system configuration. We first assume the AMGS will be used in a covert manner to remotely monitor target signals from a stand-off distance. It is not intended to roam widely over the target locale to triangulate or home in on the target source. The robotic vehicle(s) is assumed to travel from its deployment site to the monitoring site under its own power for a distance that could be up to a kilometer or two. It will set up an initial sensor array in a localized area and monitor the target signals to validate assumptions on seismic conditions, target bearing, and target frequencies. Once actual signal conditions are identified, the system will autonomously compute its best estimate of a modified array layout based on measured conditions and it will reconfigure the sensors by moving them short distances in the same monitoring area. This is the real value of the system we are developing. Not only does this enable on-the-fly correction for inaccurate initial assumptions concerning site and target conditions, but it also enables the use of very different array configurations for the initial target evaluation and subsequent operational characterization. Operationally the sensors are intended to create a phased array tuned to specific target frequencies to create narrow beams to help in source localization. Additionally the phased array is intended to use coherent superposition of multiple sensors to increase gain for longer-range detection and scanning for specific frequencies. Reconfiguration of the sensors is intended to occur only once to place the sensors in a configuration that reflects actual conditions rather than the best guess used for the initial array configuration. Reconfiguration is not intended to be repeated numerous times in an optimization process. At the completion of the mission, the vehicle may be commanded to recover the sensors and return to its deployment site.

The AMGS sensors use interferometric methods to compute bearing to the target and to enhance signal strength. Thus sensor position relative to each other need to be well controlled with errors in positioning minimized and characterized so software adjustments in relative signal timing can be made. It is more important to know the actual sensor's position relative to the other sensors in the array than to have the sensor be in an exact predetermined location relative to the other sensors.

The operational physical environment needs to be defined as it directly impacts system design. Soil type and vegetation impacts design of the seismic sensor installation system and the hardware used to couple the sensor to the seismic signals in the ground. It directly impacts vehicle mobility and seismic sensor implantation, if the vegetation is in the way, and it indirectly impacts seismic and acoustic signal strength via root and brush density and size. Soil density and composition affects the effort needed to insert a seismic sensor coupling device. We assume the soil is arable and the vegetation is brush in a semi-open environment, similar to that surrounding Sandia. Wind impacts the overall vehicle/sensor configuration as it is an important source of acoustic and seismic noise in addition to direct mechanical load considerations. As wind speed increases the turbulence around the acoustic sensor creates noise. Wind can also cause cables to bang or vibrate if not properly secured, and it can cause vibration of vegetation and vehicles that can be transmitted into the ground causing significant seismic noise. Rather than specify an operational wind speed, we tested various system and sensor configurations to identify the lowest noise ones, and then characterized the noise floor these can achieve as a function of wind speed.

Sensor Array Constraints

Characteristics of the target and the mathematics of signal processing both determine the features of the sensor arrays necessary to achieve target characterization goals. The same equations apply to seismic and acoustic arrays, but due to the different mediums in which the signals travel, the two types of sensor arrays are laid out with very different spacing. Here we review some of the assumptions made concerning target signals and characterization goals and then trace these assumptions through the appropriate mathematical translations to determine how this affects the sensor array parameters.

Signal Wavelength

The wavelength of the measured signal of interest directly impacts the design of the sensor array because the sensors are nominally spaced at half-wavelength intervals to achieve the best results. Wavelength is proportional to the velocity of the medium and inversely proportional to the frequencies of interest. Table 1 gives frequency and velocity ranges for the targets of interest, and the wavelength range that results from these. It also selects a target frequency and signal velocity within this range for use in sensor array design. The frequencies are those found from prior experience in monitoring underground facilities to be ranges likely to yield interesting information. Acoustic velocities are calculated using the formula $c = 331.5 + 0.58°C$ for temperatures ranging from 0°C to 40°C. The seismic velocities range from that of representative of unconsolidated alluvium to a value measured in granite. Basing our sensor array design on the target wavelengths given in Table 1 and spacing the sensors at half-wavelength intervals, we arrive at an acoustic sensor spacing of 1.1m and a seismic sensor spacing of 5m.

Table 1- Target Wavelength

	Frequency Range	**Velocity Range**	**Wavelength Range**	**Target Frequency**	**Target Velocity**	**Target Wavelength**
Acoustic	50 to 250Hz	330 to 350m/s	1.3 to 70m	150 Hz	340m/s	2.3m
Seismic	10 to 100Hz	500 to 3300m/s	5 to 330m	100 Hz	1000m/s	10m

While seismic wavelengths can be quite long, emphasis should be given to shorter wavelengths, on the order of 10-30m. There are two reasons for this. First, if sensor spacing is on the order of a hundred meters, the arrays would in turn be very large, a large fraction of a kilometer. The logistics resulting from long wavelengths dictate a very different system design than a system deploying arrays with sensor spacing on the order of 10m. Second, long wavelengths are associated with high velocities that in turn are associated with solid rock. However, solid rock is rarely exposed at the surface. The unconsolidated materials found at the earth's surface tend to have lower velocities resulting in shorter wavelengths

Array Geometry

Different two-dimensional arrangements of sensors can provide different information on the target after signal processing. The intention of this effort is to enable the robotic vehicle to implant whatever array configuration is necessary to maximize the information desired from the target signal. A circular array may be selected initially because it eliminates ambiguity in target bearing, while a linear array may be selected for longer-term monitoring once the target bearing is determined. The configuration for the acoustic array mounted on the robot is constrained to a circular array (with a center sensor) because the target separation of 1.1m makes it impractical to mount a linear array on the vehicle. As mentioned previously, the acoustic sensors can also be deployed in stand-alone packages like the seismic sensors if so desired.

Number of Sensors

The goal of using arrays of sensors is to identify target bearing to help in target localization, and to increase signal gain to aid in target long-range detection and characterization. Increasing the number of sensors will increase the signal-to-noise ratio (SNR), aiding target detection. Improvement in the signal-to-noise ratio of the array output is bounded by the square root of n, the number of sensors.

$$\frac{SNR(n)}{SNR(1)} < \sqrt{n} \tag{1}$$

At best one would expect a factor of two improvement in SNR for a four-sensor array, and a factor of three improvement for a nine-sensor array. An improvement in SNR of 2 to 3 is a reasonable target, although if the sensors are small and light enough, an improvement by a factor of four could be achieved with 16 sensors. As an initial target we have decided on an acoustic sensor of seven sensors, six in a circle with one in the center. We are also targeting to deploy between 4 and 9 sensors in the seismic array as well.

Array Relative Position Accuracy

Array signal processing depends upon accurately determining the relative phases between signals measured at the elements of an array. Determining accurate relative phase in turn depends upon accurately knowing the positions of the sensors as well as the velocity of the signal in the medium in the vicinity of the array. In the application this LDRD addresses, it is difficult to accurately determine the seismic velocities. The difficulty in determining the seismic velocity bounds the degree of accuracy to which the sensor location must be determined. A key process in array signal processing is extrapolating the sensor measurements at multiple locations to estimate what the signal should be at a common “array center.” For non-impulsive signals, the extrapolation is essentially a phase shift. Phase errors caused by location inaccuracy need to be somewhat smaller, but not vastly smaller, than the low end of phase errors due to inaccurate velocities. If we assume we can estimate seismic signal velocity within 5% some portion of the time, and require the phase error induced by inaccuracies in knowing relative position to be 1/3 that of the phase error induced by the velocity error, we can determine an upper bound on the acceptable position error.

The most compact array configuration is a circular one. With sensors placed at half-wavelength spacings, the greatest separation between pairs of sensors is one wavelength (the circle's diameter). An error of 5% in velocity will cause an 18° error in signal phase (360°x0.05) across this array. This is what we are assuming is the low end of the achievable phase error due to imprecise knowledge of seismic signal velocity. One-third of this, the allowable phase error due to sensor placement error, is 6°, or 1/60 of a wavelength. Using a 10m target wavelength, this corresponds to a tolerable position error on the order of 10m/60, or 17cm. This is the upper bound we allow for relative positional error on the seismic sensors to ensure it does not contribute markedly to the overall phase error. While the original project plan did not call for developing the capability to determine relative sensor position this accurately until out years, we have been able to do so using real-time kinematic (RTK) GPS technology by Novatel, which gives a 1cm standard deviation in its relative precision.

For acoustic sensors a different approach is taken to determining allowable positional errors because the velocity error in air is much smaller. Imposing a maximum amplitude error of 10% on the in-phase and quadrature components of the signal after phase shift translates to about 6°, or 1/60 of a wavelength. Using a 2.3m target wavelength, this corresponds to a tolerable position error on the order of 2.3m/60, or 4cm. This would be significantly more difficult to achieve with stand-alone sensors implanted by a robotic vehicle, but it is easily achievable using retractable masts mounted to the vehicle.

Time Determination

Just as knowing inter-sensor spacing accurately is crucial in array signal processing, so is being able to place their signals accurately in time. We need to be able to align every sample from every sensor. We examine two major issues associated with temporal alignment of signals:

1) Estimating the maximum misalignment the array signal processing algorithms can tolerate.
2) Identifying sources of temporal misalignment.

In the previous section, 6 degrees (1/60 cycle) was established as an upper bound for phase errors arising from sources other than seismic velocity errors. This is used as a bound for the maximum tolerable phase error due to inaccurate times. We impose the same requirement on the temporal accuracy of both acoustic and seismic signals. From the highest frequency of interest we can use the maximum tolerable phase error to determine the maximum tolerable time error. Using 250Hz as our maximum frequency of interest, we take 1/60 of 1/250 of a second, giving 67usec as our maximum temporal error.

We identify two types of temporal error:

1) a constant offset between the clocks of pairs of sensors
2) a time varying difference between the clocks of pairs of sensors.

The array processing algorithms require that the sum of both errors not exceed the maximum temporal error of 67μsec. We take the conservative approach and require that the sensor portion of the robot be able to align samples to better than 67μsec accuracy for both types of error unless it can be demonstrated that one of the above types of error is guaranteed to be less than this. The following are possible sources of temporal errors:

1) digitizers that sample at slightly different rates
2) clocks that drift over time
3) improper initial alignment of clocks.

Discussion of Coupling

Probably the most critical issue that needs to be answered in this LDRD effort is achieving good signal coupling between the seismic sensor and the ground while minimizing signal coupling to noise sources such as wind-induced vegetation, cable, or vehicle vibration. Good coupling is generally achieved by driving or burying a portion of the sensor package in the ground where it can pick up seismic vibrations. Air dropped sensors are generally dart shaped and bury themselves upon impact. Hand implanted sensors can be buried or driven into the ground by stepping on them. Industrial geophones generally have a 3-inch tapered spike, which is driven into the ground by foot before covering the exposed sensor with a sandbag. There is general agreement that the deeper the sensor or coupling spike is driven the stronger the target signal. This is primarily due to avoiding the root zone, which can both dampen target seismic signals and channel wind-induced vibration of surface growth into the ground. However, the longer the spike is on a sensor, the harder it is to store the sensor on a small robotic platform. As a starting point we decided to use the standard geophone as the baseline and investigate alternative coupling devices that are roughly the same 3” length.

The requirement to insert and remove these sensors autonomously dictates a close look at the process to ensure maximum reliability. Insertion of a tapered spike using normal (vertical) force can be difficult especially in packed or rocky soil. In a packed roadbed we were unable to drive the spike in more than about an inch even with over 100 pounds of force. Anecdotal information indicated a plain tapered spike sometimes bonded with the earth so well it was nearly impossible to remove by hand with an upward pull. With a robotic vehicle, normal force for insertion must be provided by the weight of the vehicle or by impulse (jackhammer) forces. Impact methods were ruled out as they can be hard to implement in a covert manner and more importantly they may damage the seismic sensor. If the insertion mechanism is offset from the vehicle’s center of gravity, the available force can be much less than the weight of the vehicle. We are targeting a vehicle weight loaded with payload of only 50 pounds, so it would not be able to generate a steady normal force in the 100 to 150 pound range. An alternative approach is to use torque rather than normal force to insert and remove the coupling device. A vehicle of moderate length can create a significant torque, and if more is needed, small penetrators can be used to anchor the vehicle to the ground against rotation. Torsional insertion force is accomplished by using some form of screw mechanism. Three variations are investigated:

1) Threaded body, like a lag bolt. This approach provides a stiff shaft with moderate threads to translate rotational motion into a downward or upward thrust.

2) Auger, like a wood auger. This approach carves a hole and removes dirt to the top of the hole.
3) Spiral earth anchor, like a dog's leash stake. The approach just spirals into the ground.

Earth Penetration Tests

Tests were performed on March 13, 2001 to characterize the performance of a number of earth coupling devices in a range of earth types. The penetrators tested and their characteristics are listed below in Table 2.

Table 2 - Penetrator Option Parameters

Description	Length (in.)	Thread diam. (in.)	Shaft diam. (in.)
Spike	3	3/8	
#8 drywall screw	3		
#10 screw	3		
5/8" lag screw	4	5/8	1/2
3/8" lag screw	4	3/8	1/4
1/4" lag screw	2-1/2	1/4	3/16
3/4" wood auger		3/4	

The penetrators were tested on three surfaces, a packed dirt/gravel roadway, a somewhat packed dirt test track (RVR motocross test track) with a high fraction of small fragmented rock, and a soft dirt earth with a moderate fraction of small fragmented rock. All three surfaces were just outside the robotic vehicle range (RVR) compound.

In each test the spike was pushed in manually and the rest of the penetrators were screwed in with a battery powered drill while a normal force was applied manually. The stability of the inserted penetrator was determined by wiggling the projecting head. A torque wrench was applied and insertion and removal torques were measured. A dog leash spiral earth anchor was tried a few times, but it tended to wander and just create a large area of soft dislodged earth as it hit rocks and walked around. It was dropped from further consideration.

The packed dirt/gravel roadway was the densest and most difficult to penetrate. We were unable to insert the spike more than about an inch with about 100 pounds of normal force. All the various screws inserted without difficulty, although the large 5/8" lag screw visibly raised the ground around it as it hit and dislodged larger rocks under ground. With the exception of the spike, normal force was nominally about 10 pounds, but up to about 40 pounds when obstacles were encountered. Stability of most of the screw devices varied with attempt, sometimes very tight and stable, and sometimes a bit wobbly due to wiggling or hitting a rock as it was inserted, creating a shaft of looses dirt around it.

Table 3 – Penetrator Dirt Roadway Results

Penetrator	Notes	Stability	Insert Torque	Remove Torque
Spike	Only 1" penetration			
#8 screw		Tight	2 in-lb	2 in-lb
#10 screw		Tight	2	1
1/4" lag		Tight	2	1
3/8" lag		Tight	10	5
5/8" lag		Slight wobble	7	4
3/4" auger		Slight wobble	5	3

The motocross dirt track was less compacted than the roadway, but still presented a significant challenge for inserting the spike. With about 100 pounds of normal force it inserted only about 2 inches.

Table 4 – Penetrator Motocross Track Results

Penetrator	Notes	Stability	Insert Torque	Remove Torque
Spike	Only 2" penetration			
#8 screw		Tight	1 in-lb	1 in-lb
#10 screw		Tight	2	2
1/4" lag		Wobbly	1.5	<1
3/8" lag		Tight	8	3
5/8" lag	Cracks in the ground	Slight wobble	13	5
3/4" auger		Wobbly	12	4

The final test was run in the dry earth. Penetrators inserted fairly easily, although all bounced around to some degree as they hit rocks on the way in.

Table 5 – Penetrator Dry Earth Results

Penetrator	Notes	Stability	Insert Torque	Remove Torque
Spike	Only 2" penetration			
#8 screw		Tight	<1 in-lb	<1 in-lb
#10 screw		Tight	1	1
1/4" lag		Wobbly	1	1
3/8" lag		Tight	4	2
5/8" lag		Wobbly	4	2
3/4" auger		Slight wobble	7	7

Our conclusion from observations made during these tests are that the screw-type penetrators worked much better than the spike penetrator in rocky soil. Using a drill rather than hand-insertion minimized wobbling during entry and led to a tighter coupling. The spike penetrator had a very tight coupling with the ground when we could get it inserted. This is undoubtably due to its wedge shape. A wedge-shaped screw would probably work better than a straight-shaft screw, but those are not readily available. Perhaps in the future we could get such a screw fabricated for tests. The larger screws displaced too much ground and were not as secure as the smaller screws. The best size appeared to be the 3/8” lag screw, followed by the #10 screw. Although we tested 4” long screws for the most part, we could easily go longer if handling considerations on the vehicle permit.

Configuration Evaluation

The LDRD proposal advanced one concept for the platform/sensor configuration to be used in achieving the goal of an adaptable sensor array system. In fact there are several possible system configurations, which we will discuss here. The system configuration involves the platform (robotic vehicle) size and number, the seismic sensor design (stand alone or wired to vehicle), and the deployment vehicle location during measurements (stand-off or remain with the sensor). While the nominal design we are pursuing mounts the acoustic sensor array on a single vehicle using extendable masts, the acoustic sensors could be deployed and reconfigured in precisely the same manner as the seismic sensors. They could even be deployed on the same unit as the seismic sensors with the unit monitoring either the acoustic or seismic sensor depending on the deployment spacing. With this in mind, when discussing deployment of the seismic sensors it will include the option for acoustic sensors.

There are four basic platform deployment options:

- Option 1) Multiple identical platforms, each with one sensor.
- Option 2) Single “mother” platform with multiple deployable “daughter” vehicles to deploy sensors.
- Option 3) Single platform with multiple sensors that it implants and retrieves.
- Option 4) Single platform with multiple sensors that it abandons when establishing a new array.

To make some reasoned conclusions concerning the mix of these different platform, sensor, and stand-off options, we need to review some information concerning platform and sensor design. Robotic vehicles come in a range of sizes, and size directly affects 1) the scale of navigable objects, 2) the payload capability, and 3) the energy storage capability. Stored energy determines the range of travel and the duration of on-station activities. Table 6 gives specifications for three sizes of RATLER™ vehicle, and Fig. 2 shows RATLER™ vehicles in the three different sizes.

Table 6 – Robotic Vehicle Specification Estimates

Platform		**RATLER™**	
	Mini-Sized	**Mid-Sized**	**Maxi-Sized**
Dimensions			
Body			
Length	6"	15.5"	36"
Width	4.5"	11"	24"
Height	2.5"	6.5 to 7.5"	11"
Total	67.5 cu in	1278 cu in	9504 cu in
Vehicle incl. wheels			
Length	8"	21"	52"
Width	8"	21"	45"
Height	3.5"	10 to 11"	23"
Battery		2 liters	
Weight			
Vehicle (Total)	5.5lbs	35lbs.	200lbs
Battery	0.5lb? NiCd	11.4lbs SLA	124 lbs SLA
Speed			
Max.	2 mph	1.5 to 4mph	1.4 mph
Avg. w/ obst. nav.	1 mph	1mph	1 mph
Energy Storage	17Wh NiCd	168Wh SLA	1920Wh
Range (Pb-acid)	1 to 1.5 mile	4-6 miles	6 mile (est.)
Power			
Avg. Drive	10W (est.)	50W avg.	100 to 150W
Components (Active)	unknown	10-20W	unknown
Components (Sleep)	<1W	<1W	unknown
Payload			
Dim/Size	external	750 cu. in.	0.34 Cu ft.
Weight	2 to 3 lbs	40lbs (est.)	100 lb

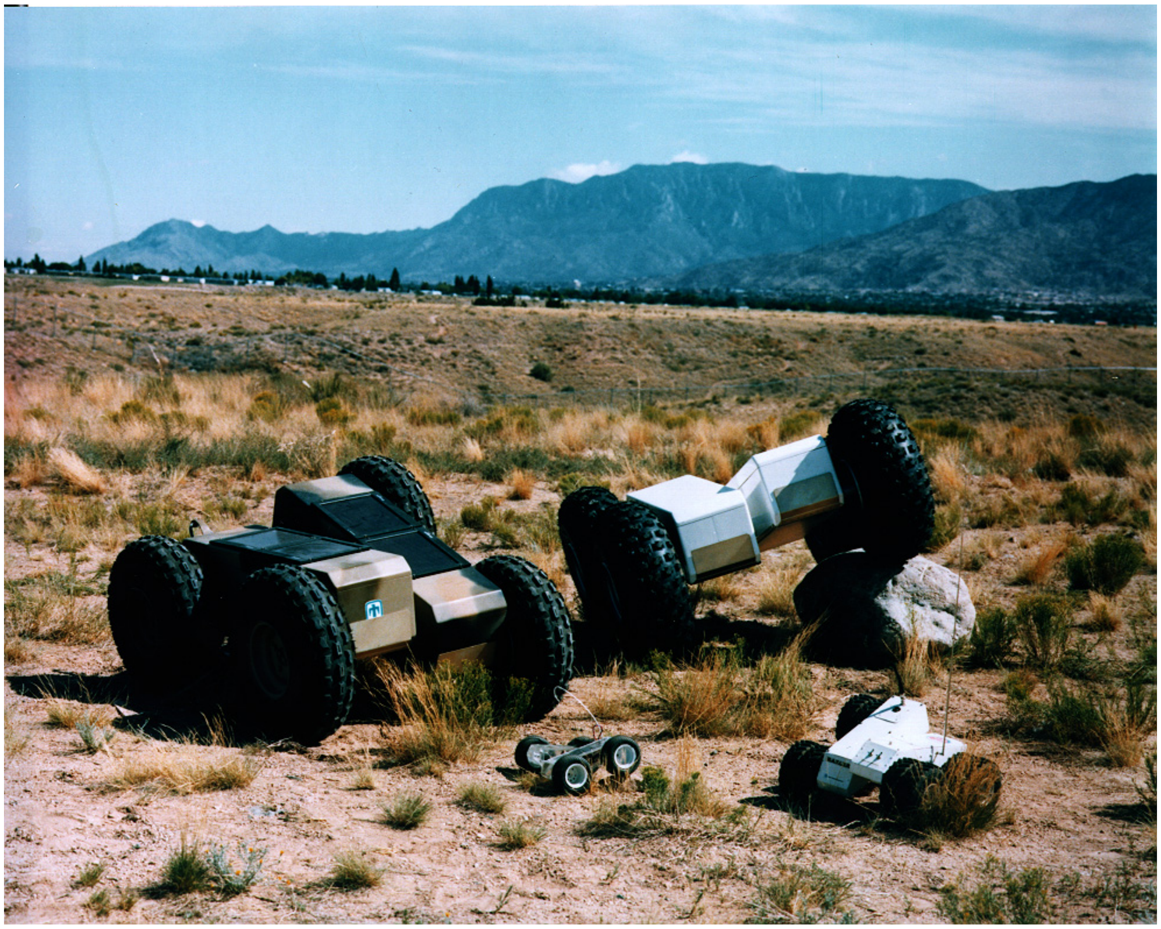


Figure 2 – The RATLER™ Robotic Vehicle Family

Obstacle Navigation

The scale of obstacle a vehicle is able to negotiate is dependent on the scale of the vehicle. A small vehicle such as the mini-RATLER™ would have difficulty navigating the obstacles present in a multi-kilometer, off-road mission even if it had the necessary energy. It would probably be able to navigate the few meters needed to disperse a sensor from a larger "mother" vehicle as in Configuration 2. The mid-sized and maxi-sized vehicles fitted with autonomous obstacle avoidance software and traveling a course selected to avoid large terrain hazards could negotiate such a route.

Payload Capability

The main payload of the vehicles in the different configurations will be the sensors and the signal processing unit, which will receive and process data from each of the sensors. A sensor consists of a small geophone (or microphone). If the unit is intended to operate in a stand-alone mode where it is not connected to the signal processing unit by a wire, the sensor package needs to include analog-to-digital (A/D) conversion electronics, a low-power radio, a small control processor such as a PIC processor to manage the sensor, and batteries. The A/D unit, PIC processor, and radio can fit on a small circuit board, while the batteries consume the bulk of the sensor package volume and weight.

Stand-alone Sensor Package Design

A low-profile sensor package was scoped out that would contain the necessary components. The nominal configuration of the sensor package is a round can 7” in diameter (6”ID) and 3” high (85 cu. in.). That should hold a dozen D-cell batteries with room in the center for the geophone (Fig. 3). The electronics board would lay across the top of the batteries. Tadiran makes a D-cell lithium thionyl chloride ($Li/SoCl_2$) primary (single use) battery with an energy density of 460Wh/kg when used at low discharge rates. It has a 13.5 Ah rating at a discharge rate of C/1000 (1000 hours to discharge completely), and a nominal voltage of 3.5V. The cell is 33mm in diameter (1.29”) and 62mm tall (2.42”). Its weight is 100g (0.22lb), so a dozen will weigh 1.2kg or 2.64 lbs and have a total energy of 552 Wh. A complete sensor package would weigh about 1.6 to 1.8kg (3.5 to 4 lbs.), and a suite of six of them on a vehicle would weigh about 10kg (22 lbs.) and have a packing volume of 14.4L (880 cu. in.). A smaller sensor with half the batteries would have a diameter of 6”, a weight of about 1kg, and a packing volume of 1.7L (108cu. in.). A suite of six of these sensors would have a weight of 6kg (13.2 lbs.) and a volume of 10.6L.

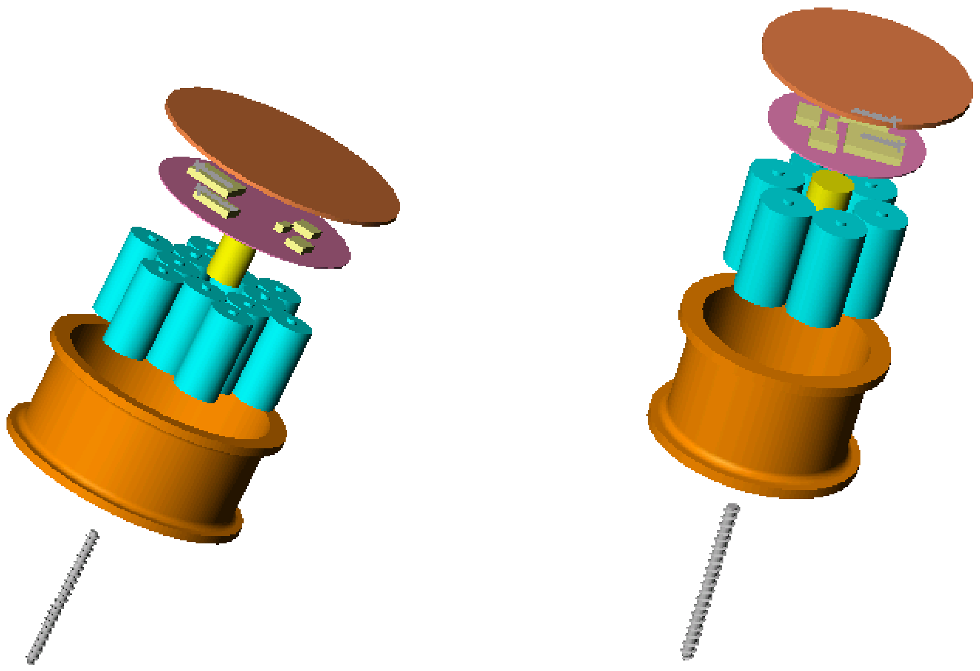


Figure 3 – Exploded view of large and small stand-alone sensor packages

Signal Processing Unit Package

The signal processing unit consists of a digital signal processing unit (DSP) with the capability to rapidly process large volumes of digitized geophone or microphone signals. One package is required per array, which could include up to 16 sensors. The package would also require a radio to communicate with the sensors, and batteries for power. The processor itself is small but it requires up to 3.7W of power during operation. A 1 mega-bit-per-second (Mbps) local low-power communication unit will draw 300mW in receive mode. Assuming it operates at a 50% duty cycle, it would require 1440Wh of energy per month of mission, or about 3.2 kg of $Li/SoCl_2$ batteries.

Mobility Energy

The energy required to traverse a given distance is dependent on the rate of energy consumption. We have measured the energy consumption of a mid-sized (50 lb.) vehicle at between 16 Wh/km (27 Wh/mile) for a wheeled vehicle, and 22Wh/km (37 Wh/mile) for a tracked vehicle on a hilly road under tele-operation. When operating in an automated obstacle avoidance mode under off-road conditions the energy consumption may be 3 or 4 times as high. A tracked vehicle traveling 4km would require 350Wh. Additionally, the type of battery needs to be one that is capable of producing high power in order to supply the power demands of the drive motor. Unless they are quite large, $Li/SoCl_2$ batteries do not have sufficient power to be used for drive motors. Some rechargeable batteries such as sealed lead acid (20-30Wh/kg), nickel-metal hydride (60-70Wh/kg), or lithium ion (100-110Wh/kg) batteries have acceptable power characteristics, and so does the primary cell (single-use) lithium manganese dioxide Li/MnO_2 (300Wh/kg) which has been configured in a large pouch-cell battery by Eagle Picher. Only about a kilogram of the Li/MnO_2 battery would be required for mobility purposes.

Insertion of sensors would require the vehicle be active, consuming on the order of 10 to 20W for position sensors (GPS, compass, tilt sensor, etc.), control electronics, cameras, and insertion motors. Insertion or removal may last up to 15 minutes per sensor (mainly for position adjustment) for six sensors. They may be inserted and removed up to three times, giving a total of about 100Wh plus moving and positioning, another 50 to 100Wh, for a total of up to 200Wh.

Table 7 gives a summary of battery load requirements for a hypothetical mission. The weight for this battery load is a minimum of 4 to 5 kg using expensive primary batteries. This requires at least a mid-sized vehicle.

Table 7 – Battery load requirements for a hypothetical mission

Load Category	Amount	Type	Quantity
Mobility	4 km	Li/MnO_2	350Wh
Sensor Insert/Remove	6 units/3 times	Li/MnO_2	200Wh
Signal Processing	50% duty, per mth	Li/$SoCl_2$	1440Wh
Station Keeping			
Total			1990Wh

Configuration Review

With this background we are now able to make some conclusions regarding system configuration, narrowing the number of options under evaluation from a full matrix of all possibilities to a few configurations which merit further evaluation.

Platform Scale

The first consideration in vehicle selection is scale. To travel the few kilometers over off-road terrain required by the mission and to provide the necessary battery weight, the vehicle will have to be at least of the mid-sized RATLER™ scale. This vehicle has a base weight of 35 to 40 pounds for its body, sensors, controller, communication, wheels or tracks, and motors, and is capable of carrying another 20 to 40 pounds. With the right design, a vehicle of this scale could perform the proposed mission, carrying at least one sensor package and the signal processing unit.

Sensor/Platform Connection

There are four possible sensor/platform connection options we considered.

Option 1) Vehicle stays with sensor – sensor physically attached to platform
Option 2) Vehicle stays with sensor – sensor attached by wire
Option 3) Vehicle stands off from sensor – sensor attached by wire
Option 4) Vehicle stands off from sensor – sensor communicates by radio

Using a wire cable between the sensor and the vehicle dramatically reduces the size of the implanted package since both power and data can be sent via the cable. The only components needed are the coupling spike, geophone, and housing, leaving a package that is at most two inches cubed. The A/D converter, radio, PIC processor, and batteries would be eliminated from the sensor. Although the A/D converter would still be needed on the vehicle in this case, the radio and PIC processor would not, eliminating the need for a significant amount of accompanying battery. The benefits of using a wire include having a smaller payload on the vehicle, reducing the size and visibility of the implanted sensors, and eliminating RF signature. However, exposed cables present very real issues with vehicle entanglement, visibility, and, most importantly, coupling of wind turbulence into the seismic sensor. For these reasons we decided not to pursue any option with wires strung between a sensor and a vehicle standing off from the sensor, eliminating Option 3.

Staying with the sensor is only an option for Options 1 and 2 where there is one sensor per vehicle. In this case the sensor can remain attached to the vehicle or be vibrationally isolated from it using a wire. Because of its small size, there is some question as to whether a mini-RATLER™ sized vehicle has the mass and torsional resistance to enable a sensor to be implanted, but the sensor could still either be pressed to the ground with a hard connection to the vehicle (Option 1), or dropped on the ground with a wire connection (Option 2). Option 4, standing off from the sensor, while minimizing the chance for coupling wind-induced vehicle vibrations into the sensor, does involve the non-trivial complexity of finding and grabbing the sensor to be able to remove it.

Option Discussion

Vehicle Option 1 was proposed in the original LDRD proposal. It uses multiple identical vehicles each having one sensor. If the system were configured so that each vehicle had to make it to the deployment site and then communicate and coordinate with the other vehicles to deploy the sensor array, one could argue that the system reliability would be reduced from that of a single vehicle. On the other hand, if the vehicles were redundant and cooperative, with any particular vehicle being expendable and the rest able to compensate for its failure to make it to the deployment site, one could argue the system reliability of this configuration is enhanced over that of a single vehicle. However, this configuration does involve a significant amount of redundant hardware in the form of the vehicles themselves. One mid-sized vehicle capable of travel over off-road terrain can be configured to carry multiple sensors. Additionally, the visible, track, and RF signature of so many vehicles increases the probability of detection. Even though Sandia has significant experience with cooperative vehicles and has developed software to enable vehicles within a swarm to adaptively communicate and to locate themselves relative to each other (although the acoustic ranging hardware needs further development for outdoor use and carries its own signature issues), the use of numerous vehicles adds complexity to an already complex mission.

This configuration could employ any of the sensor/vehicle communication options. It could implant a seismic sensor and remain physically coupled to it, or it could remain attached only by a wire. The first option has a high likelihood of coupling wind vibrations into the sensor. The second option could still do so, indirectly, by vibrating the ground next to the sensor. This needs to be tested using actual hardware in empirical tests. Both of these options place the battery pack, A/D converter, and radio in the vehicle rather than in a stand-alone sensor package. The third option uses a stand-alone sensor package to communicate directly with the signal processing unit, and the implanting vehicle stands-off so that it doesn't couple wind vibrations into ground adjacent to the sensor package.

Vehicle Option 2 uses many identical vehicles to implant sensors similar to Option 1, but the vehicles are small and only designed for short-ranged mobility. A larger "mother" vehicle is needed for the signal processing and remote communication units that need a substantial battery. The small "daughter" vehicles, which could be the size of the mini-RATLERs™, could be delivered on site by the larger "mother" vehicle or by some other means. This approach has many of the same benefits and drawbacks as Configuraton 1, but the overall amount of vehicle hardware is less since the vehicles are smaller. This option only makes sense if the daughter vehicles stay with the implanted sensors eliminating the complexity in finding and retrieving the sensors. If the daughter vehicles stand-off from the implanted sensors, there is no benefit over Option 3.

Vehicle Option 3 uses one vehicle to implant and retrieve at least multiple and hopefully the full array of sensors. This configuration minimizes duplication of mobility, navigation, control, and communication hardware by using only one vehicle instead of multiple vehicles. It minimizes total system weight and complexity and minimizes system signature (tracks, RF etc.) This configuration does require advancement of our capability in the area of locating and retrieving the sensors, and it requires developing the capability to store and access multiple sensors on the vehicle. Since the vehicle will stand-off from the sensor and we have eliminated the option of using wires, this configuration only uses the RF communication stand-alone sensor package option.

Vehicle Option 4 is similar to Option 3 except that it abandons sensors rather than retrieving and relocating them. It was initially conceived for use with wired sensors where only the small thumb-sized geophone is implanted without any stand-alone hardware. Many more of these small sensors can be carried on a single vehicle than the larger stand-alone sensors, making it easy to carry replacements. This option could also be considered when using stand-alone sensors. While it is an acceptable fall-back position if it becomes apparent that retrieving sensors is too difficult, the goal would be to reuse sensors (Option 3), and when the mission is complete, to remove the sensors leaving no sign of the system.

The result of this evaluation of configurations is that Vehicle Option 3 is the preferred arrangement. In order to minimize vehicle coupling of wind vibrations into the ground or sensor, the vehicle should stand off from the sensor. We decided to abandon wired sensor options due to creating vibrations from the wire and due to the high visibility and chance for entanglement that wires introduce. Thus the sensors need to stand alone and be large enough to contain A/D converters, a small processor, radio, and batteries for the mission life. This makes them large enough that it is hard to carry lots of them on a vehicle, and it becomes unattractive to abandon them. Having one vehicle per sensor is achievable, but it makes the complete system more complex, with a higher signature, and involves significantly more hardware than using a single vehicle. Also it is not necessary for reasons of rapid array deployment since deployment over a few hours is acceptable. To traverse a few kilometers over rugged terrain, insert and remove sensors, and have the load capability to support the signal processing unit, the vehicle has to have a minimum size that is roughly the size of a mid-sized RATLER™. Carrying several stand-alone seismic sensors and an acoustic sensor array won't make it much larger than carrying just one. Using small "daughter" vehicles to insert sensors would be a viable option if they could remain with the sensor without coupling noise into the sensor and if they could be made small enough that they do not constitute a significant payload increase, but there isn't any significant benefit to this approach.

Outdoor Testing

Outdoor tests were conducted to obtain empirical evaluations comparing the performance of a number of sensor-to-ground coupling configurations and of several vehicle configurations. Outdoor tests were also conducted to gather acoustic and seismic data that could be used for algorithm development.

Sensor-to-Ground Coupling

The main issues addressed were a) the ground coupling efficiency, i.e. the magnitude of the signals using various coupling approaches, and b) the noise floor generated by the wind coupling into the sensor via the vehicle and/or sensor package body either directly or through the adjacent ground. To make the configurations as realistic as possible, we fabricated dummy housings to perform as wind models for the stand-alone sensors. The stand-alone housing was simulated by a plastic shape machined to fit the sensor up inside it so the bottom of the plastic shape is flush with the ground. The plastic machined housing with sensor weighs 3.55 lbs, which is about what is expected for a complete system housing 2.6 lbs of batteries. Outdoor tests were conducted at the seismically quiet FACT site east of the Sandia Solar Power Tower. Each configuration used an identical commercial geophone and was recorded on an 8-channel recording device.

The configurations we planned to test were as follows:

1) Geophone with standard 3” spike alone or with a sandbag. This is the reference configuration normally used in sensing and it will serve as the standard against which the other configurations are measured.
2) Geophone attached to a metal plate laid on the ground.
3) 1/4” lag screw coupling device with a simulated housing. The 1/4” lag screw perfomed the best of the screw options. This arrangement simulates Options 3 and 4 (or Option 1 with the vehicle standing off).
4) 1/4” lag screw coupling device with a simulated sensor housing placed next to it. This evaluates the effect of separating the sensor package from the sensor to see if this reduces wind vibration coupling from the housing directly into the sensor. This arrangement adds significant complexity to the deployment and retrieval design, but needs to be evaluated to determine wind coupling effects.
5) 1/4” lag screw coupling device with a robotic vehicle sitting over top of it. This simulates Option 1 with the vehicle attached to the sensor by a wire.
6) 1/4” lag screw coupling device with a simulated housing but with the lag screw slightly wiggling in the hole representing a wobbly insertion or “stripped” thread condition.
7) 3/4” auger with a simulated housing. This represents the auger coupling option for Option 3 and 4 (or Option 1 with the vehicle standing off).
8) Geophone laid on the ground with a mini-RATLER™ vehicle pressing it to the ground. This tests Option 2 that uses small “daughter” vehicles deployed from a mother vehicle to place the sensors. In this configuration the vehicles do not stand-off from the sensor, and they may not be able to insert a penetrating coupling device since the vehicle is so small and lightweight (4.8 pounds).

The actual test configurations were a slight variation of this plan. By channel number the configurations tested were as follows. Only seven channels were available.

1) 3/4” auger with stand-alone sensor package
2) 1/4” x 3” lag bolt with stand-alone sensor package
3) standard 3” spike on bare geophone
4) 1/4” x 3” lag bolt with stand-alone sensor package
5) standard 3” spike with stand-alone sensor package
6) metal plate laying on ground
7) geophone with no coupling device with mini-RATLER™ on top

Configuration 7 was located in soft dirt while the other configurations were initially located in a packed roadbed and were then relocated to soft dirt for a second test. Figure 4 shows how the sensors were laid out for the test. Results for the first test showing the signal detected after a hammer blow to a plate on the ground 50 yards away are shown in Figure 5. Problems were encountered during the tests including excessive noise on the recorder and a faulty time stamp on the environmental recorder that prevented correlation with wind speed. Still, some conclusions could be drawn. First, all configurations were within 30% of each other in signal strength, so it is probably acceptable to eliminate the ground coupling spike altogether. Second, we found the signal was greater in the soft powdery dirt as compared to the packed roadway dirt.

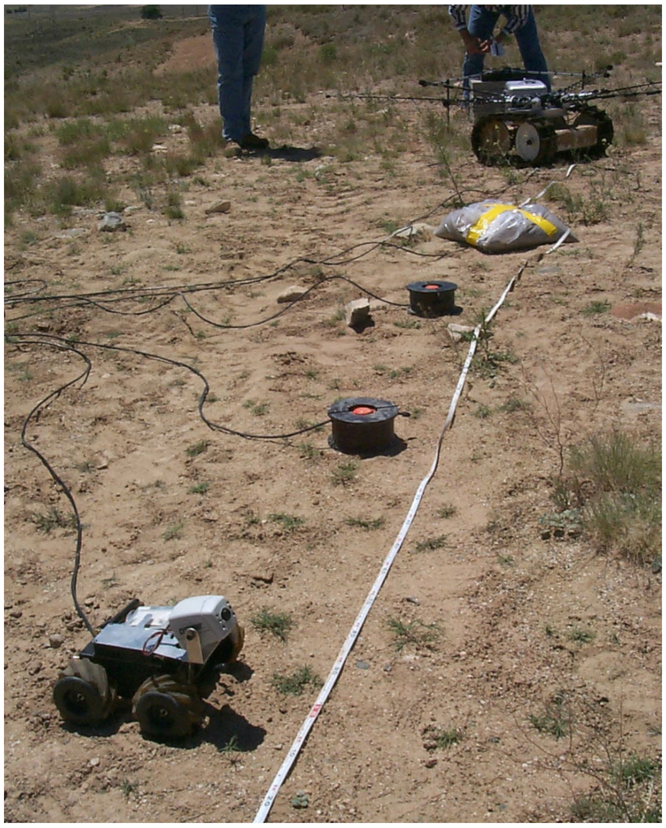


Figure 4 – Seismic Coupling Test Layout

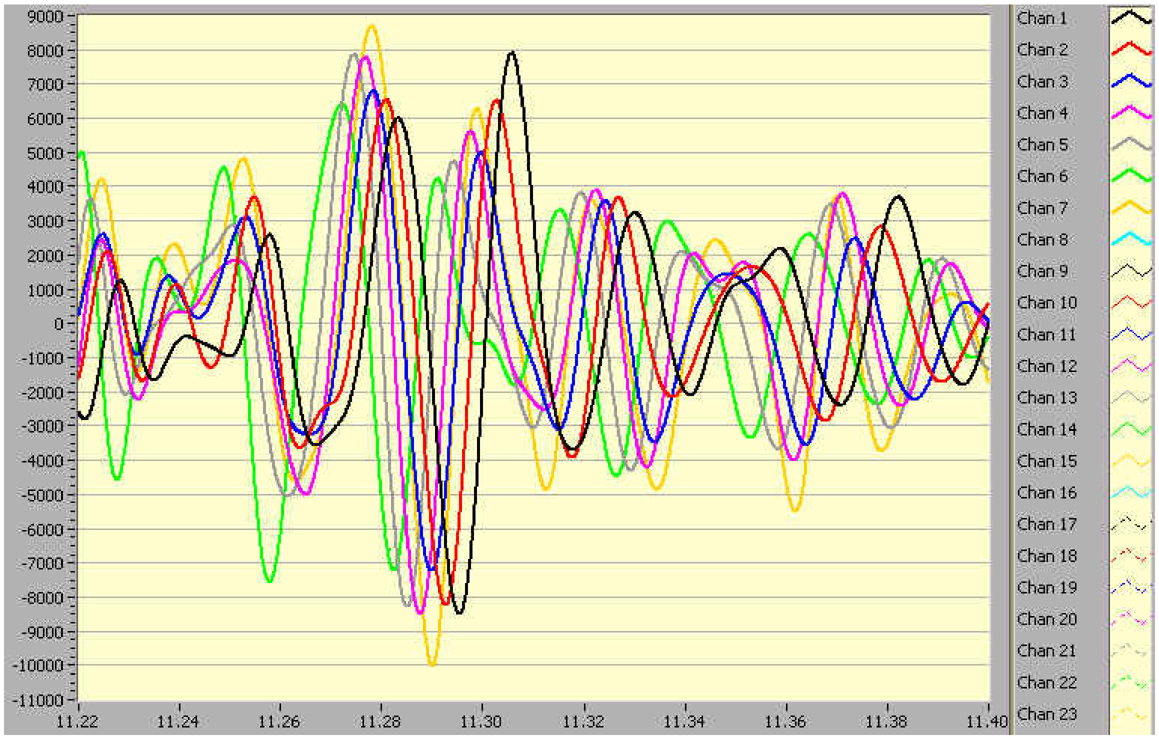


Figure 5 - Seismic/Acoustic Empirical Test Results of Signal Coupling Devices

Wind Noise Tests

On May 15, 22 and June 21, 2001 field experiments we conducted to obtain quantitative data on the impact wind has on tentative design of the acoustic array subsystem. Since heuristics and operational considerations were used to generate a tentative design for the acoustic array, the impact of wind on several elements of the array design needed further exploration by quantitative means. The questions to be answered were:

- The tentative design called for the acoustic sensors to be mounted on thin, retractable booms. Since the booms are flexible, will an unacceptable amount of noise result if wind blows on the booms causing them to flex and will the displacement be large enough to produce a significant position error?
- The initial design of the acoustic array identified the microphones used in cell phones as the best candidates for use as the acoustic sensors because they are small, lightweight and rugged. Would these microphones provide an adequate signal for this application?
- The retraction capability of the booms on which the acoustic sensors are mounted presents some engineering challenges as far as getting the signal from the microphone to an A/D on the robot. Coiling the leads around the boom in a manner similar to a telephone cord is advantageous from a mechanical standpoint. Would wind acting on the leads coiled around the booms introduce noise in the acoustic signal, either through mechanical impact on the boom or increasing the amplitude of the boom's flexing in the wind?

To answer these questions, the signals from a partial prototype acoustic array were recorded over a period of a several hours resulting in data representative of a variety of wind conditions. A speaker emitting a constant 200Hz tone 10m away from the array provided a reference acoustic signal to facilitate comparison across different sensor configurations and wind conditions. Figure 6 shows the acoustic array configuration on a robotic vehicle during the test. Figure 7 shows the test configuration with an acoustic generator speaker in the rear.

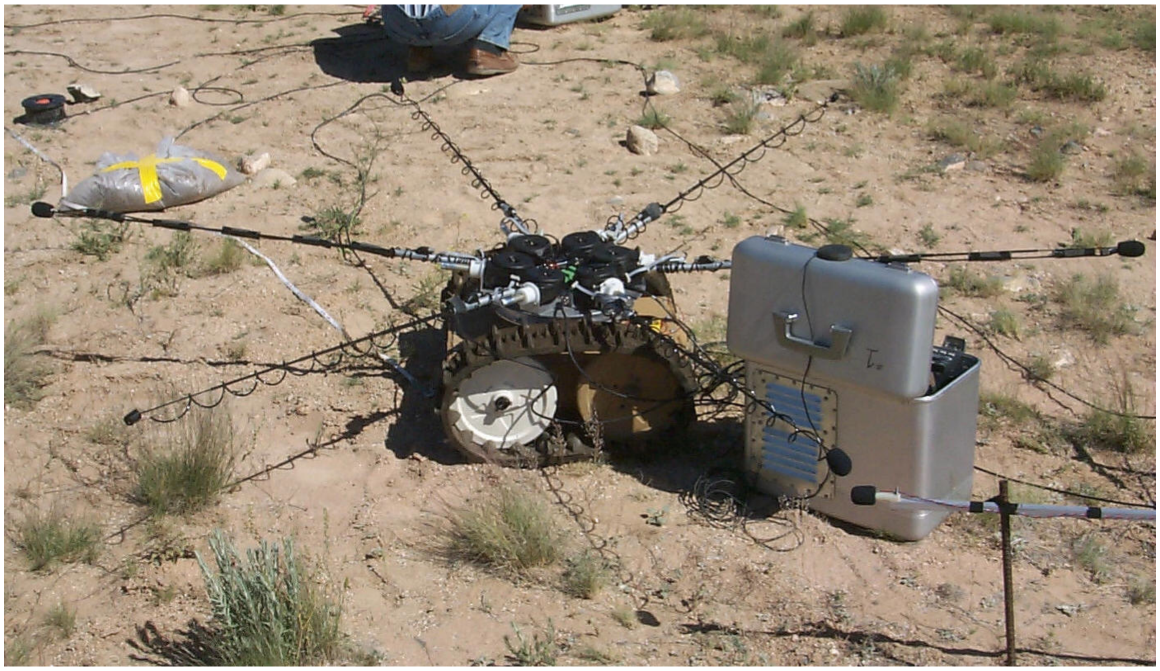


Figure 6 – Acoustic Sensor on Robotic Vehicle During Test

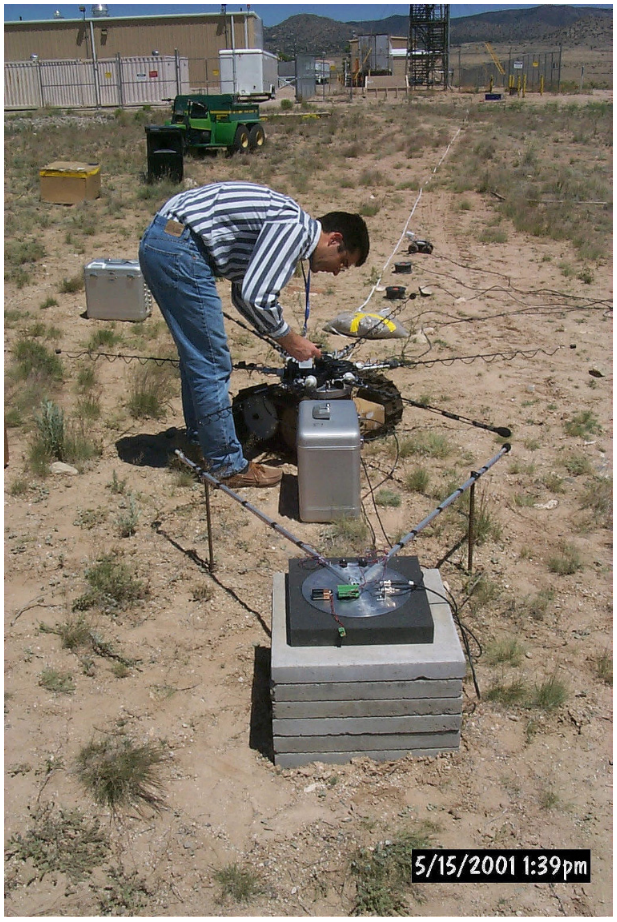


Figure 7 – Acoustic Test Configuration

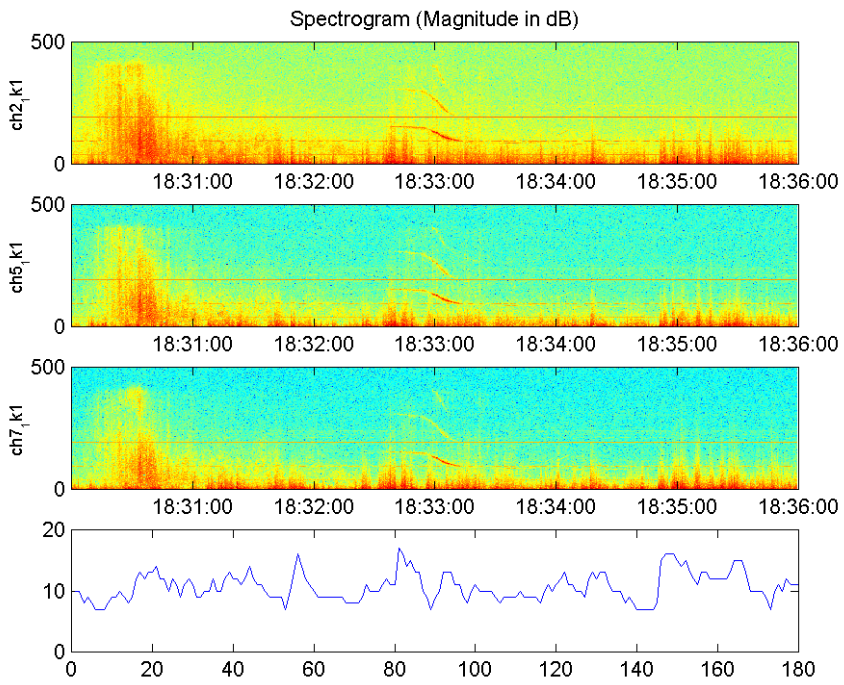


Figure 8 - Spectrograms and associated wind speed for a representative segment of data.

Figure 8 shows three spectrograms and the corresponding wind speed for a representative six-minute segment of data. The top spectrogram is for a microphone on a retractable boom with the lead coiled loosely around the boom. The middle spectrogram is for a microphone mounted to a stiff rod with the lead securely fastened to the rod. The bottom spectrogram comes from a microphone mounted on a tripod. All microphones were located within 60cm of each other. The spectrograms plot the frequency content of the recorded signal as a function of time. The constant 200Hz reference signal shows up on the three spectrograms as a horizontal line. There are three time segments of interest. The period from 18:30:20 to 18:30:40 shows the broadband signal produced by a low-flying jet airliner flying by. The discrete spectral lines from 18:32:45 to 18:33:15 exhibit the Doppler effect as a propeller aircraft passed by. Finally, there is increased low frequency content from 18:34:50 to 18:36:00 due to sustained winds. As illustrated by these three spectrograms, the signal from microphones on flexible booms is only slightly more susceptible to wind noise than rigidly mounted microphones. Moreover, no significant difference was found between the signals from microphones whose leads were securely fastened compared to those whose leads were loosely coiled about the flexible booms. The following conclusions were drawn from the experiment.

- The observed amplitude of the wind-induced vibrations did not exceed approximately 2cm indicating that wind induced vibrations will not result in significant position error with respect to the nominal position of the microphone. Nor did wind induced flexure of the retractable booms cause significant levels of noise.

- The performance of the cell phone microphones were deemed to be acceptable when covered by simple windscreens. It is expected that more expensive acoustic sensor options would not provide a significant performance advantage for the operational environment the robot is being designed for.
- Coiling of the microphone leads did not induce significant noise.
- Wind introduces noise, primarily at low frequencies. However, below 10mph, the wind noise in the range of target frequencies is minimal. The acoustic array will be able to operate at some reduced capacity at wind speeds between 10 and 20mph, especially for high frequency tones. The acoustic array is expected to have poor performance at wind speeds above 20mph.

These results indicate that the features in question of the current acoustic array design do not result in significant additional noise in the signal when the array is subjected to wind, especially winds below 10mph.

Array Adaptation Simulation Field Experiment

On July 19 and August 14, 2001, field experiments were conducted which simulated the data the robot would record during a mission. The benefits of this test were threefold.

- By simulating the array deployment and modification sequence of activities, a better understanding was obtained of the actual tasks that a robot would perform. This understanding was intended to aid the LDRD participants in designing a system that performs these tasks with the greatest degree of autonomy and effectiveness.
- Raw data was collected that could be input to signal processing algorithms under development. This was of great utility to the software development effort because real data could be included in with the data used to test the software without actually deploying the software on the robotic platform. Thus the array signal processing software could be further developed before being deployed on the robot.
- The tests provided early performance data on signal levels, gains from array adaptations, etc. This early performance data helped identify strategies with the greatest benefit as well as areas that required attention.

One aspect that the field experiment brought into better focus was the fact that array signal processing requires the determination of certain parameters from the signal environment that are used in the array design process, in particular medium velocity, signal of interest and approximate source location. These parameters are generally supplied by the human who specifies the sensor array configuration, but when the array is configured autonomously it must be identified through other means. This field test forced us to identify these design parameters and make provision for strategies for their evaluation by automated means. Figure 9 shows traces from this field experiment. It shows a seismic impulse as it travels past increasingly distant seismic sensors in the array. This data will be used in the velocity estimation routines that will estimate velocity by relating the pulse’s travel time and the distance to the sensors.

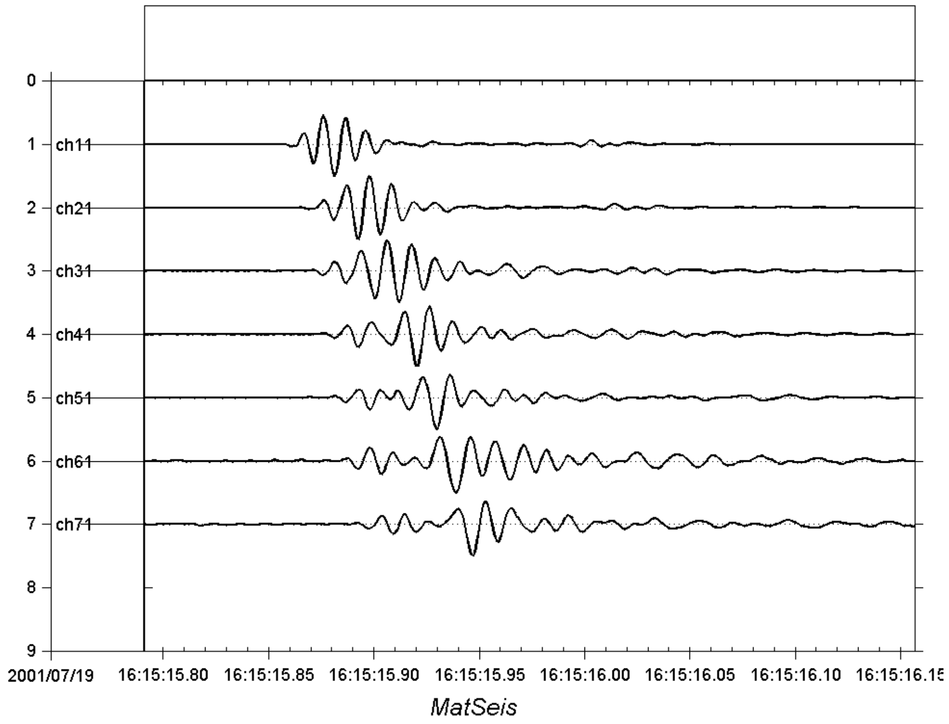


Figure 9 - Propagation of an impulse across a linear array from which velocity can be estimated

Simulation of the process of evaluating array design parameters was just one step simulated in this field experiment. The field experiment also simulated the process by which the robot makes an initial mapping of the signals in terms of frequency content and approximate location and then modifies the array to refine the initial characterization of the environment. It also simulated the process of adapting an array to respond to the introduction of a signal not present during the initial mapping phase.

As part of the initial process of determining the effectiveness of different array configurations on signal processing, the graph in Fig. 10 was generated showing the array sensitivity vs. source angle. The graph shows two array configurations, one with a normal population of sensors and one with a sparse population of sensors. The graph gives the sensitivity (y axis) vs. source angle (x axis) for a signal coming from a direction of 5°, and it illustrates the different benefits each configuration offers. The sparse array has a very narrow main lobe resulting in good angular resolution while the regular array has lower side lobes resulting in less interference from energy from other directions.

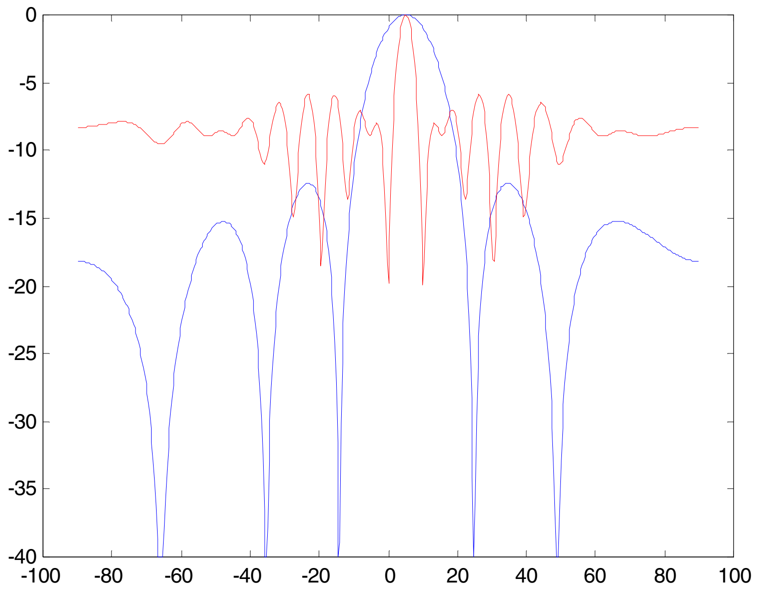


Figure 10 - Signal Processing Model Showing Benefits of Reconfigurable Array

Signal Algorithm Analysis and Design

During Year One, high-level analysis and design of software for the array signal processing subsystem was begun, including identifying the interface between this subsystem and the rest of the vehicle. The array signal processing subsystem interfaced with the rest of the system through an embedded real-time software subsystem. Efforts were begun on the design of the interface that supports the transmission of data into and out of the array signal processing subsystem with the sensors and the vehicle controller.

An important element guiding the analysis and design process was the creation of a scenario of usage. In this case, the scenario of usage outlines the mission goals and the major tasks performed by the robot in pursuit of those goals. This was used to guide more detailed design of the array signal processing software.

The overall array signal processing software subsystem was decomposed into approximately twenty software modules. Each module performs a specific task. “Stubs” were created for these software modules. The stubs embody the overall high-level architecture of the array signal processing subsystem without implementing the details of actually performing the work. The stubs helped validate the decomposition of the high-level architectural features of the array signal processing subsystem in terms of task partitioning, calling sequence/hierarchy, and data flow.

Array Configuration Placement Sequence

A meeting was held May 4, 2001 with representatives from the sensor array signal processing department and the robotic department to jointly define a robust autonomous array placement sequence. The sensor department had already defined a flowchart of events and decisions mapping the control sequence necessary to perform a site survey and then reconfigure for optimal target bearing determination and then gain enhancement for site monitoring. The main issues are that while the signal processing algorithm can define the desired new array configuration and communicate that to the robotic vehicle to implement, real world conditions will prevent placement of sensors in some locations. The meeting explored options to achieve a satisfactory, if not ideal, array configuration with the least use of resources, given the limited knowledge base the vehicle will have of the site and given operation in an autonomous mode.

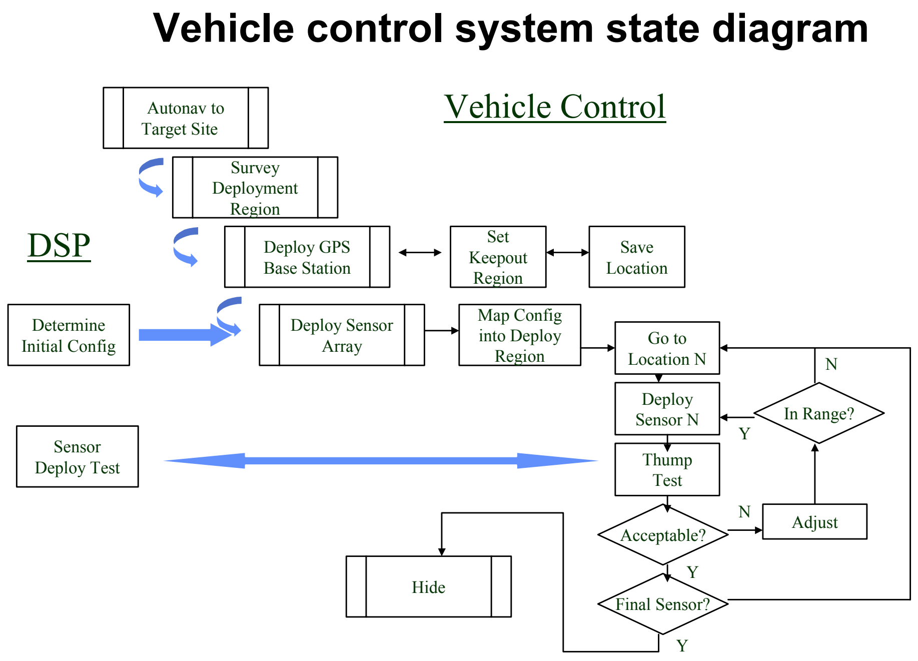


Figure 11 – High-level Sensor Deployment Sequence State Diagram Showing Vehicle/DSP Interaction

The discussion started with a review of available resources and conditions, and a review of the desired outcome. From the signal processing side, priorities were identified that it is crucial to know the final location of the sensor well and less crucial to have the sensor placed exactly in the ideal location. Sensors need to be placed in a configuration relative to each other, not in absolute locations. From the vehicle side, we can assume that because the vehicle will be navigating autonomously it will have an internal map of the region with elevations and some large obstacle data based on the best available photos or SAR data. The vehicle will also have obstacle avoidance sensors onboard and will be able to locate at least above-ground obstacles (vegetation, rocks, gullies, etc.).

In order to minimize interaction between the signal processor and vehicle control systems, we decided that the vehicle should be responsible for installing the array according to guidelines given by the digital signal processing (DSP) unit. The signal processor will provide the vehicle

1) relative placement information for the sensors,
2) an acceptable radius about the desired placement point for sensors, and
3) the minimum number of sensors that must be placed to enable the array to function.

One possibility is to have the vehicle perform an above-ground site survey prior to deciding where to place the array. Alternatively, the vehicle could use its internal map with whatever existing data is available to determine the optimal array placement. Optimal placement would probably include factors such as ground slope, and distance from known obstacles. The vehicle could also place sensors in a sequence that would allow it to take advantage of array symmetries (i.e. shifting longitudinally for linear arrays in case a sensor could not be placed. As the vehicle worked in the operating space it would add to its internal map of known obstacles. If a sensor can not be placed, the vehicle would move outwards in a spiral from the target trying to place the sensor until the maximum error is achieved. If the vehicle fails to insert the minimum number of sensors, it removes all the sensors and adjusts the array position. Once it completes the array installation, the vehicle communicates to the signal processor which sensors were installed and each sensor's actual location.

System Configuration

Once we arrived at a design concept, we needed to select specific hardware and components. This included the robotic vehicle body and component hardware, ancillary operation components, vehicle software, the seismic and acoustic hardware, and signal processing electronic hardware and software. This occurred during the latter part of the first year and during the second year of the LDRD program. The first task was deciding on a vehicle design.

Robotic Vehicle Hardware

Rather than develop a new vehicle, we decided to use a SandDragon robot as the mobile platform. This vehicle was of the necessary size and configuration, and it had the processing capability to support autonomous navigation. Appendix A provides a Fact Sheet on the SandDragon vehicle. This decision enabled the project to leverage ongoing development work on other projects, and we were able to acquire a dedicated vehicle with a minimum of additional development work. Thus the project was able to focus on advancing the navigational and sensor capabilities needed for this project rather than put its efforts on vehicle development.

The decision to deploy multiple sensors from a single vehicle rather than make each vehicle be a separate sensor required the development of a sensor deployment and retrieval mechanism. Initially we used an existing arm that had become available to develop our proof of concepts. By the end of the first year, the project had arrived at a first cut configuration of the vehicle concept, shown in Fig. 12.

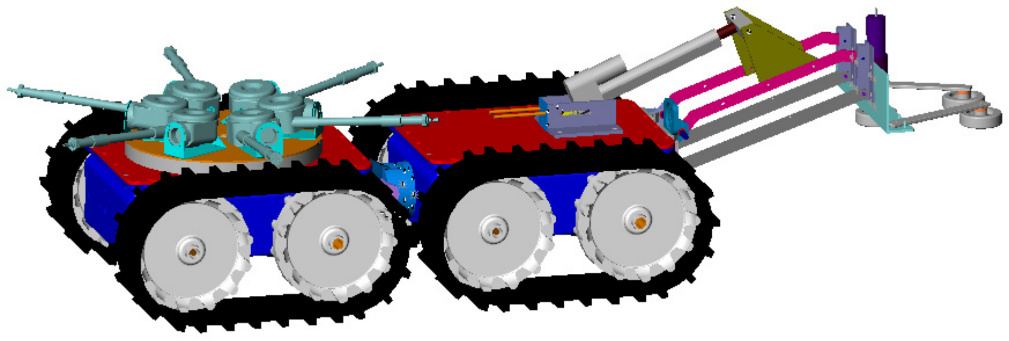


Figure 12 - Acoustic and First Prototype Seismic Sensor Array Deployment Hardware on SandDragon Vehicle

This development design was intended for concept and hardware evaluation only. The deployment arm was a quick solution available without much development effort, and it was intended to be replaced with a more suitable and integrated arm later in the project. The project also planned to design a multi-sensor stack loader to enable autonomous multi-sensor array deployment. However, inadequate funding in subsequent years kept this feature at a low priority and it was eventually never implemented. The SandDragon vehicle was large enough to accommodate the acoustic sensor on the rear half and the seismic stack loader on the front.

SMART Software Architecture

The next task was to develop a modular software architecture to replace the single-threaded software architecture that had be used by Sandia's Robotic Vehicle Range for vehicle control software until this time. Modular software architecture is essential to being able to rapidly implement new vehicle control systems. It enables previously developed software to be reused with a minimum of effort so only new capabilities need to be developed. It also speeds the debugging and quality control of software since each module of the software can be individually validated. The existing vehicle code used on previous robotic vehicles did not separate or modularize the different capabilities or functions so it was time consuming to adapt the code for new applications.

After some review, this project settled on adapting SMART (Sandia's Modular Architecture for Robotics and Teleoperation) for use in this project and as the basis for future robotic vehicles developed at the Robotic Vehicle Range. SMART has undergone extensive development over more than a decade, and offers many of the advanced yet flexible features this project was seeking. It was originally developed for use with manually manipulated robotic arms, enabling the use of a wide range of operator control input devices and teleoperated arms while guaranteeing dynamic stability. Some of the beneficial features for use as a robotic vehicle control system are given in Table 8.

Table 8 – Desirable Modular Features of SMART

Modular
- Clearly delineated inputs and outputs
- Easily tested and validated – improves software QC
- Portable and reusable code which can be connected with other modules to rapidly reconfigure code or develop new code
- Rapid and clean development and integration of new capabilities (behaviors and hardware) with existing code

Multi-thread
- The operating system handles scheduling of thread execution so it is transparent to the code developer leaving him to focus on module functionality
- Supports modularity architecture

Multi-OS, Multi-platform ready
- Enables rapid compilation for multiple operating systems. This enables cross platform development work, flexibility in vehicle and target selection, rapid upgrade in future evolution or variants
- Longer code life as systems evolve.

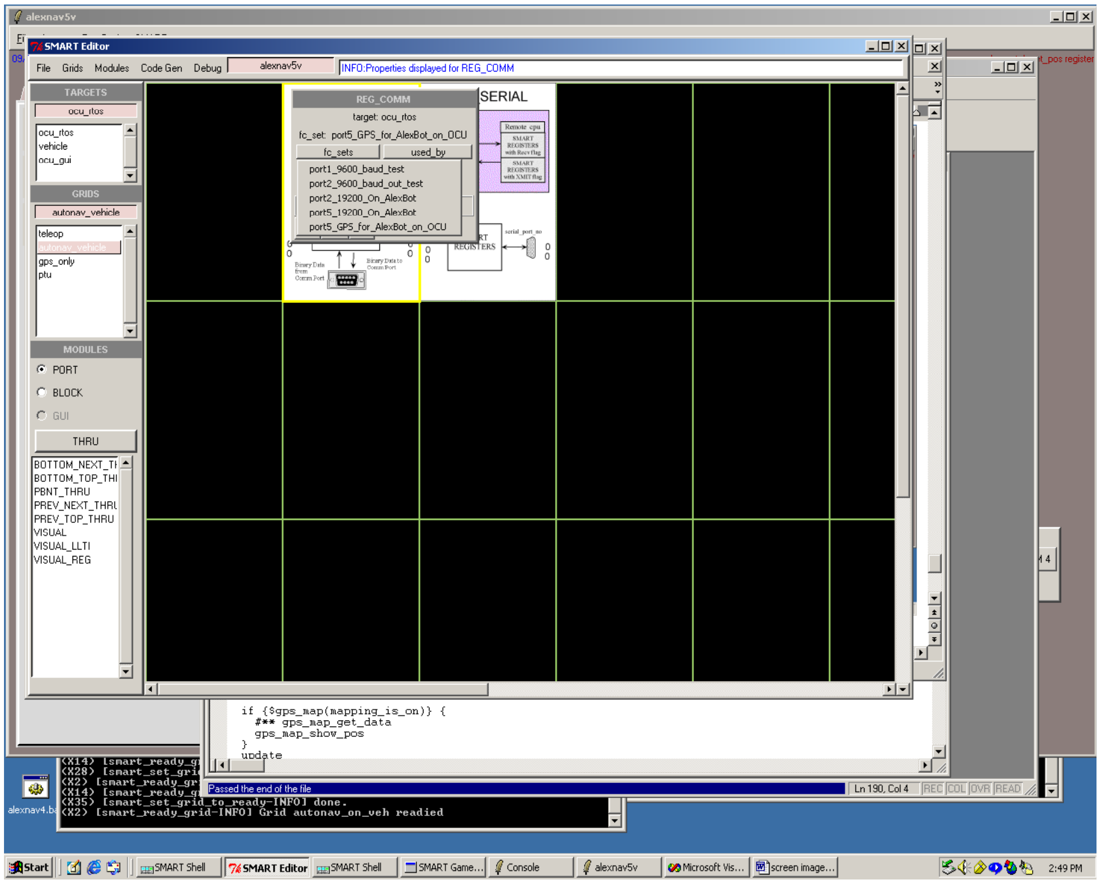
Control stability
- Control stability for tele-operated manipulators for a wide range of input and actuator devices that can be interconnected and used in a wide range of configurations.

Multiple behaviors
- Multiple behaviors can co-exist on the controller and can be switched on and off independently.

SMART was written to be highly modular. Each software module encapsulates a certain functionality and contains the code to implement that functionality and user-accessible configuration parameters to adapt it to a range of applications. Each module contains its own data structure with parameters it uses locally, and each includes a test code to individually exercise its functionality before it is integrated with other modules. Not only is the software modular, but it is very standardized. This makes it easy for the programmer to read and understand any of the code. Code functions for initialization, the body functionality or thread loop, access to the module's data structure, and exiting are all standardized and follow a set naming convention. Each SMART module consists of a standard core set of files, with the flexibility to include additional files needed for functionality.

When running, SMART actually operates as an operating system on the target. It is fairly mature and sophisticated in that many desirable support features are already developed and available to the programmer through function calls. SMART consists of a core set of modules that handle SMART's execution, including initialization of all the user's modules, communication between modules using built-in "connectors" to pass standard position and velocity data, and regular calls to each module to exercise its body code in a looping manner. SMART provides support for common data registers that can be accessed by any module to pass generic data between modules. SMART includes communication modules to transparently pass these data register values between processor platforms (targets). SMART enables a module to “register” functions or commands so they can easily be accessed by other modules or an operator at a console by calling the function with the appropriate parameters. Several utility features have also been developed, including print message support, clock/timing support, and support for multi-tasking and semaphores. Message printing support provides three levels of message priority, message buffering, and the ability to turn message printing on and off. Messages include the file and function initiating the call, which facilitates debugging. SMART has abstracted calls to functions that are specific to the operating system (OS) or processor hardware and remaps them automatically to the OS- or hardware-specific call through the use of sophisticated ‘make’ files that automate code compilation. Thus the SMART code can be compiled for several different operating systems with no change to the SMART code, only by resetting a flag in the make file.

SMART supports the assembly of modules into compiled code using a graphical user interface (GUI) code called the SMART Editor. (Fig. 13) The core SMART modules are included automatically behind the scenes, and the application-specific functional modules are added and connected graphically on the screen using the cursor and copy/paste techniques. Application-specific parameter sets can be selected for each module by double clicking the module icon to pull up a selection box, or new selection sets can be added by clicking the Edit button to pull up the file needed to add the code. The user creates one or more ‘grids’ of SMART modules for each processor/OS platform (target). Behaviors are defined by enabling one (or more if they are not mutually exclusive) grids on each target. This can be done when operating by using the registered commands.



```
if {$gps_map(mapping_is_on)} {
  #** gps_map_get_data
  gps_map_show_pos
}
update
```

```
(X14) [smart_ready_g
(X28) [smart_set_gri
(X2) [smart_ready_gr
(X14) [smart_ready_g
(X35) [smart_set_grid_to_ready-INFO] done.
(X2) [smart_ready_grid-INFO] Grid autonav_on_veh readied
```

Figure 13 – SMART Editor showing application-specific modules being assembled into a grid, and the pop-up window enabling one to select a parameter set for a module

For this project we have three targets; the vehicle RTOS (real time operating system), the base station RTOS, and the GUI (graphical user interface). The operator interacts with the GUI which is coded in Tcl/Tk, a scripting and graphical toolbox software language and is designed to run on the same processor as the base station RTOS, while the two RTOS targets perform the operational functionality at the base station and vehicle. There are two behaviors; teleoperation and auto-navigation, that use different grids on the base station and vehicle targets.

While much of the core code was available in SMART, a significant amount of work was needed to add new modules and functionality to enable this project to achieve its goals. These included creating the ability to perform waypoint navigation using GPS sensor readings for location, falling back to dead reckoning navigation when the GPS data is not sufficiently accurate, controlling the sensor deployment/retrieval hardware, creating the GUI interfaces for the vehicle and the payload status and commands, and creating the interfaces to the compass and GPS sensors and the payload processor to begin with. Other tasks included transmitting the GPS differential correction data to the vehicle GPS from the base station during the development stage and from a deployable GPS base station unit for the final product, and translating the sensor deployment requests of the sensor payload into deployment command sequences for the vehicle.

Sensor Payload Configuration

The sensor payload configuration evolved significantly as the AMGS configuration coalesced to a radio-connected (for the seismic sensors) central-DSP design. In addition to the major challenge of implementing real-time autonomous signal data processing in hardware and developing array optimization algorithm software, the payload challenge also included developing the hardware and software for multiple real-time data transmission and time synching by radio for the seismic signals. None of this had been accomplished before. The hardware configuration converged the design shown in Fig. 14. It uses a TI C6711DSP processor for the signal processing and array optimization software mounted in the vehicle and talking to the vehicle control processor over an RS-232 serial port. The sensor pod would contain the seismic sensor geophone and a TI C5510 processor. Communication would use a wireless LAN (local area network) PCMCIA transceiver card on either end. The transmission rate was selected to support up to 16 separate sensor pods all transmitting at once.

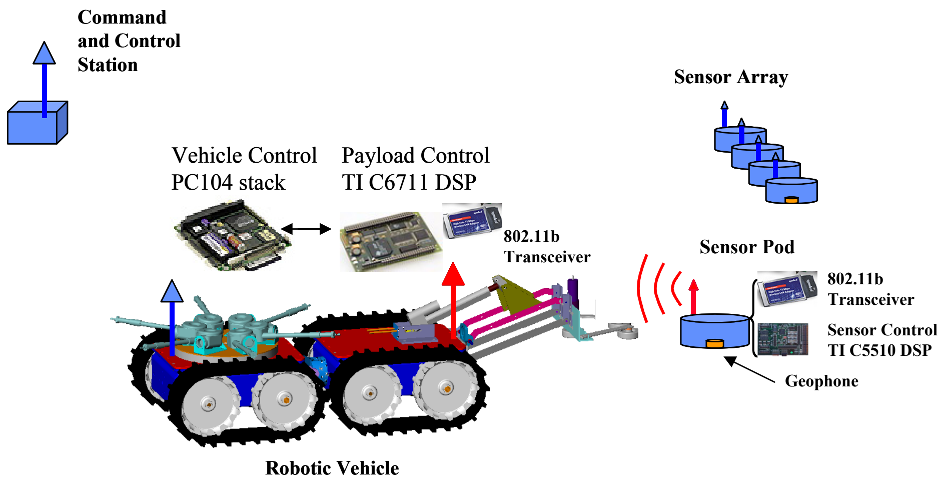

Command and Control Station
Sensor Array
Vehicle Control PC104 stack
Payload Control TI C6711 DSP
802.11b Transceiver
Sensor Pod
802.11b Transceiver
Sensor Control TI C5510 DSP
Geophone
Robotic Vehicle

Figure 14 – Sensor System Component Configuration

A modified version of this would be used for the GPS base station which would be deployed by the vehicle at the sensor deployment site. Instead of the seismic sensor and TI C5510 processor, the GPS base station sensor pod would incorporate a GPS receiver and a Tern A-Engine single-board computer interfacing to the wireless LAN transceiver. (Fig. 15). Binary differential correction data and ASCII log data from the GPS base station and ASCII commands being sent to the GPS unit would be passed through the DSP processor to and from the vehicle PC104 stack processor. During the development phase, the base station GPS unit is located at the Vehicle Command and Control Station, and differential correction data is passed to the vehicle through the control station-to-vehicle data link.

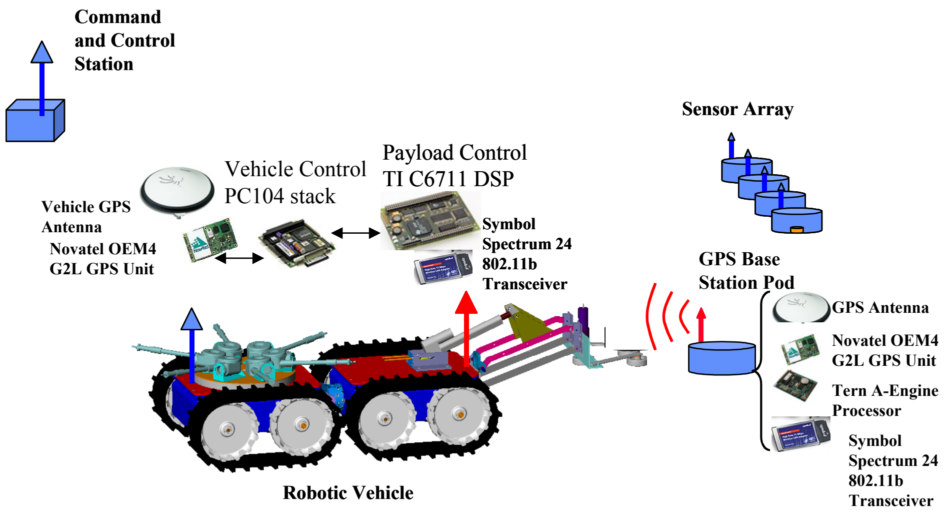

Command and Control Station
Sensor Array
Payload Control TI C6711 DSP
Vehicle Control PC104 stack
Vehicle GPS Antenna
Novatel OEM4 G2L GPS Unit
Symbol Spectrum 24 802.11b Transceiver
GPS Base Station Pod
GPS Antenna
Novatel OEM4 G2L GPS Unit
Tern A-Engine Processor
Symbol Spectrum 24 802.11b Transceiver
Robotic Vehicle

Figure 15 – GPS Base Station Sensor Pod Configuration

Vehicle System Hardware Design

The SandDragon vehicle was selected to be the platform on which the hardware for this LDRD was mounted. The base vehicle consists of two bodies that were always connected. The front vehicle was used to hold the seismic sensor deployment/retrieval hardware and the seismic puck(s). The rear body was designated to hold the acoustic sensor mechanism.

SandDragon Robotic Vehicle Hardware

The existing SandDragon vehicle was modified somewhat for the specific requirements of this project, but the core functional features remained essentially unchanged from the original design. Figures 16 and 17 shows the main functional systems and interconnects for the front and rear bodies of the vehicle as configured for this LDRD. The front vehicle contains two PC-104 form factor stacks of boards and one custom sensor stack mounted on a custom designed motherboard that acts as the interconnect and distribution for the various electrical busses. The rear body contains another custom sensor stack that includes the Novatel Pro-Pak-4E-RT2 OEM3 L1/L2 RTK GPS unit, which was installed after removing the GPS unit's Pro-Pak housing. Both bodies contain batteries, motors, and motor amplifiers.

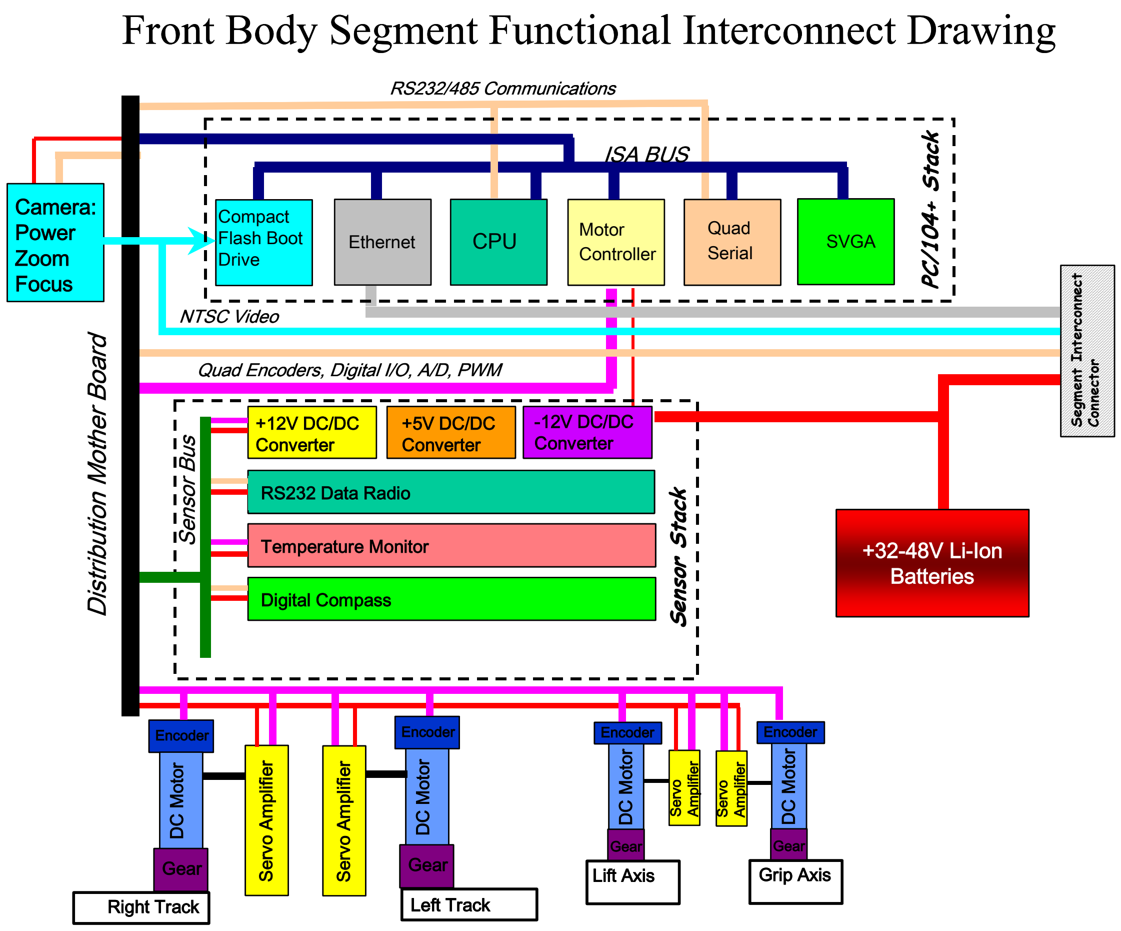


Figure 16 - SandDragon Robotic Vehicle Front Body Functional Diagram

Rear Body Segment Functional Interconnect Drawing

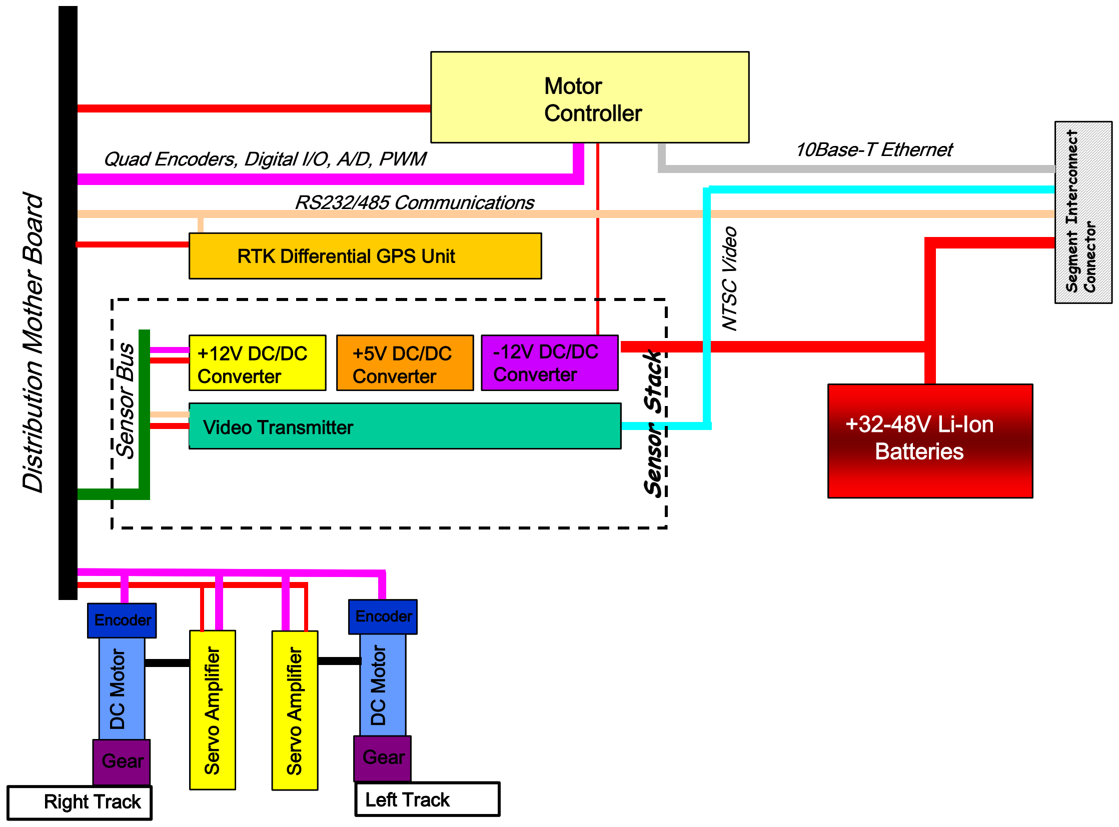


Figure 17 -SandDragon Robotic Vehicle Rear Body Functional Diagram

Batteries are configured so that the vehicle can be run on as little as two or as many as six BB-X90 form factor batteries, where the X is replaced by a number designating the battery chemistry. (Fig. 18) When running with two batteries, they are mounted in one body on the left and right outboard locations leaving the center location free. (Fig. 19) Although it is possible to operate when stationary and running forward, this configuration tends to lack sufficient power for more aggressive maneuvers, such as spin turns, which can cause the computer to reboot. Running with four batteries, in the four corner positions works better for any kind of maneuvering. BB-390 Nickle-metal hydride (NiMH) batteries and lithium-ion (Li-Ion) rechargeable batteries are available from sources such as Brentronics and Ultralife. We used Li_Ion batteries in this project as they have twice the capacity and a lower weight than the NiMH batteries. Each 1.3kg battery has a nominal 16V operating voltage (in parallel, 32V if wired in series) and 8 amp-hrs of capacity (wired in parallel and measured down to 10V). They can be recharged in a unit such as is shown in Fig. 20. The battery wiring diagram is given in Fig. 21, shown for BB-390 NiMH units.

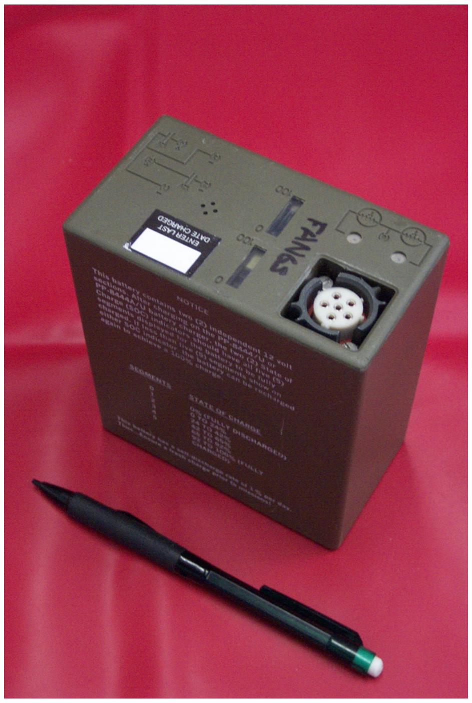


Figure 18 - BB-X90 Form Factor Battery

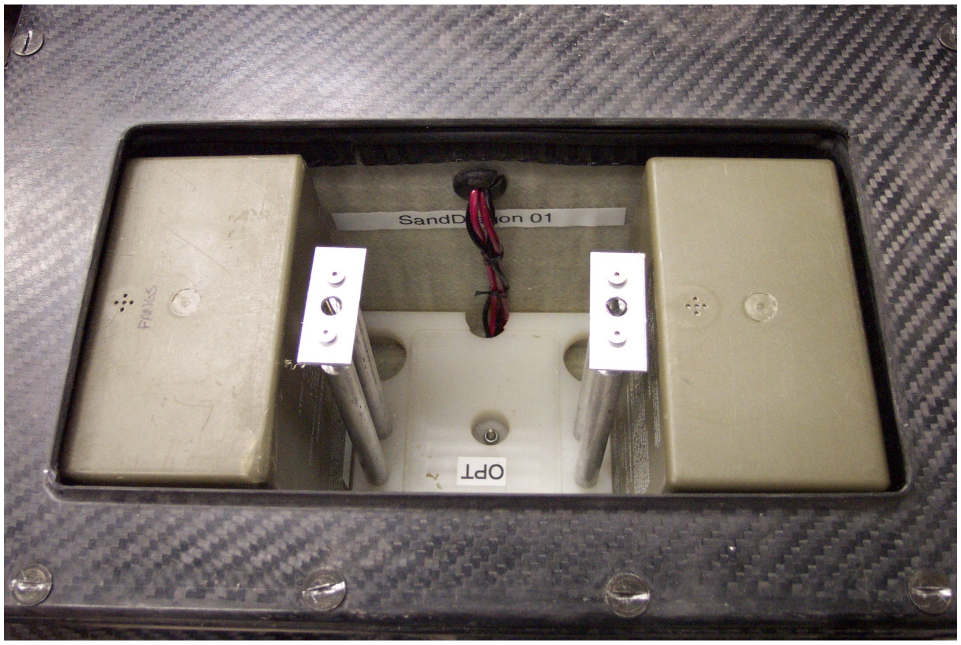


Figure 19 - SandDragon Battery Compartment with Two BB-2590 Batteries in Place

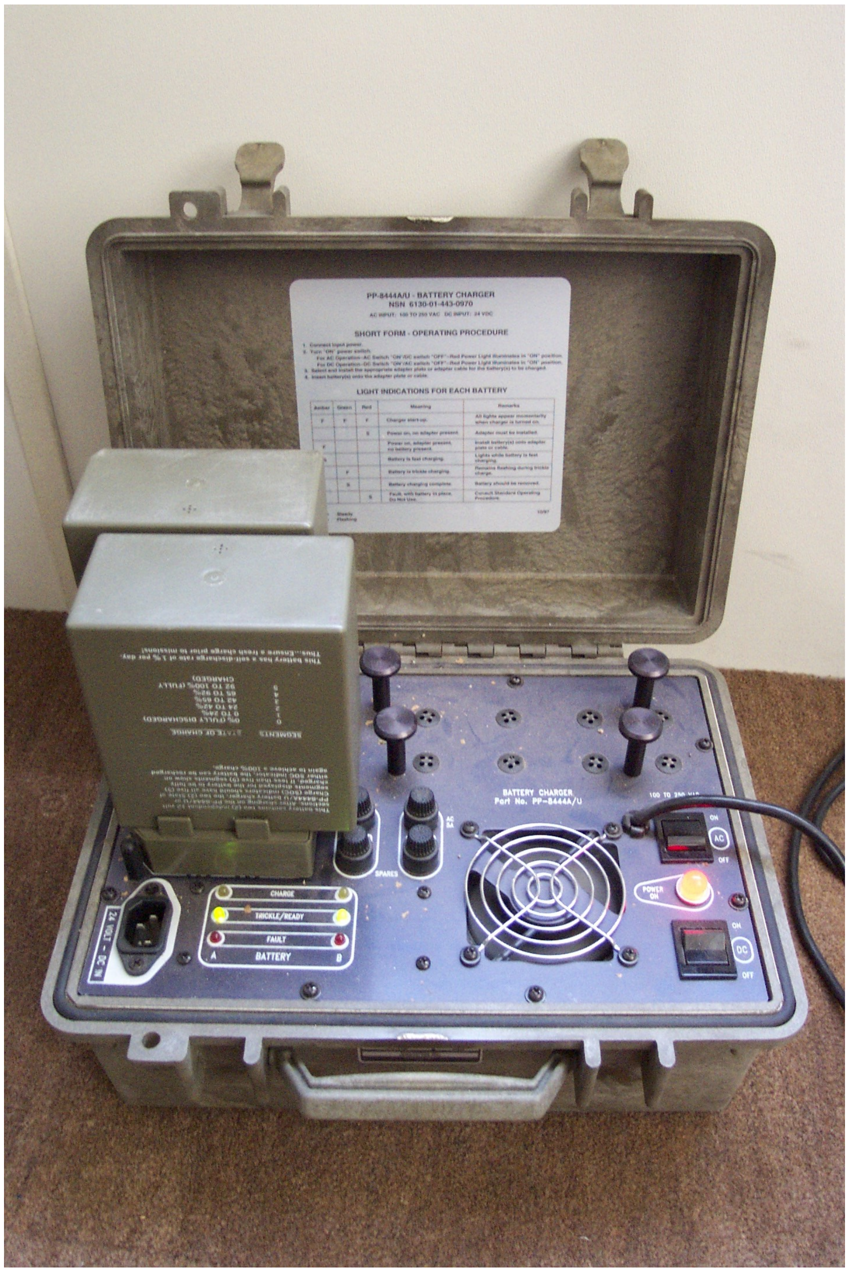


Figure 20 - Charger for Li-Ion Rechargeable BB-2590 Battery

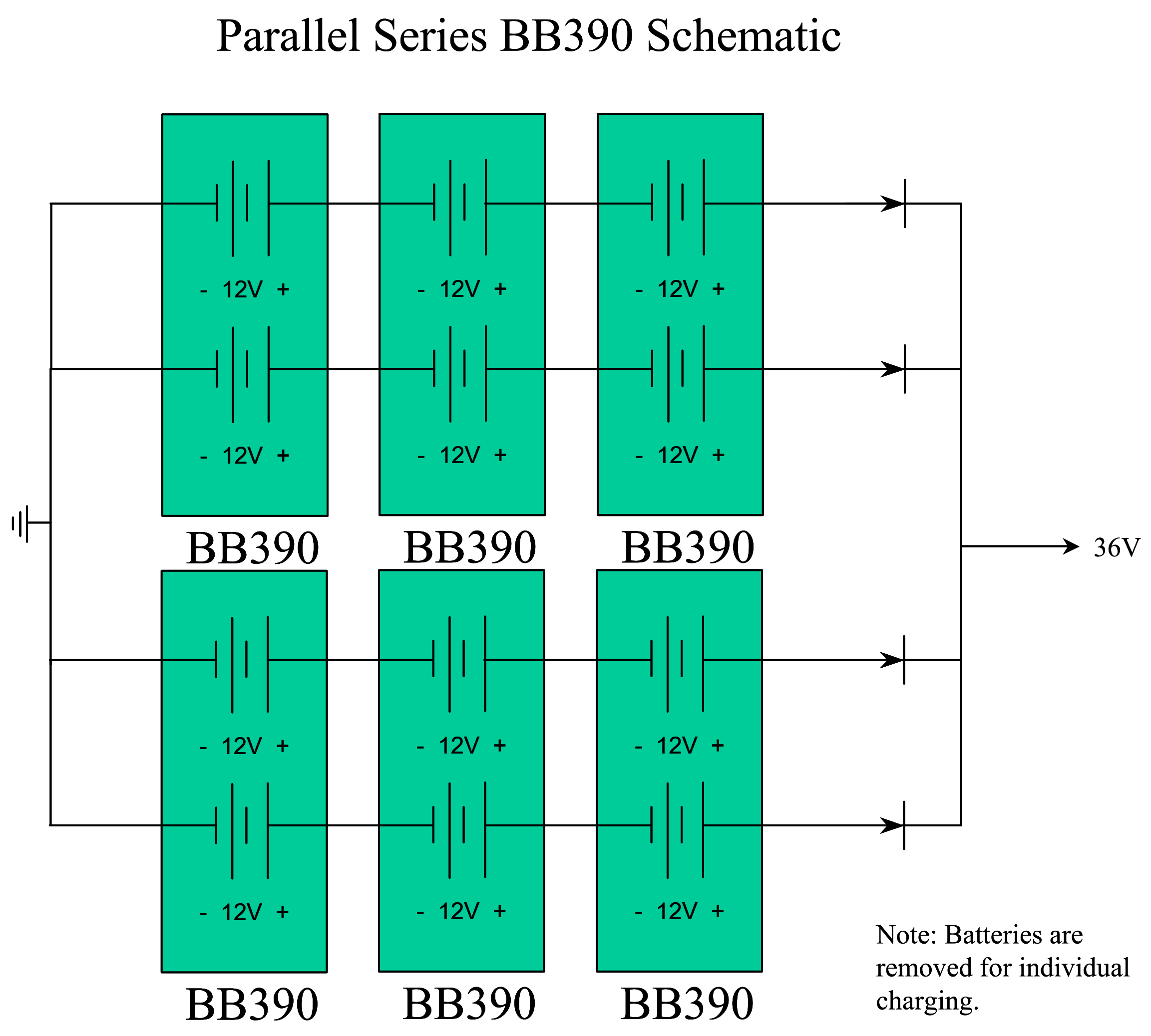

Parallel Series BB390 Schematic
- 12V +
BB390
36V
Note: Batteries are removed for individual charging.

Figure 21 - Wiring Diagram for the SandDragon Batteries

Figure 22 gives the configuration of the front body PC104 stacks, and Fig. 23 gives the configuration of the front body sensor stacks. The settings and configuration of all the PC104 and custom boards is given in Appendix B, SandDragon Configuration and Assignment Document, created by Willy Morse and modified by Alex Maish. Views of the front and rear body interiors with the covers removed are given in Figs. 24 and 25. The front left PC104 stack contains a 233MHz Pentium CPU board, a utility module containing interfaces for an external SVGA monitor, an IDE hard drive, and a floppy drive, and a quad serial board adding four serial ports to the two on the CPU board. All three are from Real Time Devices (RTD). The front right PC104 stack contains a CompactFlash Boot module which holds the operating system and control code, an Ethernet utility module, and a Directed Motion four-axis servo motion controller consisting of a two-card Galil 1240 set. The Ethernet module is connected to a multi-outlet Ethernet Hub mounted on top the right PC104 stack allowing simultaneous Ethernet connections to the rear Galil Servo Motor PC104 board set and to the external base station control computer during development. During development a PC104 Floppy Drive board could be connected to the RTD utility module, but once boot software was loaded onto the CompactFlash card it was removed and booting occurred from the CompactFlash boot drive. In the custom sensor stack, the front body contains a power supply board using Vicor +5V and +12V regulators, a Freewave DGR09 digital radio board mounted on a custom interface board, and a custom interface board for the Honeywell HMR 3000 Compass and the Sony camera. Magnetic interference within the vehicle body led to moving the compass to the sensor platform on the roll bar on the top of the vehicle. The HMR3000 custom sensor stack card was left since it put power on the appropriate serial line for the compass.

The rear body contains a Directed Motion two-axis servo motion controller consisting of a Galil DMC1425 board. It also contains a Southern California Microwave video transmitter and another power supply board using Vicor regulators.

The Novatel OEM3 RTK GPS unit was mounted in the rear body after the cover was removed.

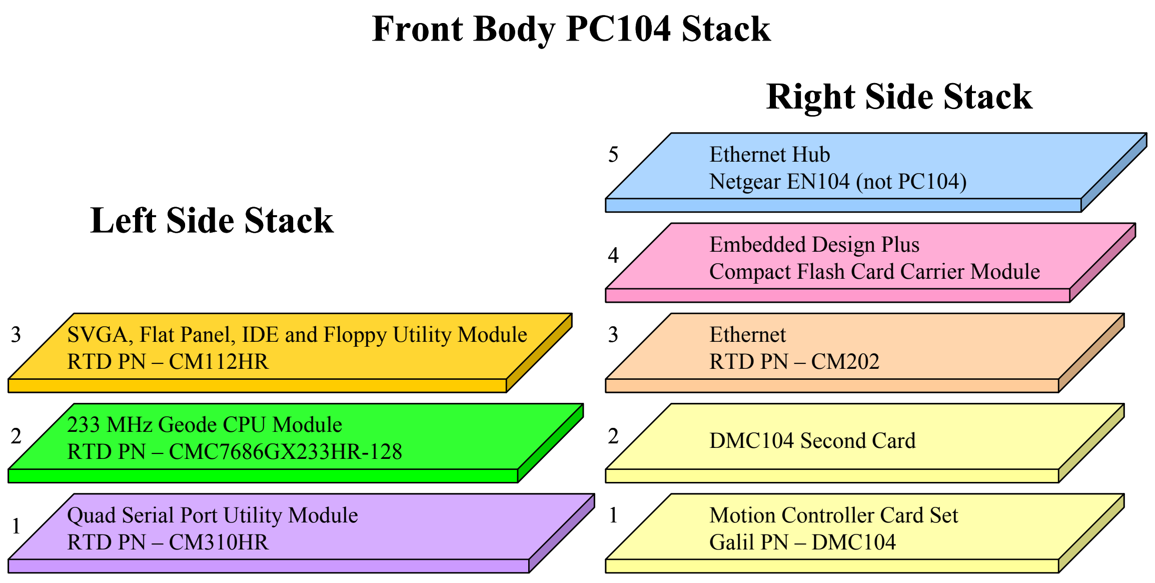

Front Body PC104 Stack
Right Side Stack
5 Ethernet Hub Netgear EN104 (not PC104)
4 Embedded Design Plus Compact Flash Card Carrier Module
3 Ethernet RTD PN – CM202
2 DMC104 Second Card
1 Motion Controller Card Set Galil PN – DMC104
Left Side Stack
3 SVGA, Flat Panel, IDE and Floppy Utility Module RTD PN – CM112HR
2 233 MHz Geode CPU Module RTD PN – CMC7686GX233HR-128
1 Quad Serial Port Utility Module RTD PN – CM310HR

Figure 22 – SandDragon Robotic Vehicle Front Body PC104 Stack Configuration

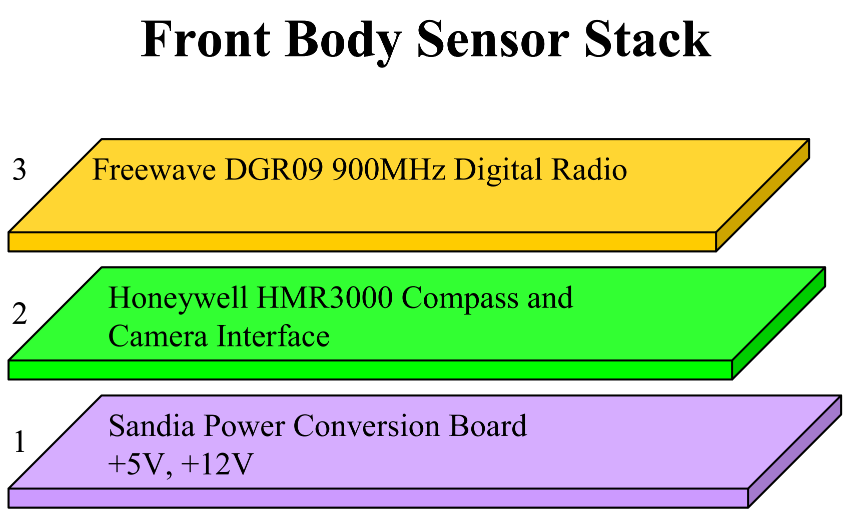

Front Body Sensor Stack
3
Freewave DGR09 900MHz Digital Radio
2
Honeywell HMR3000 Compass and Camera Interface
1
Sandia Power Conversion Board +5V, +12V

Figure 23 – SandDragon Robotic Vehicle Front Body Sensor Stack Configuration

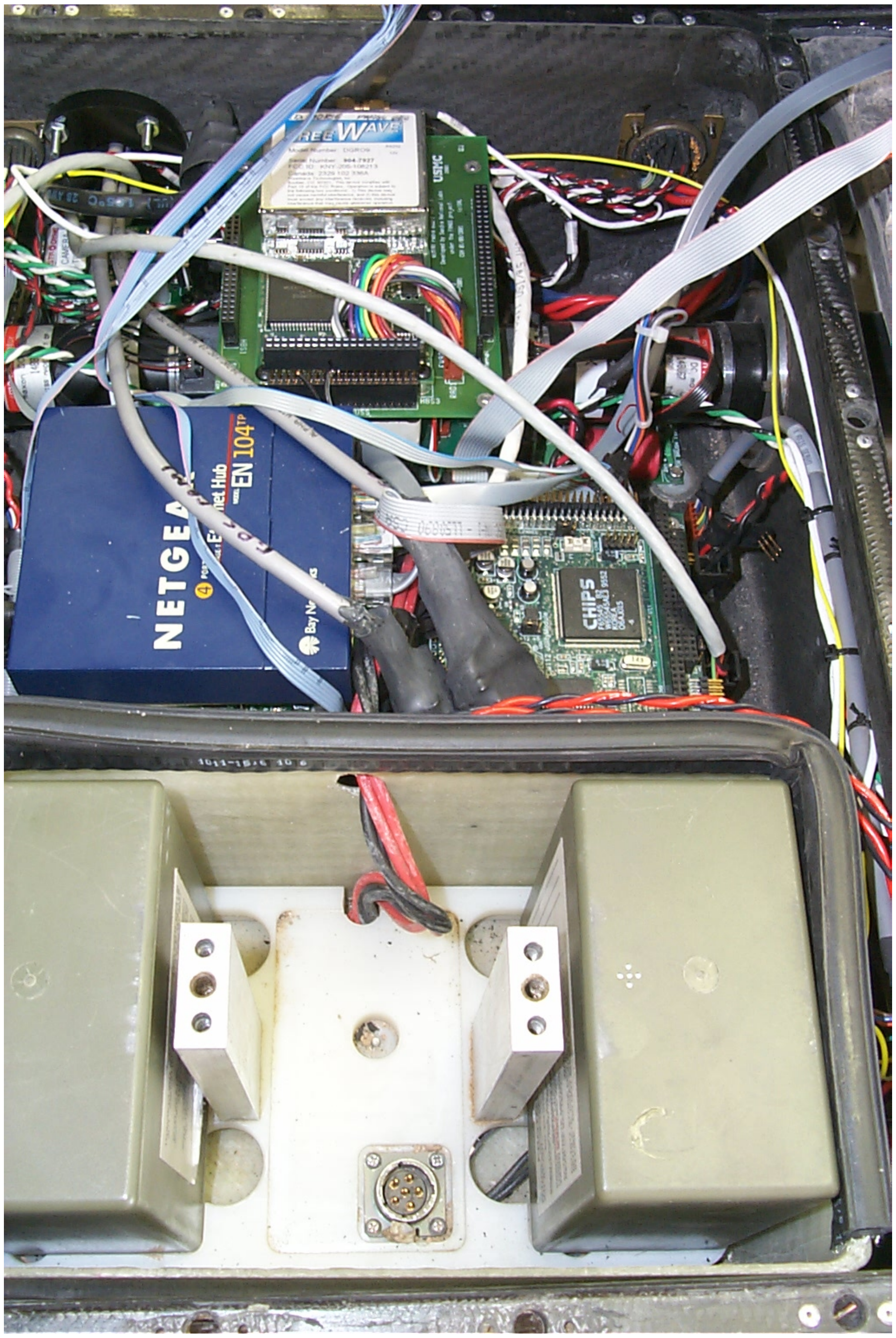


Figure 24 – SandDragon Robotic Vehicle Front Body Interior View

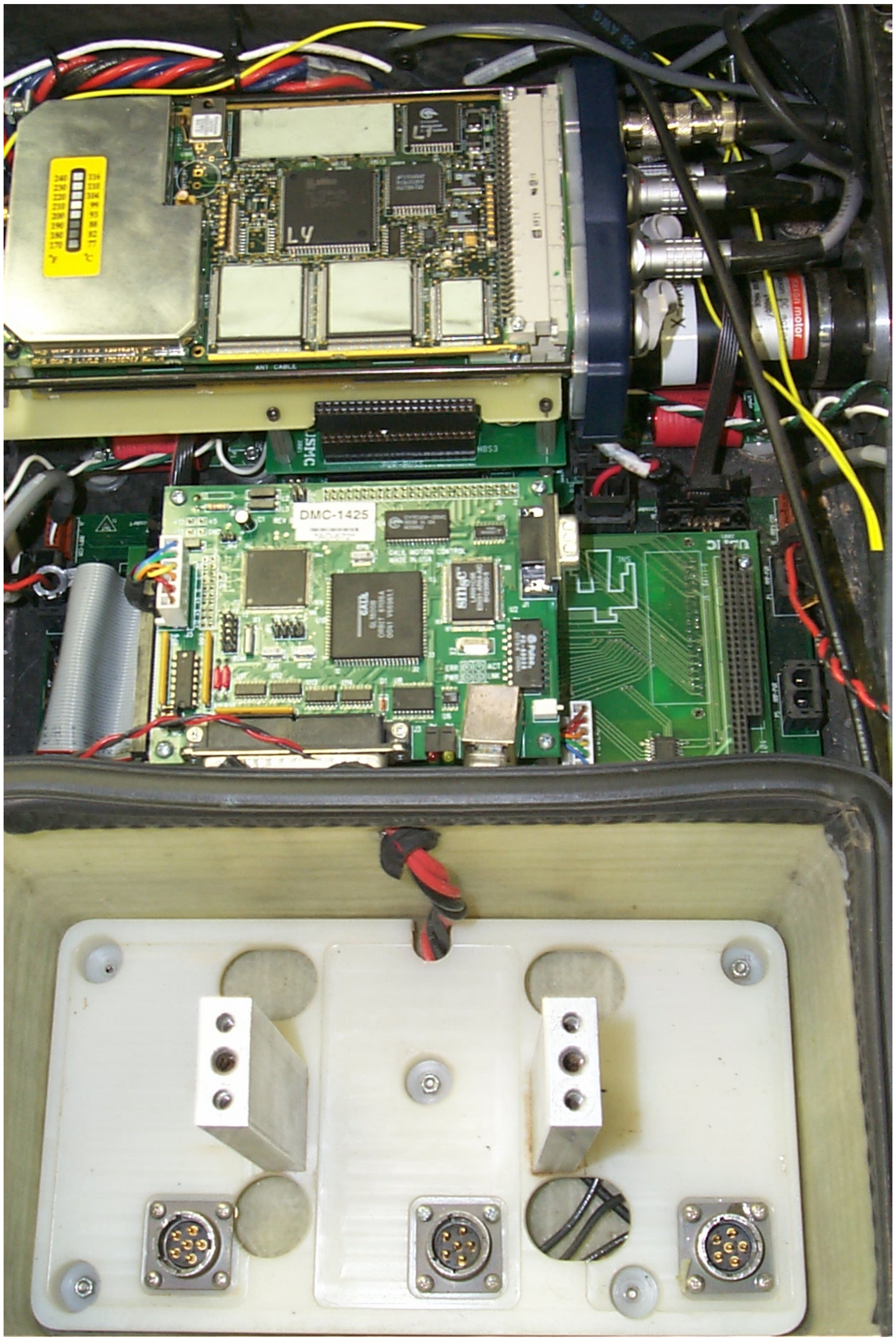


Figure 25 – SandDragon Robotic Vehicle Rear Body Interior View

The serials ports were assigned as shown in Table 9.

Table 9 – Serial Port Assignments on SandDragon Vehicle

Port	Assignment
1	Data Radio Communication
2	GPS Differential Corrections
3	Camera Control
4	Compass
5	GPS Log/Command Data
6	Payload DSP Interface

A structure was fabricated on the rear body of the SandDragon vehicle to mount the driving camera, the GPS antenna, and the compass mechanism. It was assembled using PVC tubing, as shown in Fig. 26. The Novatel GPS-512 L1/L2 aircraft antenna was moved from this structure to the on top of the sensor pod deployment/retrieval mechanism to place it closer to the deployment location.

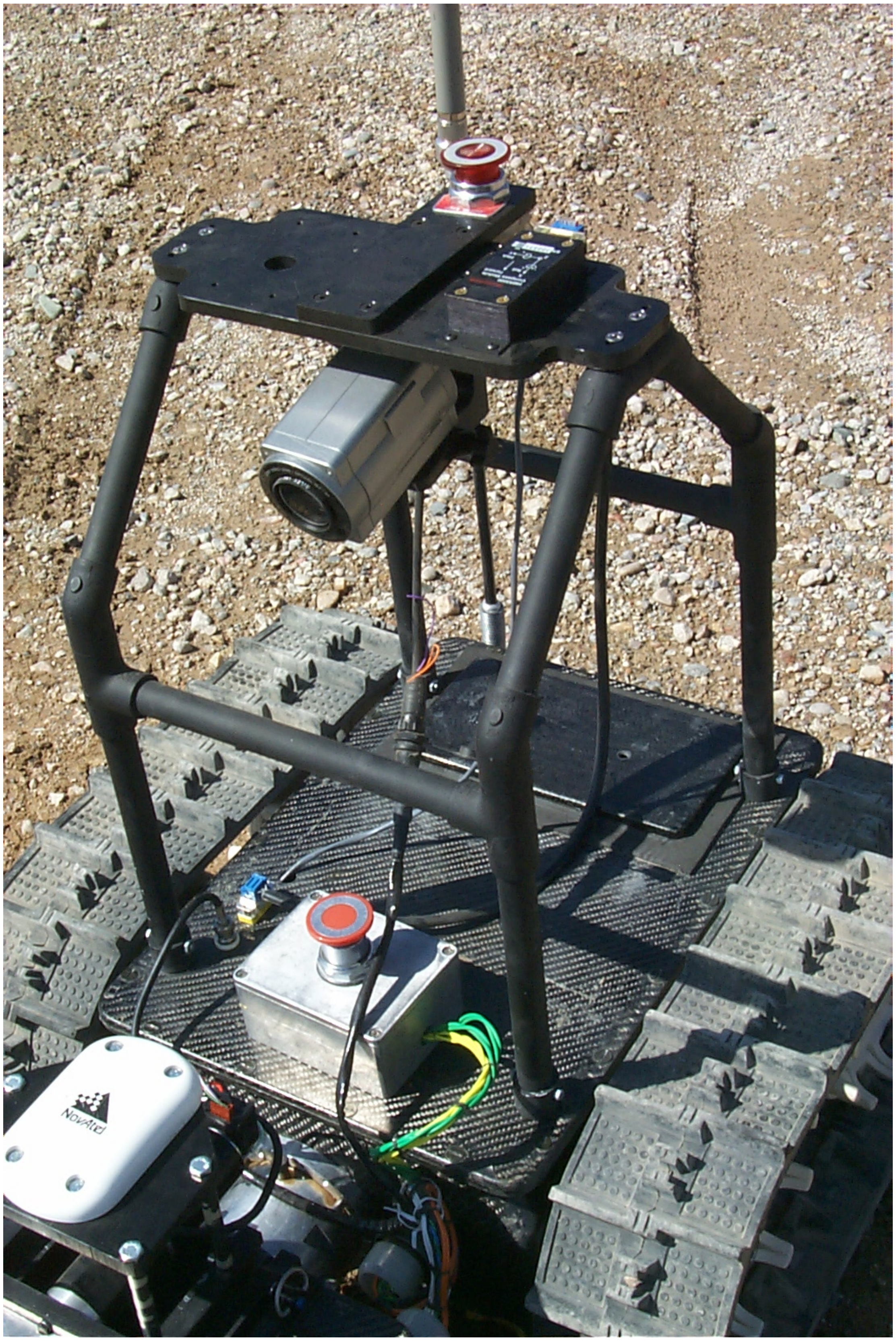


Figure 26 – SandDragon Robotic Vehicle Rear Body Sensor Support Structure

Sensor Pod Deployment/Retrieval Mechanism

The sensor pod deployment/retrieval mechanism underwent some evolution during the course of this project. Initially it was expected that we would need to drive or screw the sensor pods into the ground to achieve good vibrational coupling between the sensor and the ground. The sensor pod was designed as a round puck that could be held between three wheels and spun by driving one of the wheels so that it rotated the puck about a protruding threaded spike. The trial deployment mechanism, mounted on an arm, proved this concept would work. The three wheels that held the sensor puck were roller blade wheels. Lips on the top and bottom of the sensor puck kept the puck from slipping out from between the three wheels. When testing determined that screwing the puck into the ground did not confer a significant advantage in signal coupling, the rotating mechanism was dropped, but the puck design was retained in case it was necessary to revisit this approach.

Once the approach was selected, a new deployment mechanism was designed and built. This mechanism did not include an arm but instead raised the sensor puck around the body of the vehicle and tilted it horizontally so that it could be placed in a horizontal stack with additional pucks. The initial design did not include a multi-sensor stacker, but the concept was designed to be easily adaptable to one. A lifting mechanism was fabricated that worked like a garage door, with a pair of curved tracks guiding rods that held panels. The lower panel supported the gripper mechanism, which included a pair of arms and a motor. The motor, acting through a gear, would open and close the gripper arms. The original in-line skate wheels were dropped in favor of a configuration of spring-loaded curved arms with smaller rollers that could adapt to up to 3 inches of misalignment. The final configuration, shown in Fig. 27, was successful in grabbing a sensor puck with several inches of misalignment.

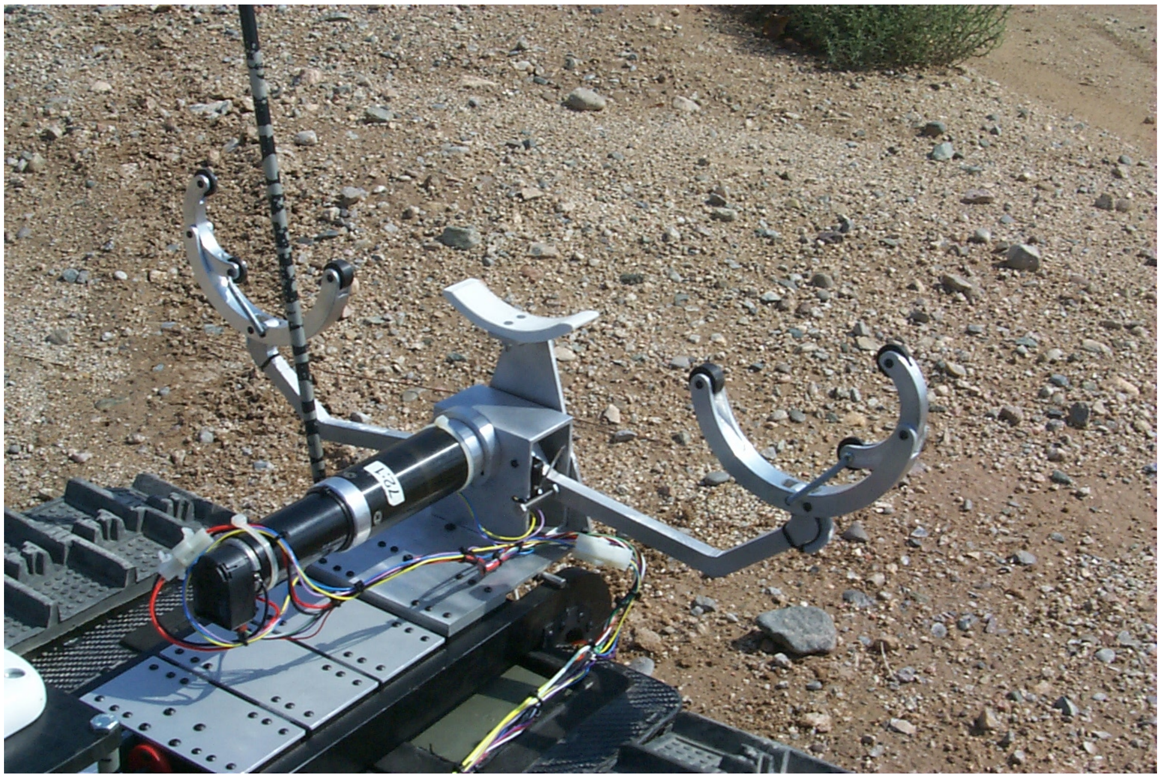


Figure 27 – Sensor Pod Lift and Grip Mechanism

Although both the lift and grip motors were adapted to use encoder feedback, we needed to add limit switches so we could initialize the encoders to zero at one end of travel. The lift mechanism used a non-backdriving ball screw that would hold the sensor pod in place if the motor was deactivated. To conserve power, the lift mechanism motor software was coded to turn off the lift servo control when the encoder stopped changing. This didn't work with the lower-geared grip mechanism, however, and when the motor was deactivated the grip mechanism relaxed, dropping the sensor puck. Thus we had to program the grip motor to stay on whenever the sensor puck was in the gripper. This could be avoided if the jaws were redesigned using a non-backdriving gear element. Although we were successful in adding power limit switches to the lift motor to cut power to the lift motor whenever it hit the limit, this did not work with the grip motor. Instead, when a grip limit was hit, it caused the gripper to drive at full speed in the opposite direction due probably to some interaction involving the motion servo system. So we hooked the gripper limit switches directly into the Galil motion servo limit inputs, which stopped motion in that direction.

Operator Control Unit Hardware

The Operator Control Unit (OCU) consists of a PC computer, a radio communication unit with a small video monitor housed in a portable box, and a gamepad joystick input device used during manual teleoperation. (Fig. 28) Additional hardware used in development included an additional monitor and keyboard to connect directly to the vehicle, and a large monitor to view the vehicle video camera image. An Ethernet connection between the vehicle and the OCU base station was also used to facilitate debugging and code transfer. A standard desktop computer with keyboard and monitor was used in this project for the base station OCU computer, but a laptop or ruggedized PC could just as easily suffice. The radio communication unit contains the digital and video radio communication boards and antennas for the base station. It can be run off an internal 12V battery or it can be plugged into the wall. It connects to the OCU computer through an RS232 serial link, and to the video display with a video connector. The DI gamepad joystick unit provides separate joysticks with deadman switches for driving the vehicle and driving the lift/grip sensor deployment/retrieval unit. (Fig. 29)

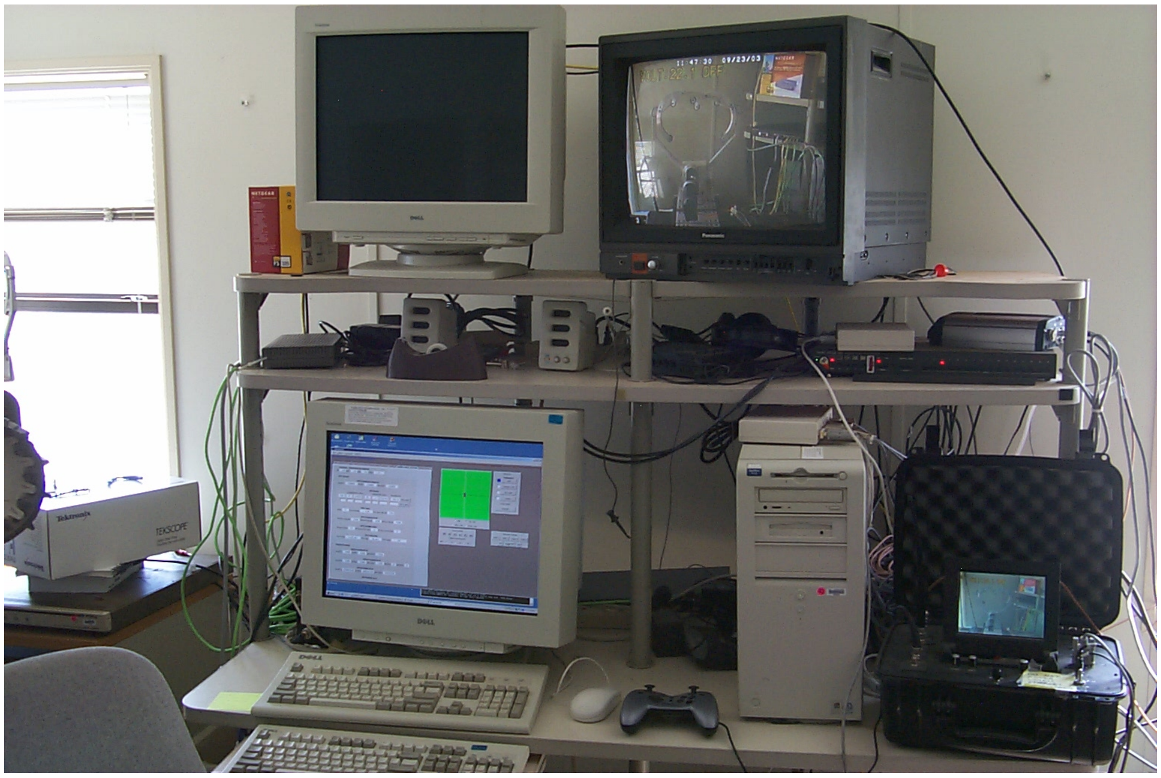


Figure 28 – Operator Control Unit and Development Hardware

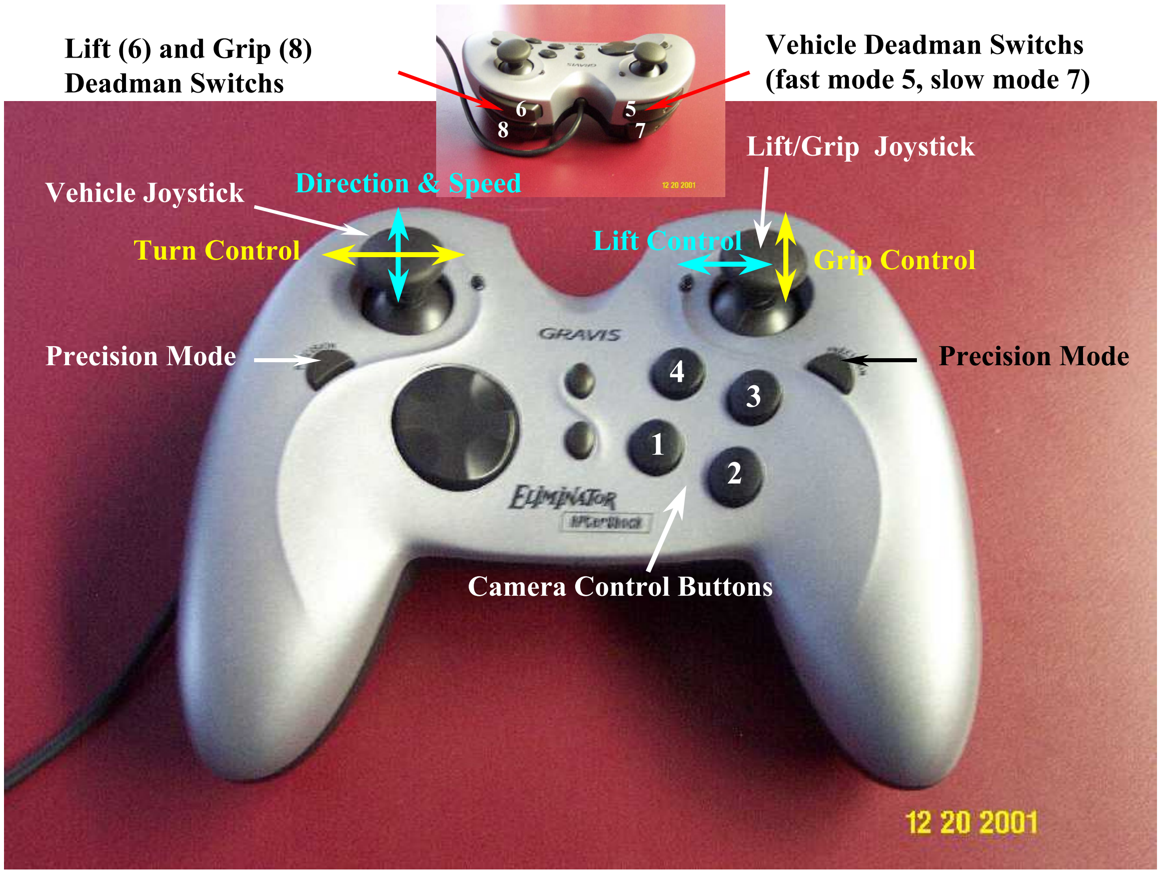


Figure 29 – DI Gamepad Unit

Acoustic Sensor Mechanism

During the first year of the program an acoustic sensor mechanism was developed that supported an array of seven sensors, one in the center and six in a circular pattern about it. The target half-wavelength for a 150Hz acoustic signal is only 1.1m as compared to the 5m spacing for the seismic signals due to the slower signal velocity in air. Thus the full array could be mounted on the vehicle rather than having to place sensors on the ground. The acoustic sensor spacing could be adjusted between 14 and 39 inches enabling it to be tuned to various frequencies. The array is shown in Fig. 6. A turntable is used to rotate the array to different directions, and power automotive radio antennas are used to extend and retract the sensor arms. The sensors are WP-3502 headset microphone units for outdoor use from Emkay Innovative Products and are covered with an acoustic foam ball to reduce wind noise. They are designed to survive immersion in water.

Vehicle System Software Design

The vehicle system software was written using Sandia's SMART software Version 0.9. Although the code is written in C, not C++, it implements a number of the object oriented concepts such as encapsulation. Make files are used to compile the code using the Gnu gcc compiler. (Version?)

The SMART software for the SandDragon vehicle for this LDRD consists of three separate targets defined in the sanddragon.lab file. (Fig. 30) These targets are the base unit's real-time operating system (RTOS) running on a Windows OS, the vehicle RTOS running on a QNX Neutrino real-time OS, and the base unit graphical user interface running under Tcl/Tk on a Windows OS.

```
#****************************************
# sanddragon.lab
# This lab is located all around the RVR
# and consists of the Neutrino based SandDragon system
# two mounted computers, and a graphical host.
#  OS options: irix, irix62, linux, lynxos,
#     neutrino, qnx, solaris, vxworks,
#     winnt, win98
#  ARCH options: alpha, 68k, x86, mips, ppc, sparc
#  defaults: irix->mips, irix62->mips, linux->x86,
#      lynxos->ppc, neutrino->x86, qnx->x86,
#      solaris->sparc, vxworks->68k, winnt->x86,
#      win98->x86.
#****************************************
SMART_LAB sanddragon

TARGET ocu_rtos RTOS
  OS winnt
  ARCH x86
  PORT 2060
  CPU_NAME "teleopbase"
TARGET_END

TARGET vehicle RTOS
  OS neutrino
  ARCH x86
  PORT 2060
  CPU_NAME "sanddragon143"
TARGET_END

TARGET ocu_gui GUI_HOST
  OS winnt
  ARCH x86
  CPU_NAME "teleopbase"
TARGET_END

SMART_LAB_DONE
```

Figure 30 - Sanddragon Lab File used by the SMART Editor

Application-specific software configurations are generated graphically in the SMART Editor, and consist of grids of SMART modules that implement specific functionalities on each platform. Behaviors define what grids (groups of SMART modules) are operable on each RTOS platform and enable coordinated activity of the different platforms. There are four behaviors defined for the SandDragon system. The active grids on each target for each behavior are shown in Table 10. The GUI grid runs all the time independent of which behavior is active on the RTOS platforms, so the table only shows the RTOS grids that are active for each behavior. The *teleop* behavior was named *joint* due to a code requirement that the first behavior be named *joint*. We didn't track this down and change it. The *camera* grid can run simultaneously with the *teleop* or *autonav_on_veh* behaviors. The final version of this configuration information developed under this project is saved under the workcell named alexnav5v.wk2.

Table 10 – Grids Active in Each Behavior on each Target

	Behaviors			
Target	Joint (Teleop)	Autonav_veh	Gps_only	Camera
Base RTOS	Teleop	Autonav_vehicle	Gps_only	ptu
Vehicle RTOS	Nav_on_base	Autonav_on_veh	Nav_on_base	

Each module has a function and customizing parameters, called filter constants in SMART. These parameters are stored in a configuration file with a configuration name. SMART modules communicate either through registers, which are named common storage locations, or through connect structures graphically depicted as line connections on each side of the module icons in the SMART Editor. SMART implements the graphical connection behind the scenes. The functions, filter constant settings, and connection values are described for each behavior below.

Tele-operation Behavior

The tele-operation behavior consists of cooperative grids on the base unit and the vehicle. The base unit runs the *teleop* grid shown in Fig. 31, and the vehicle runs the *nav_on_base* grid shown in Fig. 32.

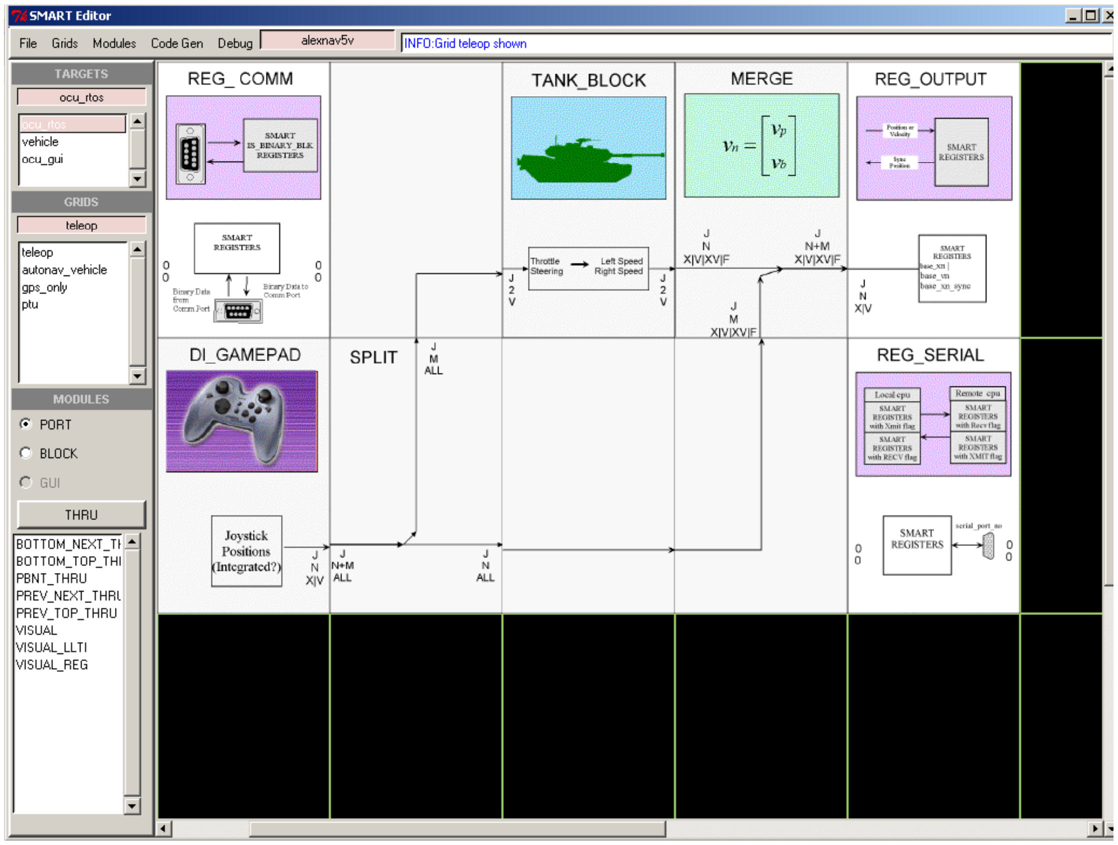


Figure 31 – Base Platform’s *teleop* SMART grid

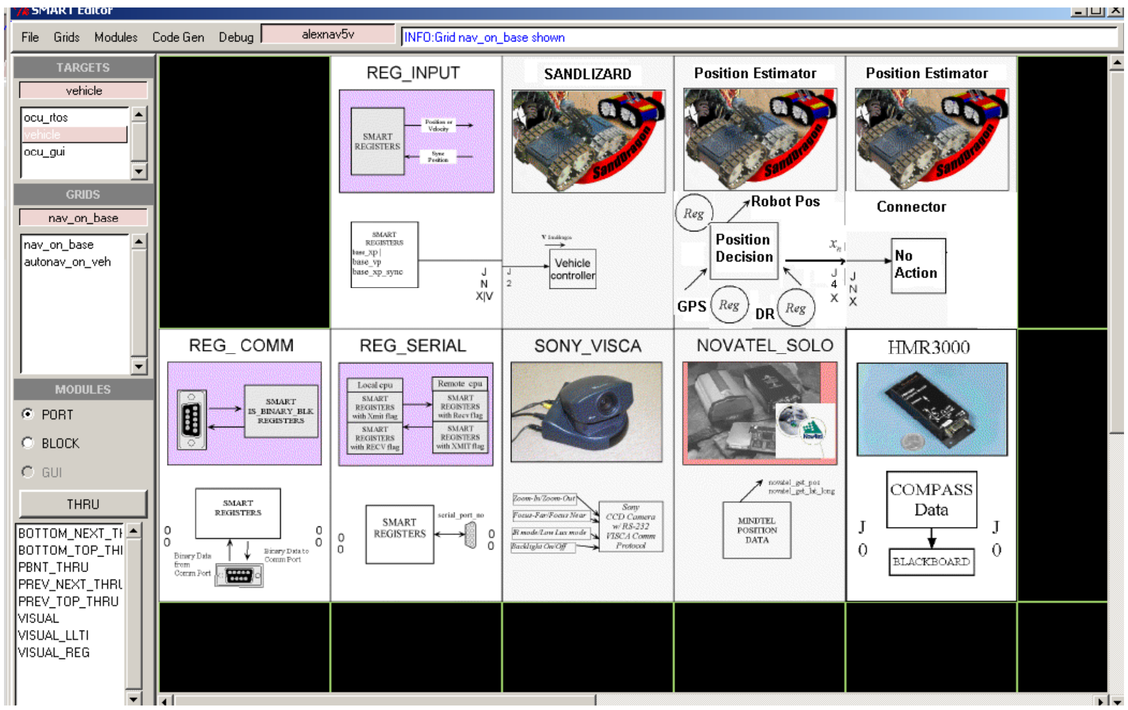


Figure 32 – Vehicle Platform’s *nav_on_base* SMART grid

Table 11 shows the named variable storage registers used by the *joint* behavior. Many are flagged to be passed between the vehicle and the base platform over the radio serial link by the Reg_Serial SMART modules on each platform's grid.

Table 11 – Joint (Teleop) Behavior Registers

Register Name	Type	Contents	Owner Module / Target	Used By Module / Target
gpsdiff1	bin_blk	GPS differential binary log	Reg_comm / Base	Reg_comm / Veh
abase	twist	Vehicle velocity in x and y and lift and grip directions.	Reg_output	Reg_input /Veh
DR_travel	twist	Travel distance, left, right, lift, grip.	Sandlizard /Veh	Pos_estim / Veh, Sandlizard_status / GUI
traj_io_stat e	integer	Not implemented. Used to feed back deploy state to traj_io module.	Not used in this behavior	Sandlizard /Veh
traj_io_cmd	integer	Bit commands used to signal raise, lower, open and close sensor deploy mechanism.	Not used in this behavior	Sandlizard / Veh
gps_pos	vector	Distance from home in m, x (E is pos), y (N is pos), z using GPS information.	Novatel_solo / Veh	Pos_estim / Veh Sandlizard_status / GUI
gps_home	vector	HOME position in decimal degrees and m	Novatel_solo / Veh	Sandlizard_status / GUI
gps_base	vector	Base GPS fixed position in decimal degr and m	Novatel_solo / Veh	Sandlizard_status / GUI
gps_pos_de c	vector	Vehicle position in decimal degrees and m	Novatel_solo / Veh	Sandlizard_status / GUI
gps_sats	vector	Number of satellites	Novatel_solo / Veh	Sandlizard_status / GUI
gps_std_de v	vector	Std deviation of rtk msmt in m for x, y, and z	Novatel_solo / Veh	Pos_estim / Veh, Sandlizard_status / GUI
gps_rdg_ti me	vector	Satellite time in week and sec of week.	Novatel_solo / Veh	Pos_estim / Veh, Sandlizard_status / GUI
gps_rtk_sta t	vector	Rtk status code	Novatel_solo / Veh	Sandlizard_status / GUI
gps_base_l og	string	Base GPS unit log text	Not used	Novatel_solo / Veh
hmr_rpy	vector	Roll, pitch, and yaw in degrees	HMR3000 / Veh	Pos_estim / Veh Sandlizard_status / GUI
compass_st atus	integer	Status byte	HMR3000 / Veh	Sandlizard_status / GUI
robot_pos_ x	twist	Distance from home in m, x (E is pos), y (N is pos), z using better of GPS or DR information. Also lift and grip in % of travel from up and open.	Pos_Estim / Veh	Sandlizard_status / GUI
DR_pos	vector	Distance from home in m, x (E is pos), y (N is pos), z using dead reckoning information.	Pos_Estim / Veh	Sandlizard_status / GUI

Base Platform Teleop Grid

The base platform communicates with the vehicle platform using a pair of Reg_serial SMART modules. The base unit is set to a baud rate of 19200 and uses port 3 on the base station to talk to the radio. (These parameters are set in the configuration files for each module, which are listed in Appendix C.) The Reg_serial SMART module was substantially modified under this program to improve communication with the limited bandwidth available and the numerous operating values to be monitored at the base station. Originally the Reg_serial module only made one pass through the registers each minimum-send-time period (which is set to 0.1 seconds in the configuration file) to fill the buffer. If there was more data to be sent, it waited until the next time period. This was changed so that the software looped through the registers repeatedly, refilling and sending the buffer each time period until all awaiting registers were sent. When each register is saved, the time_at_last_update timestamp is updated in the reg.c core file, but we had to add a 'reg_sent_timestamp' to the Reg_serial data structure for each register to enable the change. These timestamps are associated with the target being sent to, so they were made part of the Reg_serial module instead of the reg.c core file that is associated with the registers in order to accommodate multiple Reg_serial modules sending to different targets. After this change there was still a problem because some SMART modules updated registers repeatedly even if the data hadn't changed, and this flagged the register to be re-sent even if the data hadn't changed. A significant reduction in register data traffic was finally achieved when another timestamp, named time_at_last_change, and an associated reg_get_change_timestamp routine were added to reg.c and used to enable sending the register data only if it had actually changed since the last send. A full send of all the register values is sent every minute (as set in the full_send_time parameter in the configuration file) even if the data hasn't changed.

Next, the Reg_Comm SMART module receives binary differential correction data from the base station Novatel GPS unit. The communication rate is 19200 baud over serial port 5. Data is stored in the gpsdiff1 binary block register and is communicated via the Reg_serial modules to the same register name on the vehicle where the correction data is passed on to the vehicle's Novatel GPS unit with another Reg_Comm module. This data enables the vehicle's GPS unit to maintain centimeter accuracy relative to the fixed base station position. The base GPS unit is initialized by sending it the commands given in file base.gps shown in Fig. 33. This is done using the novatel_test.exe executable with the 'base' option before running the SandDragon executable code. Once the settings are sent to the Novatel GPS base unit and saved to non-volatile memory using the SAVECONFIG command, the unit will start up with these settings. The commands set it to send its position log over its COM1 line (base unit port 4) to the base station every 5 seconds. This log, which gives the latitude and longitude of the fixed base unit position, isn't monitored by the SMART software but can be read by Novatel's GPSolution software. The base.gps file also commands the Novatel unit to send two differential correction logs over its COM2, connected to the Base Station's serial port 5. These are put in gpsdiff1 register by the Reg_Comm module. The first is the RTCAOBS log sent every two seconds, and the second is the RTCAREF log sent every 10 seconds, offset by a second. Offsets can only be on the whole second, so to avoid having both logs sent at once, which would overfill the gpsdiff1 binary block register size, we had to set the RTCAOBS log to every other second rather than every second. This is sufficient to achieve the desired accuracy according to Novatel.

```
#**** This is the base station setup file for the RVR base station
#****  The Fixed position needs to be adjusted
# This command should be called from within the GPSolution environment
# followed by a SAVECONFIG
# COM1,19200,N,8,1,N,OFF,ON
UNLOGALL
UNFIX

#**** THE LOG MESSAGES Depend on which windows are open! ****
LOG,COM1,POSA,ONTIME,5.00

#**** The position below defines the base position ****
#To get a FIXED position run the $POSAVE,com1 command within GPSolution
#Let it run for 24 hours, then come back and record the data off of the Position window.
#COM is redundant since it won't accept it unless it is already talking at correct rate
#COM2,19200,N,8,1,N,OFF,ON
#The value below was averaged over 2 days on 4/15/02 using the antenna on top of RVR control room*
FIX,POSITION,35.04113166,-106.52210904,1672.318
#The RTCAREF needs to be done every 10 seconds. Offset from RTCAOBS to not overflow the
#buffer in SMART, which can be filled by one RTCAOBS reading. Offset may have to be on full sec
#boundaries.
LOG,COM2,RTCAREF,ONTIME,10.0,1.
#The RTCAOBS needs to be sent every second or two.
LOG,COM2,RTCAOBS,ONTIME,2.0
SAVECONFIG
```

Figure 33 – Novatel GPS Base Unit Command String

Next, the DI_Gamepad SMART software module interfaces with the hardware gamepad unit (Fig. 29) with its two joysticks and multiple switches used for tele-operation control. Its output is a connection with 4 degrees-of-freedom (dof) of velocity, for the forward and lateral motion of the hitch point, and the lift and grip motion of the sensor deployment and retrieval mechanism. In the adjacent 'split' SMART module, the first two dofs, forward and lateral motion, are split off and sent to the Tank Block, which translates the motion into left and right track velocity commands. These are recombined in the SMART merge module and put into register *abase* in the Reg_output module to be sent to the vehicle platform.

Vehicle Platform Teleop Grid

The vehicle platform has the Reg_serial module that is complementary to the one on the base platform. Its configuration filter constants are set up (Appendix C) to use port 1 to talk to the radio at a 19200 baud rate. It also has a Reg_comm module that is set up to relay the gpsdiff1 binary block register values to the Novatel GPS unit on port 2 at a rate of 19200 baud. Figure 34 lists the setup command string used with the onboard Novatel GPS unit. The Reg_input module extracts the four velocity values from the *abase* register and passes it to the Sandlizard module through its right side connection.

```
#**** This is the Vehicle Setup file for the Novatel GPS
#**** It is read from SMART_RTOS_DIR/novatel/<filename>
#**** where SMART_RTOS_DIR is defined in rtos/smart_include_dirs
#**** and is currently set to "/demos" on the vehicle.
#**** This is read from the vehicle even if telneting from base.

# The COM1 and COM2 settings aren't really needed.
# If you are talking, it is already correct
#COM1,19200,N,8,1,N,OFF,ON
#COM2,19200,N,8,1,N,OFF,ON
UNLOGALL
UNFIX

# POSA and GPGGA not normally used
#LOG,COM1,POSA,ONTIME,1.00
#LOG,COM1,GPGGA,ONTIME,1.00
# Accept incoming diff corrections on COM2
ACCEPT,COM2,RTCA
# Send out position log on COM1 every second
LOG,COM1,PRTKA,ONTIME,1.00

# Send out status log every sec offset 0.5 sec for debug
#LOG,COM1,RCSA,ONTIME,1, 0.5

# This puts values into nonvolatile ROM
#SAVECONFIG
```

Figure 34– Remote.gps command file used to setup the onboard Novatel GPS unit

The Sandlizard module interfaces with the Galil motion controllers to send it commands for the four track motors and two sensor deployment/retrieval motors to respond to the velocity commands passed in by the Reg_input module. It also retrieves the encoder positions for the four front-body motors, left and right, lift and grip, and puts them into register *DR_travel* for use by the Position_estimator module and to be sent back to the OCU GUI.

Three other SMART modules in this grid interface with vehicle sensors and components. The Sony_Visca module provides control and status communication with the Sony camera. The control pendant DI_Gamepad button commands are not put into a register, but rather are sent as commands over the Reg_serial link to the Sony_Visca module, which translates these into camera commands that it sends over comm port 3. The Novatel_solo_block module retrieves position and status information from the Novatel GPS unit and puts the data in registers for use by the Pos_Estim position estimation module and the OCU GUI for display purposes. The HOME position is the GPS position from which distances are measured. It is set initially to the default in the configuration file, which is the location of the Robotic Vehicle Range control room GPS antenna. The HMR3000 module interfaces with the Honeywell HMR3000 digital compass and puts the tilt, roll, and yaw in degrees into the *hmr_rpy* register as a vector. It communicates over serial port 4. It also puts the compass status information into register *compass_status*.

The *pos_estim* or position estimation SMART module arbitrates between the GPS position and the dead reckoning (DR) position. The algorithm it uses in its decision is a rudimentary one at present. It checks the GPS standard deviation error value and if the x and y components are within 10 cm, it uses the GPS value and updates the DR value to it. Otherwise it uses the DR value. The *pos_estim* block module provides the robot_pos values in its 'next' or right side connection which is used in the autonav_on_veh grid, so instead of having two versions, this connection is sinked to a do-nothing module called *pos_estim_conn_block*.

Autonavigation Behavior

The autonavigation behavior consists of cooperative grids on the base unit and the vehicle. The base unit runs the *autonav_vehicle* grid shown in Fig. 35, and the vehicle runs the *autonav_on_veh* grid shown in Fig. 36.

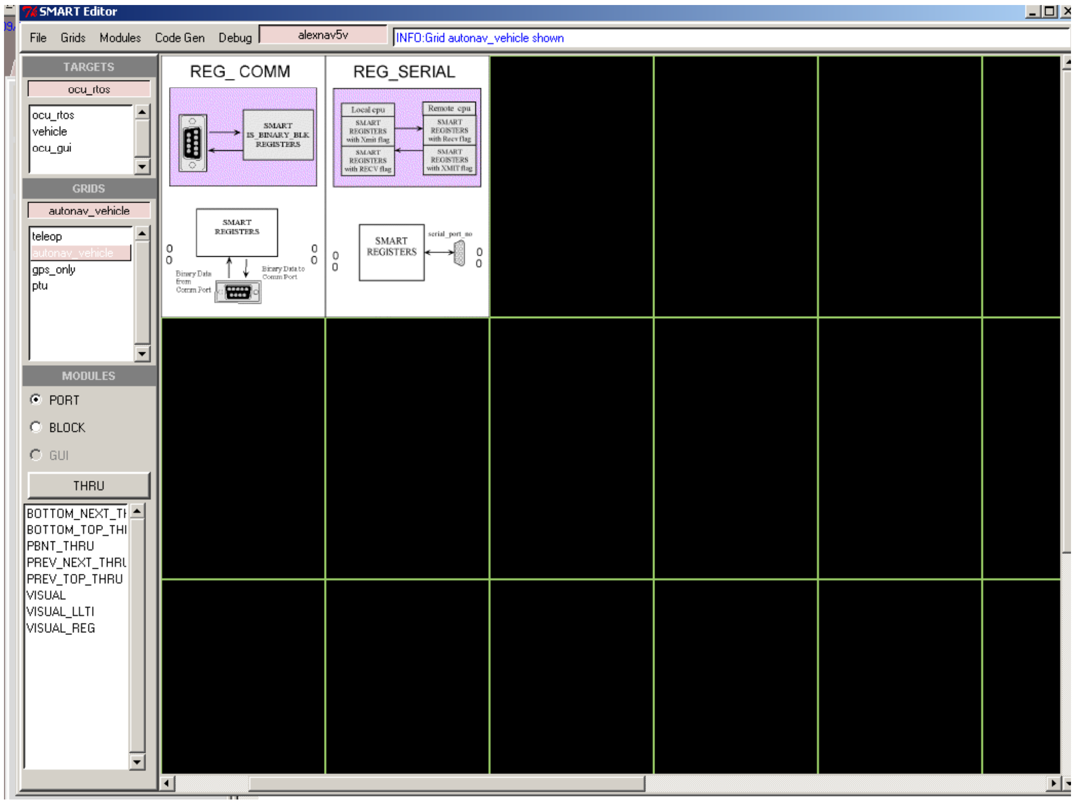


Figure 35 - Autonav_vehicle grid used on base platform during autonavigation behavior

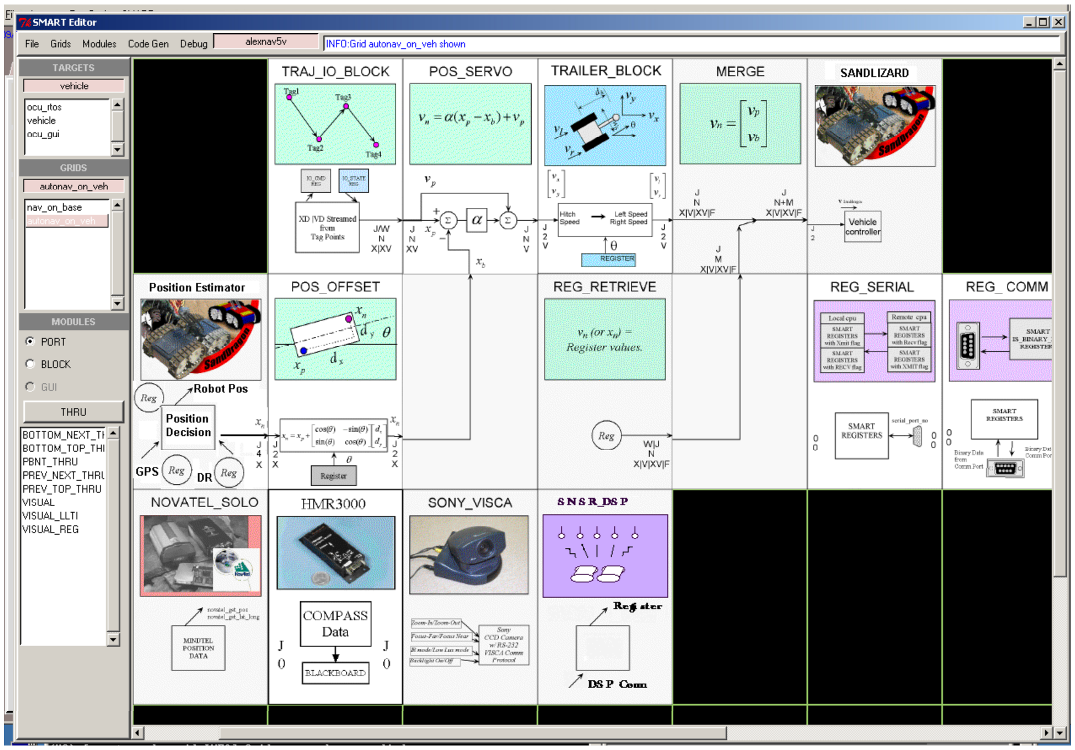


Figure 36 - Autonav_on_veh grid used on vehicle platform during autonavigation behavior

Table 12 shows the named variable storage registers used by the *autonav* behavior. Many are flagged to be passed between the vehicle and the base platform over the radio serial link by the Reg_Serial SMART modules on each platform's grid.

Table 12 – Autonavigation Behavior Registers

Register Name	Type	Contents	Owner Module / Target	Used By Module / Target
gpsdiff1	bin_blk	GPS differential binary log	Reg_comm / Base	Reg_comm / Veh
DR_travel	twist	Travel distance, left, right, lift, grip.	Sandlizard /Veh	Pos_estim / Veh, OCU GUI
traj_io_state	integer	Not implemented. Used to feed back deploy state to traj_io module.	Traj_io_block / Veh	Sandlizard /Veh
traj_io_cmd	integer	Bit commands used to signal raise, lower, open and close sensor deploy mechanism.	Traj_io_block / Veh	Sandlizard / Veh
traj_tag_info	string	Trajectory values: %done, traj_state, tag_name, and io_cmd	Traj_io_block / Veh	Traj_tag / GUI
gps_pos	vector	Distance from home in m, x (E is pos), y (N is pos), z using GPS information.	Novatel_solo / Veh	Pos_estim / Veh Sandlizard_status / GUI
gps_home	vector	HOME position in decimal degrees and m	Novatel_solo / Veh	Sandlizard_status / GUI
gps_base	vector	Base GPS fixed position in decimal degr and m	Novatel_solo / Veh	Sandlizard_status / GUI

gps_pos_dec	vector	Vehicle position in decimal degrees and m	Novatel_solo / Veh	Sandlizard_status / GUI
gps_sats	vector	Number of satellites	Novatel_solo / Veh	Sandlizard_status / GUI
gps_std_dev	vector	Std deviation of rtk msmt in m for x, y, and z	Novatel_solo / Veh	Pos_estim / Veh, Sandlizard_status / GUI
gps_rdg_time	vector	Satellite time in week and sec of week.	Novatel_solo / Veh	Pos_estim / Veh, Sandlizard_status / GUI
gps_rtk_stat	vector	Rtk status code	Novatel_solo / Veh	Sandlizard_status / GUI
hmr_rpy	vector	Roll, pitch, and yaw in degrees	HMR3000 / Veh	Pos_estim / Veh Pos_Offset / Veh Trailer / Veh Sandlizard_status / GUI
compass_status	Integer	Status byte	HMR3000 / Veh	Sandlizard_status / GUI
robot_pos_x	twist	Distance from home in m, x (E is pos), y (N is pos), z using better of GPS or DR information. Also lift and grip in % of travel from up and open.	Pos_Estim / Veh	Sandlizard_status / GUI
DR_pos	vector	Distance from home in m, x (E is pos), y (N is pos), z using dead reckoning information.	Pos_Estim / Veh	Sandlizard_status / GUI
gps_diff	bin_blk	GPS differential correction information passed from deployed base unit through DSP	Snsr_dsp / Veh	Unused. Alternate path for gpsdiff1
gps_base_log	string	Deployed GPS unit log data, primarily its fixed position passed through DSP	Snsr_dsp / Veh	Snsr_status / GUI Novatel_solo / Veh
dsp_msg	string	Message text from DSP to operator	Snsr_dsp / Veh	Snsr_status / GUI

Base Platform Autonavigation Grid

As with the teleoperation grid, the base platform communicates with the vehicle platform using a pair of Reg_serial SMART modules. The configuration settings are the same. The base unit is set to a baud rate of 19200 and uses Port 3 on the base station to talk to the radio. The only other module in the base platform grid is the Reg_comm module used to communicate with the base unit Novatel GPS unit. The settings for this module are also identical to those used in the teleoperation grid.

Vehicle Platform Autonavigation Grid

The vehicle platform has the Reg_serial module that is complementary to the one on the base platform. Its configuration filter constants are set as in the teleop grid to a baud rate of 19200 on port 1. It also has a Reg_comm module that is set up to relay the gpsdiff1 binary block register values to the Novatel GPS unit on Port 2. The same settings used on the teleop grid are also used for the Novatel_solo_block, HMR3000_block, Sony_Visca_port, Merge, and Position Estimator block SMART modules that interface to the GPS unit, digital compass, camera, and the position estimation arbitrator modules. The position estimation module arbitrates between the GPS and dead reckoning positions and exports the best estimate of the vehicle position in meters from the HOME position in the x (east), y (north), and z (up) directions to its right or 'next' connector. This is imported into the position offset (Pos_Offset) module which uses the compass direction from the hmr_rpy register and the offset between the GPS antenna and the vehicle hitch from its configuration file to compute and export the position of the hitch in meters from HOME. The hitch position, rather than the GPS antenna position, is what we want to control, and this is fed into the Pos_Servo module.

The Traj_IO module creates and runs trajectories of waypoints. The waypoint tag files are created in the OCU GUI in the Tag_Editor module and are stored in a file on the base unit in the $SMART_HOME/rtos/save/vehicle directory. Tag records with the same base name and a number suffix (i.e. deploy1, deploy2, etc.) form the waypoints of a trajectory. They are downloaded to the vehicle in the Tag_Editor module by highlighting the trajectory to send, copying it, and pasting it to the Download window. This causes each named tag point to be sent to the Traj_IO module using the tag_add_point command. The file *vehicle_start.tags* is downloaded automatically on startup, but other files can be created and downloaded manually. The Traj_IO module stores the tag points in memory. The operator then uses the Traj_Tag module in the OCU GUI to send commands to the Traj_IO module on the vehicle to run, pause, rewind, or stop available trajectories. When the Traj_IO module receives a run command from the OCU Traj_Tag module, it first finds the tag records with the same base name, creates a trajectory by inserting necessary waypoints to implement transitions at each waypoint (accelerate, decelerate, or round off corners as indicated by the tag file record), and runs the trajectory sequence by sending out the requested position in its right or 'next' connection which is hooked to the Pos_Servo module. It puts status data into a register to be sent back to the OCU Traj_Tag module for display to the operator. This includes the percent done, the trajectory state (PLAYING, REWINDING, AT_END, etc.), the tag number being executed, and the io_command being performed. The Pos_Servo module compares the requested position with the hitch position provided by the Pos_Offset module, and generates a velocity that is proportional to the difference. It also implements a speed limit, in meters per second, which is set in its configuration file. This limit is currently set to 0.5 m/s.

Initially we considered implementing the sensor pod deploy/retrieve motion commands using the vehicle movement servo loop expanded from two degrees of freedom (2dof) to 4dof by adding the lift and grip positions. This SMART module is shown in Fig. 37, but it was not used. We decided instead to implement a smaller servo loop by sending a flag to the SandLizard module to raise, lower, open, or close the sensor gripper. This caused the control loop to be handled entirely between the SandLizard module and the Galil motion controller rather than involving the SMART module sequencing, which nominally runs at 100Hz. While this provides faster control response, it does eliminate the possibility of servoing to intermediate lift and grip positions. The binary flag is stored in the io_cmd integer as part of the waypoint tag. When the Traj_IO module runs the tag point as part of a trajectory, it stores the io_cmd value of the tag in the io_cmd register for use by the SandLizard module. The SandLizard module reads this value and runs the sandlizard_io_lift_stow (value 1), sandlizard_io_lift_place (value 2), sandlizard_io_grip_open (value 4), or sandlizard_io_grip_close (value 8) command as directed. The command sets the lift or grip motor speed command, which is sent to the Galil motion controller by the sandlizard_io.c file. For the lift motor, motion stops when the mechanism runs into the limit switch, which cuts power. For the grip motor, motion stops when the grip encoder position reaches its limit within the operating range and the sandlizard_io.c file turns off the appropriate motor.

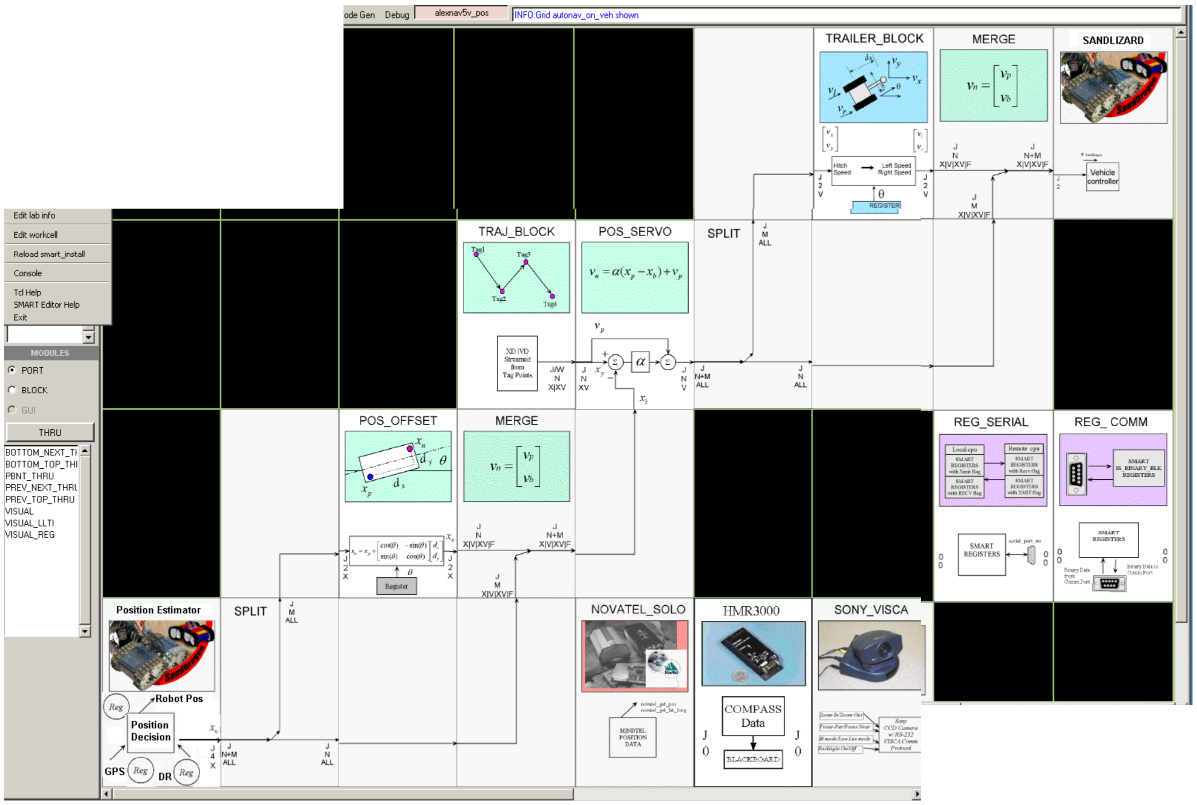


Figure 37 – Sensor Pod Control Included in SMART Servo Loop

The io_status flag in the tag file and its corresponding register were created to enable the status of a directed command to be reported back to the Traj_IO module so that it would wait for a status event before proceeding. This could be the success of a directed action, or an external event. At present this capability has not been implemented, so instead of waiting for the SandLizard module to report back that the lift or grip mechanism has successfully completed its command, the waypoint tag file is programmed with a delay to enable the command to complete. Likewise, there is no way to redirect the trajectory command if the vehicle does not successfully follow the requested path. If the vehicle gets too far behind the requested position, the SandLizard module will stop sending a velocity request to the Galil motion controller, which stops its servo loop from incrementing the requested encoder position, and instead it will just send a motor voltage to keep the motor moving. However, if the vehicle gets stuck, say in deep sand, there is presently no mechanism for it to recognize this and to implement an alternate maneuver to free itself.

The Trailer_Block module translates the hitch velocity back to left and right track velocities based on the dimensions of the vehicle saved in its configuration file. The Reg_Retrieve module retrieves a velocity for the lift and grip motors from an unused register, filling them with zero values and essentially placeholding the lift and grip velocity request. They get merged with the vehicle velocity request in the Merge module and passed to the Sandlizard module where they are sent to the Galil motion controller.

The Snsr_dsp module was written to interface with the payload digital signal processor (DSP) module. Appendix E provides the interface document detailing the communication interface and sequence of interaction. This interface was tested with the DSP software, but the software to implement the sensor deployment sequences was not done. The Snsr_dsp module performs several functions. When it is turned on, the DSP sends a New_Array command followed by a number of Sensor_Add commands to tell the vehicle the relative position of the sensors it wants deployed. The Ack and Resend commands are used in response to these data passing commands. As each sensor is placed, the vehicle sends a Sensor_Status command to tell the DSP where the sensor is placed. Once the array is placed and the vehicle has driven to a stationary 'hide' position, it sends an Array_Done command indicating it is OK to process signal data. The DSP uses the Send command to send text status and results information through the Vehicle control system and back to the OCU GUI where it is displayed in the DSP_Status module.

During the development program, the base station GPS unit was located at the operator control unit (OCU) station, and differential correction messages were sent from the OCU to the vehicle. In operation, the vehicle would drive without differential corrections or with satellite-broadcast differential corrections to the deployment site where it would place the base station GPS unit. This base GPS unit would have the same size and shape as a sensor pod, and would be the first pod that the vehicle would deploy. (Fig. 15) It would be set to monitor its position for a while and automatically switch into being a base station once it reached a certain position accuracy or after a designated elapsed time. (This is supported by a Novatel GPS command.) The unit would then start sending binary differential correction signals over one COM line and its fixed position in a log over the other COM line to the Tern A-Engine computer board embedded in the base station GPS unit. The A-Engine merges these signals into a third COM line which connects to the same type of radio (Signal Spectrum 24 802.11b wireless LAN) as is used by the sensor pods to relay data back to the DSP radio. The DSP extracts this information and passes it to the Snsr_dsp module over a serial line. The Snsr_dsp module can also relay GPS commands back to the base station GPS unit over this same route. The command originates with the operator interacting with the Snsr_status OCU GUI module. This module sends the GPS command embedded as a text string in a command to the Snsr_dsp module on the vehicle. The Snsr_dsp module then sends it out the serial communication port to the DSP module, which passes it to the sensor radio, which sends it to the GPS base station.

Camera Behavior

The camera behavior is independent of the vehicle operation behaviors previously discussed, and it can be active while either the teleoperation (joint) behavior or the autonavigation behavior is active. It enables some modules that control a remote tower camera located at the Robotic Vehicle Range (RVR). This camera can be zoomed and aimed using GUI controls to display the vehicle as it is tested outdoors on the RVR grounds. Its grid is shown in Fig 38.

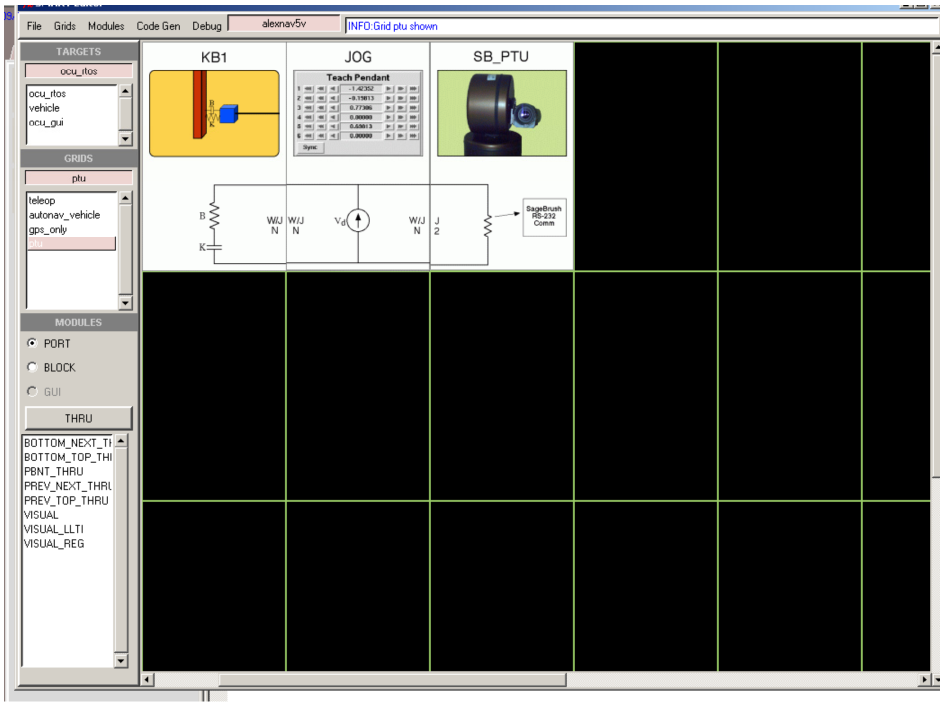


Figure 38 – Camera Grid

GUI Software Grid

The GUI software is active during all behaviors, and in fact it contains a module that enables the operator to control the vehicle behavior by sending commands to the SMART RTOS state engine code on the vehicle to change its own behavior. Unlike the RTOS code which is written in C, the GUI code is written the Tcl/Tk script language. It is created in the SMART Editor as a grid, as shown in Fig. 39.

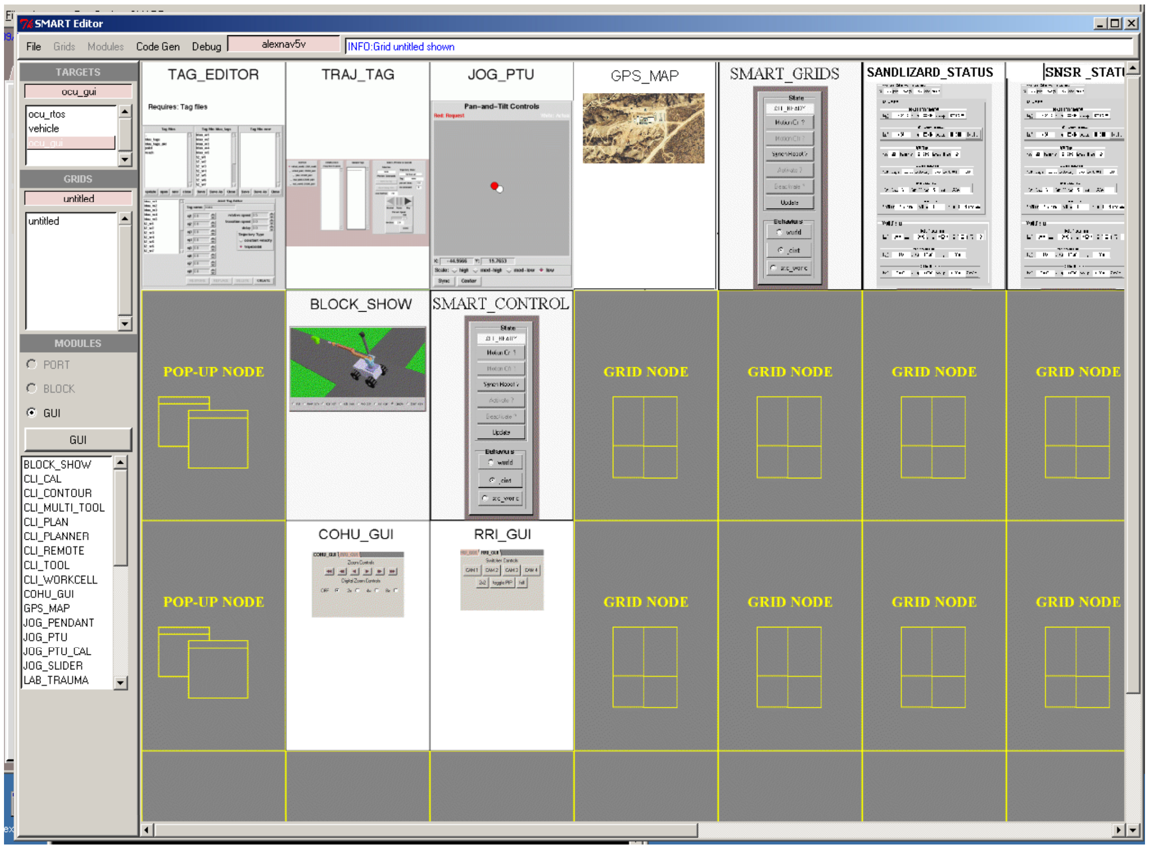
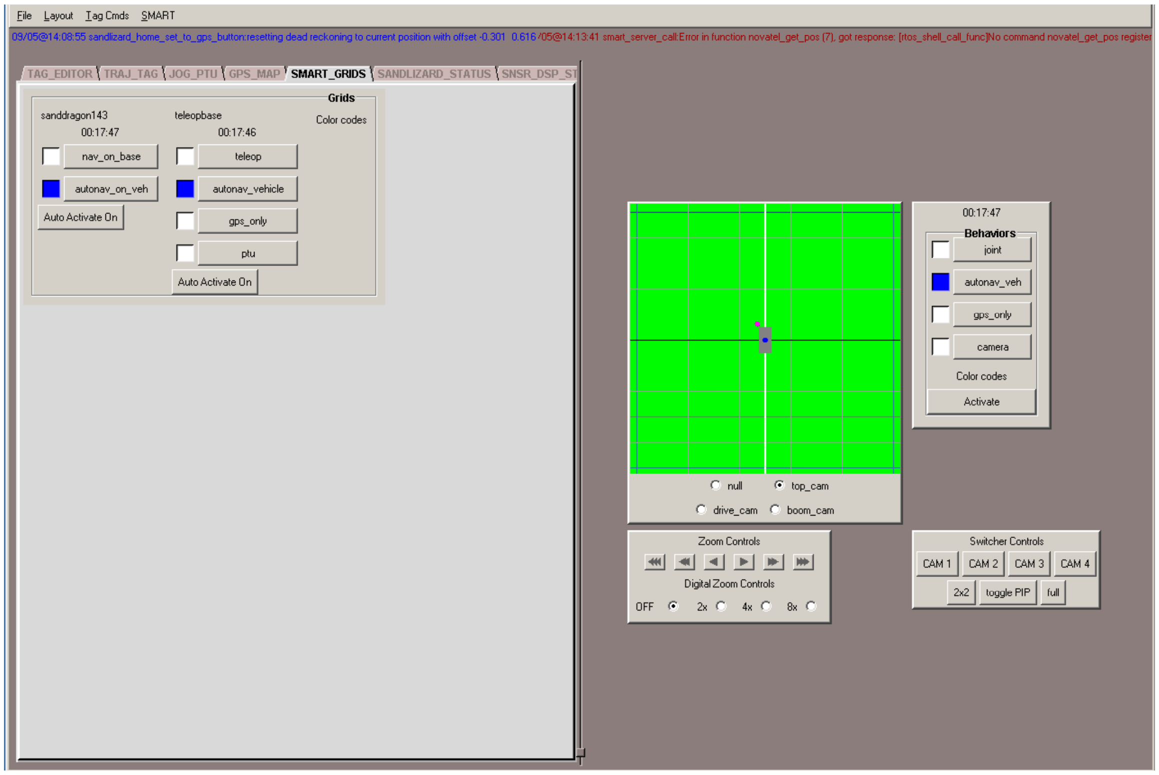


Figure 39 – GUI Grid for the SandDragon Vehicle

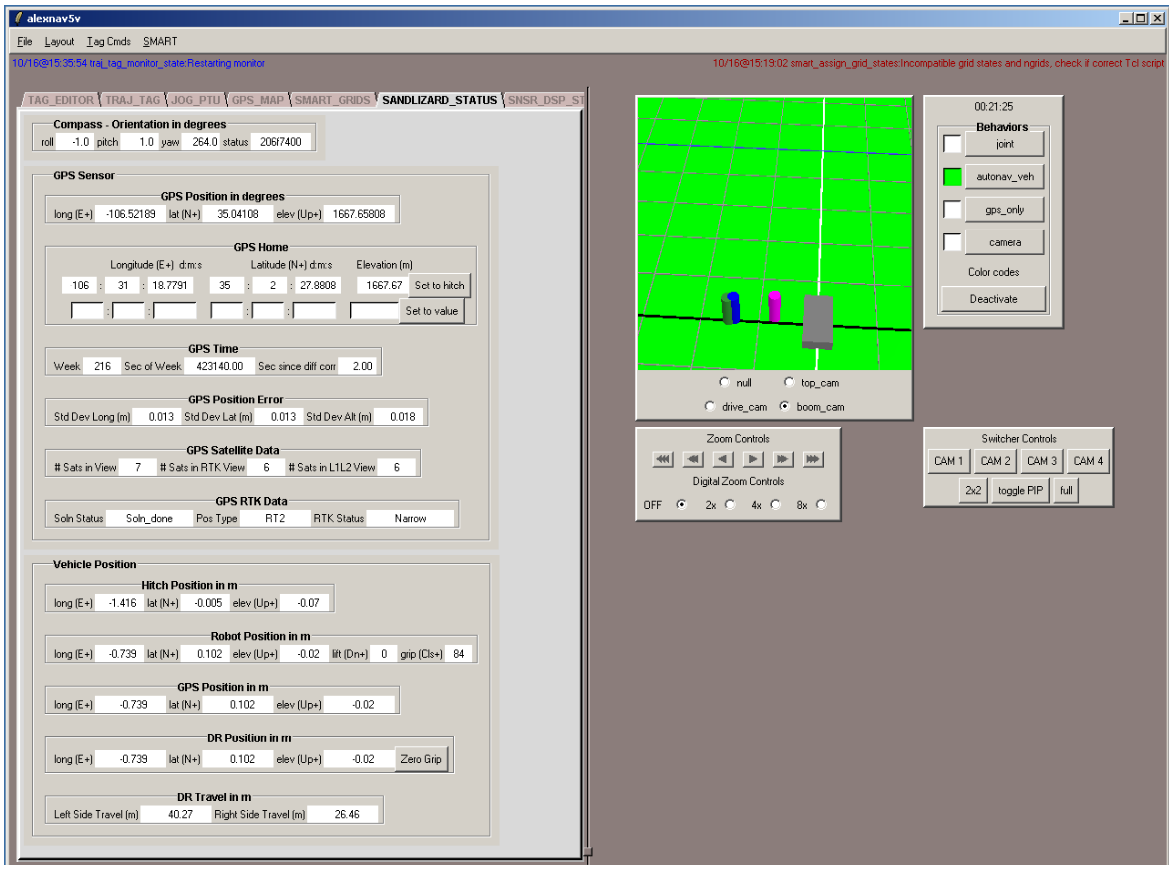


09/05@14:08:55 sandlizard_home_set_to_gps_button:resetting dead reckoning to current position with offset -0.301 0.616 /05@14:13:41 smart_server_call:Error in function novatel_get_pos (7), got response: [rtos_shell_call_func]No command novatel_get_pos register

Figure 40 – GUI Display Showing SMART Grids Tab

The SMART Editor generates the single executable Tcl script from the component GUI modules, and the resulting display is shown in Fig. 40 showing the SMART Grids tab page. The Behaviors window on the right side of the display enables the operator to view and select the active behavior for the system, and the SMART Grids tab page allows the operator to select or view the active grid on each target. In the view shown, the behavior is *autonav_veh* and the grids are *autonav_on_veh* on the sandddragon143 vehicle target, and *autonav_vehicle* on the teleopbase target. The color indicates the state of the grid, with blue being Flow_On and green being Activated. Flow_On allows all data to be processed but the motors are not energized while the Activated state enable motion. The green box is the Block_Show module that gives a visual indication of the position of the vehicle. In this application, three cylinders represent the position of the vehicle GPS antenna (purple), the vehicle hitch (blue), and the requested servo position for the hitch (green). These values are set up in the Block_Show configuration file. The operator can switch views from the top view, shown, to a boom camera or a drive camera view. The Zoom Control and the Switcher Controls modules control the tower camera zoom and the display of one or more images from the tower camera or vehicle camera on the monitor.

Other tabs can be viewed on the GUI tab page. The Sandlizard_Status tab displays vehicle status information. (Fig. 41) This figure also shows the boom camera view for the Block Show module.



```
10/16@15:35:54 traj_tag_monitor_state:Restarting monitor
10/16@15:19:02 smart_assign_grid_states:Incompatible grid states and ngrids, check if correct Tcl script
```

Compass - Orientation in degrees

roll -1.0 pitch 1.0 yaw 264.0 status 206f7400

GPS Sensor

GPS Position in degrees

long (E+) -106.52189 lat (N+) 35.04108 elev (Up+) 1667.65808

GPS Home

Longitude (E+) d:m:s			Latitude (N+) d:m:s			Elevation (m)
-106	31	18.7791	35	2	27.8808	1667.67

GPS Time

Week 216 Sec of Week 423140.00 Sec since diff corr 2.00

GPS Position Error

Std Dev Long (m) 0.013 Std Dev Lat (m) 0.013 Std Dev Alt (m) 0.018

GPS Satellite Data

Sats in View 7 # Sats in RTK View 6 # Sats in L1L2 View 6

GPS RTK Data

Soln Status Soln_done Pos Type RT2 RTK Status Narrow

Vehicle Position

Hitch Position in m

long (E+) -1.416 lat (N+) -0.005 elev (Up+) -0.07

Robot Position in m

long (E+) -0.739 lat (N+) 0.102 elev (Up+) -0.02 lift (Dn+) 0 grip (Cls+) 84

GPS Position in m

long (E+) -0.739 lat (N+) 0.102 elev (Up+) -0.02

DR Position in m

long (E+) -0.739 lat (N+) 0.102 elev (Up+) -0.02

DR Travel in m

Left Side Travel (m) 40.27 Right Side Travel (m) 26.46

00:21:25

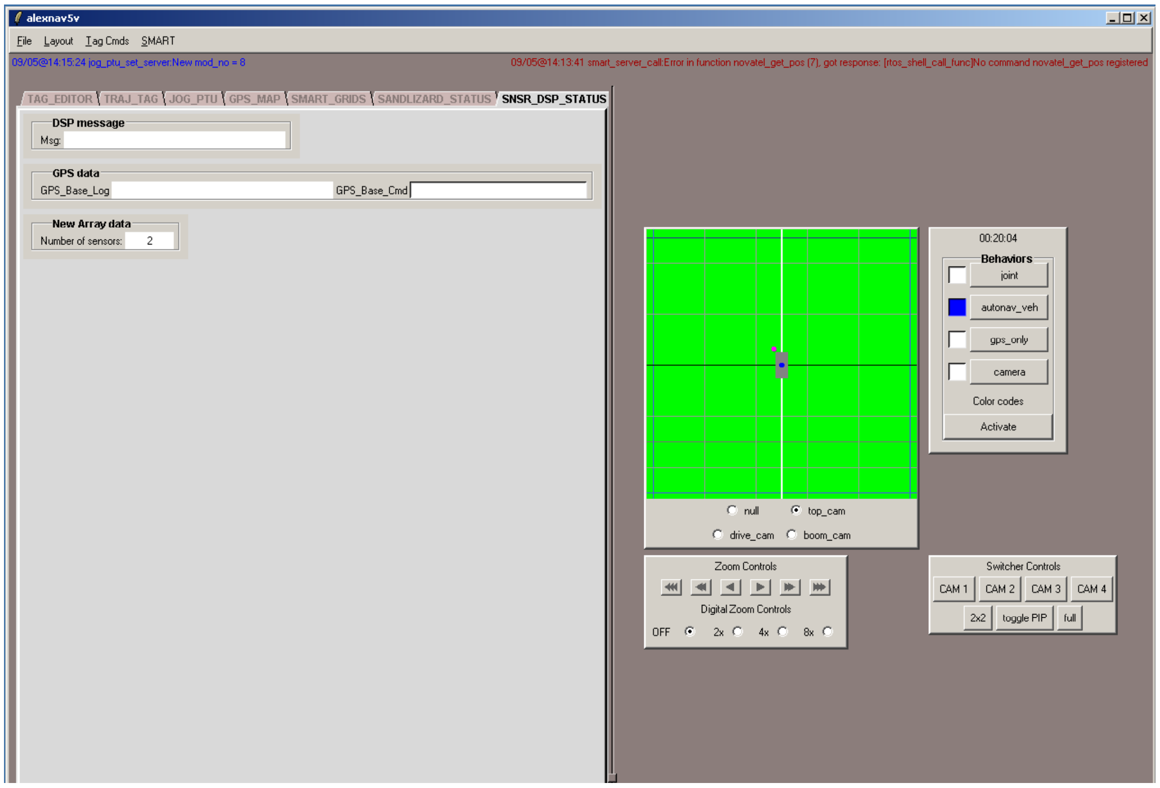
[✓ boom_cam]

OFF [✓] 2x 4x 8x

Figure 41 – Sandlizard Status Tab

The data displayed in the Sandlizard Status tab page is broken into separate named frames. This data is originally stored in registers on the vehicle and is passed to the base station registers by the pair of Reg_Serial modules. The Sandlizard Status Tcl module then displays the data in real time. The named frames contain data on the compass reading, the GPS data, and the vehicle position. The digital compass provide roll, pitch, and yaw data in degrees, and a status byte is used to turn the value color red if the data is not valid. The GPS frame contains several sub frames displaying status data from the GPS unit. The first frame gives the GPS position in digital degrees. The next provides the HOME position in degrees, minutes, and seconds and is used to reference the vehicle position in meters. The default HOME position is set in the configuration file to be the RVR base GPS position, however the HOME position can be reset to be any location and is best set to be in the vicinity of vehicle operations. The first button in this frame sets the HOME position to the current GPS position of the hitch point, which requires that the software read the compass yaw direction and the hitch offset from the Pos_Offset module. If the vehicle is not in the *autonav* behavior, the position defaults to setting HOME to the current position of the GPS antenna. The second button sets the HOME position to the value entered in the adjacent text entry boxes. The HOME frame is followed by frames giving the GPS time, position error, satellite data, and real-time-kinematic (RTK) status of the solution. The position error is particularly useful as it displays when the GPS unit has locked into the 1-cm accuracy range. The vehicle position data includes several sub frames displaying position in meters relative to HOME. These include the vehicle hitch position, the best estimate of the GPS antenna (robot) position, the indicated GPS antenna position from the GPS unit, and the dead reckoning estimate of the GPS antenna position. The final sub frame displays the left and right track odometer readings used by the dead reckoning algorithm.

The next tab, Fig. 42, displays the data from the payload DSP unit, and enables the operator to send commands to the remote GPS base station when it is deployed by the vehicle. The first line displays messages sent by the DSP processor unit. The next displays log data sent by the remote GPS unit, and has a text entry box for sending commands to the remote GPS base station. During the development phase the base station GPS unit is located at the operator control station, but during operation it will be carried on the vehicle and deployed at the sensing site. In this case, communication with the GPS unit is through the DSP unit. The final frame displays the New Array command parameters. Other frames can be created to display the sensor requested position and deployment position.



```
09/05@14:15:24 jog_ptu_set_server:New mod_no = 8
09/05@14:13:41 smart_server_call:Error in function novatel_get_pos (7), got response: [rtos_shell_call_func]No command novatel_get_pos registered
```

Msg:

GPS_Base_Log

GPS_Base_Cmd

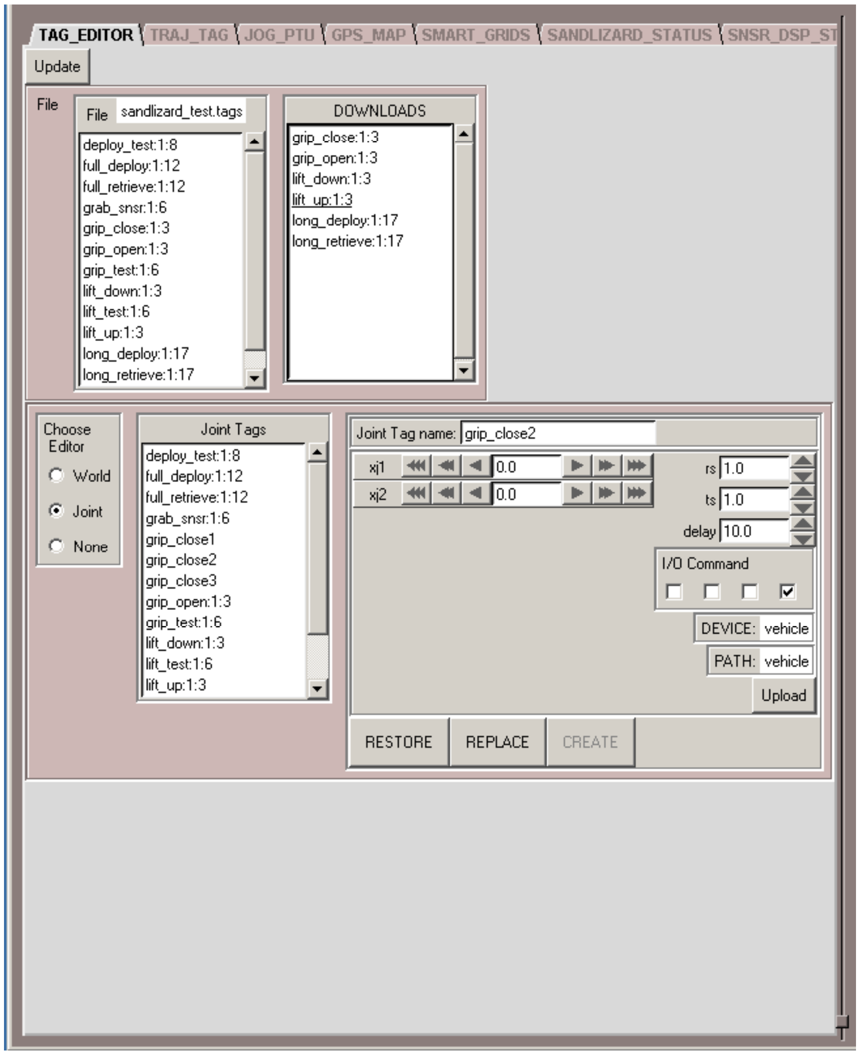
Number of sensors: 2

00:20:04

null [✓ top_cam] drive_cam boom_cam

[✓ OFF] 2x 4x 8x

Figure 42 – DSP Status and Communication Tab



File: sandlizard_test.tags

deploy_test:1:8
full_deploy:1:12
full_retrieve:1:12
grab_snsr:1:6
grip_close:1:3
grip_open:1:3
grip_test:1:6
lift_down:1:3
lift_test:1:6
lift_up:1:3
long_deploy:1:17
long_retrieve:1:17

DOWNLOADS

grip_close:1:3
grip_open:1:3
lift_down:1:3
lift_up:1:3
long_deploy:1:17
long_retrieve:1:17

Choose Editor: World [✓ Joint] None

Joint Tags

deploy_test:1:8
full_deploy:1:12
full_retrieve:1:12
grab_snsr:1:6
grip_close1
grip_close2
grip_close3
grip_open:1:3
grip_test:1:6
lift_down:1:3
lift_test:1:6
lift_up:1:3

Joint Tag name: grip_close2

xj1 0.0
xj2 0.0
rs 1.0
ts 1.0
delay 10.0
I/O Command: ☐ ☐ ☐ ☑
DEVICE: vehicle
PATH: vehicle

Figure 43 – Tag Editor Tab

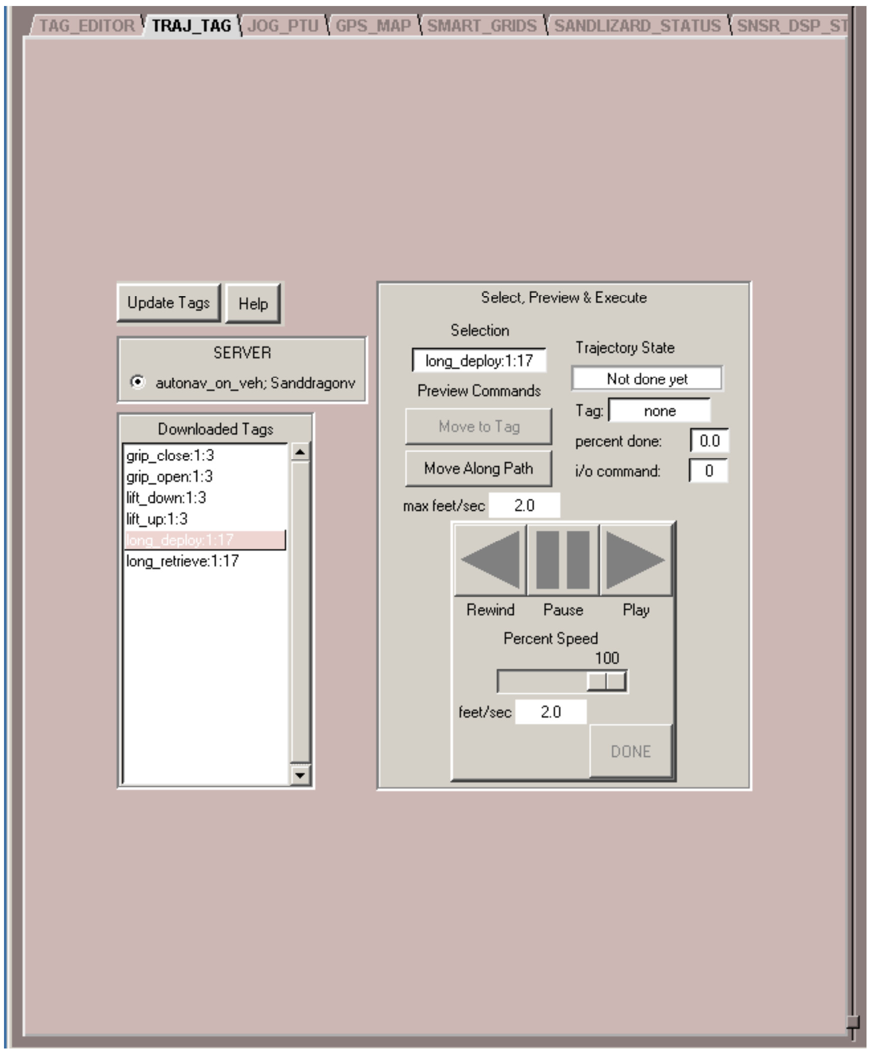
The Tag Editor tab (Fig. 43) provides an interface for the operator to create waypoint tags which combine to define trajectories. Clicking on the upper left File label accesses a drop-down menu to open saved tag files. The displayed tag file contains several tag sequences, in the left display box. Each tag sequence consists of sequential tags with the same base name and a sequential suffix number. They are displayed collapsed, with the first and last sequence number shown, or they can be expanded to show all the component tags by double clicking them. A tag sequence can be sent to the vehicle by copying and pasting it to the adjacent Downloads window. The tags can be created or edited using the lower part of the window. Here, the tag *grip_close2* is being edited. The tag entry specifies the x and y position, corresponding to the east and north position of the vehicle relative to HOME, the relative speed (rs), the transition speed (ts), where a value of 0 means the vehicle stops and the tag and a value of 1 means the vehicle goes through the point without slowing, and the delay at the tag point if the vehicle stops. The IO_Cmd boxes set the binary flags that signal the Sandlizard module to raise (1) or lower (2) or open (4) or close (8) the lift and grip mechanism. Table 13 contains some tag sequences for the vehicle. The tags can be edited in the Tag Editor tab, or in any word processor in the tag file. If one is using the Tag Editor, the new or edited tags must be resaved, and copy and pasted to the File and Downloads windows.

Table 13 – Sample Tag File with Deploy Trajectory Sequences

```
#**************************************************
# Name: sandlizard_test.tags
# Programmer: Generated from tag_file_save by ratler
# Date: Thu Aug 14 15:42:54 Mountain Daylight Time 2003
#**************************************************

# io_cmds 1= lift up, 2=lift down, 4=grip open, 8 = grip closed
DEVICE: vehicle
PATH: vehicle
COLOR: red
FIELDS: name interp io_cmd ts rs delay xj1 xj2
deploy_test1 3 0 1.0  1.0 0.0 0.0 0.0
deploy_test2 3 0 1.0  1.0 5.0 0.0 0.0
deploy_test3 3 2 1.0  1.0 25.0 0.0 0.0
deploy_test4 3 4 1.0  1.0 10.0 0.0 0.0
deploy_test5 3 1 1.0  1.0 25.0 0.0 0.0
deploy_test6 3 8 1.0  1.0 10.0 0.0 0.0
deploy_test7 3 0 1.0  1.0 5.0 0.0 0.0
deploy_test8 3 0 1.0  1.0 0.0 0.0 0.0
full_deploy1 3 0 1.0  1.0 0.0 4.0 0.0
full_deploy2 3 0 1.0  1.0 5.0 4.0 0.0
full_deploy3 3 2 1.0  1.0 30.0 4.0 0.0
full_deploy4 3 4 1.0  1.0 10.0 4.0 0.0
full_deploy5 3 1 1.0  1.0 30.0 4.0 0.0
full_deploy6 3 8 1.0  1.0 10.0 4.0 0.0
full_deploy7 3 0 1.0  1.0 5.0 3.0 0.0
full_deploy8 3 0 1.0  1.0 0.0 2.0 -1
full_deploy9 3 0 1.0  1.0 5.0 1.0 0.0
full_deploy10 3 0 1.0  1.0 0.0 -2.0 0.0
full_deploy11 3 0 1.0  1.0 5.0 -1 -1
full_deploy12 3 0 1.0  1.0 0.0 0.0 0.0
```

The next tab, the Traj_Tag tab, sends command to the Traj_IO module on the vehicle to run or stop trajectories that have been downloaded to the vehicle by the just-mentioned copy and paste process. The tab is shown in Fig. 44. The Update Tags button is used to display the tag sequences previously downloaded to the vehicle. (If the vehicle software has rebooted since the tag files were downloaded, the tag files will still show in this window, but they are no longer on the vehicle and will need to be downloaded again.) Clicking on the tag sequence in the window loads it into the Trajectory Control window where it can be run by clicking on ‘Move Along Path’. The trajectory status, tag name, percent done, and IO_Command value are displayed as the data is sent back from the vehicle in the traj_tag_info register string.



SERVER

● autonav_on_veh; Sanddragonv

Downloaded Tags

- grip_close:1:3
- grip_open:1:3
- lift_down:1:3
- lift_up:1:3
- long_deploy:1:17
- long_retrieve:1:17

Selection: long_deploy:1:17

Preview Commands

max feet/sec 2.0

Trajectory State: Not done yet

Tag: none

percent done: 0.0

i/o command: 0

Percent Speed 100

feet/sec 2.0

Figure 44 – Traj Tag Tab

The Jog_PTU tab (Fig. 45) provides an interface for the user to control the aiming of the tower camera by selecting the motion sensitivity and moving the red aiming ball with the cursor.

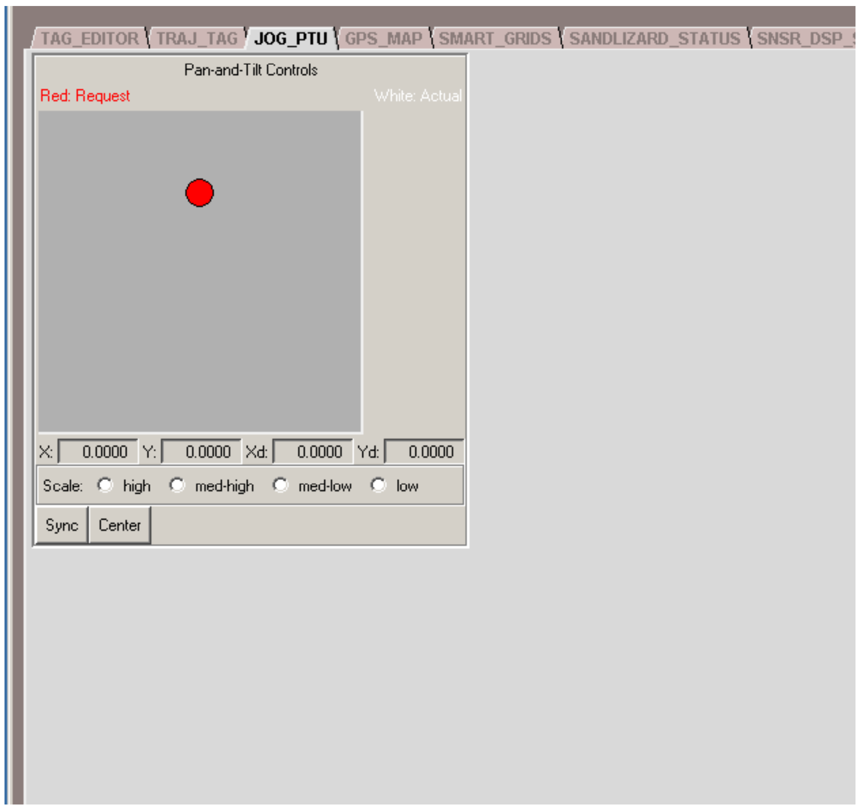
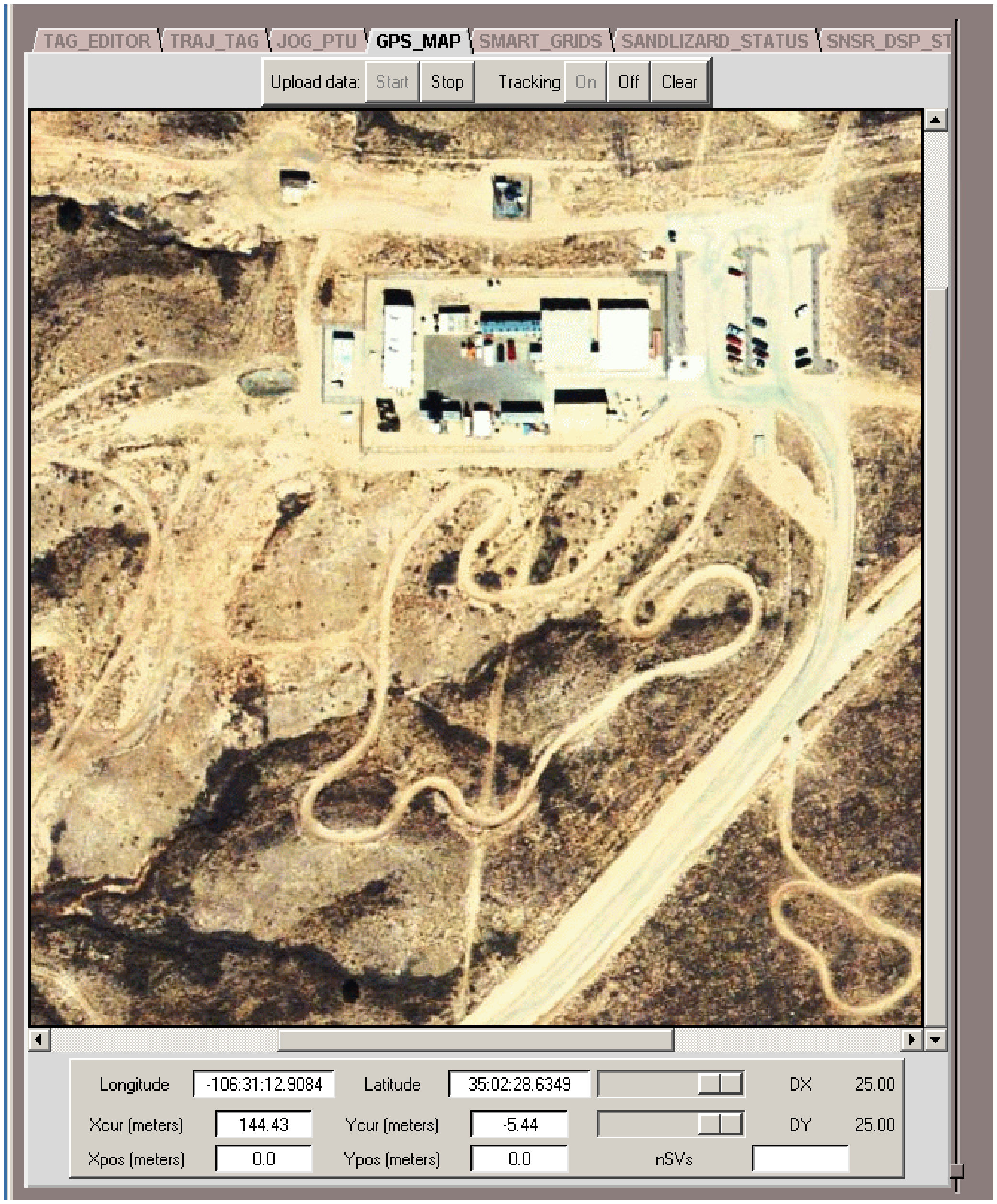


Figure 45 – Jog PTU Tab

The GPS_Map Tab (Fig. 46) is not fully implemented for selecting waypoints or displaying the path followed, but it provides an interface between an aerial view and GPS coordinates that can be selected with the cursor. We intend to implement this tab to aid in creating tag records visually rather than by x and y coordinates.



Longitude	-106:31:12.9084	Latitude	35:02:28.6349		DX	25.00
Xcur (meters)	144.43	Ycur (meters)	-5.44		DY	25.00
Xpos (meters)	0.0	Ypos (meters)	0.0	nSVs		

Figure 46 – GPS Map Tab showing the RVR Vicinity

Vehicle Testing and Demonstration

Testing proceeded through a sequence of trial and error steps identifying and fixing problems. Eventually we reached a point at which we were able to successfully drive along waypoints specified in the tag files and to raise, lower, open, and close the sensor gripper mechanism. At that point we were able to refine our procedures and tags to perform core maneuvers such as approach and recover a previously deployed sensor pod.

The sequence of actions to perform a demonstration is given below.

1) Power up the vehicle by raising the stop button on the vehicle body.
2) Open a SMART shell window (click on the SMART_Shell icon), change (cd command) to the smart_0.9/rtos/demos directory, and run the appropriate base station executable, such as alexnav5v_2.exe.
3) Power on the radio communication link (portable radio box) and check for video from the vehicle on the comm box display. After about 90 seconds, if the vehicle boots successfully, the video image will have the voltage and other system data superimposed on the video image. If it doesn't, the vehicle code may have hung up (generally during the Galil board initialization) and needs to be restarted by cycling the stop button on the vehicle.
4) The base station code and the vehicle code exchange register data, which can be seen on the base station shell window. This takes a minute or two.
5) Once the base station code has ended its exchange of register data, open a SMART_GUI Tcl/Tk console window (SMART_GUI icon) and start the Tcl script by typing 'source alexnav5v.tcl' substituting the correct file name.
6) As the Tcl script boots, it creates some registers and passes the default tag file to the vehicle. Once it successfully boots, the vehicle behavior module shows the vehicle to be in the joint (teleop) behavior mode in the 'Flow_On' (blue color) state. If it does not successfully boot, generally because the script was launched too early, all four behaviors will display a blue box. Click on the console window (using the strip at the bottom of the page), type 'exit', and then relaunch the Tcl script (Step 6).
7) Click on the Sandlizard_Status tab of the GUI to see the vehicle status variables.
8) If it isn't there already, move the vehicle outdoors where it has a clear view of the sky. Watch the GPS status variables and wait until the vehicle has acquired sufficient GPS satellites and has locked in so that the standard deviation of the position error drops to 0.01 meters (1 centimeter). This takes up to 5 minutes.
9) Click on the autonav behavior and check that the blue box moves to the *autonav_veh* behavior mode.
10) Click on the *Set to Hitch* button in the GPS Home frame of the Sandlizard_Status tab. This will set the HOME position to the current hitch position. The hitch position should go to zero in the Vehicle Position frame.
11) To operate in the Teleop mode, click on the Joint (Teleop) button in the Behavior window and then click on the Activate button. The box next to the Joint mode should turn green, indicating the grids on the base station and the vehicle are enabled for motion. Depress the deadman button on the left side of the gamepad and operate the drive with the left joystick. To operate the lift/grip mechanism, use the deadman buttons and joystick on the right side of the gamepad. To deactive the vehicle, press the Deactivate button in the Behavior window.

12) To operate in the Autonav mode, you first need to download a tag or tag sequence to the vehicle.
    a) Click on the Tag_Editor tab and then on the File label. Click Open and open the saved tag file, which will load saved tag sequences into the window.
    b) To edit or create new tag sequences, click the Joint button in the lower half of the window and create or edit the tags. Tags in a sequence have the same base name and sequential suffix numbers. There is more information on this module in the write-up above.
    c) Download the tag sequence to the vehicle by highlighting the sequence in the file window, copying the sequence (Ctrl-C) and pasting to the Downloads window (Ctrl-V). This will initiate sending the tag sequence to the vehicle. Once this is complete, after a few seconds, the tag sequence name appears in the Downloads window. Download all tags you may want to use.
    d) Switch to the Trag_Tag tab and click Update to display the tag sequences on the vehicle. (If the vehicle has been rebooted since the tags were last downloaded they will need to be downloaded again even though they appear in this window.)
    e) Click on a sequence to select it, or double click a sequence to expand it into its component tags. You can select an individual tag to go to.
    f) In the Behavior window, select the *autonav_veh* behavior. The hitch and robot GPS antenna position should appear as cylinders in the Block_Show window. You may need to restart the Block_Show update using the entry under the SMART menu. Due to timing issues, sometimes it tries to start before the initial position register handshaking has been completed, and it shuts itself down.
    g) Check that the vehicle hitch position is zero, or rezero the HOME position to the hitch position (Step 10). Check that the GPS standard deviation of error is in the centimeter range, if possible.
    h) Click on activate to enable the motors. The vehicle should move around slightly, on the order of a centimeter as it adjusts slightly for GPS errors.
    i) On the Traj_Tag tab, Select, Preview, & Execute window, click on the Move Along Path button to begin motion. The status of the trajectory will be displayed, and the Block_Show display will update the motion progress relative to the requested position.

The method of interacting with the vehicle code is through commands that are registered with the RTOS in the code. Some of these commands are accessed through the Tcl GUI code using buttons. Others that are not accessible through buttons can be accessed through the RTOS console by using the ‘reg_serial_send_cmd’ command to send the command and its parameters to the vehicle. Available commands that are not part of the core SMART architecture are listed in Appendix D.

Deployment/Retrieval Demonstration

Once the vehicle was operating correctly, were able to successfully deploy and retrieve a demonstration sensor pod that was fabricated out of a block of plastic. The demonstration pod has roughly the same weight as the planned active sensor pod. (Fig. 47) Positional accuracy of the vehicle was excellent once the GPS locked in to its centimeter accuracy range. The vehicle had a tendency to overshoot and then back up to the correct position, which sometimes pushed the pod a few centimeters, but this was reduced by programming the vehicle to stop short of the pod, open and lower its gripper, and then advance at a reduced speed before closing the gripper about the sensor pod. The gripper has about 3 inches of lateral and axial tolerance for pod misalignment, which easily accommodates the GPS positional accuracy and some movement of the pod as well. We were able to repeatedly return to the same location given the GPS centimeter accuracy relative to a fixed base station.

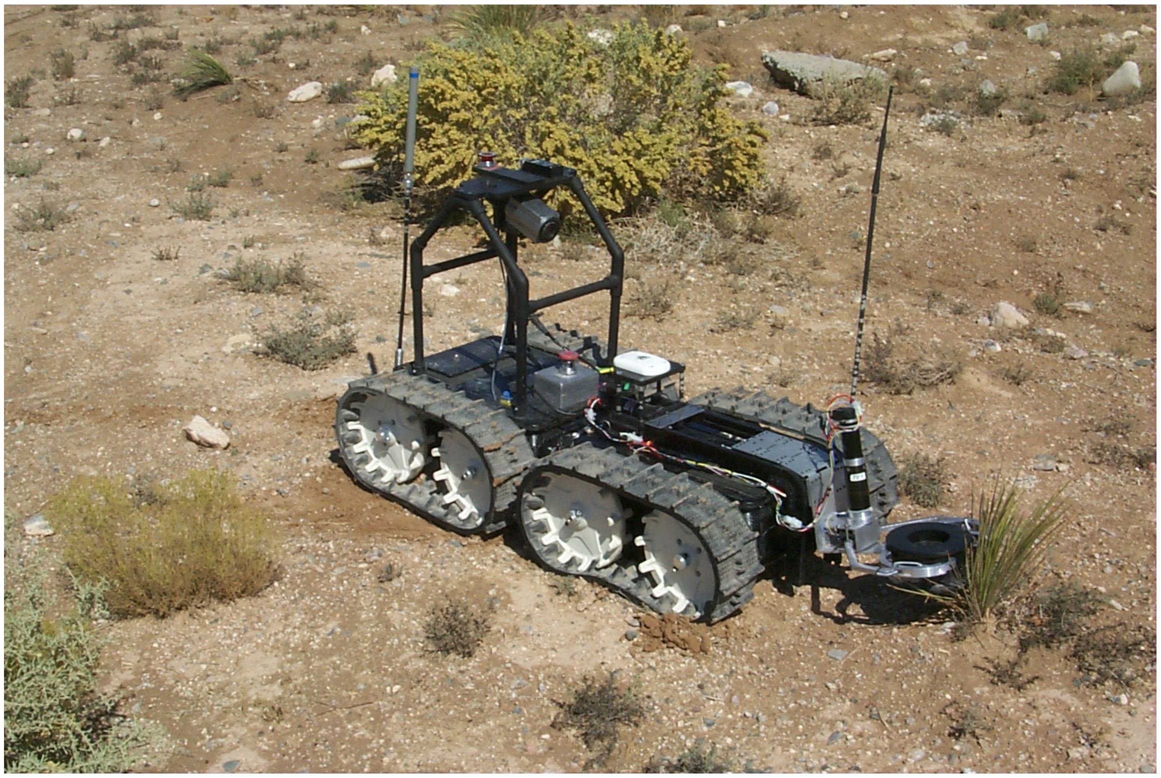


Figure 47 - SandDragon Vehicle Holding Sensor Pod Replica

Sensor Array and Signal Processing Hardware

The payload system is comprised of a controller unit mounted on the robotic vehicle, and multiple sensor pods configured in an interferometric array. The primary function of the controller pod is to execute a ‘beam-forming’ algorithm designed to determine the location and frequencies of a stationary signal-generating object by processing signals measured at each of the sensors. Performing this analysis requires accurate temporal (time) alignment of the signal data from each sensor, knowing the exact relative positions of the sensors, and having a good estimate of the signal velocity. The signal gain can be increased and signal direction determined by properly phase-aligning and analyzing the data. Once an initial estimate of the signal source direction and frequency is obtained, the controller pod then is responsible for generating and transmitting to the vehicle an optimized array configuration for improved signal monitoring. The vehicle is then responsible for picking up and repositioning the sensors.

As discussed in the analysis portion of this report, two types of signals can be monitored, acoustic and seismic. The acoustic sensor was fabricated and used to conduct field tests to gather signal data for the algorithm development (Figs. 6-8). With the exception of the signal parameters such as propagation speed, the processing of the two types of signal is identical. This project put most of its effort on the harder problem involved with seismic sensors, made difficult because the signals need to be transmitted by radio, while preserving their time synchronization, back to the controller pod that does the signal processing.

For this development program, the sensor pods were simulated using five instrument boxes housing the seismic sensors, C5510 processors, radios, and battery power systems. In an operational system these would be repackaged into a much smaller optimized sensor pod.

Payload Controller Unit

The controller unit consists of an Orsys® micro-line™ C6711 DSP board (Fig. 48) and a SymbolOEM™ Spectrum 24 802.11b wireless radio in a PCMCIA package used to communicate with the deployed seismic sensor pods. The micro-line™ board has a Texas Instruments® TMS320C6711 32-bit floating-point DSP with 100MHz CPU. It also has 64MB off-chip SDRAM and a peripheral UART device used for RS-232 communications with the vehicle controller. Much of the board is not used for this application, and for a final product the board could be shrunk significantly by redesigning it with only the necessary components.

Figure 48 – Orsys DSP Board Controller Unit

The Spectrum 24 radio is connected via the External Memory Interface (EMIF) of the DSP, which allows the radio registers to be accessed a byte at a time. The radio interface is interrupt-driven. The IRQ line of the radio is connected to external interrupt 0 on the DSP.

Figure 49 is a conceptual view of the DSP software configuration. The code structure that holds it all together is a high-level model consisting of C++ code that was generated using the Rational Rose Realtime™ modeling tool. The high-level model is shown in Fig. 49 as the Data Flow Manager. The model links five main capsules: the radio callback interface (low-level control interface to the signal pod radio), the UART callback interface (low-level control interface to the UART that communicates with the vehicle), the beam former (signal processor), the robot interface (high-level state engine) and the sensor system interface (high-level state engine). The beam former capsule implements the array-processing algorithm and is represented by all the blocks in Fig. 49 above the Data Flow Manager. The robot interface capsule has the state machine that coordinates communications between the robot and the beam former, while the system interface capsule coordinates communications between the controller and each of the sensor pods.

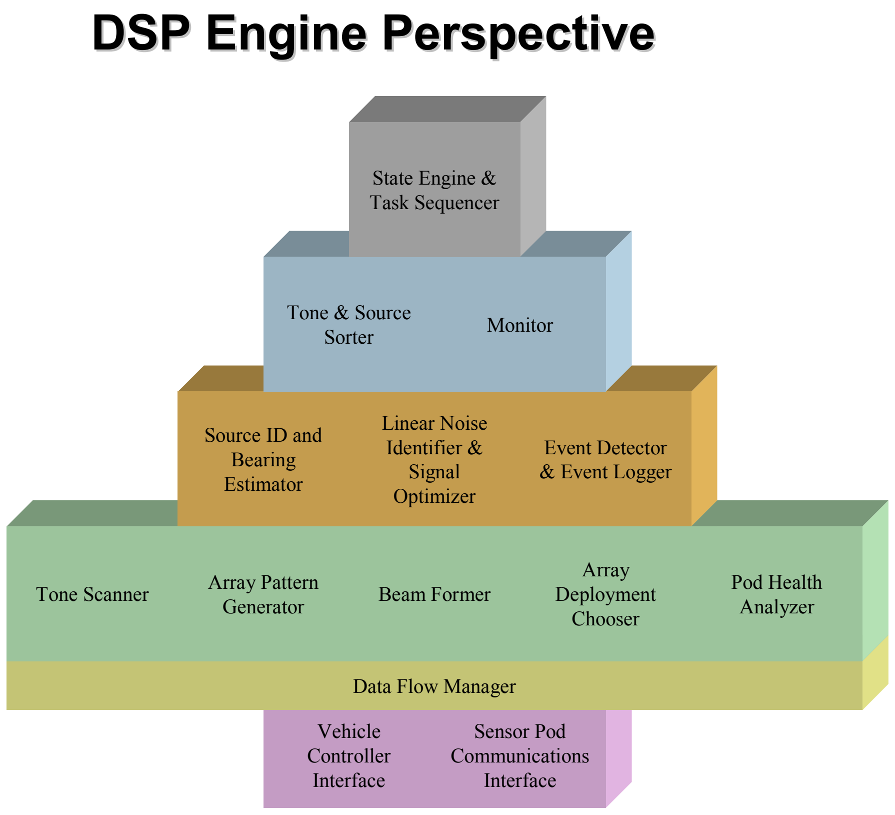

DSP Engine Perspective
State Engine & Task Sequencer
Tone & Source Sorter
Monitor
Source ID and Bearing Estimator
Linear Noise Identifier & Signal Optimizer
Event Detector & Event Logger
Tone Scanner
Array Pattern Generator
Beam Former
Array Deployment Chooser
Pod Health Analyzer
Data Flow Manager
Vehicle Controller Interface
Sensor Pod Communications Interface

Figure 49 - Conceptual Diagram of the DSP Software

The UART and radio callback capsules are written in C. The UART code is interrupt-driven and uses separate tasks to configure the UART, transmit data to the robot, and receive data from the robot. Callbacks are used in the transmit and receive tasks to hook into the UART callback capsule of the model.

Similarly, the radio code is interrupt-driven and uses separate tasks to process the data. Due to the high volume of data being received over the radio, there are two tasks that deal with received data: a task that removes the data from the radio and stores it, and a concurrent task that parses the data to figure out what is in the message.

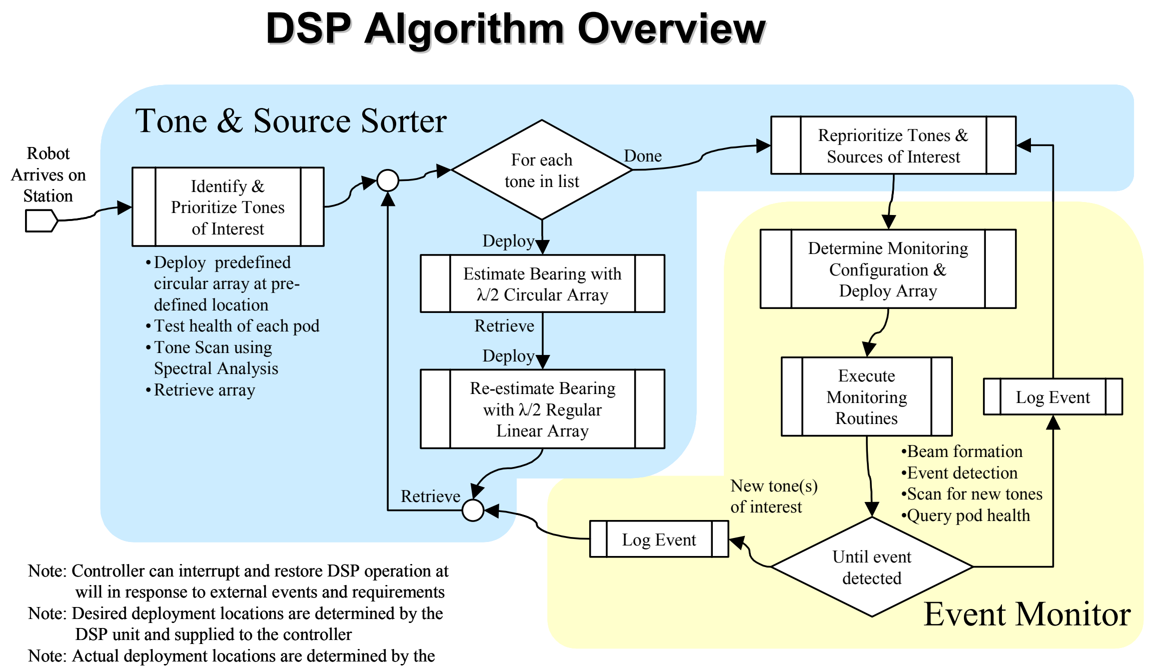
Real-time vs. Pseudo-real-time

Initially, the system design called for fully real-time acquisition and processing of the data. There were some issues that prevented this method from being feasible in the project time frame. One issue was that there were numerous tasks running on one processor. This created a situation where the processor-intensive beam former was stealing CPU cycles from the radio tasks trying to keep up with the incoming data flow, causing missed packets and incomplete data sets. A possible solution is to have a faster DSP. However, even without the beam former running, the sheer volume of incoming data from all five pods simultaneously was too much for the master radio to handle. Each pod was trying to send 1024 samples of data broken up into 4 packets. Due to the high number of incoming packets to the DSP, data packet collision occurred, preventing the C5510 pod processor and the DSP processor board from operating in real-time. This issue points out the fact that we must have collision avoidance and detection algorithms in place before real time DSP algorithms can be implemented. To solve that problem, radios with the ability to buffer larger amounts of incoming data are necessary. Another alternative to a faster DSP is to have a distributed computing environment that would relieve the processing load on the main DSP. This could be accomplished by pushing more of the workload down to the pod processors. Since the pods are only sampling, they can do a few more operations and maybe even extract features that are currently done on the central DSP. The idea here is to reduce the number of packets sent to the DSP. Doing so will also reduce the size of these packets because ideally these feature sets would be smaller than the raw sampled data.

To remedy the problem in the available time, a pseudo-real-time approach was adopted. Since the beam former needs a certain amount of time-continuous data samples to generate new coordinates for the sensor pods, each pod now buffers 180 seconds worth of continuous data samples. When the samples are collected, each pod notifies the controller that it is ready to send the data. The controller then serially sends requests to each pod individually, which enables the start of the data transfer. After all five sets of data are collected on the controller, the data is then fed into the beam former one second at a time. Only after the robot has received and acknowledged new coordinates for the sensor pods, and has placed them appropriately, will the controller inform the sensor pods to resume buffering real-time sample data. The major difference between this method and a fully real-time method is that there will be a time gap between each 180s data set. Since the source is stationary, the time gap does not cause any issues with the validity of the array processing.

Sensor System Control Algorithm

The algorithm originally envisioned for the DSP-based signal processing unit was a comprehensive one that not only sorted through and identified signals of interest, but it would iteratively prioritize and select tones to monitor more carefully, and optimize an array configuration to implement, which it would pass to the vehicle. This comprehensive algorithm is depicted in Fig. 50. During this project we were able to implement the core parts of this algorithm, the seismic/acoustic signal processing and beam-forming algorithms, and the array optimization and layout algorithm, and subsequent communication of the configuration to the vehicle.


DSP Algorithm Overview
Tone & Source Sorter
Robot Arrives on Station
Identify & Prioritize Tones of Interest
•Deploy predefined circular array at pre-defined location
•Test health of each pod
•Tone Scan using Spectral Analysis
•Retrieve array
For each tone in list
Done
Deploy
Estimate Bearing with λ/2 Circular Array
Retrieve
Deploy
Re-estimate Bearing with λ/2 Regular Linear Array
Retrieve
Reprioritize Tones & Sources of Interest
Determine Monitoring Configuration & Deploy Array
Execute Monitoring Routines
•Beam formation
•Event detection
•Scan for new tones
•Query pod health
Log Event
New tone(s) of interest
Until event detected
Log Event
Event Monitor

Note: Controller can interrupt and restore DSP operation at will in response to external events and requirements
Note: Desired deployment locations are determined by the DSP unit and supplied to the controller
Note: Actual deployment locations are determined by the Controller and supplied to the DSP unit
Note: Retrieval is a function only of the controller

Figure 50 - DSP Algorthm Flowchart

Beam-forming / Array Optimization Algorithm

The purpose of the beam-forming algorithm is to extract source characteristics to be used in deriving an optimal array for direction of arrival estimation. The source is assumed to be a mono-tonal seismic signal traveling along the surface. Frequency domain conventional (delay-and-sum) beam forming is used to perform frequency-slowness (FK) analysis on the received signals. Frequency domain analysis is chosen over time domain to eliminate the need for up-sampling. Array optimization is performed by iteratively adjusting the array geometry until the minimum angular resolution is achieved. The geometries are adjusted based on the source characteristics obtained by the beam-forming algorithm.

Algorithm Block Diagram

Figure 51 gives a block diagram of the beam-forming algorithm sequence. Each block is discussed below.

Block 1 - Initial frequency and velocity will be preset parameters based on the source and soil characteristics. From these parameters an initial array configuration (sensor location) will be determined.
Block 2 - The channel signal will then be filtered using a bandpass filter at initial frequency.
Block 3 - The FFT of the filtered signal.
Block 4 - The spatial correlation matrix is computed using the positive frequency values of the FFT results.
Block 5 - Steering vectors are generated based on the current frequency, velocity and sensor locations.
Block 6 - The algorithm, delay and sum beam-forming, is used to calculate the weights (future).
Block 7 - The beam pattern is generated using steering vectors and spatial correlation matrix.
Block 8 - A direction of arrival is determined using the beam pattern.
Block 9 – Beam pattern values are stored until all frequencies are processed and a target determination is made. A new array configuration is determined based on the frequency, velocity and bearing that result in the highest beam pattern power (FK analysis).

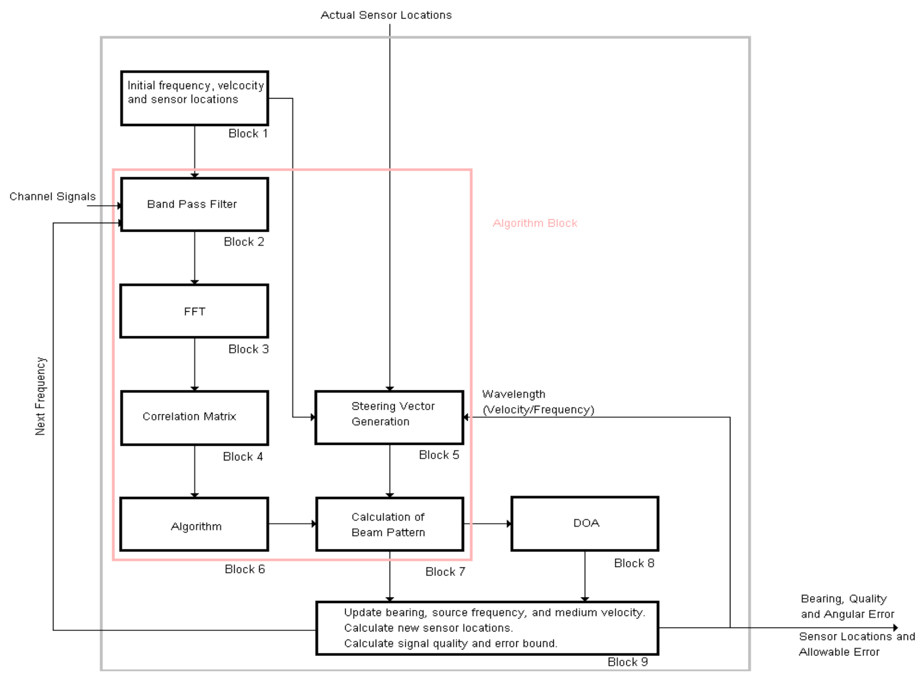

Actual Sensor Locations
Initial frequency, velcocity and sensor locations
Block 1
Channel Signals
Band Pass Filter
Block 2
Algorithm Block
FFT
Block 3
Next Frequency
Correlation Matrix
Block 4
Steering Vector Generation
Block 5
Wavelength (Velocity/Frequency)
Algorithm
Block 6
Calculation of Beam Pattern
Block 7
DOA
Block 8
Update bearing, source frequency, and medium velocity.
Calculate new sensor locations.
Calculate signal quality and error bound.
Block 9
Bearing, Quality and Angular Error
Sensor Locations and Allowable Error

Figure 51 – Beam Forming Algorithm Block Diagram

Preconditions

To ensure the implemented algorithm performs properly and within time constraints, certain preconditions are assumed.

- Sensor locations are known to within 1cm.
- Source frequency is known +/- 10%.
- Medium velocity is known +/- 20%.
- Analog anti-aliasing filter phase shift variance less than 2%.
- Acoustic signal is suppressed, at least 3dB below seismic.
- Source seismic signal is at least 6dB above the noise level (6dB above the grass).
- Sampling lag between pods is less than 1 millisecond (approx).

Algorithm Speed vs. Accuracy

Real-time systems required that all processing be completed within a single window of time before the next set of data is available. A necessary tradeoff is made between speed and accuracy to ensure that this time constraint is achieved. Some of the factors that are used in determining these tradeoffs include the following:

- Algorithm complexity.
- Filtering requirements.
- Number of sensors.
- Desired angular resolution.
- Acoustic coupling.
- DSP optimized functions.
- Sampling Rate.

Each of these factors were addressed and some compromises were made in order to minimize computational complexity and to maximize detection confidence. Some of these factors were determined using simulation and synthetic data, while others were made using real data obtained from a data collect.

<u>Algorithm Complexity.</u> The computational complexity, of beam forming algorithms, ranges from the most efficient (delay and sum) to the most computationally expensive algorithms (constrained optimization). Not surprisingly, the efficiency typically compromises accuracy. The delay and sum technique was chosen both because this is a first attempt and the algorithm is kept simple.

Filtering Requirements. To prevent erroneous/ambiguous direction of arrival (DOA) estimates, the source signal must be filtered. Filtering is performed using a first order Butterworth IIR bandpass filter (combines a first order highpass and lowpass Butterworth) . IIR filters tend to reduce computation due to the lower order required to perform. Additionally, where a second order IIR filter requires five coefficients (3 FIR coefficients and 2 IIR coefficients), an FIR filter with similar frequency response requires 128 coefficients.

Number of Sensors. The number of sensors directly affects the wireless network traffic and the computational cost on the processor. The angular resolution increases and the signal-to-noise ratio decreases with the number of sensors. However, for this project, the number of sensors was limited by the hardware available, and in total, five sensors were used.

Angular Resolution. Angular resolution is the minimum angle at which the source direction can be detected. This is determined by several factors, which include the following:

- Beamforming algorithm.
- Number of sensors.
- Sensor placement (configuration).
- Signal to noise ratio (SNR).
- Array unknowns (Sensor misplacement, channel phase and gain loss).
- Interference signals and multipath.
- Number of Steering vectors.

With the algorithm choice and number of sensor predetermined, the only controllable factors at this point are the sensor placement and number of steering vectors. The algorithm cannot control the remaining factors. The sensor placement is derived as described earlier. The number of steering vectors (180) is chosen based on computation expense and the minimum discernable angle with all other factors in place.

Acoustic Coupling Issues. With truly seismic sources, the wave energy is contained below the surface. A negligible amount of energy propagates into the air to produce an acoustic wave. Acoustic coupling occurs when the source is above ground and produces an acoustic wave. The acoustic energy is coupled into the geophone through the ground in the immediate vicinity of the geophone or through the case in which the geophone is contained. From data collected at the Robotic Vehicle Range, the acoustic power coupled into the geophones for a 25 kW generator at 300 ft from the closest sensor in the array was roughly 2 dB below the seismic wave. This results in an apparent seismic wave traveling at the speed of sound. Typically, acoustic waves are much smaller than seismic waves and may cause spatial aliasing if the array aperture is too large. Spatial aliasing is prevented by keeping the array aperture small. The spacing between the sensors that make up the array is no more than half the wavelength of an acoustic wave at that frequency (minimum velocity is assumed to be 330 m/s). This allows the acoustic wave to be identified and ignored.

DSP optimized functions (6711 DSP Library Function). Some of the most common signal functions/transforms can be calculated efficiently (under certain conditions) using specialized code available for DSPs. Texas Instruments provides a signal processing library which contains C callable functions that are assembly code hand-optimized for the 6711 DSP. The functions used are the radix-2 FFT and the 2nd order IIR Biquad Filter. These functions reduce the computation time by one order of magnitude over the C/C++ equivalent code.

Sampling. The wireless network used to send sensor data to the 6711 processor obviously has a limit on the data transfer rate. This in turn places a limit on the sampling rate. Early in the project, the sampling rate was 1024 samples per second. This was later lowered to 512 samples per second due to the restrictions discussed above. This sampling rate is still well above the Nyquist rate for the sources of interest (less than 100 Hz) and above the rule-of-thumb 5-times the highest frequency of interest.

Assuming that the noise is Gaussian wide-sense-stationary, certain assumptions can be made on the statistical estimators on the data. The beam forming algorithm uses a second order statistic (Correlation). Still, the estimate error of the spatial correlation matrix increases as the number of samples decrease. To remedy this, the algorithm averages several windows of data in the calculation of the spatial correlation matrix.

Algorithm Testing and Validation

The algorithm was tested in three steps. First, the algorithm was written in the form of a Matlab script to simulate the functional blocks of the DSP Algorithm. Using synthetic data, the DSP output was compared with the Matlab output at different points in the algorithm. The IIR filter, FFT and the beam pattern outputs were compared and the errors fell within the numeric precision of the DSP. Next, the Matlab blocks were connected to mimic the algorithm on the DSP. The Matlab and DSP results were compared for various sources in multiple scenarios (for example; low SNR, high acoustic coupling, inaccurate sensor placement, unstable source and various combinations of these). The DSP was able to mimic the Matlab output well within acceptable error. Additionally, this step allowed adjustment of the parameters to maximize accuracy without exceeding computation time. Finally, all parameters were set to their predetermined values and real data was used to test the DSP vs the Matlab. The outputs were remarkably similar. Source DOA detections fell well within minimum performance, given the preconditions described earlier.

Sensor System Testing and Validation

Most of the seismic/acoustic sensor system hardware and software was developed and tested as components, but funding ran out before a full system demonstration could be accomplished. This was probably only a few weeks away. The accomplishments were significant.

1) End-to-end tests were conducted in the laboratory of five C5510 sensor pod processors transmitting data over the 802.11b radios to the central DSP processor.
    a. The C5510 processors for the sensor pods were completed.
    b. The radio firmware was completed, and radio hardware was tested and integrated.
    c. The analog board was designed, completed, and integrated.
    d. The radio hardware and software necessary for transporting signal data from each pod to the central DSP processor was completed and tested.
    e. While real-time transmission from all the pods simultaneously was not accomplished, the project did accomplish collecting near-real-time data from all five sensors, which was time synchronized and processed by the central DSP.
2) An algorithm was implemented in a stand-alone DSP system that performed conventional beam-forming (bearing determination) and additionally identification of an optimal array configuration for the measured signal.
    a. The algorithm was tested and validated in Matlab on a workstation, using data taken in the field from multiple stationary sources and from a generator. The results are that the algorithm gave the correct bearing, and an optimal array configuration was identified.
    b. The algorithm was ported to C++ using UML the Rational Rose Real Time Tool.
    c. The algorithm was subsequently tested and validated on the DSP processor board. The results of the validation test on DSP board consisted of using the same data as had been used on the workstation running Matlab and verifying that the same answers were obtained on the DSP board.
3) The C6711 DSP software design architecture was completed.
    a. It implemented a basic mediator pattern software control architecture.
4) The full system simulator was completed, i.e. the project demonstrated all software and hardware to go from the signal sensing to communicating the results and commands to the robotic simulator.
    a. Signal data was transmitted from the sensor pods to the DSP processor
    b. The beam-forming algorithm performed the bearing estimate and determined the optimal array configuration.
    c. The DSP then sent the messages on moving the sensors to the robot controller simulator.
5) One of the five fully built sensor boxes was completed. The other four would probably required less than a day to complete.

Follow-On Work

The component elements required to demonstrate a fully functional autonomous system to deploy a phased array sensor array, and to autonomously process the signals, determine the optimal array for the *in-situ* conditions, and to redeploy the array into that configuration, were successfully developed under this project. Due to funding constraints the project was unable to demonstrate the fully integrated system, however the project was able to demonstrate the vehicle's ability to deploy and retrieve individual sensors. Follow-on activities needed to achieve the full demonstration originally envisioned include completing assembly of the sensor demonstration boxes that simulate the sensor pods, integrating the DSP into the vehicle, and testing the full system to ensure correct communication from the sensor pods to the DSP and then through the vehicle control system to the Operator Control Unit. Software to pass the vehicle-deployed GPS base station unit's signals through the DSP unit and its radio link also need to be added and tested. Finally, software is needed to autonomously generate waypoint tag files from the deployment location commands generated by the DSP and to run the waypoint tag files. At the conclusion of the project, tag files were generated and run manually. Generating and running the waypoint files could most flexibly be done using the Tcl scripting language that runs the GUI. The project had investigated creating a mission script software engine to run sequences of tag files under the Tcl operating system on the vehicle's control processor rather than on the base station. This requires compiling Tcl along with the C code used by SMART in the controller software. Although this approach is very flexible and practical, we were unable to complete this task with the available funds. Funds also prevented us from fabricating a multi-sensor stack loader for the vehicle, so the current hardware only allows one sensor to be carried and deployed or retrieved at a time.

Despite not demonstrating all the developed capabilities as a cohesive system, the accomplishments that were achieved are very significant and provide the core capabilities needed for substantial follow-on programs and investment. Our ability to rapidly assemble vehicle control code using SMART enables Sandia to respond rapidly and flexibly to new project requirements. We are already benefiting from this ability in other projects. Our ability to use GPS and dead reckoning to very accurately control vehicle position is an important requirement for a wide range of commercial, defense, and government missions. We have used this capability with at least two prospective customers to propose projects. The ability to deploy individual or phased-arrays of sensors and to autonomously analyze the data on the vehicle is also a high-value enabling capability that will enable Sandia to pursue future projects. In summary, this LDRD project has significantly advanced Sandia's technical capabilities in a number of strategically important areas in the mobile robotic and deployed sensor fields.

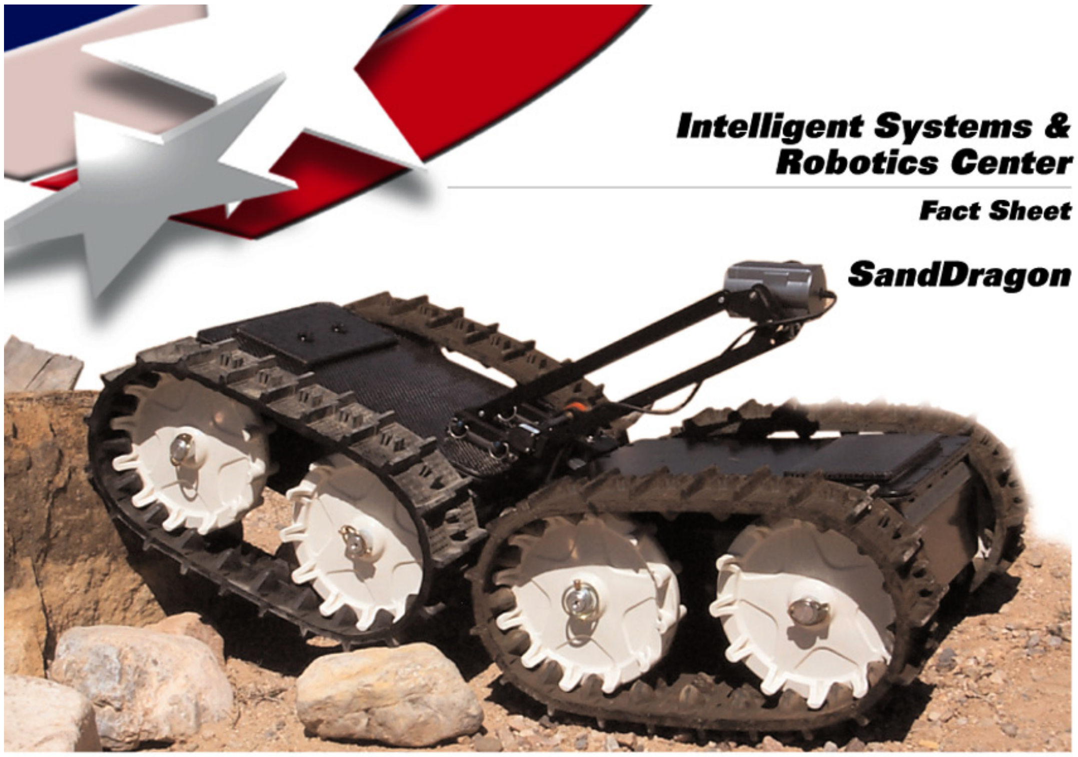
This page intentionally left blank.

Appendix A – SandDragon Vehicle Fact Sheet

Intelligent Systems & Robotics Center

Fact Sheet

SandDragon



Description

SandDragon is a man-portable ground robot developed for the Marine Corps Warfighting Lab (MCWL) as part of the Forward Attack Networked Ground System (FANGS) project. Its mission is to conduct networked surveillance, reconnaissance, target acquisition, and response (lethal and nonlethal) in coordination with other sensors and robots (ground and aerial) in support of Marine Expeditionary Forces (MEF) conducting Operational Maneuvers from the Sea (OMFTS).

The SandDragon has two segmented bodies joined longitudinally by a single axis passive pivot joint. This configuration provides unprecedented mobility for climbing stairs, fording shallow mud and water, crossing aggressive rubble fields, navigating dense foliage, and the potential for swimming. SandDragon's mobility design is derived from the Gemini platform, first developed for the Special Operations Command (SOCOM).

SandDragon uses the patented Sandia Modular Architecture for Robotic Teleoperation (SMART) software suite and toolkit originally developed for the Department of Energy (DOE). SandDragon's Modular Sensor and Component Bus (MSCB) and Local Area Network (LAN), in concert with SMART, enables users to rapidly reconfigure and graphically program new behaviors for arbitrary combinations of sensors, input devices, and vehicle and manipulator kinematics. Potential missions include complex mobile manipulation tasks, communication relay nodes, and engineered collectives.

SandDragon Features

Mobility

- Maximum speed is approximately 4.5 mph on level terrain.
- Traversal through mud and water up to approximately 2 feet deep.
- Weight: Each battery weighs 3.875 pounds, each body segment weighs approximately 30 pounds without batteries.
- Maximum payload is approximately 40 pounds while traversing aggressive terrain and 80 pounds in moderate terrain.
- Operational radio range is approximately 1km in high desert terrain using 3dB omni-directional antennas.

Endurance

- Scalable hot swappable NiMH BB-390A/U battery array (24 or 36 Volts):
  - Can run on 2, 3, 4, or 6 batteries
  - 21.5 Amp Hours of capacity at 36 volts
  - Approximately 5 hour endurance while traversing motocross course terrain at a constant 2 mph speed
  - Approximately12 hours endurance if in a stationary surveillance mode with all electronics energized

Design Architecture

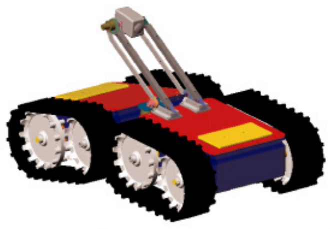
- Vehicle software is the patented Sandia Modular Architecture for Robotic Teleoperation (SMART) using the QNX RTOS v6.0 realtime operating system.
- Modular Component and Sensor Bus (MCSB) for rapid physical insertion and extraction of system devices while minimizing required cabling.
- Ethernet Local Area Network for distributed control, communications, and expansion.
- PC/104+ open architecture with a 233MHz Geode™ 686 Class Pentium CPU Module.
- Color and Near-IR dual mode camera with 18X optical zoom, 4X digital zoom, and IR LED illumination.
- Rail mounted articulated camera mast.
- Carbon fiber body construction.
- 902-928 MHz spread spectrum data transceiver radio.
- 1710-1830 MHz frequency agile video transmitter with digital and audio subcarriers.
- System is capable of supporting manipulation and cooperative behavior extensions.

Operational Control Unit

- Xybernaut 233MHz pentium class PC that can be used with either a Head Mounted Display (HMD) or a daylight readable touch panel display.
- Two NiMH BB-390A/U hot swappable batteries.
- Battery Life is approximately 8 hours.

Additional Features

- Fiber-optic control tether (currently up to 100 m length) for both video, data, and command transmissions, interchangeable with RF relay.
- MultiRAE gas monitoring system
  - Oxygen
  - Combustible gas
  - Carbon monoxide
  - Hydrogen sulfide
  - Sulfur dioxide
  - Nitrogen dioxide
  - Chlorine
  - Ammonia
  - Hydrogen cyanide, phosphine
- Indigo Alpha IR wide field of view imaging system directly interchangeable with visible camera



Contact
William Morse
Phone: 505-845-9696
email: wdmorse@sandia.gov

Appendix B

SandDragon Configuration and Assignment Definition Document

September 15, 2003

1. CPCMC7686GX233HR-128 Real Time Devices Pentium

Table 1 CMC7686 Jumper Settings

Jumper	Setting	Use
J1	2-3	PCI Bus Signal Voltage: 1-2 3.3V; 2-3 5.0V
JP1	Open	Enable/Disable 120ohm termination resistor on COM1 for RS-422/485 mode
JP2	Open	Enable/Disable 120ohm termination resistor on COM1 for RS-422/485 mode
JP5	Open/RESERVED	Factory Use
JP6	Open/RESERVED	Factory Use
B1	RESERVED	Factory Use
B2	RESERVED	Factory Use
B3	RESERVED	Factory Use
B4	RESERVED	Factory Use
B5	RESERVED	Factory Use
B6	1-2	PCI Bus Voltage supply – 1-2 on-board voltage regulator; 2-3 voltage supplied from pins 10 and 12 on P9
B7	2-3	Watchdog time out: Open 600ms, 1-2 150ms, 2-3 1.2 seconds.
B8	RESERVED	Factory Use

Table 2 CMC7686 Watchdog Timer

Watchdog Timer
I/0 Register 0x1E bit 0
Set Bit0 High to Enable and Low to Disable
Timer is reset by reading Register 0x1E
Time out is 1.2 seconds

Table 3 CMC7686 Connectors

Port Name	Connector	Connected To	Address/IRQ	Settings
COM1	P1 (10 Pin)	Freewave Radio	0x3F8/IRQ4	
COM2	P5(10 Pin)		0x2F8/IRQ3	
Parallel	P3 (26 Pin)	Not used	0x378/IRQ7	
Bus Mouse	P6 (4 Pin)	RTD Mouse Cable		Connector plugs in such that the marked white side is to the right toward pin 1
Multifunction	P8 (10 Pin)	Key Board and Reset Button Cable		Connector plugs in such that the marked white side is to the right toward pin 1
Aux Pwr	P9 (8 Pin)	Not used		

Table 4 CMC7686: COM1&2 Dual-row Connecter Pin Definition Facing Head On

9	7	5	3	1
GND	DTR	TXD	RXD	DCD
GND	RI	CTS	RTS	DSR
10	8	6	4	2

Table 5 CMC6786 COM1 Connection Details

COM1 – Connector P1		Freewave COM	
Pin Number	Signal	Signal	Pin Number
1	DCD		
2	DSR		
3	RXD	TXD	5
4	RTS		
5	TXD	RXD	7
6	CTS		
7	DTR		
8	RI		
9	GND	GND	6
10	GND		

Table 6 CMC6786 COM2 RS232 Connection Details

COM1 – Connector P5		FCB-IX470 CN802	
P1 – Pin Number	Signal	Signal	Pin Number
1	DCD		
2	DSR		
3	RXD	TXD	1
4	RTS		
5	TXD	RXD	4
6	CTS		
7	DTR		
8	RI		
9	GND	GND	8
10	GND		

Table 7 CMC7686 Realtime Clock

Real Time Clock Registers Index Register Address 0x070 Data Register Address 0x071		
Registers	Number of Bytes	Function
0x00	1	BCD Seconds
0x02	1	BCD Seconds
0x04	1	BCD Hours
0x06	1	Day of Week
0x07	1	Day of Month
0x08	1	Month
0x09	1	Year
0x0A-0x31	40	RESERVED
0x32	1	BCD Century
0x33-0x3F	13	RESERVED
0x40-0x7F	64-127	User RAM

Table 8 SandDragon I/O Address Map

Real Time Clock Registers		
Address Range	Number of Bytes	Device
000 - 00F	16	DMA Controller
010 - 01F	16	Reserved for CPU
020 – 021	2	Interrupt Controller #1
022 – 02F	13	Reserved
040 – 043	4	Timer
060 – 064	5	Keyboard Interface
070 – 071	2	Real Time Clock Port
080 – 08F	16	DMA page register
0A0 – 0A1	2	Interrupt controller #2
0C0 – 0DF	32	DMA controller #2
0F0 – 0FF	16	Math Coprocessor
1F0 – 1FF	16	Hard Disk
2F8 – 2FF	8	Serial Port
300		Ethernet Card
330 - 333	1/2	GALIL 1240
378 – 37F	8	Parallel Port
3BC – 3BF	4	Parallel Port
3E8 – 3EF	8	Serial Port
3F0 – 3F7	8	Floppy Disk
3F8 – 3FF	8	Serial Port

Table 9 CMC7686GX233 BIOS Settings

Real Time Clock Registers		
Major Field	Field	Selection
Standard CMOS Setup	Hard Disk Primary Master	Auto
	Hard Disk Primary Slave	None
	Hard Disk Secondary Master	Auto
	Hard Disk Secondary Slave	None
	Drive A	1.44MB
	Drive B	None
BIOS Features Setup	Virus Warning	Disabled
	CPU Internal Cache	Enabled
	Quick power on self test	Enabled
	Boot Sequence	A, C, SCSI
	Swap Floppy Drive	Disabled
	Boot up floppy seek	Disabled
	Boot up numlock status	Off
	Gate A20 option	Fast
	Typematic rate setting	Disabled
	Typematic rate	6 char/sec
	Typematic delay	250ms
	Security Option	Setup
	PCI/VGA Pallette Snoop	Disabled
	OS select for DRAM >64 MB	Non OS2
	Report No FDD for Win95	Yes
	BIOS Shadowing	Enabled
	Cyrix 6x86/MII CPUID	Enabled
Chipset Features Setup	16-bit I/O recovery (Clocks)	5
	8-bit I/O recovery (Clocks)	5
Power Management Setup	Power Management	Disabled
	Doze Mode	Disabled
	Standby Mode	Disabled
	HDD Power Down	Disabled
	Modem Use IRQ	NA
	Throttle Duty Cycle	33.3%
	RTC Alarm Function	Disabled
	RTC on by date	NA
	RTC on by time	NA
	IRQ that will bring CPU out of Pwr Mgmt	IRQ1-ON, Others OFF
PNP/PCI Configuration Setup	PNP OS Installed	NO
	Resources controlled by	Auto
	Reset Configuration Data	Disabled
	IRQ assigned to	Level
Integrated Peripherals Setup	IDE Block Mode	Enabled
	Keyboard Controller Input Clock	8 MHz
	Onboard serial port 1	0x3F8/IRQ4
	Mode	RS232
	Onboard serial port 2	0x2F8/IRQ3
	Mode	RS232
	Onboard Parallel Port	0x378/IRQ7
	Parallel Port Mode	ECP + EPP
	ECP mode use DMA	3
	BIOS extension window	Disabled
	FailSafe DOS Boot Up (Orig D000:0000)	Disabled

2. CM202 NE2000 Ethernet utilityModule; Real Time Devices

Table 10 CM202 Software Configured Settings; NE2000 Compatible; PnP Disabled

Option	Setting
I/O Address	0x300
Interrupt	IRQ5
Media	Auto
BPROM	0x0 [No BPROM]

Table 11 CN3 RJ45 Ethernet 10Base-T Pinout (Coax is 10Base-2)

Pin	Signal	Function
1	TX+	Transmit +
2	TX-	Transmit -
3	RX+	Receive +
4	NC	Not Connected
5	NC	Not Connected
6	RX-	Receive -
7	NC	Not Connected
8	NC	Not Connected

Table 12 Diagnostic LEDS

LED	Name	Meaning	Normal State
D1	COL	Collision Detect	OFF
D2	LNK	Link Established (UTP)	On (10 Base T Only)
D3	RXD	Receive From Network	Flashing
D4	TXD	Transmitting to Network	Flashing
D5	POL	Polarity of Signal Incorrect	OFF
D6	PWR	-9 Volt Pwr Present (BNC)	On (10Base-2)

Table 13 SandDragon CPU IP Addresses

Serial No.	IP	Machine Name
4560136	134.253.218.136	SandDragon136
4560137	134.253.218.137	SandDragon137
4560138	134.253.218.138	SandDragon138
4560140	134.253.218.140	SandDragon140
4560142	134.253.218.142	SandDragon142
4560143	134.253.218.143	SandDragon143
4560144	134.253.218.144	SandDragon144
4560145	134.253.218.145	SandDragon145

3. CM112 Super VGA + Flat Panel utilityModule; Real Time Devices

Table 14 CM112 Jumper Settings

Jumper	Setting	Use	Result
JP1	1-2	Floppy Drive Controller	Enabled (address 0x3F0-0x377)
JP2	2-3	IDE Hard Drive Controller	Disabled
JP3	2-3	IRQ9 from VGA controller to Bus	Disabled
JP4	1-2	VGA BIOS	Factory BIOS
JP5	1-2	VGA BIOS Programming	Disables User BIOS programming
JP6	1-2	Back light voltage	+12V
JP7	1-2	Flat Pannel Blank*/DE signal	M signal to Blank*/DE pin
JP8	NC	Factory Use Only	Factory Use Only
JP9	1-2	IDE Drive Primary/Secondary	Primary
JP10	NC	Factory Use Only	Factory Use Only
JP11	NC	Factory Use Only	Factory Use Only

Table 15 CM112 Connectors

Connector	Function	Size
CN1	Floppy Drive	34 Pin
CN2	IDE Hard Drive	40 Pin
CN3	HDD activity LED	2 Pin
CN4	VGA Monitor	10 pin
CN5	Flat Panel	40 pin
CN6	Flat Panel Pwr	10 pin
CN7	Pwr Down Control	4 pin
CN8	Pwr Connector	10 Pin
CN9	PC/104 XT Bus	64 pin
CN10	PC/104 AT Bus	40 pin

4. CM310 Quad Serial Port utilityModule; Realtime Time Devices

Table 16 CM310 Jumper Settings; Boards Silked Screened With IRQ Numbers

Jumper	Pin Numbers/*IRQ*	Function
JP1	3-4 / *4*	IRQ4 for 1st COM (SandDragon COM3)
JP2	1-2 / *3*	IRQ3 for 2nd COM (SandDragon COM4)
JP3	5-6 / *5*	IRQ5 for 3rd COM (SandDragon COM5)
JP4	13-14 / *10*	IRQ10 for 4th COM (SandDragon COM6)
JP5	2-3 / NA	1.8432 MHZ input clock select

Table 17 CM310 Switch Settings

DIP Switch	No.	Position	Function/Setting
Switch 1	1	Down (Closed)	COM3 0x3E8
	2	Down (Closed)	COM4 0x2E8
	3	Down (Closed)	COM5 0x280
	4	Down (Closed)	COM6 0x288
Switch 2	1	Up (Open)	COM3 Enabled
	2	Up (Open)	COM4 Enabled
	3	Up (Open)	COM5 Enabled
	4	Up (Open)	COM6 Enabled
	5	Down (Closed)	COM3 RS232
	6	Down (Closed)	COM4 RS232
	7	Down (Closed)	COM5 RS232
	8	Down (Closed)	COM6 RS232
Switch 3	1	Up (Open)	COM3 RXD No Termination Resistor RS232
	2	Up (Open)	COM3 CTS No Termination Resistor RS232
	3	Up (Open)	COM4 RXD No Termination Resistor RS232
	4	Up (Open)	COM4 CTS No Termination Resistor RS232
Switch 4	1	Up (Open)	COM5 RXD No Termination Resistor RS232
	2	Up (Open)	COM5 CTS No Termination Resistor RS232
	3	Up (Open)	COM6 RXD No Termination Resistor RS232
	4	Up (Open)	COM6 CTS No Termination Resistor RS232

Table 18 CM310 Port 1 (COM3) RS232/485 Signals

CN3-Pin Number	RS232 Signal	RS485 Signal
1	DCD	RTS-
2	DSR	RTS+
3	RXD	RXD-
4	RTS	TXD+
5	TXD	TXD-
6	CTS	RXD+
7	DTR	CTS-
8	RI	CTS+
9,10	GND	GND

Table 19 CM310 Port2 (COM4) RS232/485 Signals

CN4-Pin Number	RS232 Signal	RS485 Signal
1	DCD	RTS-
2	DSR	RTS+
3	RXD	RXD-
4	RTS	TXD+
5	TXD	TXD-
6	CTS	RXD+
7	DTR	CTS-
8	RI	CTS+
9,10	GND	GND

Table 20 CM310 Port3 (COM5) RS232 Signals

CN3-Pin Number	RS232 Signal
1	DCD
2	DSR
3	RXD
4	RTS
5	TXD
6	CTS
7	DTR
8	RI
9,10	GND

Table 21 CM310 Port 4 (COM6) RS232 Signals

CN3-Pin Number	RS232 Signal
1	DCD
2	DSR
3	RXD
4	RTS
5	TXD
6	CTS
7	DTR
8	RI
9,10	GND

5. SandDragon Motherboard

Table 22 SandDragon Motherboard Connectors

Connector	Function	Connected To	Controls
SER1	Serial Connection	CMC7686 PORT1 (COM1) P1	Links to FreeWave Transceiver
SER2	Serial Connection	CMC7686 PORT1 (COM2) P2	Novatel GPS COM2, send differential corrections
SER3	Serial Connection	CM310 PORT1 CN3	FCB-IX470 Camera
SER4	Serial Connection	CM310 PORT2 CN4	HMR3000 compass
SER5	Serial Connection	CM310 PORT2 CN5	Novatel GPS COM1, log and command data
SER6	Serial Connection	CM310 PORT4 CN6	For use with payload DSP processor
P1	Amp Pwr	Port Fwd Motor (X-Axis)	Unregulated Battery
P2	Amp Pwr	Stbd Fwd Motor (Y-Axis)	Unregulated Battery
P5	Amp Pwr	Camera Tilt Motor	Unregulated Battery
P6	Amp Pwr	Camera Mast Motor	Unregulated Battery
J3	Amp Control Lines	Port Fwd Motor (X-Axis)	Galil X-Axis
J4	Amp Control Lines	Stbd Fwd Motor (Y-Axis)	Galil Y-Axis
J16	Amp Control Lines	Camera Tilt Motor	Galil Z-Axis
J17	Amp Control Lines	Camera Mast Motor	Galil W-Axis
J6	Motor Encoders	Port Fwd Motor (X-Axis)	Feedsback to Galil X-Axis
J7	Motor Encoders	Stbd Fwd Motor (Y-Axis)	Galil Y-Axis
J18	Motor Encoders	NC	NC
J19	Motor Encoders	Camera Mast Motor	Galil W-Axis
9,10	GND	GND	

Table 23 SandDragon Motherboard Jumper Settings for setting RS485 or RS232 communications. Note that Jumper 1 is closest to the outer board edge; it is not clearly labeled.

Jumper Series	Jumper	Use	Result
J20	1	Installed	RS232
	2	Removed	
	3	Removed	
	4	Installed	
	5	Installed	
J21	1	Installed	RS232
	2	Removed	
	3	Removed	
	4	Installed	
	5	Installed	

6. DMC1240 4-Axis PC/104 Motion Controller; Galil

I/0 Address: 0x330 or decimal 816
IRQ: None
X-Axis – Port Maxon Drive Motor (Servo with reversed polarity & reversed quadrature)
Y-Axis – Stbd Maxon Drive Motor (Servo with reversed polarity & normal quadrature)
W-Axis – Camera Mast Motor
Z-Axis – Camera Tilt Motor

Table 24 DMC1240 Jumper Settings

Jumper		Setting	Use
JP5	A8	Out	Sets the Memory Address: A2-A5 set the second byte, A6-A8 Set the Firts byte, Third byte is zero. Therefore 011 0011 000, or 0x330 or 816.
	A7	In	
	A6	In	
	A5	Out	
	A4	Out	
	A3	In	
	A2	In	
JP3	IRQ5	Out	Interrupt Request Lines; None Selected
	IRQ9	Out	
	IRQ10	Out	
	IRQ11	Out	
	IRQ12	Out	
	IRQ15	Out	
JP4	UPGRD	Out	MRST, Master Reset Enable. Returns controller to Factory Default settings and erases EEPROM. Requires power-on or RESET active
	MRST	Out	
	FEN	Out	
JP21	OPT	Out	Sets Servo Motor (SM) type. In specifies a stepper motor. Label for each axis SMX, SMY, SMZ, SMW
	SMW	Out	
	SMZ	Out	
	SMY	Out	
	SMX	Out	
	E-H	Out	
JP 7	All	OUT	Factory

Table 25 DMC-1240 Connector J8 (A-D Axes) 50 Pin IDC Interconnection

DMC1240		Motor Amps and Encoders Axis			
Pin	Signal	Signal	Pin	Amp	Encoder
1	Analog Gnd				
2	Ground	-REF_IN	5 3	X,Y Z,W	
3	+5V				
4	Error Output				
5	Reset				
6	Encoder-Compare Output				
7	Ground				
8	Ground				
9	Motor Command D(W)	+REF_IN	1	W	
10	Sign D / Dir D(W)				
11	PWM D / Step D(W)				
12	Motor Cmd C(Z)	+REF_IN	1	Z	
13	Sign C / Dir C(Z)				
14	PWM C / Step C(Z)				
15	Motor Cmd B(Y)	+REF_IN	4	Y	
16	Sign B / Dir B(Y)				
17	PWM B / Step B(Y)				
18	Motor Cmd A(X)	+REF_IN	4	X	
19	Sign A / Dir A(X)				
20	PWM A / Step A(X)				
21	Amp Enable D(W)	12V Control	4	W	
22	Amp Enable C(Z)	12V Control	4	Z	
23	Amp Enable B(Y)	(INHIBIT) IN	11	Y	
24	Amp Enable A(X)	(INHIBIT) IN	11	X	
25	A+ A(X)	Chan_A	3		X
26	A- A(X)				
27	B+ A(X)	Chan_B	1		X
28	B- A(X)				
29	I+ A(X)	Chan_I	4		X
30	I- A(X)				
31	A+ B(Y)	Chan_A	3		Y
32	A- B(Y)				
33	B+ B(Y)	Chan_B	1		Y
34	B- B(Y)				
35	I+ B(Y)	Chan_I	4		Y
36	I- B(Y)				
37	A+ C(Z)	Chan_A	3		Z
38	A- C(Z)				
39	B+ C(Z)	Chan_B	1		Z
40	B- C(Z)				
41	I+ C(Z)	Chan_I	4		Z
42	I- C(Z)				
43	A+ D(W)				
44	A- D(W)				
45	B+ D(W)				
46	B- D(W)				
47	I+ D(W)				
48	I- D(W)				
49	+12V				
50	+12V				

Table 26 DMC-1240 Connector J6 (A-D Axes) 50 Pin IDC Interconnection

DMC1425		Motor Amps and Encoders Axis			Other
Pin	Signal	Signal	Pin	Amp	
1	No Connection				
2	Gnd				
3	+5V				
4	No Connection				
5	Home D(W)				
6	Reverse Limit D(W)				
7	Forward Limit D(W)				
8	Home C(Z)				
9	Reverse Limit C(Z)				
10	Forward Limit C(Z)				
11	Home B(Y)				
12	Reverse Limit B(Y)				
13	Forward Limit B(Y)				
14	Home A(X)				
15	Reverse Limit A(X)				
16	Forward Limit A(X)				
17	Gnd				
18	+5V				
19	No Connection				
20	Latch A(X)	FAULT OUT	14	X	
21	Latch B(Y)	FAULT OUT	14	Y	
22	Latch C(Z)				
23	Latch D(W)				
24	Input 5				
25	Input 6				
26	Input 7				
27	Input 8				
28	Abort				
29	Output 1	Dout_1 HBus			Video TX ON/OFF Control - Radio Not Present
30	Output 2				
31	Output 3				
32	Output 4	Dout_4			Mother Board LED
33	Output 5				
34	Output 6				
35	Output 7				
36	Output 8	Dout_8			DMC1425 Reset Control
37	+5V				
38	Gnd				
39	Gnd				
40	Gnd				
41	Analog Input 1				Temp Sense (MB)
42	Analog Input 2	Current Monitor Out	8	X	
43	Analog Input 3	Current Monitor Out	8	Y	
44	Analog Input 4				Battery Voltage Sense (CB)
45	Analog Input 5				+12V Sense (CB)
46	Analog Input 6				5V Sense (CB)
47	Analog Input 7				-12V Sense (CB)
48	Analog Input 8				
49	-12V				
50	-12V				
		Continuous Current Limit (HB available)	10	X,Y, W,Z	

Table 27 DMC-1240 J7 (A-D Axes) Auxiliary Encoder; 20 Pin IDC

J7		Connection
Pin Number	Signal	
1	+5V	No Connection
2	Gnd	
3	A+ Aux A(X)	
4	A- Aux A(X)	
5	B+ Aux A(X)	
6	B- Aux A(X)	
7	A+ Aux B(Y)	
8	A- Aux B(Y)	
9	B+ Aux B(Y)	
10	B- Aux B(Y)	
11	+5V	
12	Gnd	
13	A+ Aux C(Z)	
14	A- Aux C(Z)	
15	B+ Aux B(Z)	
16	B- Aux B(Z)	
17	A+ Aux D(W)	
18	A- Aux D(W)	
19	B+ Aux D(W)	
20	B- Aux D(W)	

7. DMC1425 2-Axis Ethernet Motion Controller; Galil

X-Axis – Stbd Maxon Drive Motor (Servo with reversed polarity & normal quadrature)
Y-Axis – Port Maxon Drive Motor (Servo with reversed polarity & reversed quadrature)

Table 28 Galil 1425 IP Addresses

Serial No.	IP	Machine Name
DMC-1425-AO260	134.253.218.160	GalilAO260
DMC-1425-AO261	134.253.218.161	GalilAO261
DMC-1425-AO262	134.253.218.162	GalilAO262

Table 29 DMC1425 Board (IP Address 134 253 218 160) J3 General I/O; 37-Pin D-Type

DMC1425		Motor Amps and Encoders				Other
Pin	Signal	Signal	Pin	Amp	Encoder	
1	Reset					Pulled low by 10K Resistor; Pulled high by 1240 Dout8
2	Amp Enable	(INHIBIT) IN (INHIBIT) IN	11 11	X Y		
3	Digital Output 3	A1/MUX				Selects Analog Inputs 1-7
4	Digital Output 1	Dout_1 HBus				Video TX ON/OFF Control
5	Analog Input 1	COMA/MUX				See Mux Tab
6	Main Index (Digital In 7)	Chan_I	4		Y	
7	Reverse Limit Y (Digital In 5)					
8	Digital Input 3					
9	Digital Input 1					
10	+5V	+5V				
11	Ground	Gnd				
12	+12V	+12V				
13	Ground	Gnd				
14	X Encoder A-					
15	X Encoder B-					
16	X Encoder I-					
17	Y Encoder A-					
18	Y Encoder B-					
19	ACMDY	+REF_IN	1	Y		
20	Error					
21	ACMDX	+REF_IN	1	X		
22	Digital Output 2	A0/MUX				Selects Analog Inputs 1-7
23	Circular Compare					
24	Analog Input 2	COMB/MUX				See Mux Tab
25	Home Y (Digital In 6)					
26	Forward Limit Y (Digital In 4)					
27	Digital Input 2 (Y Latch)					
28	Forward Limit X					
29	Reverse Limit X					
30	Home X					
31	-12V	-12V				
32	Encoder A+ X	Chan_A	3		X	
33	Encoder B+ X	Chan_B	1		X	
34	Encoder I+ X	Chan_I	4		X	
35	Encoder A+ Y	Chan_A	3		Y	
36	Encoder B+ Y	Chan_B	1		Y	
37	Encoder I+ Y	Chan_I	4		Y	

Table 30 DMC1425 Jumper Settings

Jumper		Setting	Function
JP1	MR	Removed	IRQ4 for 1st COM (SandDragon COM3)
	UP	Removed	IRQ3 for 2nd COM (SandDragon COM4)
	96	Removed	96 and 12 Removed sets Baud Rate to 19200
	12	Removed	96 and 12 Removed sets Baud Rate to 19200
JP3	MC	Installed	Motor Command
	SD	Removed	Step and Direction

Table 31 DMC1425 Indicator LEDs

LED	Name	Meaning	Normal State
Green	Power	+5V has been applied properly	ON
Red	Status/Error	At least one axis has position error greater than the error limit.	Flashes on initial power up and stays lit for 1-8 seconds. Will illuminate solid red during error conditions. Normal condition is OFF
		Reset line is being held low or is being affected by noise.	
		Controller failure and the processor is resetting.	
		Output IC failure which drives the error signal.	
Green	Link	Lit when there is an Ethernet connection to the controller. Tests only for physical connection, not for active or enabled link.	ON
Yellow	Activity	Indicates traffic across the Ethernet connection. Shows both TX and RX activity.	Flashing during activity. If there is no Ethernet connection or IP address assigned, the LED will flash at regular intervals to show that the BOOTP packets are being broadcast.

Table 32 DMC1425 Analog Input Mutilpexer

A1	A0	Pin	Analog Input	Description	Pin	Axis
0	0	NO1A	2	Current Monitor Out	8	X
0	1	NO4B	1	MB Temperature		
1	0	NO3A	4	Battery Voltage Sense (CB)		
1	1	NO4A	5	+12V Sense (CB)		
0	0	NO1B	3	Current Monitor Out	8	Y
1	0	NO3B	6	5V Sense (CB)		
1	1		7	-12V Sense (CB)		

Table 33 DMC1425 J5 Power; 5(6) Pin Molex

Pin Number	Signal	Signal
1	Motor A	NC
2	Motor B	NC
3	Motor C	NC
4	Ground	GND
5	V+ Input	+12V

Table 34 DMC-1240 EEPROM Setting

Description	Galil Command	X-Axis	Y-Axis	Z-Axis	W-Axis
Proportional Constant	KP	0.0	0.0	6.0	1000.0
Derivative Constant	KD	240.0	240.0	64.0	200.0
Integrator Constant	KI	0.0	0.0	0.0	5.0
Torque Limit	TL	9.998	9.998	9.998	9.998
Off-on-error Function	OE	0	0	0	0
Motor Type (1 = Servo, -1 Servo Reversed Polarity)	MT	-1	1	1	1
Configure Encoder (0 = Normal Quadrature, 2 = Reversed Quadrature)	CE	2	0	0	2
Acceleration	AC	699392	699392	9216	9216
Deceleration	DC	699392	699392	9216	9216
Acceleration Feedforward	FA	0	0	0	0
Velocity Feedforward	FV	0	0	0	0
Integrator Limit	IL	9.9982	9.9982	9.9982	9.9982

Table 35 DMC-1425 EEPROM Setting

Description	Galil Command	X-Axis	Y-Axis
Proportional Constant	KP	27.75	27.75
Derivative Constant	KD	282.38	282.38
Integrator Constant	KI	0.0	0.0
Torque Limit	TL	9.998	9.998
Off-on-error Function	OE	0	0
Motor Type (1 = Servo, -1 Servo Reversed Polarity)	MT	1	-1
Configure Encoder (0 = Normal Quadrature, 2 = Reversed Quadrature)	CE	0	2
Acceleration	AC	699392	699392
Deceleration	DC	699392	699392
Acceleration Feedforward	FA	0	0
Velocity Feedforward	FV	0	0
Integrator Limit	IL	9.9982	9.9982

Galil Board Installation Notes

To talk to the galil code, make sure to disconnect the reset line on the back end so that the rear galil board can boot up!

All galil boards are set at the same address, GalilAO260 - 134.253.218.160, but to get the /etc/hosts file to work correctly the /etc/hosts file needed to be created without any ^M characters (not seen on Windows side!).

This was done by doing the following

```
echo '127.0 localhost' > /etc/hosts
echo '134.253.218.160 GalilAO260' >> /etc/hosts
echo '134.253.218.66 teleopbase' >> /etc/hosts
```

When the system complained about not being able to find a number of services in the inetd.conf file, it was because the /etc/services file had not been installed.

With the SandDragon build it is possible to create local passwords. From the login type

```
Passwd
```

This will then run a script which will create the necessary passwd file.

The system had a number of other complaints:
It wouldnt allow a setsockopt call in syslogd
It needed a valid location in syslog.conf , so I made the entry

```
*.*                                    /home/demos/syslog
```

When finally connected with the updated version, the system kept running and crashing after first failing to connect to the galil board, and then receiving an signal 11 error.

```
#define SIGSEGV     11  /* segmentation violation */
```

This seemed to point to a new coding error. So I recompiled the libraries,
and tried again!

Advance Motion Control Motor Amplifier 25A8

Table 36 25A8 DIP Switch Settings

25A8 Current Mode Operation		
Dip Switch	Signal	Setting
1	Voltage Feedback	OFF (open)
2	Current Integrator	OFF (open)
3	Velocity Integrator	ON (closed)
4	Test/Offset	OFF (open)

Table 37 25A8 Pot Adjustments

25A8 Pot Settings		
Pot	Signal	Setting
1	Loop Gain	Full CCW
2	Current Limit	Full CW
3	Reference In Gain	Full CW
4	Test/Offset	Using a zero offset, Galil Command (OF 0,0) and derivative gain of 0 (KD 0) adjust until all motion is stopped.

Table 38 25A8 Definitions. The Green Phoenix Contact Connector is at the bottom of the Amplifier and the bottom most pin is pin number 5.

Type	No.	Description
Trim Pot	1	Loop Gain
	2	Current Limit
	3	Reference In Gain
	4	Test/Offset Gain
DIP Switch	4	Test/Offset
	3	Velocity Integrator
	2	Current Integrator
	1	Voltage Feedback
MOLEX Connector	1	+5V 3mA Out
	2	Signal Ground
	3	-5V 3mA Out
	4	+REF IN
	5	-REF OUT
	6	- TACH IN
	7	-TACH/GND
	8	Current Monitor Out
	9	Current Reference Out
	10	Continuous Current Limit
	11	(INHIBIT) IN
	12	(+INHIBIT) IN
	13	(-INHIBIT) IN
	14	FAULT OUT
	15	NC
	16	NC
LED	1	GREEN = OK; RED = FAULT
Phoenix Contact Connector	1	- MOTOR
	2	+ MOTOR
	3	Power GND
	4	Power GND
	5	High Voltage

8. Embedded Designs Plus PC104 CompactFlash Boot Drive

Table 39 Embedded Designs Plus PC104 CompactFlash Boot Drive

Jumper Number	Description	Jumper	Condition
S2	Primary/Secondary Master Drive Selection	Removed	Primary Master
S3	LED Bus Activity Indicator	Installed	LEDs Disabled

9. APEX SA60 Compact Motor Driver Card

Table 40 Compact Motor Driver Card Jumper Settings

Jumper	Setting	Function
J1	Installed	
J5	Installed	
J6	1-2 Installed 3-4 Removed	

Table 41 Apex SA60 Interface Connector Definition

	Signal	Label	Pin	Description
Grip Motor (W-Axis)	Analog In	Ain	1	Galil J8 Pin 9 Motor Cmd D(W)
	Digital In	Din	2	
	Signal Gnd	SGnd	3	Galil J8 Pin 2 Gnd
	12 Volt Control of DC/DC Converter	12V Cntl	4	
	Vehicle Raw Voltage	Vveh	5	
	Vehicle Ground	Gnd	6	
Lift Motor (Z-Axis)	Analog In	Ain	1	Galil J8 Pin 12 Motor Cmd C(Z)
	Digital In	Din	2	
	Signal Gnd	SGnd	3	Galil J8 Pin 2 GnD
	12 Volt Control of DC/DC Converter	12V Cntl	4	
	Vehicle Raw Voltage	Vveh	5	
	Vehicle Ground	Gnd	6	

10. FreeWave Digital Radios

Table 42 FreeWave DGRO9RAS Connection Details

Freewave COM		CMD6786 COM1 – Connector P1	
Signal	Pin Number	Pin Number	Signal
B+ Input	1		
Interrupt	2		
DTR	3		
GND	4		
TXD	5	3	RXD
GND	6	9	GND
RXD	7	5	TXD
Carrier Detect	8		
RTS	9		
CTS	10		

Table 43 FreeWave DGRO Front Panel LEDs *MultiPoint* Communications (NA, For information only)

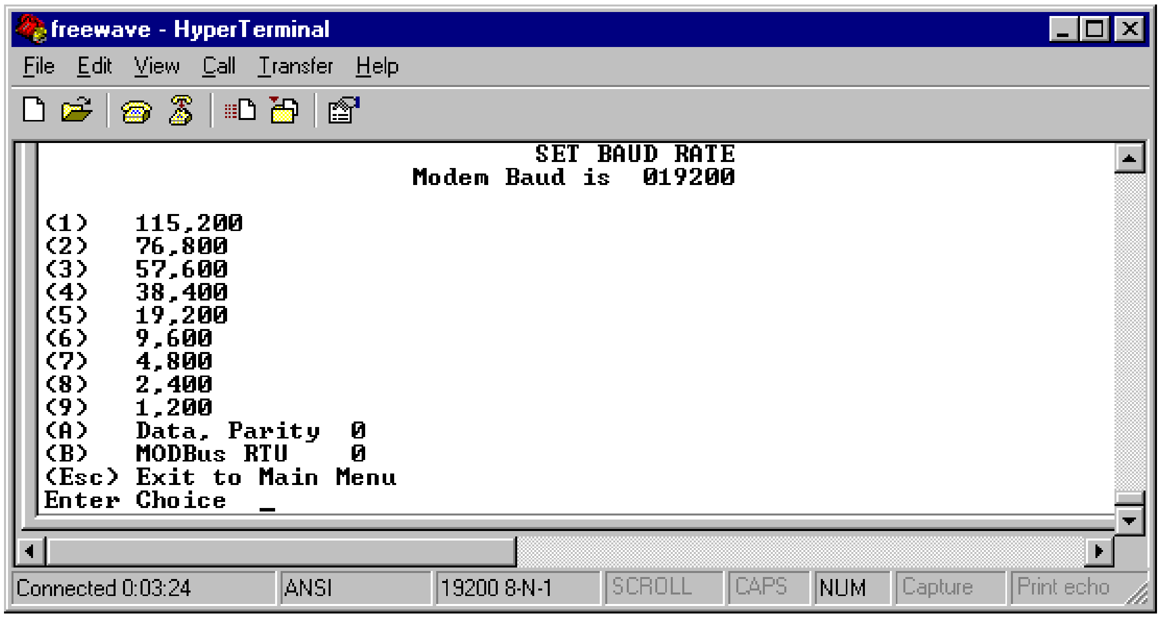
Condition	Master			Slave			Repeater		
	CD	TR	CTS	CD	TR	CTS	CD	TR	CTS
Powered, Disconnected	SR	SD	O	SR	O	BR	SR	O	BR
Repeater and slave connected to master, no data	SR	SD	O	SG	O	SR*	SG	SD	SR*
Repeater and slave connected to master, master sending data to slave	SR	SD	O	SG	O	SR*	SG	SD	SR*
Repeater and slave connected to master, slave sending data to master	SG - SR	SD	IF	SG	IF	SR*	SG	SR	SR*
*CTS will be Solid Red with a solid link, as the link weakens the CTS light on the repeater and slave will begin to flash.									

Table 44 FreeWave DGRO LED Legend

LEGEND	
BR	Blinking Red
FO	Flashing Orange
IF	Intermittent Flash Red
O	Off
SD	Solid Red, Dim
SG	Solid Green
SR	Solid Red, Bright
LED Name	
CD	Carrier Detect LED
CTS	Clear to Send LED
TR	Transmit LED

Table 45 FreeWave DGRO Front Panel LEDs Point-to-Point Communications (NA, For information only)

Condition	Master			Slave			Repeater		
	CD	TR	CTS	CD	TR	CTS	CD	TR	CTS
Powered, Disconnected	SR	SR	SR	SR	O	BR	SR	O	BR
Connected, no repeater, sending sparse data	SG	IF	IF	SG	IF	IF			
Master calling slave through repeater	SR	SD	SR	SR	O	BR	SR	O	BR
Master connected to repeater, not to slave	FO	SD	SR	SR	O	BR	SR	SD	SR
Repeater connected to slave	SG	IF	IF	SG	IF	IF	SG	IF	IF
Mode 6, disconnected	SR	O	BR	SR	O	BR			
Setup Mode	SG	SG	SG	SG	SG	SG	SG	SG	SG

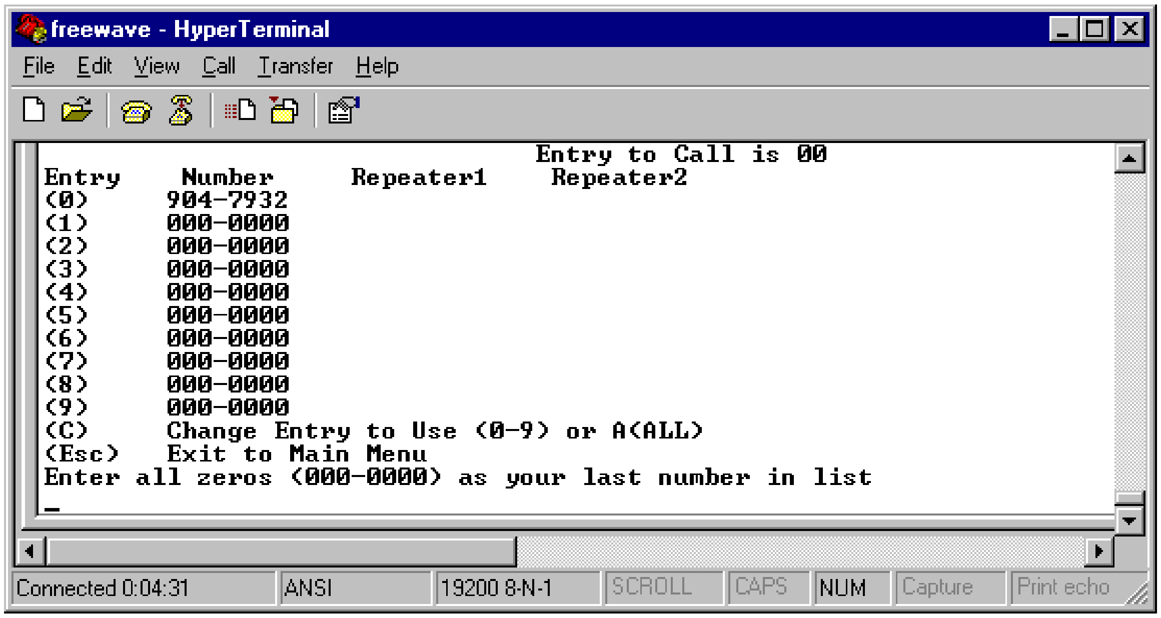


```
                              SET BAUD RATE
                         Modem Baud is  019200

(1)   115,200
(2)   76,800
(3)   57,600
(4)   38,400
(5)   19,200
(6)   9,600
(7)   4,800
(8)   2,400
(9)   1,200
(A)   Data, Parity  0
(B)   MODBus RTU    0
(Esc) Exit to Main Menu
Enter Choice  _
```

```
Connected 0:03:24   ANSI   19200 8-N-1   SCROLL   CAPS   NUM   Capture   Print echo
```

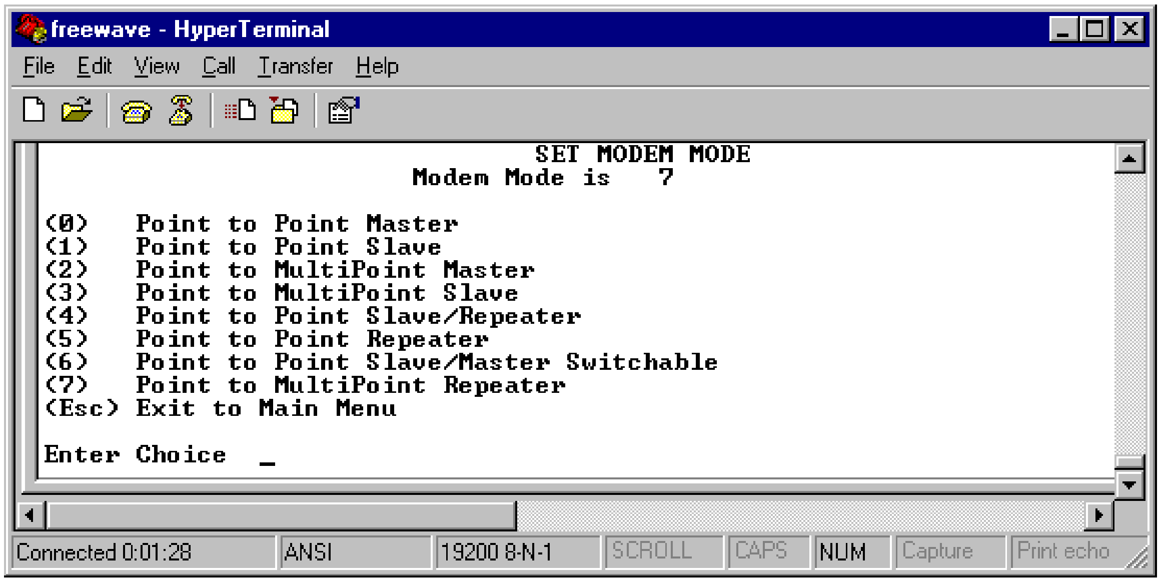
Figure 1



```
                                                  Entry to Call is 00
Entry     Number      Repeater1    Repeater2
(0)       904-7932
(1)       000-0000
(2)       000-0000
(3)       000-0000
(4)       000-0000
(5)       000-0000
(6)       000-0000
(7)       000-0000
(8)       000-0000
(9)       000-0000
(C)       Change Entry to Use (0-9) or A(ALL)
(Esc)     Exit to Main Menu
Enter all zeros (000-0000) as your last number in list
_
```

```
Connected 0:04:31   ANSI   19200 8-N-1   SCROLL   CAPS   NUM   Capture   Print echo
```

Figure 2

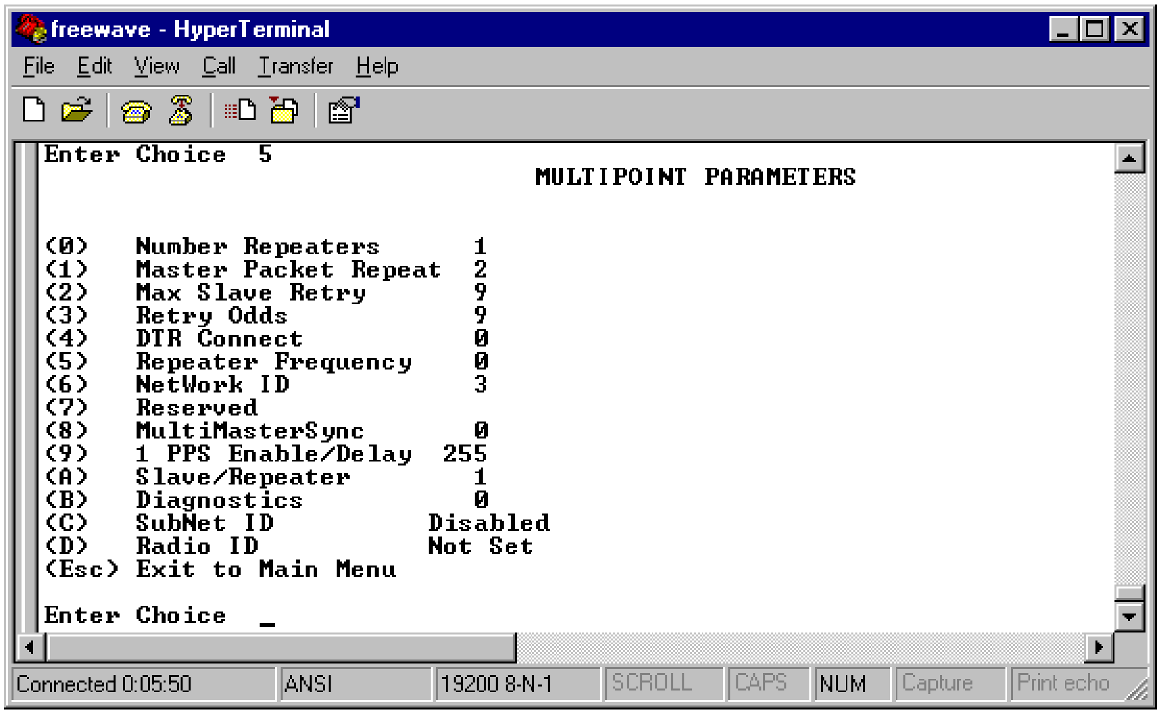


```
                          SET MODEM MODE
                   Modem Mode is    7

(0)    Point to Point Master
(1)    Point to Point Slave
(2)    Point to MultiPoint Master
(3)    Point to MultiPoint Slave
(4)    Point to Point Slave/Repeater
(5)    Point to Point Repeater
(6)    Point to Point Slave/Master Switchable
(7)    Point to MultiPoint Repeater
(Esc)  Exit to Main Menu

Enter Choice  _
```

Figure 3



```
Enter Choice  5
                               MULTIPOINT PARAMETERS


(0)    Number Repeaters        1
(1)    Master Packet Repeat    2
(2)    Max Slave Retry         9
(3)    Retry Odds              9
(4)    DTR Connect             0
(5)    Repeater Frequency      0
(6)    NetWork ID              3
(7)    Reserved
(8)    MultiMasterSync         0
(9)    1 PPS Enable/Delay    255
(A)    Slave/Repeater          1
(B)    Diagnostics             0
(C)    SubNet ID             Disabled
(D)    Radio ID              Not Set
(Esc)  Exit to Main Menu

Enter Choice  _
```

Figure 4

11. Power Distribution Board

Table 46 Pwr Distribution Board LEDs

LED Name	Inidcates
D7	Neg 12V
D6	+5V
D5	+12V

12. Video Transmitter Board

Frequency Agile 1710-1830 MHz

Table 47 Video Transmitter Board Jumper

Jumper	Condition	Result
J2	Removed	High Power TX

Table 48 Video Transmitter Freq Select Dials (from left to right, 0 on top)

Dial	Condition	Result
1	8	1,815 MHz
2	1	
3	5	

13. Camera/Compass Board

Table 49 Camera/Compass Board Jumper

Jumper	Pin	Condition	Result
J1	1	Removed	RS485 Selected when installed
	2	Removed	
	3	Removed	
	4	Removed	

Table 50 Camera/Compass Board Connector

Connector	PIN	Description	Result
J2	1	+5V	
	2	GND	
J3	1	+12V	
	2	GND	
J4	1	+12V	
	2	GND	
J5	1	+5V	
	2	GND	
J7	1	GND	Tied to Motherboard Serial 3
	2	TX	
	3	RX	
J8	1	GND	Tied to Motherboard Serial 4
	2	TX	
	3	RX	
J9	SMA	Video From Camera	
J10	SMA	Video feed to Back Half	
J11	1	12V	
	2	GND	
	3	Video	
	4	GND	
	5	TX	
	6	RX	

14. Software Installation Notes

To install the software on Sanddragon the following steps must be taken.

First, get a 64 MByte SanDisk card and configure it for Neutrino operation following the SMART Neutrino installation notes for a flash target.

```
Use fdisk add qnx 2m
fdisk add qny all
```

Copy over the x86.simple code from teleopbase.

Mount the smart directories and copy over $SMART_HOME/galil/galil_test.exe and $SMART_HOME/demos/sandblock_2.exe to the /home/demos directory.

Edit the /bin/startup script to launch the sandblock_2.exe code.

Create an /etc/hosts entry for the GalilA0260 board.

> To edit the Galil parameters execute the $SMART_HOME/galil/galil_test.exe code using galil_test.exe 1 to talk to the front end, and galil_test.exe 6 to talk to the back end.

The original sand_start script had the following:

```
echo "In /bin/sand_start"
alias rtos="cd /home/smart_nto/rtos"
echo "cd /home/x86/demos"
cd /home/x86/demos
echo "Calling ./sandblock_2.exe"
./sandblock_2.exe
```

The original /etc/hosts file had the following

```
#
# Host database
#
127.1         localhost.localdomain localhost
134.253.218.28 willysolo
134.253.219.98 fangslaptop01
134.253.79.224 vanilla
134.253.218.160 GalilAO260
```

<u>To add the keyboard support:</u> Attach the keyboard cable to port c44 of the CMC7686GX computer board. The white painted side of the berg connector should connect towards the center of the board.

<u>To add the monitor support:</u> Attach a CM112 Utility module to the top of the stack (on top of the CMC768 board). Attach the monitor cable to the CN4 port of the module. Pin 1 (square pin) should correspond to the red wire of the monitor cable.

<u>To add internet support</u>: Detach the keyboard from the back half of the robot and plug it in to one half of the triangular internet coupler (PC end). Then take two Cat 5 cables connected to the hub and plug one back into the ethernet card, and the other to the other end of the coupler.

Appendix C
SandDragon SMART Software
Filter Constant (Configuration) Settings

Target: ocu_rtos **Grid:** teleop & autonav_vehicle **Module:** reg_comm_block

```
#if MODULE_CONFIG == 4
  #define CONFIG_NAME "port5 GPS for AlexBot on OCU"

  #ifdef INCLUDE_HEADER
/**** reg_comm0 parameters ****/
static int reg_comm0_baud = 19200;
static int reg_comm0_port_no = 5;
static char* reg_comm0_recv_name = "gpsdiff1";
static char* reg_comm0_send_name = 0;
static double reg_comm0_wait_time = 0.1;
static CommType reg_comm0_comm_type = COMM_LOCAL_TYPE;
static int reg_comm0_reg_no = -1;
   #endif

   #ifdef INCLUDE_USER_INIT
    /**** REG_COMM_BLOCK definitions for base station for AlexBot ****/
  {
    int target_no;
    target_no = smart_register_get_target_no("sanddragon143");

    /**** get camera switch register and set initial value ****/
    reg_comm0_reg_no = smart_get_register(IS_BINARY_BLK, reg_comm0_recv_name);
    if (target_no < 0) {
      print_msg(fname, MSG_INFO, "No target sanddragon143 recognized\n", 0);
    } else {
      smart_register_set_xmit_flag(reg_comm0_reg_no, target_no);
    }
  }

   #endif
#endif
```

Target: ocu_rtos **Grid:** teleop **Module:** di_gamepad_block

```
/**** defines for Obstacle demo ****/
#if MODULE_CONFIG == 1
  #define CONFIG_NAME "Sanddragon"
  #ifdef INCLUDE_HEADER

  /***** Direct Input Joystick constants for Gravis Gamepad for targetting ****/
  /*** Note 0.2 is the maximum value allowed for RCP PAN1 packets ***/
static Twist di_gamepad0_gain = {{  -2.5f, -2.5f, -0.1f,  0.1f, 0.1f, 0.1f, 0.1f, 0.01f}};
static double di_gamepad0_omega = 10.0;
static int di_gamepad0_deadzone = 80;
static int di_gamepad0_type = 0;
static char *di_gamepad0_reg_name = "di_gamepad_reg";
static int di_gamepad0_mappings[8] = {1, 0, 3, 2, 4, 5, 6, 7};
static Bool di_gamepad0_integrate_dof[8] = {True, True, True, True, True,
        True, True, True};
static int di_gamepad0_zoom_switch_reg = -1;
  #endif

  #ifdef INCLUDE_USER_INIT

              /* ### 10 bits */
              /* bits 9 - 7 = camera ID, value 0 - 7            */
              /* bit 6 = camera power; 0 = off, 1 = on          */
              /* bit 5 = manual iris open; 0 = false, 1 = true  */
              /* bit 4 = manual iris close; 0 = false, 1 = true */
              /* bit 3 = focus out; 0 = false, 1 = true         */
              /* bit 2 = focus in; 0 = false, 1 = true          */
              /* bit 1 = zoom out; 0 = false, 1 = true          */
              /* bit 0 = zoom in; 0 = false, 1 = TRUE           */
  /**** get camera switch register and set initial value ****/
  {
    int target_no;
    target_no = smart_register_get_target_no("sanddragon143");
    di_gamepad0_zoom_switch_reg = smart_get_register(IS_INT, "cam_bits");
    smart_register_set_int(di_gamepad0_zoom_switch_reg, 0x0);

    /**** get camera switch register and set initial value ****/
    if (target_no < 0) {
      print_msg(fname, MSG_INFO, "No target sanddragon143 recognized\n", 0);
    } else {
      smart_register_set_xmit_flag(di_gamepad0_zoom_switch_reg, target_no);
    }
  }
  #endif

    #ifdef INCLUDE_USER_UPDATE
/**** di_gamepad 'Sanddragon' mode ****/
 {
    static int buttons, grid_no;

    /**** direct input update code for "sanddragon" ****/
    buttons = di_gamepad_block_get_buttons(smart_sp(DIPADB0));
    if (buttons & 0x200) {
      smart_module_deactivate(smart_sp(DIPADB0));
    }
```

```
   if (buttons & 0x100) {
     grid_no = smart_module_get_grid_no(smart_sp(DIPADB0));
      if (grid_no >= 0 && !smart_is_grid_activated(grid_no)) smart_module_activate(smart_sp(DIPADB0));
    }

               /**** Set buttons for camera controls ****/
               smart_register_set_int(di_gamepad0_zoom_switch_reg, buttons & 0xf);
  }
  #endif

#endif
```

Target: ocu_rtos **Grid:** teleop **Module:** tank_block

```
#if MODULE_CONFIG == 1
  #define CONFIG_NAME "Sanddragon"

  #ifdef INCLUDE_HEADER
/**** TANK_BLOCK module, Wheel base for Sanddragon ****/
static double tank_block0_d_wb = 18.25 * INCH_TO_METER;
   #endif

#endif
```

Target: ocu_rtos **Grid:** teleop **Module:** reg_output_block

```
#if MODULE_CONFIG == 2
  #define CONFIG_NAME "alexbot base"

  #ifdef INCLUDE_HEADER
/**** test parameters ****/
static char* reg_output0_base_name = "abase";
static char* reg_output0_target_name = "sanddragon143";
   #endif
#endif
```

Target: ocu_rtos **Grid:** teleop& autonav_vehicle **Module:** reg_serial_port

```
#if MODULE_CONFIG == 14
  #define CONFIG_NAME "on teleopbase to alexbot pt 3"

  #ifdef INCLUDE_HEADER
/**** Receive module robande reg ****/

/**** On Wolverine reg_serial0 parameters ****/
static int reg_serial0_baud = 19200;
static int reg_serial0_port_no = 3;
static char* reg_serial0_remote_target_name = "sanddragon143";
static double reg_serial0_min_send_time = 0.1;
static double reg_serial0_max_send_time = 1.0;
static double reg_serial0_id_send_time =  10.0;
static double reg_serial0_full_send_time = 60.0;
static Bool reg_serial0_comm_type = COMM_LOCAL_TYPE;
static Bool reg_serial0_is_server = False;
   #endif

   #ifdef INCLUDE_USER_INIT
  /**** Initialize remote server ****/
  if (smart_server_remote_init(2020, "sanddragon143") == SS_ERROR) {
    print_msg(fname, MSG_WARNING, "Unable to inialize remote server\n", 0);
    return(SS_ERROR);
  };
   #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** reg_serial_port

```
#if MODULE_CONFIG == 13
  #define CONFIG_NAME "on alexbot"

  #ifdef INCLUDE_HEADER
/**** Receive module robande reg ****/

/**** On Wolverine reg_serial0 parameters ****/
static int reg_serial0_baud = 19200;
static int reg_serial0_port_no = 1;
static char* reg_serial0_remote_target_name = "teleopbase";
static double reg_serial0_min_send_time = 0.1;
static double reg_serial0_max_send_time = 1.0;
static double reg_serial0_id_send_time =  20.0;
static double reg_serial0_full_send_time = 60.0;
static Bool reg_serial0_comm_type = COMM_LOCAL_TYPE;
static Bool reg_serial0_is_server = True;
   #endif

#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** reg_comm _block

```
#if MODULE_CONFIG == 3
  #define CONFIG_NAME "port2 19200 On AlexBot"

  #ifdef INCLUDE_HEADER
/**** reg_comm0 parameters ****/
static int reg_comm0_baud = 19200;
static int reg_comm0_port_no = 2;
static char* reg_comm0_recv_name = 0;
static char* reg_comm0_send_name = "gpsdiff1";
static double reg_comm0_wait_time = 0.1;
static CommType reg_comm0_comm_type = COMM_LOCAL_TYPE;

static int reg_comm0_reg_no = -1;
   #endif
   #ifdef INCLUDE_USER_INIT
    /**** REG_COMM_BLOCK definitions for AlexBot ****/
  {
    int target_no;
    target_no = smart_register_get_target_no("teleopbase");

    /**** get camera switch register and set initial value ****/
    reg_comm0_reg_no = smart_get_register(IS_BINARY_BLK, reg_comm0_send_name);
    if (target_no < 0) {
      print_msg(fname, MSG_INFO, "No target teleopbase recognized\n", 0);
    } else {
      smart_register_set_recv_flag(reg_comm0_reg_no, target_no, reg_comm0_send_name);
    }
  }

   #endif
#endif
```

Target: vehicle **Grid:** nav_on_base **Module:** reg_input_block

```
#if MODULE_CONFIG == 8
  #define CONFIG_NAME "Alexbot"

  #ifdef INCLUDE_HEADER
/**** Receive module robande reg ****/

/**** Wolverine base parameters ****/
static char* reg_input0_blk_target_name = "teleopbase";
static char* reg_input0_blk_base_name = "abase";
static double reg_input0_blk_max_delay = 3.0;
static Twist reg_input0_blk_vmax = { 2.0, 2.0, 2.0, 2.0 };
static Twist reg_input0_blk_amax = { 10.0, 10.0, 10.0, 10.0 };
#define DEG5 (5.0*DEG_TO_RAD)
static Twist reg_input0_blk_max_error = {DEG5, DEG5, DEG5, DEG5};
   #endif
#endif
```

Target: vehicle **Grid:** nav_on_base **Module:** sandlizard_block

```
#if MODULE_CONFIG == 2
  #define CONFIG_NAME "Sandlizard with traj on base"

  #ifdef INCLUDE_HEADER
/** Values for SANDLIZARD_BLOCK module **/
/** static const**/ char* base_name = "teleopbase";
static char* sandlizard0_galil_host_name = "GalilAO260";
static char* sandlizard0_DR_travel_reg_name = "DR_travel";
/**static const**/ char*  sandlizard0_io_state_reg = "traj_io_state";
/**static const**/ char*  sandlizard0_io_cmd_reg = "traj_io_cmd";


  #endif

  #ifdef INCLUDE_USER_INIT
   /**** SANDLIZARD_BLOCK definitions for Sandlizard with traj on base ****/

  {
    int io_cmd_reg_no, io_status_reg_no, target_no;

    io_cmd_reg_no = smart_get_register(IS_INT, sandlizard0_io_cmd_reg);
    target_no = reg_get_target_no(base_name);
    if (io_cmd_reg_no == SS_ERROR) {
      print_msg(fname, MSG_WARNING, "io_cmd_reg_no\n", 0);
              } else if (target_no < 0) {
      print_msg_with_string(fname, MSG_INFO, "No target %s recognized\n", base_name);
    } else {
      reg_set_recv_flag(io_cmd_reg_no,  target_no, sandlizard0_io_cmd_reg);
    }

    io_status_reg_no = smart_get_register(IS_INT, sandlizard0_io_state_reg);
    if (io_status_reg_no == SS_ERROR) {
      print_msg(fname, MSG_WARNING, "Can't get data register\n", 0);
              } else if (target_no < 0) {
      print_msg_with_string(fname, MSG_INFO, "No target %s recognized\n", base_name);
    } else {
      reg_set_xmit_flag(io_status_reg_no, target_no);
    }
  }
  #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** sony_visca_port

```
#if MODULE_CONFIG == 5
  #define CONFIG_NAME "Port 3 with RCP zoom"

  #ifdef INCLUDE_HEADER
```

```
/**** SONY_VISCA module (Port 3 camera 1) ****/
static CommType sony_visca0_comm_type = COMM_LOCAL_TYPE;
static int sony_visca0_port_no = 3;
static int sony_visca0_camera_no = 1;
static char* sony_visca0_device_name = NULL;

/*** Application code register (from gamepad!) ***/
static int sony_visca0_switch_reg = -1;
#define ZOOM_IN_BIT 0x01
#define ZOOM_OUT_BIT 0x02
#define FOCUS_NEAR_BIT 0x04
#define FOCUS_FAR_BIT 0x08
#define TOGGLE_FOCUS 0x0c   /** Press 3 & 4 to toggle focus state **/
#define TOGGLE_BACKLIGHT 0x09  /** Press 1 & 4 to toggle backlight **/
#define TOGGLE_LOW_LUX 0x0a  /** Press 2 & 4 to toggle low lux **/
#define TOGGLE_IR 0x06  /** Press 3 & 2 to toggle IR **/
  #endif

  #ifdef INCLUDE_USER_INIT

 /**** Sony VISCA camera definitions ****/
 {
   int target_no;
   target_no = reg_get_target_no("teleopbase");

   /**** get camera switch register and set initial value ****/
   sony_visca0_switch_reg = reg_get(IS_INT, "cam_bits");
   reg_set_int(sony_visca0_switch_reg, 0x0);
   if (target_no < 0) {
     print_msg(fname, MSG_INFO, "No target teleopbase recognized\n", 0);
   } else {
     reg_set_recv_flag(sony_visca0_switch_reg, target_no, "cam_bits");
   }
 }

  #endif

  #ifdef INCLUDE_USER_UPDATE
 /**** Sony VISCA control ****/
 {
   int camera_bits;
   static Bool is_zooming = False;
   static Bool is_focusing = False;
   static int old_camera_bits = 0;

   camera_bits = 0;
   reg_get_int(sony_visca0_switch_reg, &camera_bits);
   camera_bits &= 0xf;
          if (camera_bits != old_camera_bits) {
     if (camera_bits == TOGGLE_FOCUS) {
       if (sony_visca_is_manual_focus(0)) sony_visca_send_command(0, "auto_focus_on");
       else sony_visca_send_command(0, "auto_focus_off");
     }
     if (camera_bits == TOGGLE_IR) {
       if (sony_visca_is_ir_on(0)) sony_visca_send_command(0, "ir_off");
       else sony_visca_send_command(0, "ir_on");
     }
     if (camera_bits == TOGGLE_BACKLIGHT) {
       if (sony_visca_is_backlight_on(0)) sony_visca_send_command(0, "backlight_off");
       else sony_visca_send_command(0, "backlight_on");
     }
     if (camera_bits == TOGGLE_LOW_LUX) {
       if (sony_visca_is_low_lux_on(0)) sony_visca_send_command(0, "low_lux_off");
       else sony_visca_send_command(0, "low_lux_on");
     }
     else if (camera_bits == ZOOM_IN_BIT) {
       sony_visca_send_command(0, "zoom_in");
       is_zooming = True;
     }
           else if (camera_bits == ZOOM_OUT_BIT) {
       sony_visca_send_command(0, "zoom_out");
       is_zooming = True;
     }
     else if ((camera_bits == FOCUS_FAR_BIT) && sony_visca_is_manual_focus(0)) {
       sony_visca_send_command(0, "focus_far");
       is_focusing = True;
     }
     else if ((camera_bits == FOCUS_NEAR_BIT) && sony_visca_is_manual_focus(0)) {
       sony_visca_send_command(0, "focus_near");
       is_focusing = True;
     }
           else {
       if (is_zooming) sony_visca_send_command(0, "stop_zoom");
       if (is_focusing) sony_visca_send_command(0, "stop_focus");
       is_zooming = False;
       is_focusing = False;
     }
     old_camera_bits = camera_bits;
   }
 }

  #endif

#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** novatel_solo_block

```
#if MODULE_CONFIG == 2
  #define CONFIG_NAME "Alexbot port 5"

  #define KEY_WORDS " virtual "

  #ifdef INCLUDE_HEADER
/**** Novatel constants for "RVR site control room" ****/
#include "comm.h"
static CommType novatel0_comm_type = COMM_LOCAL_TYPE;
static int novatel0_port = 5;
```

```
/**** These values are for Albuquerque, RVR site, 6970 Pad ***/
const NovatelLatLongType novatel0_latitude = {35, 02, 28.073976, 'N'};
const NovatelLatLongType novatel0_longitude = {106, 31, 19.592544, 'W'};
const Vector novatel0_scale = {{-25.36, 30.83, 1.0}};
const double novatel0_altitude = 1667.289;
const char* novatel0_pos_reg_name = "gps_pos";
const char* novatel0_gps_home_reg_name = "gps_home";
const char* novatel0_gps_base_reg_name = "gps_base";
const char* novatel0_gps_pos_decimal_reg_name = "gps_pos_dec";
const char* novatel0_gps_pos_sats_reg_name = "gps_sats";
const char* novatel0_gps_std_dev_reg_name = "gps_std_dev";
const char* novatel0_gps_rdg_time_reg_name = "gps_rdg_time";
const char* novatel0_gps_rtk_stat_reg_name = "gps_rtk_stat";
const char* novatel0_gps_base_log_reg_name = "gps_base_log";
const Bool novatel0_is_upload = True;
  #endif

#endif
```

Target: vehicle **Grid:** autonav_on_veh **Module:** sandlizard_block

```
#if MODULE_CONFIG == 1
  #define CONFIG_NAME "Sandlizard with traj onboard"

  #ifdef INCLUDE_HEADER
/** Values for SANDLIZARD_BLOCK module **/
static char* sandlizard0_galil_host_name = "GalilAO260";
static char* sandlizard0_DR_travel_reg_name = "DR_travel";
static const char*  sandlizard0_io_state_reg = "traj_io_state";
static const char*  sandlizard0_io_cmd_reg = "traj_io_cmd";


  #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** hmr3000_port

```
#if MODULE_CONFIG == 2
  #define CONFIG_NAME "Port 4 hmr_rpy"

  #ifdef INCLUDE_HEADER

/**** Honeywell HMR3000 Compass port 4 ****/
#include "comm.h"
static int hmr30000_port_no = 4;
static char *hmr30000_rpy_reg_name = "hmr_rpy";
static char *hmr30000_status_reg_name = "compass_status";
static Bool hmr30000_is_upload = True;

  #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** pos_estim_block

```
#if MODULE_CONFIG == 1
  #define CONFIG_NAME "Sandlizard with gps hmr compass"

  #define KEY_WORDS " virtual "

  #ifdef INCLUDE_HEADER
/*** values for POS_ESTIM_BLOCK ***/
static char* gps_pos_reg_name = "gps_pos";
static char* gps_std_dev_reg_name = "gps_std_dev";
static char* gps_rdg_time_reg_name = "gps_rdg_time";
static char* DR_travel_reg_name = "DR_travel";
static char* rpy_reg_name = "hmr_rpy";
static char* robot_pos_reg_name = "robot_pos_x";
static char* DR_pos_reg_name = "DR_pos";
static Bool is_upload = True;

  #endif
#endif
```

Target: vehicle **Grid:** autonav_on_veh **Module:** pos_offset_block

```
#if MODULE_CONFIG == 4
  #define CONFIG_NAME "alexbotv"

  #ifdef INCLUDE_HEADER

static double pos_offset_block0_dx = (28.0 * INCH_TO_METER); /** Distance from gps antenna to hitch
**/
static double pos_offset_block0_dy = (0.0 * INCH_TO_METER); /** Same in y, positive to left **/
static const char* pos_offset_block0_reg_name  = "hmr_rpy";
static const SmartRegisterType pos_offset_block0_reg_type = IS_VECTOR;
static int pos_offset_block0_heading_map_type = 1;  /** Use compass heading **/
  #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** traj_io_block

```
#if MODULE_CONFIG == 2
  #define CONFIG_NAME "Sanddragonv"

  #ifdef INCLUDE_HEADER
/**** TRAJ_IO_BLOCK module,  ****/

/**** traj_io_block parameters ****/
static /**const**/ char* robot_name = "sanddragon143";
static /**const**/ char*  traj_io_block_state_reg = "traj_io_state";
static /**const**/ char*  traj_io_block_cmd_reg = "traj_io_cmd";
static /**const**/ char*  traj_io_block_tag_info_reg = "traj_tag_info";
static const Twist traj_io_block_vmax = {{2.0f, 2.0f, 0.2f, 3.9f, 4.16f, 3.93f}};
static const Twist traj_io_block_amax = {{2.0f, 2.0f, 18.7f, 69.0f, 74.0f, 70.0f}};
static const Real traj_io_block_percent_speed = 0.5f;
  #endif
  #ifdef INCLUDE_USER_INIT
   /**** TRAJ_IO_BLOCK definitions for Sanddragonv ****/
  {
    int io_cmd_reg_no, io_state_reg_no, io_tag_info_reg_no, target_no;

    io_cmd_reg_no = smart_get_register(IS_INT, traj_io_block_cmd_reg);
    target_no = reg_get_target_no(robot_name);
    if (io_cmd_reg_no == SS_ERROR) {
      print_msg(fname, MSG_WARNING, "io_cmd_reg_no\n", 0);
            } else if (target_no < 0) {
      print_msg_with_string(fname, MSG_INFO, "No target %s recognized\n",
robot_name);
    } else {
      reg_set_xmit_flag(io_cmd_reg_no,  target_no);
    }

    io_state_reg_no = smart_get_register(IS_INT, traj_io_block_state_reg);
    if (io_state_reg_no == SS_ERROR) {
      print_msg(fname, MSG_WARNING, "Can't get state register\n", 0);
            } else if (target_no < 0) {
      print_msg_with_string(fname, MSG_INFO, "No target %s recognized\n",
robot_name);
    } else {
      reg_set_recv_flag(io_state_reg_no, target_no, traj_io_block_state_reg);
    }
    io_tag_info_reg_no = smart_get_register(IS_STRING, traj_io_block_tag_info_reg);
    if (io_tag_info_reg_no == SS_ERROR) {
      print_msg(fname, MSG_WARNING, "Can't get tag_info register\n", 0);
            } else {
      reg_queue_for_upload(io_tag_info_reg_no);
    }
  }
  #endif

#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** pos_servo_block

```
#if MODULE_CONFIG == 2
  #define CONFIG_NAME "low speed tankpos"

  #ifdef INCLUDE_HEADER
/**** pos_servo0 parameters for tankpos  ****/
   static double pos_servo0_alpha = 0.5;
   static Twist pos_servo0_vmax = {{1.0, 1.0, 1.0, 10.0, 10.0, 10.0, 10.0, 10.0}};
  #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** trailer_block

```
#if MODULE_CONFIG == 3
  #define CONFIG_NAME "alexbot"

  #ifdef INCLUDE_HEADER

static double trailer_block0_d_wb = (22.0 * INCH_TO_METER); /** spacing of treads
**/
static double trailer_block0_d_lh = (25.0 * INCH_TO_METER); /** vehicle center to
hitch **/
static const char* trailer_block0_reg_name  = "hmr_rpy";
static const SmartRegisterType trailer_block0_reg_type = IS_VECTOR;
static const int trailer_block0_heading_map_type = 1; /** Use compass for heading **/
   #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** reg_retrieve_block

```
#if MODULE_CONFIG == 2
  #define CONFIG_NAME "armpos"
  #ifdef INCLUDE_HEADER
/****  parameters for REG_RETRIEVE ****/
static char* reg_retrieve0_base_name = "armpos";
static SmartRegisterType reg_retrieve0_reg_type = IS_VECTOR;
   #endif
#endif
```

Target: vehicle **Grid:** nav_on_base & autonav_on_veh **Module:** merge_block

```
#if MODULE_CONFIG == 3
  #define CONFIG_NAME "prev 0 1 bot 2 3"

  #ifdef INCLUDE_HEADER
/**** merge0 parameters prev 0 1 bot 2 3 ****/
static int merge0_prev_map[MAX_DOFS] = {0, 1};
static int merge0_bottom_map[MAX_DOFS] = {2, 3};
  #endif
#endif
```

Target: vehicle **Grid:** autonav_on_veh **Module:** snsr_dsp_block

```
#if MODULE_CONFIG == 1
  #define CONFIG_NAME "Alexbot"

  #ifdef INCLUDE_HEADER
/**** parameters for SNSR_DSP  module ****/
#include "comm.h"
static CommType snsr_dsp0_comm_type = COMM_LOCAL_TYPE;
static int snsr_dsp0_port = 6;

static char *snsr_dsp0_diff_reg_name = "gps_diff";
static char *snsr_dsp0_log_reg_name = "gps_base_log";
static char *snsr_dsp0_dsp_msg_reg_name = "dsp_msg";
static char *snsr_dsp0_gps_cmd_reg_name = "gps_cmd";
static Bool snsr_dsp0_is_upload = True;

  #endif
#endif
```

This page intentionally left blank

Appendix D

SandDragon Module Registered Commands

All the available commands can be viewed using the command 'info' at the console prompt. There are numerous commands that are part of the standard SMART core modules, and they will not be detailed here. A problem occurs when running the vehicle remotely and there isn't a console available on a display. If a command is not instantiated through the GUI, it is still possible to send it to the vehicle over the reg_serial communication link from the remote (base station) console using the command:

```
reg_serial_send_cmd <string>
```

where the string contains the command name, the argument type(s), the return type, and the argument value(s). For example, to send the command "gps_lat_long" which takes the smart module type as an argument (type number 0x1, value 5) and returns a string (type 0x4), use the command:

```
reg_serial_send_cmd "gps_lat_long 0x1 0x4 {5}"
```

A command which takes multiple arguments looks like:

```
reg_serial_send_cmd "gps_prtk_repeat_pos 0x22 0x2 {0 10}"
```

The commands specific to the modules for this project are shown below.

Module: novatel

Cmd: gps_prtk_pos <SMART module number>
Use: Remote call returning string with GPS status data
Arg Type: 0x1
Return Type: 0x4
Calls: novatel_solo_get_pos_string
Returns: String:
NSVS %d SDEV %.3f x %.3f y %.3f alt %.3f wk %d sec %.3f age %.3f\n
Containing
NSVS: number of satellites,
SDEV: sqrt of sum of squares of lat and long standard deviation of position error in meters.
x: longitudinal distance from HOME in meters (East is positive)
y: latitudinal distance from HOME in meters (North is positive)
alt: altitude above HOME
wk: GPS week
sec: GPS second of the week

age: Age of the differential correction in seconds

Cmd: gps_prtk_repeat_pos <instantiation number (normally 0)> <# loops>
Use: Multiple printouts of GPS status to Console
Arg Type: 0x22
Return Type: 0x2
Calls: novatel_gps_prtk_pos_display
Returns: Status: SS_OK or SS_ERROR
Action: Prints to console
NSVS: number of satellites,
SDEV: sqrt of sum of squares of lat and long standard deviation of position error in meters.
x: longitudinal distance from HOME in meters (East is positive)
y: latitudinal distance from HOME in meters (North is positive)
alt: altitude above HOME
wk: GPS week
sec: GPS second of the week
age: Age of the differential correction in seconds

Cmd: gps_lat_long <SMART module number>
Use: Returns string with lat/long of vehicle position in deg:min:sec
Arg Type: 0x1
Return Type: 0x4
Calls: novatel_solo_get_lat_long
Returns: string

Cmd: gps_get_home <SMART module number>
Use: Returns string with lat/long of HOME in deg:min:sec
Arg Type: 0x1
Return Type: 0x4
Calls: novatel_solo_get_home_string
Returns: string

Cmd: gps_home_set_prtk <instantiation number (normally 0)> <lon_offset_in_m> <lat_offset_in_m>
Use: Sets HOME to the current GPS position and adds the offset in m
Arg Type: 0x233
Return Type: 0x2
Calls: novatel_gps_home_set_prtk
Returns: Status: SS_OK or SS_ERROR

Cmd: gps_home_set_lon <instantiation number (normally 0)> <lon_degr> <lon_min> <lon_sec>
Use: Sets HOME longitude to the specified deg:min:sec. Negative degree is W.
Arg Type: 0x2333
Return Type: 0x2

Calls: novatel_gps_home_set_lon
Returns: Status: SS_OK or SS_ERROR

Cmd: gps_home_set_lat <instantiation number (normally 0)> <lat_degr> <lat_min> <lat_sec>
Use: Sets HOME latitude to the specified deg:min:sec. Negative degree is S.
Arg Type: 0x2333
Return Type: 0x2
Calls: novatel_gps_home_set_lat
Returns: Status: SS_OK or SS_ERROR

Cmd: gps_home_set_elev <instantiation number (normally 0)> <alt_m>
Use: Sets HOME elevation to the specified height in m.
Arg Type: 0x23
Return Type: 0x2
Calls: novatel_gps_home_set_elev
Returns: Status: SS_OK or SS_ERROR

Cmd: gps_config_file_download <instantiation number (normally 0)> <filename>
Use: Sends the named file to the GPS. The file needs to already be on the vehicle.
Arg Type: 0x24
Return Type: 0x2
Calls: novatel_config_file_download
Returns: Status: SS_OK or SS_ERROR

Cmd: gps_config_cmd_download <instantiation number (normally 0)> <command>
Use: Sends the command to the GPS.
Arg Type: 0x24
Return Type: 0x2
Calls: novatel_config_cmd_download
Returns: Status: SS_OK or SS_ERROR

Module: sandlizard

Cmd: sdb_print
Use: Prints the sandlizard_io structure to the console
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_print
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_reset
Use: Turns off the motion, clears the galil buffer, and resets the rear galil board.
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_reset

Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_set_volt_max <max voltage>
Use: Sets the max voltage value that is sent by the Galil motion controller for the track drive motors. Output values range from 0 to 10V.
Arg Type: 0x3
Return Type: 0x2
Calls: sandlizard_io_set_voltage_max
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_reset_pos
Use: Zeros the position encoders for the left and right tracks and the lift and grip encoders.
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_reset_pos
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_stop
Use: Sends a stop command to the Galil motion controllers, and sets the speed parameters to zero.
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_stop
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_set_speed <left speed> <right speed> <lift speed> <grip speed>
Use: Sets the requested speed parameters for the left, right, lift, and grip motors to the indicated value (by calling sandlizard_io_assign_speed), and sends the values to the Galil motion controllers. Left and right are in m/sec, lift and grip in % of travel range per second.
Arg Type: 0x3333
Return Type: 0x2
Calls: sandlizard_io_set_speed
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_assign_speed <left speed> <right speed> <lift speed> <grip speed>
Use: Sets the requested speed parameters for the left, right, lift, and grip motors to the indicated value. Left and right are in m/sec, lift and grip in % of travel range per second.
Arg Type: 0x3333
Return Type: 0x2
Calls: sandlizard_io_assign_speed
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_motion_on
Use: Sends 'servo_here' command to the Galil motion controllers to turn on and servo to the present location .

Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_motion_on
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_motion_off
Use: Sets the velocity command parameters to zero, and sends a zero “offset” command to the Galil motion controllers followed by a ‘stop’ command and a ‘motor_off’ command.
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_motion_off
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_get_speed
Use: Reads the speed from the Galil motion encoders and puts them in the local structure variables.
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_get_speed
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_get_pos
Use: Reads the position from the Galil motion encoders and puts them in the local structure variables.
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_get_pos
Returns: Status: SS_OK or SS_ERROR

Cmd: sdb_zero_l_and_g
Use: Drives lift and grip motors to their up and open positions, and zeros their counters.
Arg Type: 0x0
Return Type: 0x2
Calls: sandlizard_io_zero_lift_grip
Returns: Status: SS_OK or SS_ERROR

Module: pos_estim

Cmd: robot_pos <SMART module number>
Use: Returns string containing robot distance from HOME in meters in longitude, latitude, elevation directions, and also includes reading time.
Arg Type: 0x1
Return Type: 0x4
Calls: pos_estim_block_get_pos_string
Returns: String containing text and data

Cmd: robot_repeat_pos <instantiation number (normally 0)> <# times to print>
Use: Prints out pos_estim structure the indicated number of sequential times.
Arg Type: 0x22
Return Type: 0x2
Calls: pos_estim_pos_display
Returns: Status: SS_OK or SS_ERROR

Cmd: pos_estim_zero_dr_pos <instantiation number (normally 0)> <offset long> <offset lat>
Use: Sets the dead reckoning position to the offset value indicated.
Arg Type: 0x233
Return Type: 0x2
Calls: pos_estim_zero_dr_pos
Returns: Status: SS_OK or SS_ERROR

Module: snsr_dsp

Cmd: gps_base_cmd_send <instantiation number (normally 0)> <text msg>
Use: Packages the msg with the necessary header and footer and sends to the DSP to pass on to the GPS base unit.
Arg Type: 0x24
Return Type: 0x2
Calls: snsr_dsp_gps_cmd_recv
Returns: Status: SS_OK or SS_ERROR

Appendix E
Adaptive and Mobile Ground Sensors LDRD DSP/PC104 Communication Protocol Interface Document

Introduction
This interface document discusses the protocol and message sequencing to be used between the SandDragon PC104 stack processor and the payload DSP processor as part of the AMGS LDRD. A sequence of messages and acknowledging handshakes are required to be passed back and forth as part of the planned operation. Communication will be over a serial communication line (COM line) at a baud rate to be agreed upon (9600 or 19200 most likely).

Packaging:
Each message string will be packaged with start sequence consisting of four ASCII dollar sign characters ($$$$), a message type integer character, data characters specific to that message type, and a CRC (cyclic redundancy check) sequence. The CRC will fill two bytes of a four byte integer value. Some messages will be of known and fixed length, while others will necessarily be of variable length. The variable length messages will have as its first data integer byte the length of the full message, including the CRC value but excluding the initial preamble '$$$$' characters and the message type integer.

Messages and Sequences
The following 10 message types (shown with their integer type representation) are passed between the processors. Each message has a sequence of data sent with it, shown in the subsequent header file.

SEND	**1**
GPS_DIFF	**2**
GPS_CMD	**3**
GPS_LOG	**4**
NEW_ARRAY	**5**
ACK	**6**
SENSOR_ADD	**7**
SENSOR_STATUS	**8**
ARRAY_DONE	**9**
RESEND	**10**

The sequence used in communicating these commands is given by the following exchange information.

Command	Arguments	Usage
Sent by DSP Unit to PC104		
SEND	unsigned char "$$$$"	Alpha/Numeric message of up to 128 characters to be sent on to the base station. (The PC104 puts it into a register which gets passed to the OCU and displayed.) PC104 will reply with ACK msg.
	unsigned long *msgType*	
	unsigned long *length*	
	unsigned char[128] *msg*	
	unsigned long *checksum*	
GPS_DIFF	unsigned char "$$$$"	The Novatel gps unit sends a variable-length binary string of up to 250 characters. This needs to be packaged by the sensor processor and sent along with gps ASCII data to the DSP which forwards it along to the PC104. When it gets to the DSP it will have been packaged by the sensor processor with a leading length byte, so it just needs to be additionally wrapped by the DSP and passed on to the PC104. (The PC104 puts it into a register for the Reg_Comm module to forward to the GPS.) No ACK msg will be sent in reply.
	unsigned long msgType	
	unsigned long length	
	unsigned char[250] msg	
	unsigned long checksum	
GPS_LOG	unsigned char "$$$$"	The Novatel gps unit sends a variable-length log file which is already packaged in a NEMA package. The DSP will further package it in the presently described format and pass it to the PC104. The PC104 puts this into a register, which is parsed by the GPS module to extract the base position, which is put into another register and passed to the OCU. No ACK msg will be sent in reply.
	unsigned long *msgType*	
	unsigned long *length*	
	unsigned char[128] *msg*	
	unsigned long *checksum*	
Sent by PC104 Unit to the DSP		
GPS_CMD	unsigned char "$$$$"	Commands can be sent from the OCU through the PC104 (via a register) to the DSP to be passed on to the sensor processor which sends it to the base station GPS unit. This command is already packaged and just needs to be passed on. No ACK msg will be sent in reply.
	unsigned long *msgType*	
	unsigned long *length*	
	unsigned char[128] *msg*	
	unsigned long *checksum*	

Deploy/Retrieve Exchange Commands/Data:

Category	Arguments	Usage

<Vehicle drives to deployment site using mission script.>
<Vehicle powers up the DSP, which sends its first array configuration to the PC104.>

NEWARRAY	unsigned char "$$$$"	The DSP decides the configuration and
	unsigned long *msgType*	number of sensors. It sends this number and
	long int *num_sensors*	the "style," which indicates what type of
	*unsigned long *style*	array this will be. This is used by the PC104
	unsigned long *checksum*	to determine whether to deploy the array or
		if it is the initial list of what it is carrying.
		The first NEWA list will be style 1, what is
		on the vehicle. The second will be style 3,
		what is to be deployed. The vehicle
		determines what sensor goes where. The
		PC104 responds with an ACK msg that
		includes the index number assigned by the
		PC104, for use in future references.
	* *Style (not implemented):*	
		EMPTY = 0, No sensors in the array
		VEHICLE = 1, Stack loader on the vehicle
		is an array
		STORE = 2, Used to hold inactive sensors
		DEPLOY = 3, Used to hold deployed array
		of sensors
ACK	unsigned char "$$$$"	
	unsigned long *msgType*	
	unsigned long *array_index*	
	unsigned long *checksum*	
RESEND	unsigned char "$$$$"	If the CRC checksum doesn't match, this is
	unsigned long *msgType*	sent (by either the PC104 or the DSP)
	unsigned long *checksum*	instead of the ACK.

<The DSP then sends the *num_sensors* number of SENSOR_ADD messages>

SENSOR_ADD		
		unsigned char "$$$$" The DSP
		sends this info for each sensor.
	unsigned long *msgType*	The stimulus tells if stimulation is to be
	unsigned long snr*_index*	(thumping the ground) before moving on.
	float *x_pos_req*	X and Y location are all relative. The PC104
	float *y_pos_req*	can place the array anywhere locally as long
	*float *circular_error*	as it preserves the relative position of the

*unsigned long *stimulus*	sensors. The PC104 sends an ACK msg with
unsigned long *checksum*	the *sensor_index* number when it receives a msg.
	* These two parameters unimplemented

<Vehicle proceeds to set up array when it can. It puts a sensor in an array and if array was marked for thump testing, calls DSP before sending out a thump. At present we have not defined the interface for stimulus testing. Once the sensor is place, the PC104 sends the sensor status:>

SENSOR_STATUS

unsigned char "$$$$"
unsigned long msgType The PC104 sends information on each
unsigned long *list_index* sensor as it is
placed. The DSP responds
unsigned long *snr_ID* with an ACK
that includes the *list_index*.
float *x_pos_act*
float *y_pos_act*
unsigned long *checkSum*

<This get repeated for each sensor until the array is completed>
<Then the vehicle sends the array data>

ARRAY_DONE

	unsigned char "$$$$"
unsigned long *msgType*	The PC104 sends this to tell the DSP the
unsigned long *checksum*	array is in place and the vehicle in parked so that monitoring can begin. The DSP responds with an ACK msg that includes the *array_index*..

Distribution:

1	MS0767	Marshall, Billy (04110)
1	MS1003	Spletzer, Barry (15221)
1	MS0859	Stalker, Terry (15351)
1	MS0491	Hoffman, John (12345)
1	MS1004	Harrigan, Ray (15221)
1	MS1003	Feddema, John (15211)
1	MS1079	Scott, Marion (07700)
1	MS1007	Shipers, Larry (15272)
1	MS1125	Bennett, Phil (15252)
1	MS1219	Taylor, John (05907)
1	MS1219	Craft, Charlie (05941)
1	MS0859	Williams, Robert (15351)
1	MS1003	Lewis, Chris (15211)
1	MS0870	Nguyen, Hung (15351)
1	MS0519	Holzrichter, Michael (02348)
1	MS1003	Carlson, Jeff (15211)
1	MS0986	Gallegos, Daniel (02664)
1	MS0196	O’Rourke, William (02666)
1	MS0196	Zenner, Jennifer (02666)
1	MS1125	Maish, Alex (15252)
1	MS1126	Anderson, Robert (15252)
1	MS1125	Hayward, David (15252)
1	MS1125	Hobart, Clint (15252)
1	MS1125	Weber, Tom (15252)
1	MS0323	LDRD Office (01011)
1	MS9018	Central Technical File (8945-1)
2	MS0899	Technical Library, 9616