Army Research Laboratory

A Platform for Developing Autonomy Technologies for Small Military Robots

by Gary Haas, Jason Owens, and Jim Spangler

ARL-MR-709 December 2008

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-MR-709 **December 2008**

A Platform for Developing Autonomy Technologies for Small Military Robots

Gary Haas, Jason Owens, and Jim Spangler
Vehicle Technology Directorate, ARL

REPORT DOCUMENTATION PAGE		***Form Approved* *OMB No. 0704-0188***

1. REPORT DATE *(DD-MM-YYYY)* December 2008	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** January 2008–August 2008
4. TITLE AND SUBTITLE A Platform for Developing Autonomy Technologies for Small Military Robots		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Gary Haas, Jason Owens, and Jim Spangler		**5d. PROJECT NUMBER** 1L162618AH80
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory, ATTN: AMSRD-ARL-VT-UV, Aberdeen Proving Ground, MD 21005-5066		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-MR-709
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In order to study autonomous behaviors in small military robots, researchers at the U.S. Army Research Laboratory (ARL) renovated an existing but outdated ATRV research robot. Commercial sensors with capabilities resembling those anticipated from the military tech base were selected and integrated, and the computing capability was substantially enhanced by judiciously selecting commercial components. Support electronics were upgraded or replaced as necessary. Safety elements common in larger robotic vehicles were integrated into the small ATRV chassis. Systems software was selected to provide a stable foundation for the advanced functions envisioned. Player, a middleware widely used by academic robot researchers, was incorporated as a springboard to the agent-based behaviors believed necessary for the next phase of development in robotics. A distributed development environment was implemented to enable parallel software efforts. Issues in software architecture were identified, and architectures from the literature were investigated in search of a foundation for future work. Without major investment, the antiquated research robot has become a key element in ARL's quest to develop technologies for a highly capable robot to team with soldiers on tomorrow's urban battlefield.

15. SUBJECT TERMS
unmanned ground vehicle, robot, perception, robotic agent architecture

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Gary Haas
a. REPORT UNCLASSIFIED	**b. ABSTRACT** UNCLASSIFIED	**c. THIS PAGE** UNCLASSIFIED	UL	34	**19b. TELEPHONE NUMBER** (*Include area code*) 410-278-8867

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Objective and Program Background[*]

In late 2007, the U.S. Army Research Laboratory (ARL) initiated a research thrust targeting robotics technologies for asymmetric warfare in urban terrain. The vision for this tactical domain calls for small robots suited to maneuver indoors as well as outdoors, perhaps sharing space with troops, with sensors to perceive the immediate surroundings, and sufficient intelligence to enable (at least short periods of) unsupervised operation. Such a robot must embody substantial autonomy, e.g., be able to "see" and "understand" its environment so that it can perform its function with minimum burden on the soldiers it supports. It requires sensors capable of detecting the immediate surroundings with high fidelity and richness, and powerful onboard computing systems. At the inception of the new thrust, such capabilities were unavailable in research robots currently in ARL's robotics labs. As a first step toward exploring this new mission space, scientists and engineers at the Vehicle Technology Directorate's Unmanned Vehicle Technologies Division (UVTD) set out to create the capability.

A test bed for developing autonomy technologies, at least early in the program, can be based on commercial sensing technologies and a mobility platform of limited performance. The lab had in its inventory 8-year-old ATRV Jr. research robots once built by Real World Interface, Inc. (RWII). At the time of acquisition, these robots were quite advanced and offered skid-steer wheeled mobility, global positioning system (GPS) and electronic compass for navigation, ladar (a portmanteau of laser radar and often used interchangeably with lidar) and sonar sensors for obstacle detection, and a software development environment based on linked server modules. These robots were at the core of in-house robotics research at ARL. By 2007, the ATRVs were well worn. RWII (renamed iRobot Corporation) had discontinued production and support, and the Pentium III processor and Red Hat 6.2 operating system at the core of the robotics had been superceded by several generations.

Research supporting ARL's new robotics thrust calls for research robots similar in scale to the old ATRVs and with power and payload to support quantities of sensors and computing. New research robots have become available from several vendors, but, in general, the function of these products is not substantially different from that of the old ATRVs. The decision was made to renovate the old robots rather than invest in new research robots. This report describes the upgrade of the ATRV robot for its new role.

[*]The products described in this report are believed to be suitable for the intended use, but their use in this endeavor does not constitute an endorsement by the government. Other products may perform as well or better, or be less expensive.

2. System Description

The stock ATRV robot is a 25-in-long × 24.5-in-wide × 21-in-high 110-lb vehicle. Two deep-draw, gel-cell, lead-acid batteries power a pair of servo motors that drive its four 12-in tires in skid-steer fashion by means of toothed belts. A sturdy rectangular sheet-metal chassis houses the internals and supports the mounting rails front and rear and on the deck.

Of the rest of the original major components, only the pan-tilt unit remains. The rest have been replaced by functional equivalents and supplemented with functional extrapolations. Figure 1 depicts the components of the upgrade at a block diagram level. Details of the upgrade follow.

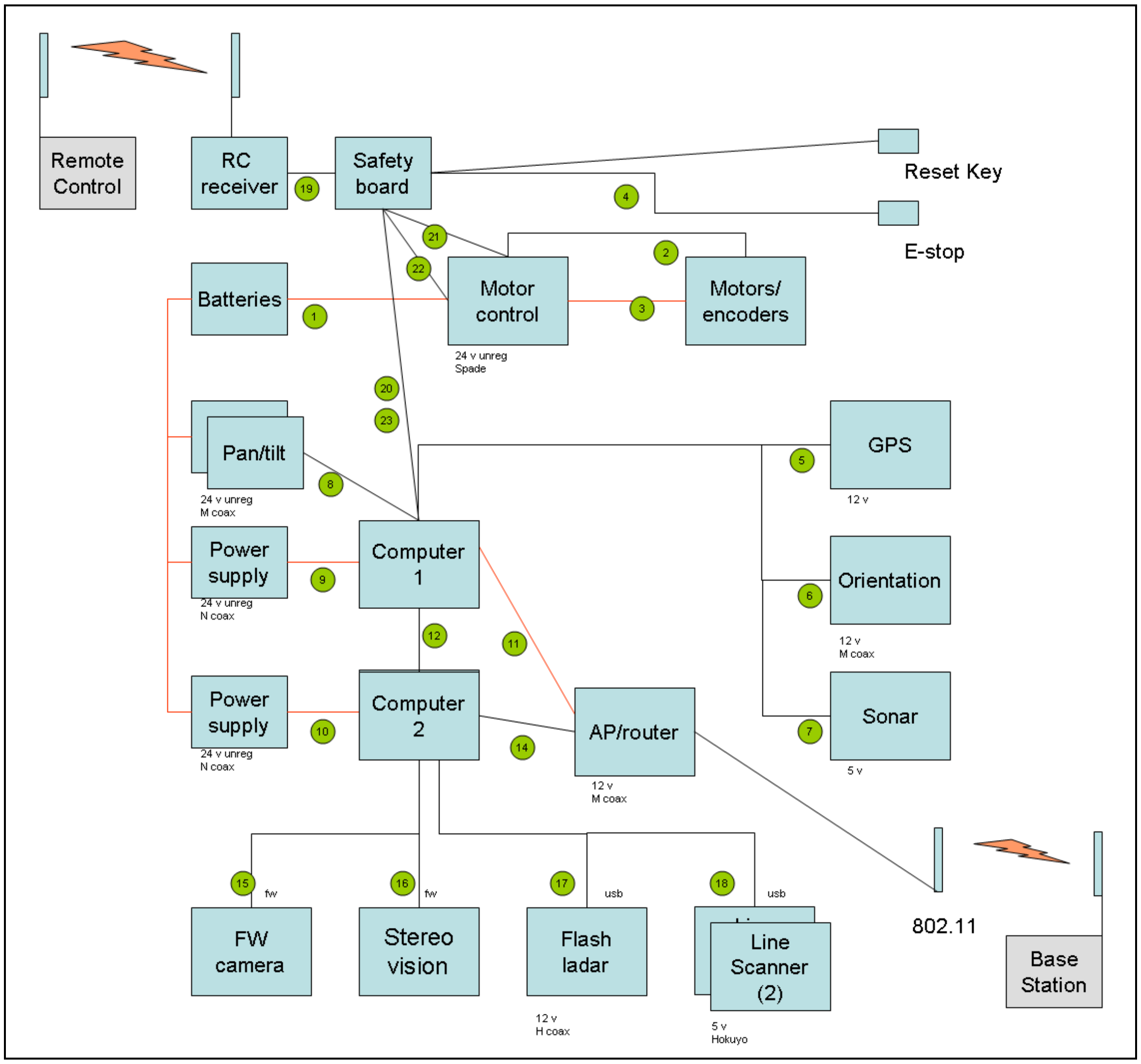

Figure 1. Major components of the upgraded ATRV. Gray indicates the component is off-board, linked by radio.

3. Upgrade

3.1 Hardware

3.1.1 Computational Hardware

Increasing the computational capacity was a requirement for the ATRV upgrades. In order to accommodate the planned sensor-processing research, the computers needed to represent the latest technology available off the shelf. In addition, they needed to be small and power-efficient but possess sufficient processing capability to minimize the processing bottleneck. The current trend in the hardware and software community is to leverage horizontal scale (more processing elements on a die), more so than clock speed. Thus the new computational hardware design takes full advantage of the space and processors available.

3.1.1.1 Central Processing Units (CPUs). A Mini-ITX (a motherboard format popularized by Via Technologies, Inc. [*1*]) was located that supported dual- and quad-core Intel processors. The only one available at the time was the Commell Core 2 Quad[*] with dual gigabit ethernet ports, six USB 2.0 ports, an 8-bit digital general purpose input/output[†] port, and Serial Advanced Technology Attachment.[‡] The ATRV Jr. has enough room for two reasonably sized mini-ITX cases, so the computers are relatively modular and easily replaceable. Each computer is equipped with 4 GB of RAM (although the computer system bus allows access to only 3 GB). A gigabit ethernet switch connects the two machines together and essentially yields a miniature computing cluster with eight processor cores.

3.1.1.2 Storage. Storage is provided by a single 16-GB solid-state drive for each CPU. Solid-state drives are used to increase system performance (relative to standard rotating platter 4200 RPM laptop drives) and increase system reliability with respect to vibration issues. While the storage size is small compared to today's large drives, it is more than enough to hold an hour's worth of raw data from the primary sensors and can be easily expanded with larger sizes if the need arises.

3.1.1.3 Network. As mentioned previously, the computers interface with each other through the network switch and with external machines (i.e., control stations, logging/debugging systems, development systems) through either a 100-Mbs wired or 54-Mbs wireless interface provided by an Alfa Network's AWAP608 wireless access point (*2*).

[*]Model No. LV-676.

[†]Usually an 8-bit digital interface.

[‡]A computer bus for attaching mass storage devices to a computer.

3.1.1.4 Sensor Interfaces. The ATRV Jr. has a variety of onboard sensors that utilize several connection interfaces on the CPU: serial, USB 2.0, and IEEE 1394.[*] Many of the serial devices are connected through RS-232[†]-to-USB adapters, while others (like the motor controller) are connected directly to the motherboard. Several perception sensors on the ATRV Jr. use USB and IEEE 1394 directly.

3.1.2 Sensors for Reaction and World Modeling

The purpose of sensors on a mobile robot is to create an analog of the nearby environment in data structures used by the computer programs that control the robot. These data structures are collectively termed the "world model." While some robots simply react to sensor inputs to alter some behavior ("obstacle ahead, turn left"), the objective of this project is to enable the robot to sense the geometry of its environs, store the sensed elements in data structures based on a self-constructed local map, and plan its behaviors based on the map. The map will have different layers, populated with geometric elements (extracted and abstracted from its geometry sensors), spectral elements (extracted from its cameras), elements fused from the sensor-derived elements, and iconic elements (extrapolated from the sensor-derived elements and filtered based on a mission-based context).

The universe of sensors appropriate for small robots is not a large one. Sensors considered for the robot are described further in this section, and those selected for the upgrade are pictured in figure 2.

3.1.2.1 Video Camera. Video cameras are widely available and relatively inexpensive. Imagery is dense (high pixel count) but only spectral in nature. The information content of the image is rich but lacks the immediate geometric significance needed for safe mobility. Significant processing is necessary to convert a stream of spectral images to the geometric world model needed. However, given a geometry by some other sensor, the richness of the video imagery can be overlaid. Video imagery is also the most easily interpreted sensor mode for a human. Augmented by feature-tracking software, the video sensor can provide a direction reference and can support algorithms such as direction-only simultaneous localization and mapping.

Given the widespread availability of video cameras, there are a number of parameters that can be used as a selection criterion. Field of view, a function of lens selection, is probably the most important (the wider the better). In general, the image resolution (number of pixels) is not important, as even the least capable cameras have sufficient resolution for the application (except for stereo vision, which will be treated separately). A key parameter is the ease of integration with a computer. While analog cameras today dominate the market, cameras with a built-in digital interface are more suitable for the application. This is partly due to ease of interfacing, but the primary reason is that most digital video cameras have progressive scan technology,

[*]Also known as Firewire (another serial bus standard for computer interfaces).

[†]A standard serial interface to a computer, once common but recently supplanted by USB.

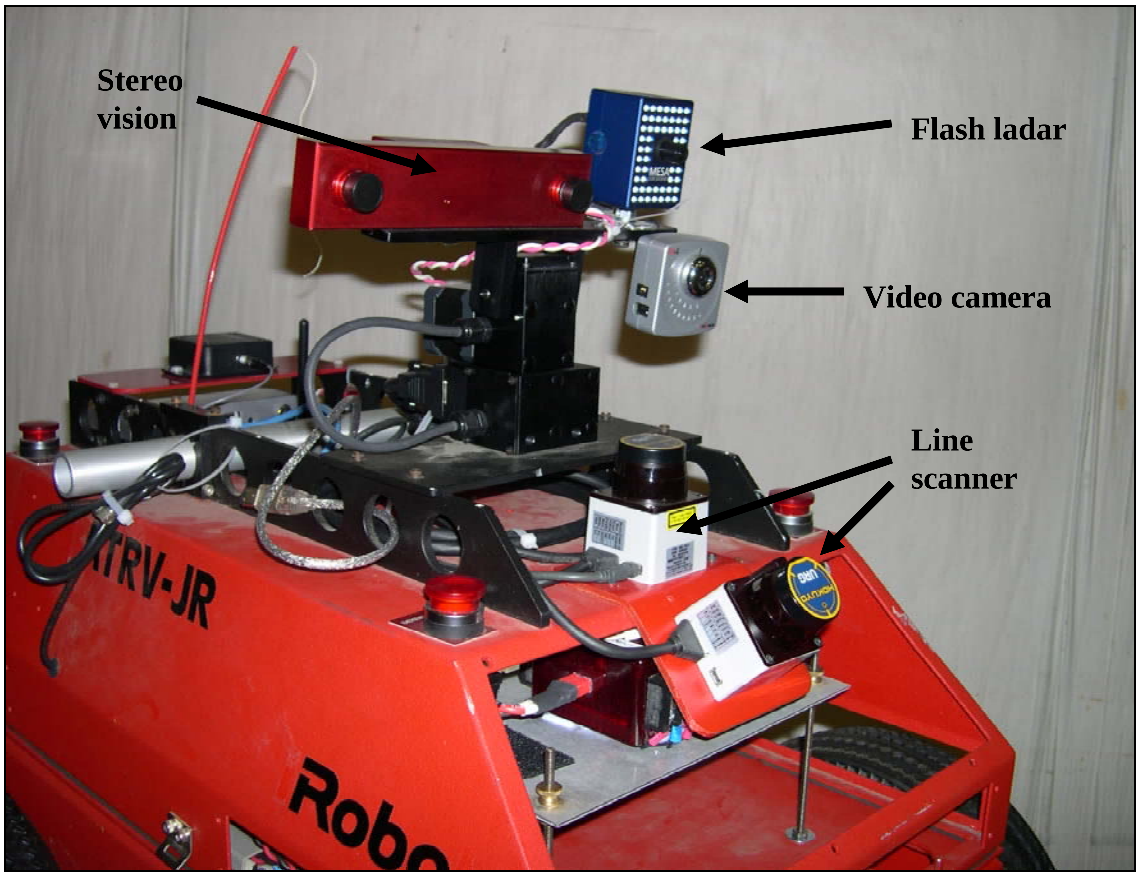

Figure 2. Principal sensor of the ATRV upgrade.

which is important when the camera is mounted on a moving platform. Firewire (IEEE 1394) cameras are the most common. For streaming applications, Universal Serial Bus (USB) has little to offer over Firewire, which was designed with video applications in mind. Both offer 400-Mbs rates, adequate for video graphics array* (VGA)-quality resolution or a little more. The newer IEEE 1394b, at 800 Mbs, is not yet widely available but will become so.

A representative video camera is the Unibrain Fire-i, priced at around $120 for VGA-resolution imagery with a plastic case and glass lens (*3*). A 4.3-mm focal length *f* 2.0 lens was selected, specified to deliver VGA-resolution imagery at 30 Hz, covering a horizontal field of view of 42.25°. A number of vendors offer competing products; this selection was based on low price for adequate performance.

3.1.2.2 Near-Range Geometry Sensing. For a number of years the only sensor capable of detecting the geometry of the environment, in a package of suitable size and cost for use on a small robot, was a sonar sensor such as the Polaroid product of the early 1980s. Modern sonar sensors for robotics differentiate between empty and occupied volumes in a cone subtending as little as 15°, so the spatial resolution perpendicular to the cone is limited, and the “occupied”

*A standard for computer display hardware.

region of the cone can be a tiny fraction of the cross section of the cone at the detected range (*4*). Sonar units work well enough for simple reactive obstacle detection, and several will be incorporated in the finished upgrade. However, a denser scan is necessary to be useful as a geometry sensor.

3.1.2.2.1 Line Scanner. A line scanner is one such sensor and is widely used in mobile robotics. A line scanner is a laser range finder, which is swept through an arc by a spinning mirror. The sensor detects the return from the laser and calculates distance from time of flight at discrete angular increments around the disk so described. A line scanner oriented so the plane of detected points is horizontal (e.g., the axis about which the mirror spins is vertical and the angle between the mirror and its axis is 45°) is useful in real-world terrain where objects of interest (walls, etc.) are also vertical, such as an indoor environment. This sort of sensor is insufficient for general obstacle detection and terrain mapping but can be used to generate a useful first approximation, as vertical terrain features tend to be the most salient.

A line-scanning unit built by the German company SICK AG was used on many of the mobile robots competing in the recent Defense Advanced Research Projects Agency Grand Challenges for autonomous unmanned ground vehicles. The SICK unit, however, is too large and heavy for this application. Instead, a device similar to the principal geometry sensor was used.

A smaller line-scanning device, the URG-04LX (*5*), is available from Hokuyo. This sensor works very much like the SICK scanner but is small (2 × 2 × 3 in) and lightweight (165 g). It sweeps an arc of 240° at a rate of 10 Hz, returning range measurements at intervals of 0.36°. This corresponds to approximately one data point per inch at the maximum range of 4 m and 683 data points every 100 ms to process to maintain real time. The URG sensor is mounted to the frame of the robot so that the plane of the measurements is horizontal and at the height of the robot. This is consistent with using the sensor to avoid right prismatic (cuboid) obstacles, and it also enables the mapping of indoor terrain, which is predominantly bounded by vertical planes.

A second line scanner is mounted at the front of the robot, directed at the ground ~1 m ahead of the robot. This scanner senses the terrain the robot is just about to drive onto. The horizontal line scanner receives no sensed data from the ground, so the second scanner is depended upon to assure that there is indeed ground to drive upon and that the terrain is smooth enough for the robot to traverse. Ideally, this sensor would look out 3 m ahead so there would be time to stop if, for example, the sensor detected the top step of a flight of stairs. The look-ahead distance may be changed as researchers gain experience with the system.

3.1.2.2.2 Imaging Ladar. More detail concerning the geometry of the environs is available from an imaging ladar sensor. Such a sensor acquires range data as a set of range vectors centered at a focal point and organized as an image, e.g., rows and columns of data points. Surveying ladars, such as those available from Riegl USA, Inc., provide high-resolution three-dimensional data at ranges over 100 m, but the range measurements are sequential and too slow for mobile applications. Ladars built specifically for mobility applications, such as the product built by

General Dynamics Robotic Systems for the Army's Autonomous Navigation System, are substantially faster, but today's technology is too large and heavy for use on this small robot.

A recent technology known as "flash ladar" is based on camera technology, enabling fast data acquisition as well as light weight and compact size. Such a device illuminates the environment with light modulated at a known frequency and determines time of flight from the phase shift of the reflected energy incident on each pixel imager. Devices using this technology are available from PMD Technologies GmbH, Mesa Imaging AG, and possibly others.

The device selected for the ATRV sensor upgrade is the Mesa Imaging SwissRanger SR-3000 (*6*). This sensor collects frames of range data 176 × 144 pixels at a rate of 30 Hz over a field of view of 47.5° × 39.6° (0.27° per pixel). Maximum range is advertised as 7.5 m, limited by the nonambiguity constraints of the measurement technique, but several papers in the literature indicate a shorter useful range. Range resolution is specified by the data sheet as 1% of range. A cursory evaluation of the sensor revealed a sensitivity to bright lights, resulting in washed-out regions of the image, which must be further investigated.

The SwissRanger is mounted on an existing pan-tilt unit on the deck of the ATRV. The pan will be used to compensate for the narrow fields of view of the various sensors mounted on the unit. The tilt axis will likely be set at a fixed look-down angle, which provides a "good" amount of information about the ground immediately ahead of the robot while not sacrificing too much information about overhanging objects.

3.1.2.2.3 Stereo Vision. There will be times when it is necessary to sense geometry at ranges greater than that provided by the active sensors. Computer-based stereo vision can provide range images at distances of tens of meters, but it has been seldom utilized outside the laboratory (and in planetary exploration). In part, this is because the sensors (conventional cameras) are inexpensive, but the computing to process the camera images into a range image was "do it yourself"—the phenomenon was well understood and algorithms were widely available, but there was no integrated stereo "system" delivering range images.

The recent availability of "Stereo on a Chip," from Videre Design LLC, has changed the maximum range available from an active sensor (*7*). A field programmable gated array packaged with the complementary metal oxide semiconductor video imagers computes disparity (a function of range) at each pixel of the VGA-resolution image at 30 Hz, reducing the workload of the host processor substantially. The 4.5-mm lens images a field of view of 59° horizontal × 41° vertical. As configured for UVTD's application, the range resolution at 8 m is computed by Videre Design's online calculator to be roughly 2.5 in; ranges closer than 0.5 m are unavailable. The stereo sensor is expected to deliver a dense sampling of a (possibly imprecise) range function, allowing ranges to be estimated beyond the ability of the ladar sensors. Reflectance values from the stereo system are also available on a frame-by-frame basis, enabling data integration and/or fusion.

The stereo sensor will be mounted on the pan-tilt unit near the SwissRanger so the region sampled by both sensors overlaps. The overlap among the fields of view of the various sensors, shown in figures 3 and 4, will be exploited in any way possible.

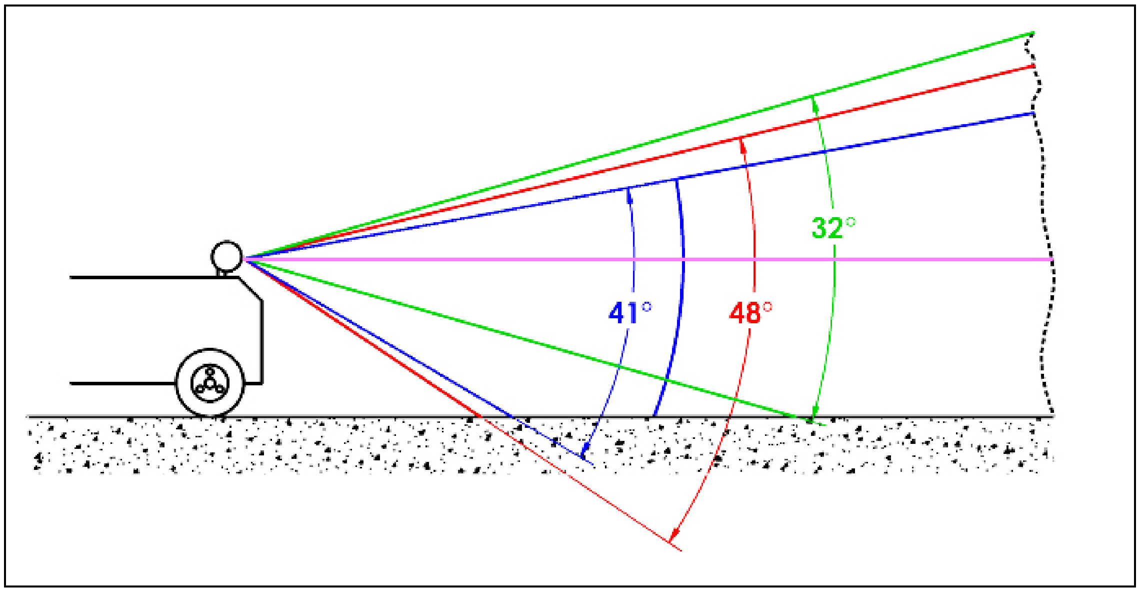

Figure 3. Sensor fields of view are shown here in side view, according to the following key: stereo in blue, color camera in green, flash ladar in red, horizontal line scanner in magenta.

3.1.2.3 Inertial Reference. All the sensors described report their sensed data in the coordinate frame of the sensor itself. In order to integrate the information over time, it is necessary to transform the data to a common coordinate system, preferably a world-fixed coordinate system. The conventional way to do this is to sense the robot location from a sensor such as GPS and the robot orientation from a compass, and augment these sensors with time derivatives of each from an inertial reference sensor suite and odometry. In the case of this ATRV, the ideal operational area is where GPS is unavailable and where compass readings may be compromised (and possibly in unmapped regions). In this case, high-quality time derivatives of position and orientation are wanted because of the integrations required.

A 3DM-GX1 inertial reference sensor (IRS) from MicroStrain, Inc., (*8*) provides orientation and acceleration for the upgraded robot. Raw data from embedded accelerometer, gyros, and magnetometers are fused by the sensor itself. Alternative IRSs are available but were not considered since sensors from the MicroStrain product line are used on other division assets, and performance was deemed acceptable.

3.1.3 Power

Power for the robot as a whole was left unchanged from the original ATRV, that is, dual 12-V deep-draw batteries with off-board recharging. It remains to be seen whether the duration available will be sufficient for research missions.

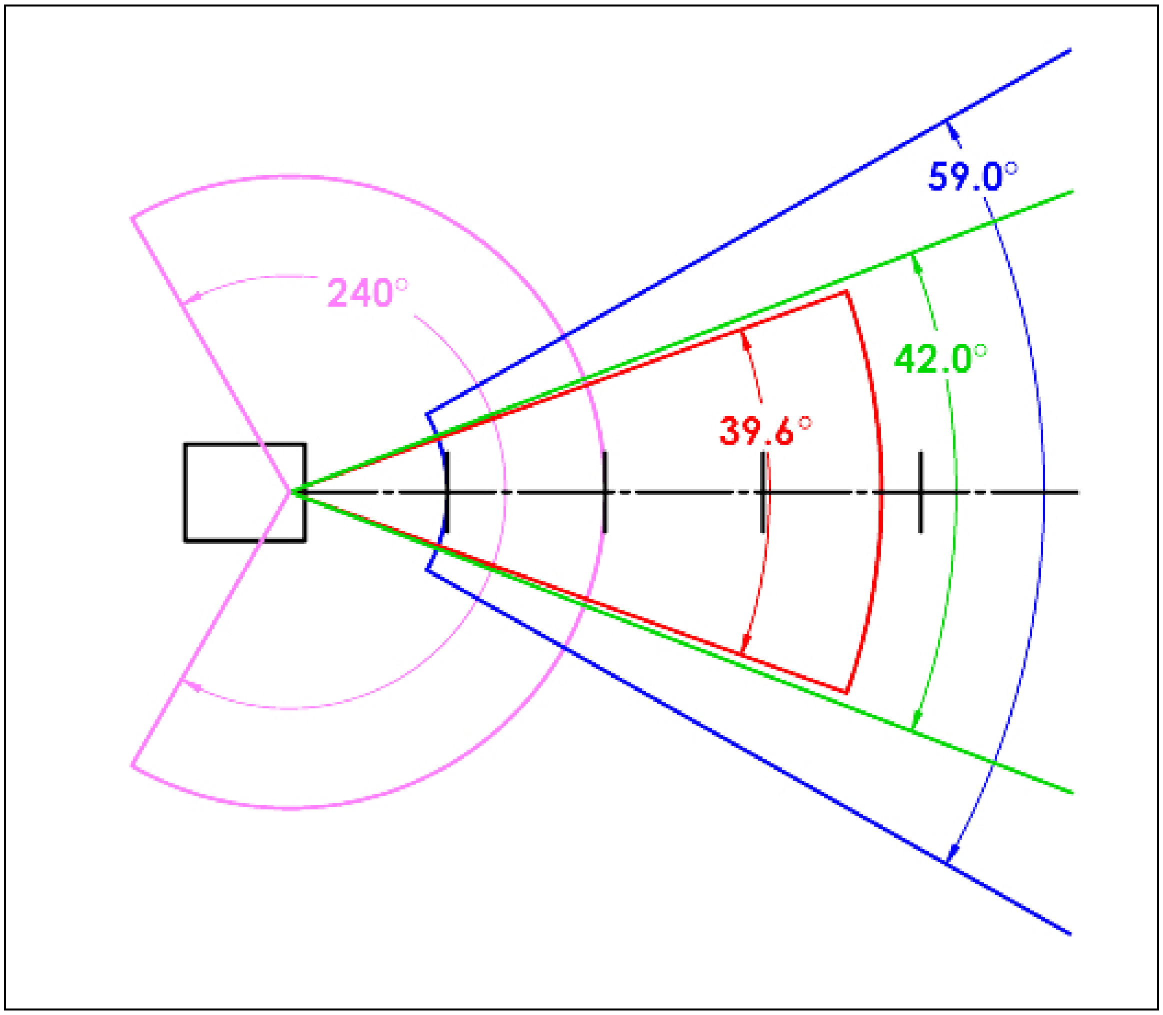

Figure 4. Sensor fields of view are shown here in top view, according to the following key: stereo in blue, color camera in green, flash ladar in red, horizontal line scanner in magenta. Black centerline shows crosshatches at 2-m intervals out to 10 m.

Power for peripherals was shifted from the computer power supply of the original ATRV power architecture to a custom power distribution board supplying regulated 5 and 12 V through bussed terminal strips.

The CPU of the original ATRV computer was rated at ~30 W, while the CPU selected for the upgrade was rated at 125 W, so the computer power supply was upgraded as well. The onboard power supply, the M2-ATX 160W (*9*), was selected based on its tolerance of a wide range of input voltages (6–24 V) and its form factor, which corresponds to the dimensions of the case selected. The maximum input operating voltage, outlined in the specifications, is 24 V, which is marginal for a battery system consisting of two 12-V batteries. A more recent release, the M2-ATX-HV, claims an even higher maximum input voltage and would be a more conservative selection.

3.1.4 Motor Control Board

The motor control board was replaced as well, though the two wheelchair motors and gearboxes driving the ATRV's four wheels were retained. The AX3500 product from Roboteq (*10*) was selected for the following reasons:

1. It is designed to run with 24-V rechargeable batteries.
2. It supplies maximum motor current of 40 A for each of two brush-type electric motors, well beyond the 11-A rating of the wheelchair motors.
3. Input to the motor control board through RS-232 is available. A translational velocity /rotational velocity command is native.
4. The board supports optical encoders, also part of the original ATRV motor suite. The encoders enable closed-loop velocity control, an essential element for control of a skid-steer vehicle where soil resistance is uncertain. In addition, the encoder counts can be monitored through the serial link, providing an odometry function.
5. It has two distinct safety shutoff modes, including one which can be easily asserted from a remote radio control (RC) controller.

While each of the elements listed are essential to the application, a number of other features bring added value to the control board, notably the ability to control the robot from an RC remote control. This capability is very useful in logistic operations, such as maneuvering the robot into its parking place at the end of the work day.

3.1.5 Safety Circuitry

The original ATRV was equipped with four e-stop mushroom buttons on the robot. To stop the robot in case of a software failure, it was necessary to approach the robot and push one of the buttons. One of the goals of the upgrade was to increase the safety of the robot by enabling a software-independent kill capability from a distance, so the robot can be brought to a halt without jeopardizing the operator.

Using a safety mode suggested by the manufacturer of the motor control board, there are now three means of stopping a runaway robot. The first is by means of the e-stop buttons on the robot. These cause an e-stop input on the motor control board to be activated. The second is a red e-stop button on the remote control. Pressing this button actuates the same input through one channel of the RC radio. Both require a reset from a key-switch on the robot body before motor control is restored.

The third mode stops the robot by a new mechanism. The motor controller can accept commands from either the serial link connected to the onboard computer or from the RC receiver linked by a dedicated radio channel to the remote control. The remote control determines which signals reach the control board input by means of an electromechanical relay.

The default control is from the remote control, and it must be ceded to the computer by a manual switch on the remote control. If the program running on the computer is judged by the operator to be in dangerous error, the operator can throw the switch on the remote control, which seizes control from the computer and returns it to the remote control in the hands of the operator. This safety paradigm is similar to that used on UVTD unmanned air assets. Figure 5 depicts the operator's remote operation and safety control. Figures 6 and 7 illustrate status monitors, allowing the operator to confirm elements of the robot control state.

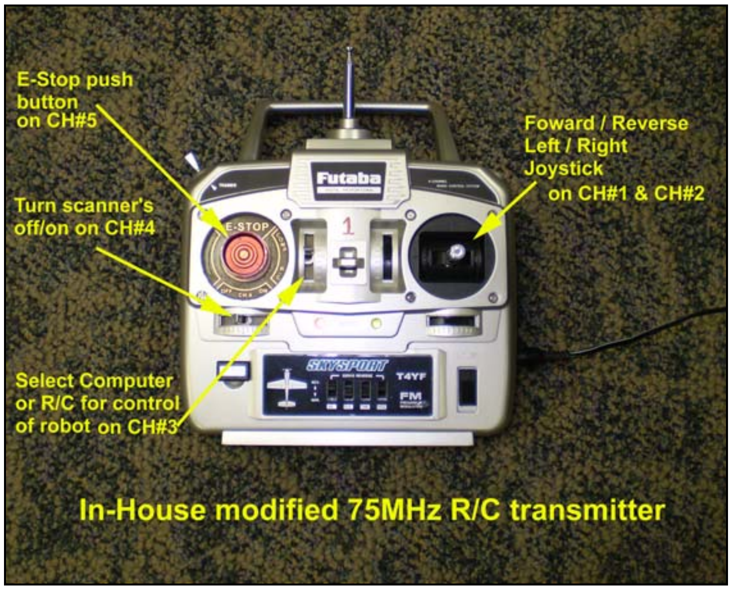
E-Stop push button on CH#5
Foward / Reverse Left / Right Joystick on CH#1 & CH#2
Turn scanner's off/on on CH#4
Select Computer or R/C for control of robot on CH#3
In-House modified 75MHz R/C transmitter

Figure 5. The remote control provides the operator the ability to stop the robot from a safe distance in case of emergency and operate it manually with the joystick.

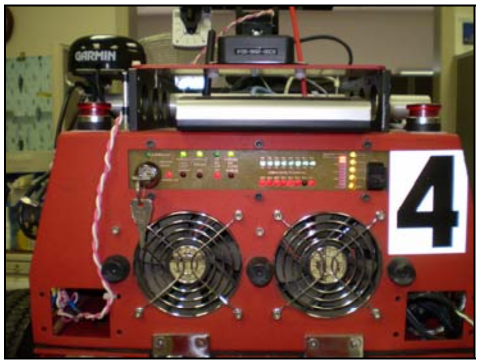

Figure 6. Display panel at ATRV rear.

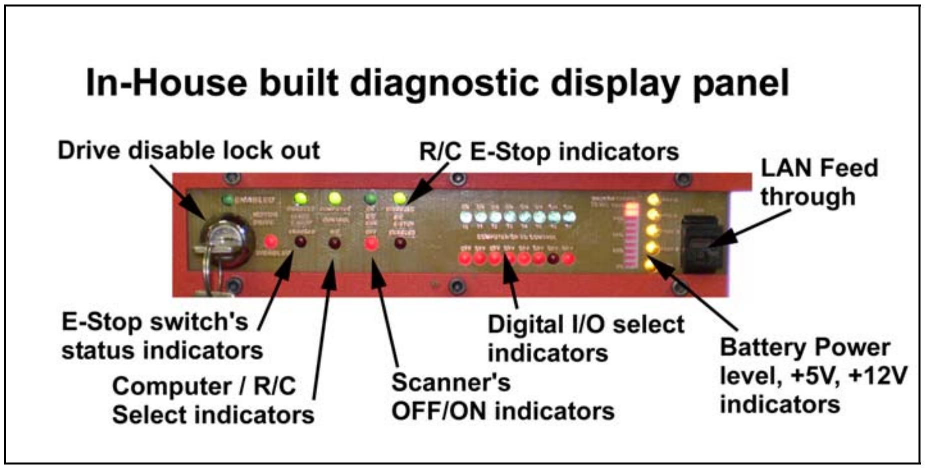
In-House built diagnostic display panel
Drive disable lock out
R/C E-Stop indicators
LAN Feed through
E-Stop switch's status indicators
Computer / R/C Select indicators
Scanner's OFF/ON indicators
Digital I/O select indicators
Battery Power level, +5V, +12V indicators

Figure 7. The display panel shows the status of a number of internal states.

3.2 Robot Software

Since the original software provided by RWII for the ATRV Jr. was proprietary, replacing the computers meant replacing the operating system and supporting software (including the device drivers). Thus the decision was made to utilize the Open Source robotics package Player (from the Player/Stage project [*11*]), which provides a convenient hardware abstraction layer and a multitude of popular robotics device drivers. Player is also used by numerous academic institutions with robotics programs including the University of Pennsylvania, Georgia Institute of Technology, and the University of Southern California (*12*).

3.2.1 Organization

The software on the robot computers is divided into three layers: base, core, and brain (see figure 8).

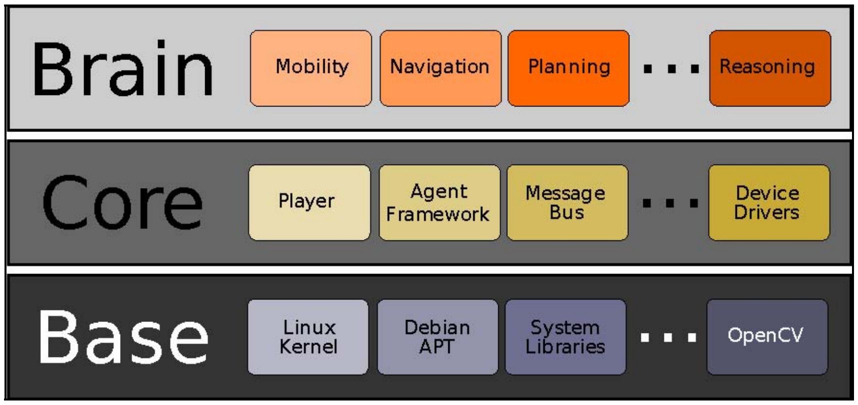

Figure 8. High-level software organization. The smaller items in each level indicate the types of components in each layer.

The base layer contains relatively static library code and the core operating system facilities that are common across all machines, including a properly configured kernel for the CPU architecture. Debian (*13*) GNU/Linux is the operating system of choice, primarily to gain the benefit of its package management system Advanced Packaging Tool (APT) as well as the ease of configuring a custom system using the very useful debootstrap tool.

The core layer is an abstraction layer providing higher-level functionality to the layer above it. It contains the Player system and any shared libraries required for robotic development, including the agent architecture under development (see section 3.4). The ATRV Jr. currently under development has a custom driver plug-in for Player that was developed for the Roboteq controller, as well as the libraries required for the SR3000 Flash Ladar. In general, more volatile libraries (rapidly changing open source and internal packages) will reside in this layer.

The brain layer is the effective application layer where agents and high-level behaviors can be implemented. Thus it will contain the custom scripts, executables, and data that compose the actual behavior of the robot. Currently, the brain layer is not implemented in the upgrade; however, work is underway to address that problem (section 3.4).

3.2.2 Operating System

The base operating system was constructed to be relatively small and boot fast. Debian GNU/Linux, however, provides an installation script that downloads a minimal Debian operating system without the Linux kernel. The minimal system includes only a small subset of a typical GNU/Linux distribution, so scripts were created to add in additional software readily available from the Debian software repository.

The omission of the Linux kernel from the minimal system is deliberate; more than likely, someone building a custom distribution will want to custom configure a kernel for specific hardware, as is the case for this project. Thus the scripts choose among several custom kernel configurations based on a given keyword, build the kernel, add it to the system base directory, and build a disk image that can be copied directly to the robot’s hard disk.

The current custom build of Debian is around 300 MB, which includes all the required kernel modules, base libraries, and extra libraries needed to comfortably support the Player system and most anything else needed (this includes the Debian-provided version of the OpenCV computer vision library). That is considerably larger than the initial goal and is mostly the result of some extraneous dependencies on GTK+ libraries within Player, which can be removed when time permits. However, the system does boot in just under 10 s, which is quite good. Boot time can be improved by optimizing the operating system.

3.2.3 Device Abstraction

As mentioned previously, the Player system is a “robot device interface and server” and acts as a device abstraction layer to elements of the robotic architecture residing in the core and brain layers. Since Player is now one of the most popular open source robotics libraries, it already has support for many of the devices used, including IEEE 1394 cameras, USB cameras, the SR3000 flash ladar, and the Microstrain inertial measurement unit (IMU). Adding a device is straightforward—pick or develop a Player interface that defines an abstract representation of a device (e.g., the laser interface defines how to talk to a laser-ranging sensor without worrying about the particulars of device initialization, configuration, or communications protocol). Most drivers then provide a specific implementation of an existing interface and a configuration file format for specifying hardware-specific parameters in a runtime-configurable format.

The ATRV Jr. currently has one custom device driver implementation for the Roboteq motor controller, described in the next section.

3.2.3.1 Roboteq Device Driver. The Roboteq device driver written by UVTD implements the Player position2d interface, which can be used to control planar mobile robots. The interface provides the facility to issue velocity commands (x_dot, y_dot, theta_dot), position commands (x,y,theta), speed/heading commands (v,theta), and car commands (v,theta), which the driver may ignore or implement according to the platform configuration. The current version of the Roboteq driver only understands the velocity commands and assumes the presence of encoders and the use of mixed mode, closed-loop serial operation to the actual motor controller device.

3.2.3.2 Player Configuration. Player is very flexible and does not assume or impose much on a system design. The core of Player is the device abstraction, but Player also provides a Transmission Control Protocol (TCP)-based server that allows multiple remote connections and controls for each device configured for that server. In most cases, there is one Player server per robot providing a connection to all the devices configured for that robot. However, the server is not implemented in a concurrent manner; therefore, the configuration in use on the ATRV Jr. takes advantage of multiple CPUs by providing one Player “server” per robotic device, e.g., a stereo vision server, an IMU server, etc. This provides the same abstraction as one server for all but allows the servers to run concurrently (and therefore block concurrently, if need be).

3.3 Development Environment

Building an essentially new robotic system from the ground up[*] requires that configuration management (CM) and software engineering issues be addressed. Section 3.3.1 highlights the motivation and derived requirements that guide the design of the environment described in section 3.3.2.

[*]At least from the software point of view.

3.3.1 Motivation and Requirements

Creating a formal CM environment is motivated by a desire to do the following:

- Keep the robot systems clean and free from version incompatibilities.
- Allow new engineers/developers to become productive with the available tools.
- Facilitate access to the code.
- Share the maintenance and development of the system across the set of contributors.
- Support more than one robot system.
- Encourage and/or enforce compliance with software engineering practices in order to improve code quality.

These goals were used to define the following high-level requirements:

1. The robotic system software must be versioned from a central server.
2. The development model will be a host-target configuration.
3. The development systems must have a common set of libraries and tools, and therefore be imaged from a central server.
4. The development systems must have access to the robot system.
5. The central server and development systems must reside on the same network.
6. The central server must provide reliable data storage.
7. At least two laptops must be available for operating, debugging, and logging data from the robots.
8. It must be easy to update the robots with newly developed software.

The system design that implements these requirements is detailed in section 3.3.2. However, two particularly important ramifications of this CM warrant further description in the following sections.

3.3.1.1 Providing a Clean Slate. Since the ATRV Jr. is a shared resource that will be utilized by multiple researchers often investigating somewhat orthogonal topics, providing a clean operating environment is essential. Extraneous software should be kept to a minimum, with an eye toward the essentials that make the robot run. Not only does this leave more storage capacity, but it speeds up the system and reduces the chance that software might conflict with mission-critical functions. While installing a programmer’s text editor and all the development libraries seems harmless, it also seems completely extraneous for a robot running a real mission.* In addition, the host-target model helps to ensure that the robot does not get out of sync with the repository. Facilities will be provided to build the relevant portions of the system (i.e., base, core, brain) and then transfer those portions to the robot (which enables a "set it and forget it" behavior).

3.3.1.2 Enabling Software Reuse. The ATRV Jr. upgrade is seen as an opportunity to begin the construction of a development environment and software platform conducive to creating state-of-the-art robotic vehicles based on the x86 architecture. Part of accomplishing this goal is providing for effective software reuse. Thus this approach relies on system connectivity, redundant data storage and automated backups, capable version control and a defined usage policy, and modular software design. Systems need to be connected in order to facilitate source code-level sharing and allow the systems to enforce a check-in policy. Reliable data storage is a requirement to prevent loss of valuable work and knowledge. Version control is a must in order to provide a well-known repository of code and a means to keep it organized and safely shareable. Finally, modular software design (see section 3.2.1) is the real key to reuse; while it is clear that a given robot will have some customized pieces of software that are likely unusable in a different robot, it is equally clear that many algorithms, frameworks, and sensor device drivers can be used across many robots. This is ultimately the purpose in open source software packages like the Player/Stage project and Yet Another Robot Platform (YARP) (*14*). A very conscious decision was made in this project to consider the packages available and reuse others' hard work as much as possible (i.e., the use of Player/Stage and the Unified System for Automation and Robot Simulation [USARSim] [*15*] as well as the ideas, algorithms, and/or code from packages like YARP and the Mobility Open Architecture Simulation and Tools [MOAST] [*16*] framework). This has, without a doubt, accelerated the ATRV Jr.'s software development significantly.

3.3.2 Environment Design

Based on the requirements stated previously, the environment consists of the following elements: an online Ubuntu GNU/Linux workstation, an offline network, a development server, multiple development laptops, robots, a version control system (VCS), a remote synchronization server, and a host of scripts implementing build and update functionality. A high-level diagram of the topology is shown in figure 9.

3.3.2.1 Online Ubuntu Workstation. Since Ubuntu (*17*) GNU/Linux is based on Debian GNU/Linux, it inherits the excellent APT and an extensive repository of software.† The online Ubuntu workstation provides access to this resource, and custom scripts generate local repository mirrors that can be transferred to the offline development server.

---

*Recently, there has been some discussion on relaxing these restrictions through careful build-time parameters; e.g., a build switch could indicate whether the robot is being used for development or "production" use.

†Often, software is not available anywhere else without a manual installation procedure: find dependencies, download, compile, and install everything in the proper order (a time-consuming process).

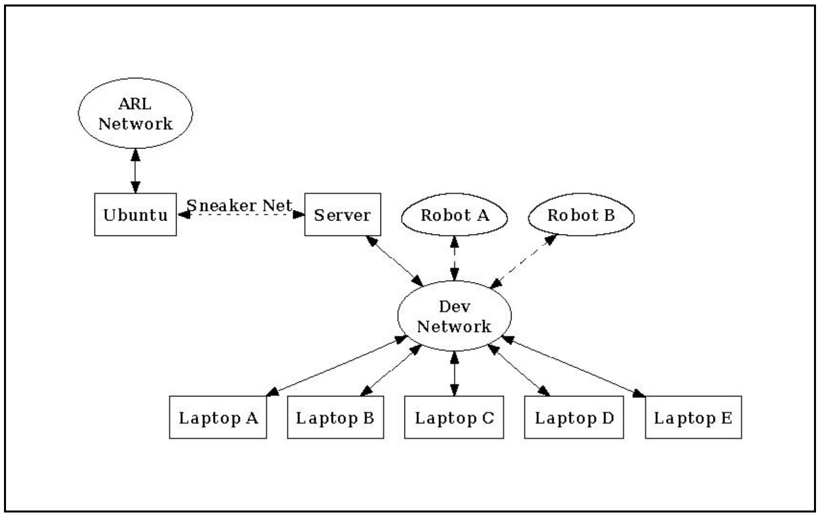

Figure 9. An Ubuntu workstation is online and provides updates to the development server through a manual process, which, in turn, provides automatic updates for the offline network.

3.3.2.2 Offline Network. An offline network is necessary since the custom robot systems (based on Debian GNU/Linux) and development systems (based on Ubuntu GNU/Linux) cannot currently gain access to the Internet.* This network connects the development server and workstations together to provide version control and system software updates, and also allows the workstations to connect to the robots.

3.3.2.3 Development Server. The development server is an Ubuntu GNU/Linux-based system that provides the master VCS repository (on a RAID-1† setup providing reliable data storage) as well as all the required libraries and tools for software development. The development server also provides a file synchronization utility (rdist‡) service for the development laptops and considerable computational power for robot simulations.

3.3.2.4 Development Laptops. The development laptops are intended to be shared resources that serve as both workstations for software development and control stations for system testing. They are configured to enforce a VCS check-in schedule in order to ensure little or no data loss should some hardware fail. The development server automatically provides remote software updates to each laptop; thus all laptops have synchronized file systems. At least two of these laptops can be used specifically as control stations during testing or demonstration.

*For reasons beyond the scope of this report.

†Raid = redundant array of independent disks.

‡Remote file distribution: a method of distributing software updates from a central location to multiple remote sites. The behavior is implemented with a client (rdist) and a server (rdistd) (*18*).

3.3.2.5 Version Control System. An important part of a basic software development environment is always the VCS. It stores the history of the project and all development artifacts (i.e., it doesn't need to hold only source code). To offer the greatest flexibility, a distributed VCS[*] is utilized. This allows easier branching and merging of individual lines of development (i.e., facilitates experimentation as well as bug fixes for stable releases) and makes it possible to develop outside the network. Since any "clone" of the repository is actually a full repository in itself,[†] one can transfer the repository to a completely different network or simply take a repository offline for development away from the server. When this "disconnected" repository needs to synchronize with the "connected" repositories, the differences are automatically merged (whether through a network connection or the exchange of patch sets through some medium like e-mail). This configuration allows for the most flexibility, especially for contributors that may not have access to the network.

3.3.2.6 Remote Synchronization System. As mentioned in section 3.3.2.3, the development server hosts a rdist daemon that ensures the file systems across the server and laptops are identical. Note that home directories are not synchronized, while all the important development tools and libraries are kept up to date. This does require some manual updates to be made to the development server itself (see section 3.3.2.2) but only when changes are required (security update or new version of a library, etc.).

3.3.2.7 Build Scripts. A library of build scripts is distributed as part of the robot software distribution's development repository. These build scripts automate and therefore simplify a large portion of the effort needed to construct aspects of the robot software package. For example, there are scripts to download and build the base component as well as scripts to automatically configure the Player library for the ATRV Jr.'s specific device configuration. These scripts are expected to be developed over time to include the most configurable aspects of the system in such a way as to greatly facilitate the construction of a software package for an entirely new piece of robot hardware.

3.3.2.8 On the Host-Target Development Model. The development environment for the new ATRV Jr. is based on a host-target environment, meaning a large portion of development happens on a developer's workstation or a stand-in platform, and the resulting product is transferred to the robot for testing. While this is a departure from the previous ATRV configuration that allowed self-hosted development, it is not as bad as it seems. By using the Player framework for hardware device abstraction, researchers get to make use of supported simulation environments, including (but not necessarily limited to) Stage, Gazebo, and USARSim. A debugger is present in the default base distribution and the host-target separation

[*]Currently, Mercurial is the distributed VCS of choice. However, Git is also being investigated for its power and flexibility.

[†]Contrast this with centralized, nondistributed VCSs like CVS and Subversion (one must have access to the server in order to check in code).

can be relaxed for situations where it makes more sense to develop directly on the robot (however, review section 3.3.1.1 for reasons to avoid that scenario).

3.4 Toward a Robotic Agent Architecture

While the software support on the ATRV Jr. can, at present, be used for application development and provides a high-level division between modules responsible for different tasks, it does not address how individual components work together to achieve the total behavior for the system. The previous ATRV Jr. software architecture is similar to the current one in that it is based on a collection of services using the Common Object Request Broker Architecture. While services can go a long way in providing reusable device abstractions and behaviors, the architecture concepts discussed in the next sections extend that paradigm to a higher level—that of cooperating agents.

3.4.1 Issues

In the context of developing a framework to control the ATRV Jr. platform, the term architecture refers to how the overall behavior, intelligence, and control of the ATRV will be arranged in software. The term "agent" refers to a self-contained entity that implements a perceive -> think -> act loop and can communicate with other agents. The idea of the agent architecture is to implement the "brain" as a collection of loosely coupled agents acting individually to perform specific tasks and concurrently acting together to enable the emergent behavior, intelligence, and control of the robot. Since an agent can be thought of as a superset of a service, an agent-based architecture can parallelize and distribute across multiple CPUs or systems as well as service architectures can. An agent-based architecture also distributes better than a monolithic system (which cannot run across multiple systems at all).

The concept of agents is not particularly new or unique. Many robot software implementations provide the capability to construct stand-alone components that can communicate with other components, sometimes within a prescribed methodology, sometimes without any guidelines for application structure at all, and almost always tied to a specific implementation language. The following questions are addressed with a new architecture design based on agents:

- What infrastructure and performance capabilities are required to effectively support advanced perception research?
- What infrastructure is required to support autonomous behavior research?
- Is there a way to integrate concepts from the cognitive artificial intelligence perspective into a cohesive software framework that can also include more primitive capabilities?
- Can the architecture be efficient enough to run aboard relatively small robots but flexible enough to support more advanced software designs (i.e., distribution, concurrency, clustering)?

- Is there a way to integrate existing software as a functioning component within the architecture?

3.4.2 Initial Design Topics

The architecture is broken down at the highest level into two components: the infrastructure and the interfaces. There really is a third component, the actual implementation of the agents, but it is orthogonal to the architecture design. Note that hardware abstraction is handled by the Player software and is therefore omitted in the following discussion.

3.4.2.1 Infrastructure. The infrastructure component includes everything that enables the agents to cooperate and perform their functions but includes and prescribes nothing related to actual robot behavior. In other words, it acts as a substrate that supports the agents both during development and runtime.

Initial requirements for the infrastructure include:

- be as lightweight as possible
- enable multilanguage agent implementations
- support appropriate peer-to-peer and service-oriented constructs
    - discovery and lookup
    - group communication/multicast
- utilize message-passing abstractions for all inter-agent communication
- allow prioritized messages to disambiguate conflicting requests
- provide efficient implementation constructs where possible
- provide a simple method for launching agents

3.4.2.2 Interfaces. The interfaces describe the kinds of agents or services one uses to build a robotic intelligence system (but not necessarily how to build the agents). By specifying interfaces, one can effectively describe the requirements of an agent without specifying implementation details and decoupling agents from other agent implementations, providing for a more fluid system design that allows for parallelism, distribution, multilanguage support, and pluggable algorithms. For example, table 1 lists some agents and services researchers would like to implement on the ATRV Jr.

Table 1. A representative example of agents and services for the ATRV Jr., roughly organized by complexity.

Low	**Middle**	**High**
Safety	Identification	Planning
Mobility	Tracking	Task
Mapping	Telemetry/reporting	Health
Navigation	Manipulation	Cooperation
Geometry	Memory	Metareasoning
Obstacle	Context	Human interaction

3.4.3 Existing Work

The following sections provide brief surveys of a number of existing robot software systems that may serve as foundations or guidelines for future work.

3.4.3.1 Player. Player provides a multiclient/server paradigm over TCP for interacting with robot hardware and does not impose or suggest any other structure or organization. The common use case involves a single server representing the devices on a robot and one or more client programs implementing behavior. Player is written in C/C++ and provides C, C++, and Python client interfaces directly, while others have provided a host of interfaces for other languages (e.g., Java, Octave, Matlab). Refer to section 3.2 for information on the way this project is already leveraging the capabilities of this software package.

3.4.3.2 The Mobility Open Architecture Simulation and Tools (MOAST). MOAST framework is based on the four-dimensional real-time control system architecture developed at the National Institute of Standards and Technology (*19*). The framework divides functionality into vertical hierarchies called echelons (e.g., primitive echelon, autonomous mobility echelon), where each echelon is further divided into functional components (e.g., sensor processing, world modeling). The Neutral Messaging Language is used for platform-independent communication between modules (which can be distributed across different systems). Like most of the systems described here, MOAST is open source.

MOAST is one of the most promising candidates for integration into this project if further investigation indicates it directly supports or allows development of the needed infrastructure and interfaces outlined in section 3.4.2.

3.4.3.3 Coupled-Layer Architecture for Robotic Autonomy (CLARAty). The CLARAty project is described as a reusable robotic software framework. While not truly open source, this project has many algorithm implementations that may be useful as standalone elements in a new system. Integration may be difficult, however, since while the design is very modular, it does not seem to support concurrency or distribution explicitly.

3.4.3.4 Other Packages. While investigating previous work, the authors have come across a large number of software packages aimed at developing mobile robots, including but not limited to YARP, RobotCub, Saphira, OROCOS, MARIE, FlowDesigner, and RobotFlow. Descriptions of these packages are outside the scope of this report; a future report describing the proposed agent architectures will provide more in-depth discussion of the state of the art.

4. Conclusion

The ATRV, even with its upgrades, is not by any means a military robot. It is a platform that provides infrastructure (mobility, power, communication) supporting the research elements of the program, i.e., perception processing and autonomous behaviors. As these behaviors take shape and mature, they will be transitioned to robots designed for the field, along with sensors, platforms, and computing elements coming from other efforts.

The upgraded ATRV has computing power and communications at the state-of-today’s practice, a suite of sensors with a variety of technologies and complementary strengths, an architecture built on the foundations of the best robotics research institutions in academe, and safety substantially improved over the baseline. Lessons have been learned in the upgrade, which strengthened the skills and understanding of the researchers involved. It is to be expected that the process, as well as the product, of the upgrade effort will enhance the research efforts of ARL’s Unmanned Systems Division for years to come.

5. References

1. VIA Technologies, Inc., Mini-ITX announcement. http://via.com.tw/en/initiatives/spearhead/mini-itx/ (accessed 1 September 2008).
2. Alfa Network, AWAP608 product page. http://dplanet.biz/alfa.com/product_info.php?cPath=159_33_55&products_id=156 (accessed 3 September 2008).
3. Unibrain, Fire-i digital camera product page. http://www.unibrain.com/Products/VisionImg/Fire_i_DC.htm (accessed 27 August 2008).
4. Maxbotix, LV-MaxSonar-WR1 Beam Plots. http://www.maxbotix.com/Performance_Data.html (accessed 27 August 2008).
5. Acroname Robotics, URG-04LX product page. http://www.acroname.com/robotics/parts/R283-HOKUYO-LASER1.pdf (accessed 27 August 2008).
6. Mesa Imaging, SR3000 product page. http://www.mesa-imaging.ch/pdf/SR3000_Flyer_Sept07.pdf (accessed 27 August 2008).
7. Videre Design, STOC product page. http://www.videredesign.com/vision/stoc.htm (accessed 27 August 2008).
8. MicroStrain, Inc. 3DM-GX1 product page. http://www.microstrain.com/3dm-gx1.aspx (accessed 27 August 2008).
9. Mp3Car.com, M2-ATX product page. http://store.mp3car.com/M2_ATX_160W_Intelligent_DC_DC_PSU_p/pwr-015.htm (accessed 27 August 2008).
10. Roboteq, AX3500 product page. http://www.roboteq.com/ax3500-folder.html (accessed 27 August 2008).
11. Gerkey, B.; Vaughan, R. T.; Howard, A. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. *Proceedings of the 11th International Conference on Advanced Robotics*, Coimbra, Portugal, June 2003; pp 317–323.
12. Player Wikipedia, Player Users Wiki page. http://playerstage.sourceforge.net/wiki/PlayerUsers (accessed 1 September 2008).
13. Debian Home Page. http://www.debian.org/ (accessed 3 September 2008).
14. YARP Home Page. http://eris.liralab.it/yarpdoc/index.html (accessed 1 September 2008).
15. SourceForge.net, USARSim description. http://sourceforge.net/projects/usarsim (accessed 1 September 2008).

16. SourceForge.net, MOAST description. http://sourceforge.net/projects/moast/ (accessed 1 September 2008).

17. Ubuntu Home Page. http://www.ubuntu.com/ (accessed 3 September 2008).

18. MagniComp, Rdist Home Page. http://www.magnicomp.com/rdist/ (accessed September 2008).

19. Scrapper, C.; Balakirsky, S.; Messina, E. MOAST and USARSim: A Combined Framework for the Development and Testing of Autonomous Systems. *Proceedings of the SPIE Defense and Security Symposium*, Orlando, FL, April 2006.

Appendix. Sensor Selection for a Fast Indoor Robot

This appendix describes mobility sensor requirements for a robot capable of speed comparable to that of a human in similar circumstances. The assumed domain is indoors or mild outdoor terrain. The implication of indoor terrain is that there is little room for evasive action (a hallway might end in stairs that span the entire width of the hallway), so the most critical response to a hazard is to stop. Sensors must be able to detect an incipient hazard at a range sufficient to allow room for a complete stop.

The speed at which a human moves indoors can be parameterized as follows. Human walking speed is roughly 2 m/s, while human running speed can be roughly bounded by the rate of a world-class sprinter, say 10 m/s. Indoors, it seems unlikely that a human will move at a sprinter's pace, so assume a maximum speed of 5 m/s.

The deceleration of the robot to a complete stop is governed by brakes (for a vehicle-like robot) and coefficient of friction. For design purposes, the coefficient of friction is assumed to be 0.5, limiting deceleration to roughly 5 m/s/s.

The distance required for the robot to stop can now be calculated. From 5 m/s, with braking acceleration limited by the coefficient of friction, it will take the robot 1 s to decelerate to a standstill, during which time it will cover a 2.5-m distance. The implication for the sensor suite is that the robot must be able to reliably detect obstacles at a distance of at least 2.5 m. A safety margin for reaction/response time should also be included. A human can react in 0.1 s; if a robot reacts in the same time (not necessarily a conservative assumption), an additional 0.5 m must be added to the sensor detection range for a total of 3 m.

In general, sensors have an easier time detecting a positive (above the local surface) than a negative (below the local surface, e.g., a hole or depression) obstacle. However, both kinds of obstacles must be detected.

NO. OF COPIES	ORGANIZATION
1 (PDF only)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LAB IMNE ALC IMS 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI OK TL 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI OK PE 2800 POWDER MILL RD ADELPHI MD 20783-1197
	ABERDEEN PROVING GROUND
1	DIR USARL AMSRD ARL CI OK TP (BLDG 4600)

NO. OF COPIES	ORGANIZATION
1	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI IA S YOUNG 2800 POWDER MILL RD ADELPHI MD 20783-1197
	ABERDEEN PROVING GROUND
1	DIR USARL AMSRD ARL VT RP M CHILDERS

INTENTIONALLY LEFT BLANK.