Rehabilitating a Nomad SuperScout Robot

Edgar Andrade, Filipe Caetano, Jose Quaresma, Luis Perdigoto, Luis Conde Bento

Abstract— **Once popular, the Nomad robots from Nomadic Technologies have become outdated and cast aside. This article presents our redesign and rehabilitation of a Nomad Superscout. We salvaged the original sensors and actuators but completely modify the internal control hardware. We propose a modular and distributed architecture, based on a CAN network, with a embedded PC/104 serving as the onboard computer, running a Linux based, real-time operating system. We describe the hardware modules that form the architecture and how the sensors and actuators are integrated, including new sensors added to the system, namely a magnetic compass, accelerometers and gyros. We also discuss our efforts in integrating the robot in the Player/Stage/Gazebo simulation environment.**

I. INTRODUCTION

In the past, mobile robots from Nomadic Technologies were used by research laboratories all around the world, with the popular models Nomad 200 and Nomad Superscout frequently being mentioned in the literature [1]-[3]. However, with the end of the company's robot division, software and hardware updates became scarce and, over time, the robots became obsolete and discarded.

In order to salvage the Nomad robots, projects like [4]-[6] appeared, that propose to remove the robots obsolete parts and replace them with more modern equipment, while maintaining the overall design and functionality.

This article presents our ongoing effort to rehabilitate a Nomad Superscout robot. Our goal was to completely redesign the robot's internal control architecture, but to salvage the original body frame, sensors and motors, all still in working order.

The rehabilitation of the robot has several advantages. First of all, it is cheaper than buying a new robot of similar characteristics, and secondly, there are educational benefits in such a project. The rebuilt Superscout can be used as an "off-the-shelf" commercial mobile robot for testing and implementing high-level navigation algorithms (obstacle avoidance, environment mapping, etc.), but students can also tweak, modify and expand the robot's internal hardware and firmware, since all the components, schematics and design options are known and documented (in a way, similar to open-hardware and open-source robots like E-puck [7]).

However, serving this dual purpose is not easy, because maintaining ongoing modifications in the low-level hardware/firmware often means that the system is not always fully functional (or compatible with previous software) for those who merely wish to use it in a high-level perspective.

To minimize system down time and maximize usability, we follow a modular and distributed architecture. In that way, new hardware (or even software) components can be easily added and removed with minimum disturbance to robot's other functionalities. This type of architecture has been successfully applied to high-performance mobile robots and its advantages verified [8],[9].

We based the new robot in the control architecture developed for the SkyGuardian Unmanned-Aerial-Vehicle (UAV) [10],[11]. The vehicle's sensors, actuators and controllers are all independent modules and communicate through a backbone control network. The main difference in the overall design is that, unlike the UAV, the new Scout is equipped with a single onboard computer and there is no controller redundancy. Many hardware components of the SkyGuardian's control network were used without modification, while others were slightly adapted to fit our particular needs.

The way the robot's control software is structured has also changed in respect to the original Scout. The onboard computer is now running a real-time kernel and we are currently rewriting the low-level sensor/actuator interface functions to run as real-time processes. This will enable, for example, the programming of sensor data filters with accurate sampling frequencies even if the user chooses to implement the high-level control as a normal, non real-time process.

Also under development is the integration of the new Scout in the open-source "Player/Stage/Gazebo" (PSG) robot simulator [17]. The integration involves creating the robot's 3D model for the virtual environment and a software layer (device driver) that enables the code written for and tested in the simulator to be used, without modification, in the real robot.

The remaining of the article is structured in the following way: Section II presents an overview of the SuperScout's original hardware. Section III describes the new architecture and the integration of new and existing (salvaged) sensors and actuators. It also outlines the software structure and real-time capabilities used in the onboard computer. Section IV discusses the integration of the robot in the "Player/ Stage/Gazebo" simulation environment. The conclusions are presented in section V.

II. THE ORIGINAL NOMAD SUPERSCOUT

The original SuperScout was a differential drive mobile robot, with a roughly cylindrical shape of 40cm of diameter by

E. Andrade and F. Caetano are undergraduate students of Electrical Engineering at the School of Technology and Management, Polytechnic Institute of Leiria, Portugal (emails:{ee17122,ee17554}@student.estg.ipleiria.pt). J. Quaresma is a researcher at the Telecomunications Institute (IT), Portugal (email:jose.quaresma@co.it.pt). L. Perdigoto and L. Bento are with the Department of Electrical Engineering at the School of Technology and Management, Polytechnic Institute of Leiria, Portugal (emails:{luis.perdigoto, luis.bento}@ipleiria.pt).

36cm of height, and powered by two 12Volt lead acid batteries. It was equipped with a tactile bumper ring and an ultrasonic range finding system composed of 16 Polaroid sonar sensors. Odometry was achieved by the use of encoders mounted on the electric motors, which provided velocity feedback for each wheel.

The high-level controller was a 200 MHz Pentium II industrial PC that ran a Linux operating system. The PC used a serial port to communicate with the low-level hardware microcontroller, a Motorola MC68332.

The robot could also be equipped with a video camera by using a PCI frame grabber installed on the onboard computer.

III. A New Scout

All the internal electronic components of the original SuperScout were removed from the robot. After testing, it was concluded that the two motors (and associated gear and wheel assembly), the bumper ring and the sonar sensor array were in working order and could be maintained. The robot's body frame was unaltered, with the exception of a few new holes drilled on the top lid to provide additional external connectors, indicator leds and buttons.

The system's power comes from two 12Volt batteries, as in the original robot. The power from the batteries is regulated to 5Volt, 12Volt and -12Volt and distributed throughout the system. A monitoring circuit outputs the current battery charge level to the user, using three different color LEDs and one buzzer alarm. The new robot can be seen in Fig. 1.

Fig. 1. The New Scout. In the left image, the SuperScout's top and bottom lid are open, revealing its new interior hardware. On the right the posterior battery bay is also open, showing the circuit-board dedicated to power distribution sided by the two 12Volt batteries.

A. The Backbone Infrastructures

The computer, the sensors and the actuators are viewed as independent modules by the control architecture. Information and commands are passed between each other through a CAN (Controller-Area Network) bus.

1) The CAN network

Originally developed by BOSCH for automotive applications, the CAN standard allows high speed, reliable and bidirectional communication in a multi-master serial bus [12]. All messages are broadcasted and do not explicitly possess an origin or destination address. Each node must decide to process, or not, the information based on the message's identification (ID) number. This addressing scheme allows for simple integration of new nodes, without requiring hardware and software updates of the remaining nodes.

The new Scout is equipped with two independent CAN buses. The onboard computer connects to both buses simultaneously, while the remaining nodes (sensors and actuators) can be placed on either one (see Fig. 2). This design allows for the division of the data rates and physical separation of nodes, and increases the expansion potential of the system.

A dedicated circuit board provides 6 node slots (connectors) for each bus. The number of nodes in the network is, of course, not limited to the number of slots and, if necessary, can be easily expanded by additional wiring.

The node connectors also provide, in addition to access to the bus, a 5Volt supply voltage, to directly power the nodes, and a global reset signal used to reboot the microcontrollers that manage the nodes (further explained below). This reset signal is activated by a push button available to the user on the robot's top lid.

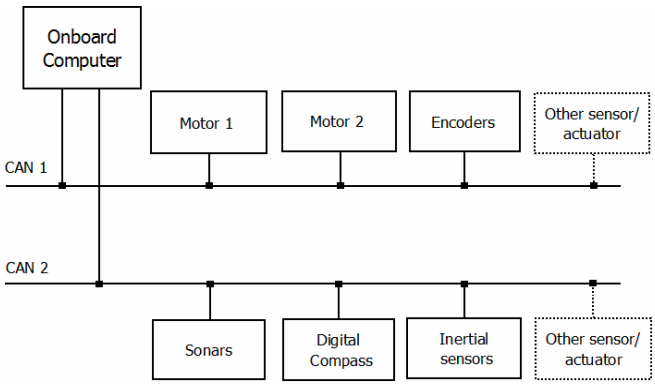

Fig. 2. The CAN network. The diagram shows the two independent CAN buses that connect the onboard computer to the robot's sensors and actuators.

2) The CAN Nodes

To enable the simple integration of new hardware (sensor/actuator) in the network, a "general purpose" CAN-node circuit-board was developed. The board relies on the PIC18F4580 Microchip microcontroller that, at the same time, communicates with the CAN network and interfaces the hardware device.

The microcontroller on the CAN-node can be programmed with varying degrees of intelligence. It can simply bridge the data between the device and the network, perform some data preprocessing or even implement a local low-level control loop when connected to an actuator.

Fig. 3 shows the CAN-node structure. A connector fits directly into the slots available in the CAN bus circuit-board, providing the microcontroller access to the network, the global

reset signal and to the power supply. It is also through this connector that the microcontroller is programmed, when detached from the robot.

The microcontroller interfaces with the hardware device using a second connector, which links to some of its I/O pins. These pins have overlapping functions, that can be defined in the software. The following I/Os are available (in a maximum of 8):

- 3 Digital I/O; or 1 I2C protocol port; or 1 SPI protocol port
- 3 Digital I/O; or 3 Analog Input
- 2 Digital I/O; or 1 Serial port

The supply voltage from the CAN bus is also available to the sensor/actuator device.

The circuit-board features a 6 bit DIP-switch to enable quick changes between predefined modes of operation without the necessity of reprogramming the microcontroller. We are currently using this capability to enable interchangeability between the various CAN-nodes. This is possible because each node has the same program, which includes the specific code for every sensor. The correct sensor is specified by the DIP-switch, using a predefined identification number. A malfunctioning CAN-node can, thus, be quickly replaced if there is a preprogrammed spare circuit-board available.

In the case of the onboard computer, the communication with the CAN buses is accomplished through a PCM–3680 Advantech board, specific to the PC/104 standard. This particular board supports dual channel communication and enables the computer to simultaneous connect to the two independent buses.

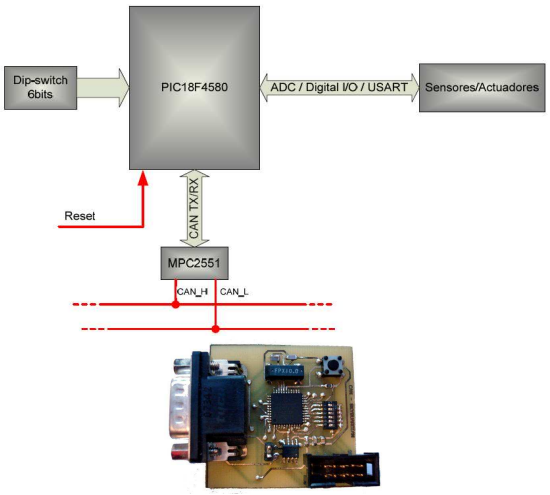

Fig. 3. The "generic" CAN-node. The diagram on the top shows the internal structure of a CAN-node. A PIC microcontroller interfaces the CAN network and the hardware device (sensor/actuator). A 6 bit DIP-switch allows the user to quickly change modes of operation. The picture on the bottom shows the CAN-node circuit-board. The left connector interfaces the CAN-network while the right connector interfaces the hardware device.

B. *Actuators and Sensors*

In this subsection we overview the salvaged and new sensors and actuators, and discuss their integration in the robot.

1) *Motors and Encoders*

The original motors, reutilized in the new Scout, are Pittman permanent magnets DC motors, equipped with a gearhead reduction box and an incremental encoder mounted on the backshaft. The encoder provides two pulsed signals, 90º off-phase to identify the direction of motion. It has a resolution of 512 pulses per turn, which combined with the reduction in the gearhead and in the transmission belt result in 12083 pulses per wheel revolution.

Each motor is associated to a Maxon ADS-50/5 motor driver. It establishes a closed-loop velocity control using encoder feedback, and receives an analog voltage signal as the reference (set-point).

The encoder feedback is also very important for higer-level control functions, such as odometry calculations. A CAN-node circuit-board is connected to both encoders and processes the encoders' pulsed signals, before outputting the computed velocities to the CAN-network. In this case, a modified version of the "generic" CAN-node was used. It follows the same basic design but provides optocoupler isolation between the signals.

2) *Bumpers*

The SuperScout's bumper ring is composed of 6 independent pressure switches located around the robot's perimeter. It was decided to use these switches mainly as a safety feature and not as an environment sensing tool. A dedicated safety circuit disables the motor drivers whenever one of the bumper switches is activated. The motors can only be re-enabled by pressing a push button located on the robot's exterior.

If desired, however, this behavior can be easily changed by disconnecting the bumper switches from the safety circuit and outputting their individual state to the CAN-network, using a "generic" CAN-node (that can handle up to 8 digital signals). In this situation, the control software on the onboard computer must take responsibility for reacting in the event of a collision.

3) *Sonar ranging system*

The SuperScout has 16 ultrasonic transducers equally distributed around the robot's body. In the original robot, the transducers were driven by a dedicated circuit-board, based on the SensComp 6500 sonar ranging module. Since the 6500 module can only drive one transducer at a time, the circuit-board multiplexed the sonars using 16 relays switches (a more detailed description of the board's operation can be found at [4]). The SuperScout's sonar ranging circuit-board is shown in Fig. 4.

After some debugging and a replaced IC, the original board was salvaged and is currently being used on the new Scout, connected to a CAN-node. The microcontroller in the CAN-node operates the board by, first, selecting one of the transducers using the multiplexer, secondly, commanding the SensComp 6500 module to start the ultrasound burst and, finally, measuring the time it takes to receive an echo (time-of-flight).

The CAN-node is programmed with several different modes of operation. It can be commanded by the onboard computer to make a single measurement on a specific transducer or it can

be instructed to start operating continuously, sweeping repeatedly the sonars in sequence while "dumping" the information to the CAN network.

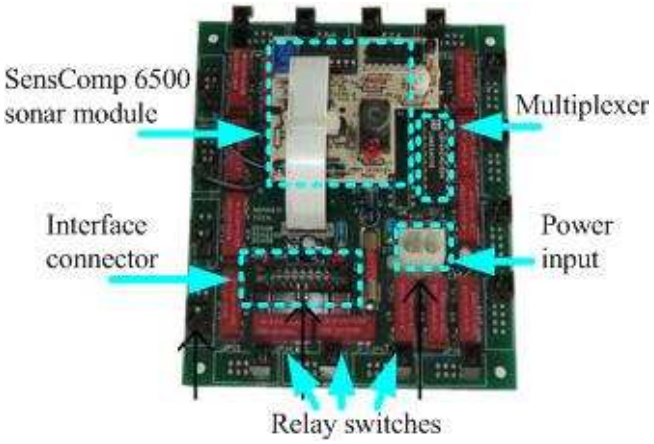

Fig. 4. The sonar ranging system circuit-board. The circuit-board, salvaged from the original robot, drives the 16 transducers by multiplexing a single SensComp 6500 sonar ranging module.

4) Inertial Measuring Unit and Compass

In the original robot, dead reckoning was achieved using odometry only. Our new Scout, however, is currently being fitted with an Inertial Measuring Unit (IMU) and a magnetic compass, which will increase its potential as an educational and research tool in areas like navigation, sensor fusion or data filtering.

The IMU consists of three Analog Devices ADXRS150 rate gyros, with a maximum sensitivity of $\pm 150^{o}/s^{2}$, and two Analog Devices ADXL320 accelerometers, ±5g of maximum range. The three gyros are mounted with the measuring axes orthogonal to each other, sensing rotation in 3D. In the case of the accelerometers, each chip measures acceleration simultaneously in two perpendicular axes, so only two chips were required to cover the three dimensions (see Fig. 5).

The sensors' output consists of analog voltage signals. Presently, we are using one "generic" CAN-node for the gyros and another for the accelerometers (three analog channels per CAN-node). Pending further testing, we are considering the creation of a specialized CAN-node that can handle the entire IMU simultaneously, so that signal sampling times can be synchronized for all the sensors.

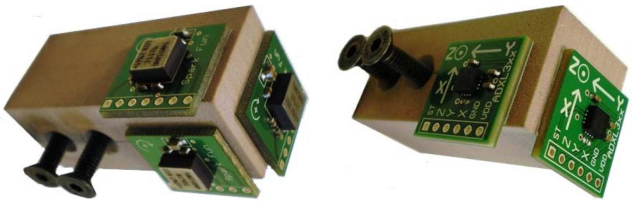

Fig. 5. The Inertial Measuring Unit (IMU) prototype. The Fig. shows the gyros (left image) and the accelerometers (right image). In the case of the gyros, each chip senses one axis. In the case of the accelerometers each chip senses two perpendicular axes simultaneously.

A 2D IMU would, of course, be sufficient for the robots' normal operation in in-doors, flat surface environments. The marginal cost of adding a third dimension is, however, justified by the gain of obtaining a fully capable IMU that can, potentially, be used in a wide range of other applications or vehicles.

In addition to the IMU, we are also using a Devantech CMPS03 magnetic compass module. It is intended to aid navigation by making the external reference provided by the earth's magnetic field available to the robot. The compass is interfaced by a "generic" CAN-node using the I2C protocol.

C. Operating System

The sensing and control algorithms have evolved and became complex. This evolution has increased as well the complexity of embedded systems projects. Consequently it required new levels of abstraction in software solutions that can interact with the hardware as efficiently as possible. In order to make the robotic platform has generic as possible so that the majority of algorithm could run in this robot, it was installed a PC with a Linux operating system. The chosen Linux distribution for the PC was the Gentoo, mainly because it is portable, easy to maintain, it has a large number of packets and it is very flexible and configurable [13]. Due to this adaptability, Gentoo is often referred as a meta-distribution. To increase the level of performance achieved by the computer the graphical component of the software was not installed and a real-time patch was added. The inclusion of this patch enabled a simple, yet effective, system in real time, providing a means of running on a GNU/Linux environment and a RTOS, side by side on the same hardware.

The PC may integrate the middle layer and top layer of the three layer architecture, where:

- the bottom layer is composed by the CAN nodes;
- the middle layer is composed by the real-time OS and the PSG server;
- the top layer is composed by the PSG client, running the higher level control algorithms.

The middle layer unit does not possess a global view of the system where it operates on. This global view will be provided by the PSG server to the PSG client, described in section IV. The PSG server and client may be both running on the PC/104 computer or in a distributed manner, being the PSG server on the PC/104 and the PSG client on an external PC.

1) Onboard Computer

The selection of the computer was driven toward a small size computer, low consumption, medium processing capacity and availability of both PCI and ISA buses. The PC/104 standard was chosen for the onboard computer, because of the previously mentioned reduced dimensions requirement, low power consumption and its modular design, meaning that it can be easily connected to other circuit boards.

These requirements led to the selection of a PC/104 "speed MOPS lcd PM" computer. The computer dimensions are 96x147 mm with a power supply of 5V, it has a fanless Intel ULV Celeron M 1.0GHz 373 (Dotam), 512KB L2, integrated graphic card Intel Extreme; 2 x USB 2.0; 1 x 10/100 Base T Ethernet; PC/104 (ISA) and PC/104plus (PCI). With the USB ports one may use a WirelessLAN or Bluetooth USB dongle to perform OTA (Over The Air) control.

The computer is equipped with a conventional hard drive but is prepared to use a solid state disk, a ISA dual CAN BUS board Advantech PCM 3680, enabling communication speeds up to 1Mbps. This board is equipped with a Philips SJA1000 controller and its connector is a standard ISA PC/104. This features where mandatory requirements since SJA1000 is a well supported controller under real-time operating systems.

2) Real-time processing

To increase the reliability of this system the PC/104 is running a real time OS. First we used a kernel (core system), with a RTAI (Real Time Application Interface) patch that allows to develop software running in real-time [14]. The code to communicate over the CAN network had to be ported to RTAI or a generic driver (can4linux) had to be used, which was not developed to work in real-time. The solution was to install a new kernel with a new patch "Xenomai" (Real-Time Framework for Linux) that contains real-time CAN drivers [15]. Xenomai also enable threads to run in real-time within the kernel space or within the user space, the space of a process in Linux. As a result the Xenomai threads fully support the Linux features (e.g. support for GDB), which is one of the major benefits for the use of Xenomai. Xenomai patches include ADEOS (Adaptive Domain Environment stands for Operating Systems), a nanokernel that allows multiple entities to exist simultaneously on the same machine, not necessarily seeing and sharing the same resources [16]. The key advantage of ADEOS is its ability to export a generic API to the kernel space, which is used by Xenomai afterwards.

This patch creates an abstract layer to Linux, which launches and execute commands for the real time tasks. To prevent Linux from interrupting real time tasks, the Linux operating system is launch as low priority interrupt of the micro kernel.

The kernel has a generic configuration that allows it to run on most hardware systems, but this makes it too large and with unnecessary features. In addition it may be the cause of some instability to Xenomai. Therefore we removed everything that was not necessary, e.g. support to various communication networks, file systems, SPI, RAID, etc. We also removed some features in the kernel that introduce high latencies when the system is working in real-time, e.g. APIC, APM, and modeling CPU frequency.

We developed software to automate the loading of kernel modules and to configure and test the CAN communications. It was also implemented software to send and receive CAN messages. This software runs on a thread in real time on the Linux environment which in turn communicates with the CAN driver in the Xenomai environment. We will develop an integrated real-time robotic environment software with several threads, running at appropriate frequencies with the proper priorities.

D. Expansion Capabilities

The new Scout's capabilities and functions can be expanded in several ways. At the low-level hardware, new sensors and actuators can be easily added to the system by using the CAN network infrastructures. Although we have only discuss the integration of new sensors (like the IMU), new actuators (e.g.101 a kicker for robot soccer) would, in the same way, be controlled locally using the CAN-node and by the onboard computer trough the CAN network.

At a higher level, off-the-shelf external devices can connect to the onboard computer though its USB or Ethernet ports. Using the appropriate USB adapter, the onboard computer can be Bluethoot and Wi-Fi enabled and communicates wirelessly to other devices or computers.

Some devices, like video cameras for example, may generate high volume of data and require more processing power than the onboard embedded computer can provide. In these situations, a laptop computer can be mounted on top of the Scout to process the information and, if desired, perform high-level control.

IV. INTEGRATION IN THE PLAYER/STAGE/GAZEBO SIMULATOR

Player/Stage/Gazebo (PSG) is a free open source software created for research in the area of robotics and sensor systems [17]. This software is divided in three distinct application software's:

PLAYER – is a network server, establishing connections through the TCP port, allowing the user to control the robots at his disposal, both real and virtual.

STAGE – is a two dimensional robot simulator that makes possible for the user to visualize the robot's movement in a 2D world previously designed, and to test the robot's programming.

GAZEBO – is a three dimensional robot simulator that allows the user to visualize the effects of the developed algorithms and programming.

One way to build programs to control the rehabilitated SuperScout is to directly communicate with the CAN nodes and actuate or sense via low level functions. This method has many disadvantages, among them are the necessity from the programmer to have the knowledge of low level architecture to implement a high level controller. The high level controller code developed in this way is non portable to other robot architecture, view Fig.6.

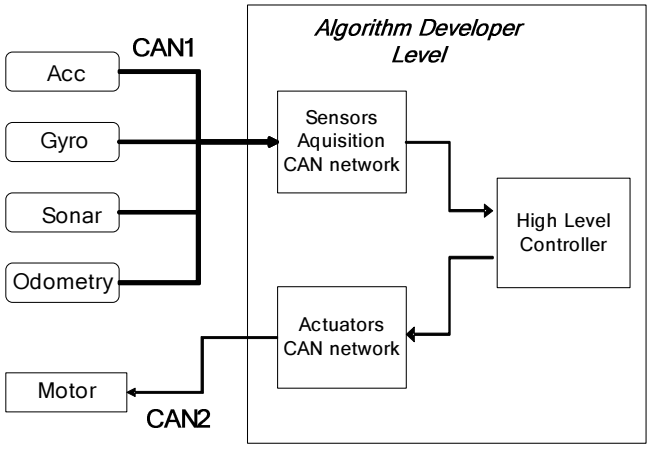

Fig. 6. Robot Architecture without PSG. Control the robot by directly communicate with the CAN nodes and actuate or sense via low level functions.

An alternative way that avoids many of the above mentioned problems is to build the high level controller using the Library Player. The structure of the Player is based on client/server model. The server interfaces with the robot sensors, obtaining data, sending them to the client and receiving instructions from the client to control the robot. The player implements an abstraction layer between the high level controller (client) and the robot (server), hence the programming is made for a generic robot, view Fig.7.

All communication between client and server are done through TCP/IP. The client enables the development of algorithms in several code languages: C, C++, Java, Python, Tcl. These features enable a broadband of developers and algorithms, reducing the development time.

This project will include, in its further stages of development, the creation of a virtual version, representing the rehabilitated SuperScout. Creating a new virtual robot requires the user to create the model, the sensors and controllers used by the real robot.

In order to be able to use the real robot and use benefits of the PSG abstraction layer, a PLAYER driver [18] and a GAZEBO physical model have to be developed. We are currently starting the creation of both driver and physical model.

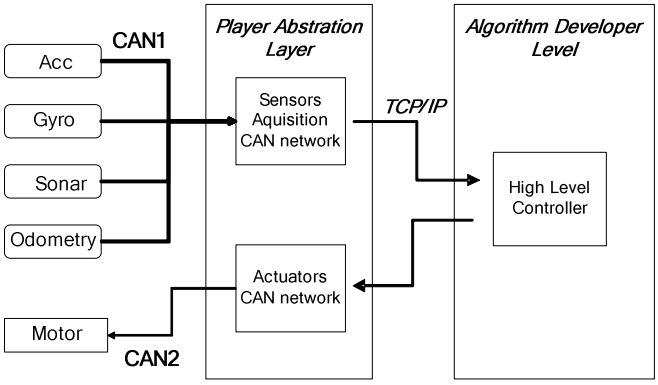

Fig. 7. PSG Robot Architecture. Player is a hardware abstraction layer for robots. This abstraction layer enables the interaction with real or simulated hardware.

A. PLAYER Driver

The PLAYER driver will provide the following data:

- *position2d* - provides an interface to control the linear and angular speed command, it also returns the odometric based pose;
- *sonar* - provides an interface to the sonar range sensors;
- *bumper* - provides an interface to the bumper sensors;
- *IMU* - provides an interface to calibrated acceleration, gyro and magnetic values.

The Player configuration file for this robot will be similar to the following:

```
driver
 (
        name "new_scout_driver"
        plugin "newlibscout.so"
        provides ["position2d:0"
                  "bumper:0" "sonar:0"
                  "imu:0"]
        portsensor   "/dev/can0"
        portactuator "/dev/can1"
)
```

The driver created will be installed on the PC/104 where player will be running, in real-time. Heavy computation algorithms using sensors or actuators such as image processing will not be included in the PLAYER driver being developed. This type of heavy computation process will run on an external PC.

B. GAZEBO Model

For the GAZEBO (3D Simulator) the creation of a model consists in creating a replica of the physical structure of the robot, sensors and actuators. This is done by creating the basic shape, which among other things, defines the measurements, the starting position of the robot, and the skin and the material chosen for the model.

There are several guidelines that should be followed during the creation of a GAZEBO model:

- Everything should be named;
- A model must contain a body;
- A model can contain only sensors and\or geometries.

After the physical structure is developed, the user should create the model joints. This small piece of code is of great use, since it allows the integration of wheels into the model.

The sensor modules in GAZEBO, are designed to simulate the data provided by real-world devices. As such, sensors do not have a physical representation, and to create a sensor for a new model, the user needs only to copy the sensor class to the new model directory. Afterwards the sensor must be loaded to the project in development.

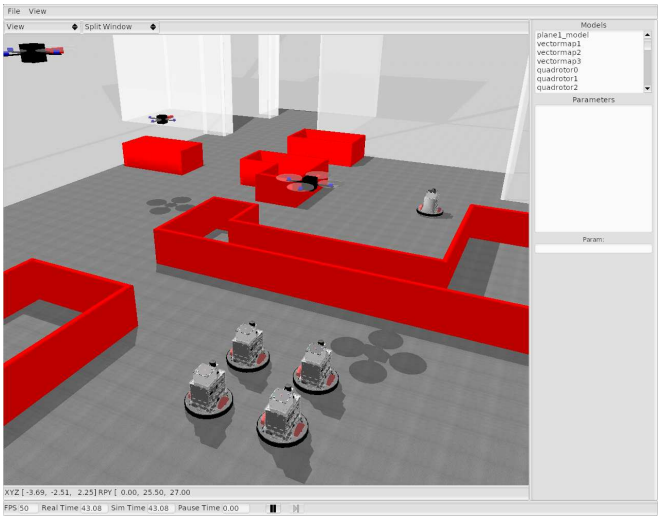

Fig. 8. Robotic environment simulated using GAZEBO. Gazebo is a 3D robot simulator, that allows the user to visualize the effects of the developed algorithms and programming.

In order to be able to move the joints, and to receive commands from the user, a controller must be created. The initial step is to decide which type of controller to used and store it accordingly. For example, camera controllers should go into the camera directory. When loading the controller the user should ensure one or more interfaces. Fig.8 presents a possible robotic environment simulated using GAZEBO.

V. CONCLUSIONS

This article presents our efforts in rehabilitating a Nomad Superscout mobile robot. The internal control hardware was completely removed and only the robot's original body frame, sensors and actuators reutilized. The new robot features a modular and distributed architecture, based on a CAN network. Every sensor and actuator is directly interfaced by a local microcontroller that communicates with the onboard embedded computer using one of two independent CAN buses available in the robot.

The new architecture is easily expanded and updated and makes repairing and replacing existing hardware simpler. The robot can also be connected to external devices or computers (wired or wirelessly) to perform complex and processor-heavy tasks (like image processing).

To further improve the robots capabilities, we are currently working on applying a real-time software architecture to the onboard computer and developing drivers to integrate the system in the Player/Stage/Gazebo simulator.

The rehabilitation of an obsolete and inoperative SuperScout has economical and educational advantages. The new robot becomes an open-hardware open-source platform and can be applied to a wide range of projects.

We hope that some of our design choices will be useful to others in the rehabilitation of Nomad robots or even in the design new robots. Pursuing that goal we plan to make the hardware and software documentation available on the WEB in the near future.

ACKNOWLEDGMENT

The authors would like to thank Carlos Arsenio and Andre Rodrigues and everyone involved in the SkyGuardian project for their invaluable contributions and sharing of know-how.

REFERENCES

[1] E. P. Silva Junior, P. M. Engel, M. Trevisan and M. A. P Idriart. *Autonomous learning architecture for environmental mapping.* Journal of Intelligent and Robotic Systems, 39:243–263, 2004.

[2] N.C. Tsourveloudis, K.P. Valavanis, and T. Hebert. *Autonomous vehicle navigation utilizing electrostatic potential fields and fuzzy logic*. IEEE Transactions on Robotics and Automation, 17(4):490–497, 2001.

[3] Araujo, R. de Almeida, A.T. *Path planning-by-learning with a Nomad 200 mobile robot* Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE97), 1997.

[4] Arjun Chopra, Mark Obsniuk, Michael R. M. Jenkin. *The Nomad 200 and the Nomad SuperScout: Reverse engineered and resurrected*, Canadian Conference on Computer and Robot Vision (CRV06), 2006.

[5] Mattias Lindstrom. *Nomadic 200 hardware guide.* Available at "http://www.nada.kth.se/~mattiasl/nomad200/".

[6] Francesco Monica. *Nomad*. Available at "http://rimlab.ce.unipr.it/".

[7] Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D. and Martinoli, A. *The e-puck, a Robot Designed for Education in Engineering*. Proc. of the 9th Conference on Autonomous Robot Systems and Competitions, 2009

[8] Azevedo, J.L., M.B. Cunha, L. Almeida, *Hierarchical Distributed Architectures for Autonomous Mobile Robots: a Case Study*. Proc. ETFA2007- 12th IEEE Conference on Emerging Technologies and Factory Automation, 2007.

[9] Kopetz, H., *Real-Time Systems Design Principles for Distributed Embedded Applications*, Kluwer, 1997.

[10] S. Faria, P. Assunção, N. Rodrigues, L. Mendes, C. Ribeiro, P. Ventura, S. Silva, C. Lopes, C. Silva, J. Pereira, L. Nero. *Sistema de Controlo e Comunicações de uma aeronave não Tripulada: SkyGu@rdian*, 2ª Conferência da Engenharia; UBIengenharias, Covilhã, Universidade da Beira Interior, 2003.

[11] Isaac Duarte, Sense and Avoid Collision System for UAV, MSc, Univ. Trás-os-Montes, Portugal, 2009.

[12] Bosch. *Controller Area Network (CAN) specification – version 2.0, part A.* Bosch, GmbH, Germany, 1991

[13] *Gentoo Linux.* Available at "http://www.gentoo.org/"

[14] *RTAI - the RealTime Application Interface for Linux from DIAPM.* Available at"https://www.rtai.org/"

[15] *Xenomai: Real-Time Framework for Linux.* Available at "http://www.xenomai.org"

[16] *The Adeos Project.* Available at "http://home.gna.org/adeos/"

[17] *The Player Project: Free Software tools for robot and sensor applications.* Available at "http://playerstage.sourceforge.net"

[18] Renato F. Garcia. *Player driver for e-puck robots.* Available at "http://code.google.com/p/epuck-player-driver/". Lab. de Visão Comp. e Robótica - VeRLab, Univ. Federal de Minas Gerais, 2008.