A Multi-Layer Distributed Development Environment for Mobile Robotics

Gregory Dudek
Research Centre for Intelligent Machines
McGill University
Montreal, Quebec, Canada

Michael Jenkin
Department of Computer Science
York University
North York, Ontario, Canada

Abstract

Mobile robotic devices combining sensors, actuators, and computers are unique, complex devices which may be difficult to model from an abstract point of view. This paper presents a software development system which builds an abstraction of a robotic environment. This abstraction allows external software to interact with either a simulated robot and environment or to a real robot complete with sensors. The implementation is distributed across a network, and allows software to run on remote hardware thus taking advantage of any specialized hardware available on the network.

This paper appears in the proceedings of the *International Conference on Intelligent Autonomous Systems (IAS-3)*, Feb. 1993, Pittsburgh, PA, pp. 542-550.

1 Introduction

Robotic systems (and in particular mobile robotic systems) embody a complex interaction of computational processes, mechanical systems, sensors, and communications hardware. The problems inherent in the system integration can be very challenging. The hardware is often designed more for "ease of design" rather that to assist with the system integration. For example: robot platforms operate in encoder counts, sonar sensors return clock ticks to the first returned echo while hardware interfaces are usually very specialized. Even if the devices have "standard" interfaces such as through VME or serial connections, or via some other "standard" protocol, the actual command stream is very device specific. These factors impede abstraction, portability, rapid prototyping, modularity and pedagogy. Software developers must write layers of code to divorce the low level interface from higher level software functions. As this layer is required for each software device, a large amount of code must be developed before the first "robotic" application can be run. Failures in higher level software can often result from unidentified bugs in this lower layer, and these bugs can be very difficult to detect.

As robotic software is highly device dependent, the system is also highly dependent on the physical availability of the hardware. This can become an issue when multiple groups are developing software for the same devices or when duty cycles are short. There are typically not enough robots to go around, or not enough space for the robots to operate in. Likewise, without controlled environments, repeatability of experiments is an issue. It is also not practical to test preliminary versions of robot control software on the real hardware for safety reasons. Robotic hardware is as weak as the weakest link and hardware failures in complex robotic systems are common. Accidentally driving the robot into stationary structures enhances the likelihood of breakdown. Not only is an abstract definition of the robot desirable, a simulation of this abstraction is very useful during the algorithm development and testing stage.

In recognition of these facts, we have developed a graphical environment for the development of algorithms and software for mobile robotics. The environment was produced as a result of an awareness that progress in mobile robotics entails a set of stages starting from basic experimentation and “fooling around” and progressing through to the development of efficiently-implemented algorithms. With this progression in mind, we have developed a control and development interface for mobile robotics experiments that permits a single robot to be controlled and/or simulated using any combination of manual experimentation, simple interactive programming and distributed computation using heterogeneous computers. The control and communication structures for a particular robot are abstracted (transparently) by the interface. The robot currently used with this system is an RWI B-12[4] equipped with a ring of 12 sonar sensors[5] (utilizing the Polaroid sonar unit[3]), controlled via a radio modem link.

Many of these concepts relating to incremental development and simulation are well established in other contexts, and the concept of an environmental simulator/controller for manipulator robots is commonplace (for example, see [1, 2]). Results from manipulator simulations suggest that for the simulation to be effective, it should accurately model as many of the real world effects as possible.

2 Functional Design

The development environment we have constructed is based on a single primary software module known simply as the robot daemon. This module forms a robot-independent gateway for all communication to the physical robot. Alternatively, it allows the physical robot to be “replaced” by a simulation. For such a simulator to be useful, it must be able not only to simulate the robot itself, but also the sensory feedback the robot receives. The daemon is capable of simulating the behavior of the robot and its sensors in the context of a prespecified simulated environment. At present, only sonar sensing is used aboard the robot and two different simulations are available providing differing levels of realism. A visual simulator is also in progress. The sonar simulation can be performed with an idealized sonar model that is much simpler than actual sonar sensors, or with a particularly realistic simulation of two dimensional time of flight sonar sensing[6, 7]. This simulation correctly models sonar diffraction effects, side lobe scatter, as well as multiple bounce specular effects.

The daemon also provides a graphic user interface (for X-windows) that displays the robot's current operating parameters, the data recently acquired, and the deamon's "view" of the world. The graphical environment also provides an editor to create and modify simulated environments.

The user may interact directly with this level of the interface by clicking buttons and pointing to locations in the world to move the robot and/or change its state. It is this level of the interface that provides the simplest abstraction of the robot for experimentation. Any changes to the robot's state are displayed using this graphic interface.

The robot daemon acts as a "service server" for the robot with a graphical inspection/interaction monitor. Communication with the daemon is accomplished via the internet. The daemon establishes an internet address and accepts messages (commands) from other processes running on machines connected to the network. The daemon defines an abstract, robot independent, communication stream between user tasks and the physical robot (or its simulation). As the communication stream is identical in either case, the robot controlling task is unaware if the robot that is running is live.

Coupled to the graphics daemon and simulator is a command-line interface that allows the user to interact with the robot using a combination of semantic primitives: primitive robot commands, abstracted robot functions (e.g. *plan a path to location* $(300, 300)$ *avoiding obstacles along the way*), system or interface commands (e.g. redraw the screen or launch a specified UNIX process), and simple control structures (e.g. **conditionals, gotos, etc.**). This allows very simple functions to be programmed interactively. These modules can be dynamically incorporated into the graphic user interface level and subsequently operated via the mouse.

The relationship between the functional components of the robot daemon, including auxiliary processes, are illustrated in Figure (1).

3 Detailed structure

Daemon graphical user interface: The graphical user interface (GUI) associated with the robot daemon can be used to manipulate both the real robot as well as a simulated one. This permits a great deal of flexibility in terms of robot testing and debugging, including such operations as

- Allowing a graphical display of the internal state of the robot, and the raw data stream being returned from the robot.
- Allowing an operator to switch instantly and invisibly from the real robot environment to the simulated one and back again. The user can interact via the GUI even when independent processes are controlling the robot. Thus real and simulated data can be integrated and used to test the controlling software.
- More complex sensors can be simulated.
- Errors can be introduced into the real robot environment (the robot can be moved independently of the controlling software).

A display from the GUI is shown in Figure (2)

The GUI can also incorporate high-level environmental models and can overlay the real robotic data (position, sonar sensor readings) with them. Such data can be manually drawn using the mouse, specified precisely as ASCII coordinates, or generated automatically by another process using the mechanism described in the next section (for example we sometimes use a “wall inferencing” process driven by sonar data). This is illustrated below (figure 3). Once entered, the world model can then be used for sonar simulation or path planning alone on in conjunction with real data acquired from the robot. It can also for verifying the accuracy of the simulation module (by switching to simulation mode and simulating the sensors given the robots current state).

As an aid to debugging, the interface can optionally display the data to and from the robot as it is transferred, display the internet communications taking place, display path planning parameters and interim results, or graphically render the sonar data as it is acquired or simulated.

Daemon internet interface: Inter-process communication between the daemon and other tasks is accomplished via internet sockets (a low level interprocess communication mechanism available under Unix), and the robot daemon operates much like the finger or telnet daemon, accepting valid connections from remote processes. When executing, the daemon monitors a particular socket address on the host machine for connections. Connections from valid hosts (by default, any machine on the internet), are accepted and a serial stream protocol is honored. One minor disadvantage of this socket-based communications protocol is that it imposes a data-dependent communications delay on interactions. For high-level control using sonar data, this is of little consequence, but for high-bandwidth communication (e.g. video data) or low-level control (e.g. local motion control) it would be problematic.

The communication protocol is line oriented, and specific commands permitted by the interface include

- rotate (relative/absolute) - rotate the robot (either clockwise or counterclockwise) a particular number of degrees. The simulation performs this operation to floating point accuracy, the real robots accuracy is limited by its proprioceptive accuracy.
- forwards, backwards - drives the robot a particular distance in either the forward or backwards direction. Once again the real robot is limited by its encoder errors.
- translate (relative/absolute) - moves the robot to a new position either relative to the current position or to an absolute position. For the real robot, the accuracy of this new position, particularly in the absolute mode is restricted by the integration error in the encoders.
- path-plan - plan and carry out a motion from the robots current location to some other specified location.

- return sensor data - return a sensor-dependent quantity for data, for example some number of sonar readings at selected angles.

It has also proven valuable to provide a facility by which a single remote task can have exclusive access to the robot for a sequence of interactions that must be accomplished "atomically". This is provided via a semaphore-like locking mechanism. In addition, this exclusive-use lock times out after a task-dependent interval so that if the requesting task itself fails for some reason, the robot can recover.

In addition to these abstract commands, any GUI command can also be sent via the internet interface, and direct control of the low level interface to the robotic controller is also available (utilized for the most part for debugging purposes).

High level control: A more powerful layer of the interface is supported via UNIX Internet sockets. User programs may be developed using stand-alone code (for example in C or LISP) that communicate via the daemon internet interface. These modules can implement sophisticated algorithms that require substantial bandwidth for communication with the robot. Examples of such a modules include one that performs navigation using a connectionist network and one that provide permits navigation using an English-like syntax ("remember this as home and go to Greg's Office"). These stand-alone modules can operate on any machine (or machines) connected to the Internet and thus can serve as an extremely convenient mechanism for distributed computation. In addition to load sharing, this can be used to provide a certain degree of robustness in operation of the robot. As long as the robots daemon remains operational and the network remains connected, any machine on the network can be used to control the robot.

In particular, the that fact higher-level processes interact with the robot (or robot simulation) via the daemon process means that advantages of the graphic display environment and interactive tools of the daemon are immediately available as diagnostic tools.

In combination, this collection of layers allows a user to develop a robotics application by first examining a hypothesis by moving a simulated robot using mouse clicks followed by graphic or numerical inspection of observed data. Once the basic properties of a problem have been examined, the command-line interface can be used to write simple control loops to elaborate the ideas and perhaps to incorporate them into the graphic interface. Finally an efficient module written in C (or some other high level language) can be developed and executed on any machine on the network to evaluate a fully developed methodology. At any point in this process, the real robot can be connected to the application program instead of the simulator. Although only a limited number of real robots and real robot environments may be available, different simulations can be run concurrently on the network.

4 Discussion

The approach to mobile robotic control presented here has several distinct strengths.

1. The three layer design allows experimentation, teaching and algorithm development to gradually incorporate increasing levels of detail in robot interaction.

2. The daemon interface to the robot allows abstraction of the physical device and should allow portable high-level modules to be constructed.

3. The internet-based task distribution allows load sharing across processors as well as increased robustness.

4. the embedded simulator(s) facilitate development by essentially increasing available resources (providing simulated robots in addition to real robots), as well as by allowing repeatable decomposable experiments to be carried out.

5. The encapsulation of high-level tasks within separate processes facilitates debugging by allow them both have simpler structure and by allowing the daemon to serve as a debugging tool by indicating the state of the robot and sensors by allowing inspection of the data flow to and from the robot.

5 Conclusion

This paper presents a networked controlling daemon for mobile robot software development. The daemon provides a device independent interface to an untethered robot complete with sonar sensors. In addition to controlling a real robot, the daemon also provides a simulated version of the robot, with accurately simulated sensors which can be used during software prototyping, testing, and algorithm evaluation.

This seems very useful and appropriate for high-level low or medium bandwidth interactions with the robot. For higher bandwidth communications, data transmission is currently possible via UNIX files, but more appropriate mechanisms are probably necessary. It may be that parallelism at the level of UNIX processes is not appropriate in such cases, at any rate.

This architecture simplifies code development and encourages rapid prototyping of software. It also provides efficient resource utilization by allowing the physical robot to be simulated hence facilitating off-line development. This is accomplished within a simple framework for distributed computation based on a "thin-wire" communications model. In addition, the gradual increase in complexity of the software layers provides a effective pedagogical environment and hides the low level complexities of communicating with the onboard sensors and actuators of the real (or simulated) robot.

Acknowledgments

Financial support from the Ontario-Quebec Exchange project and from NSERC Canada through individual operating grants to each of the authors is gratefully acknowledged.

References

[1] C. Chen, M. Trivedi, and C. Bidlack. Simulation and graphical interface for programming and visualization of sensor-based robot operation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1095–1101, 1992.

[2] W. Kim. Developments of new force reflecting control schemes and an application to a teleoperation training simulator. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1412–1419, 1992.

[3] Polaroid corporation. *a: Untitled product spec sheet on Ultrasonic ranging system, b: "Ultrasonic ranging system" users manual.*

[4] Real World Interface, Inc., P.O. Box 270, Main St., Doublin, NH. *B12 Mobile robot base: guide to operations.*

[5] Real World Interface, Inc., P.O. Box 270, Main St., Doublin, NH. *G96 Sonar Board: guide to operations.*

[6] D. Wilkes, G. Dudek, M. Jenkin, and E. Milios. A ray following model of sonar range sensing. In *Proc. Mobile Robots V*, pages 536–542, Boston, 1990.

[7] D. Wilkes, G. Dudek, M. Jenkin, and E. Milios. The simulation of sonar mapping in complex environments using multiple reflecting surfaces. In *Proc. Vision Interface 91*, pages 213–217, Calgary, Alberta, 1991.

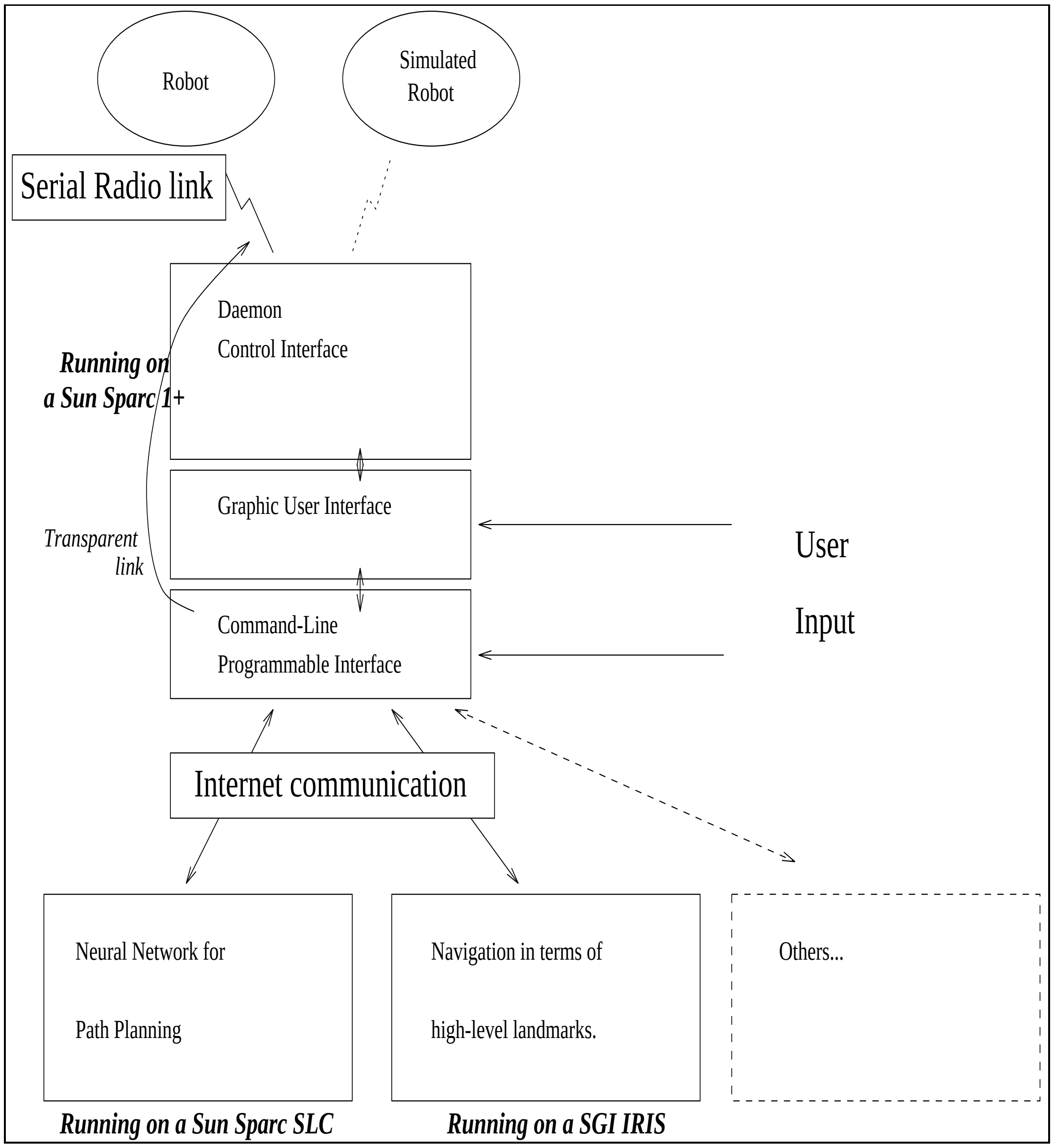
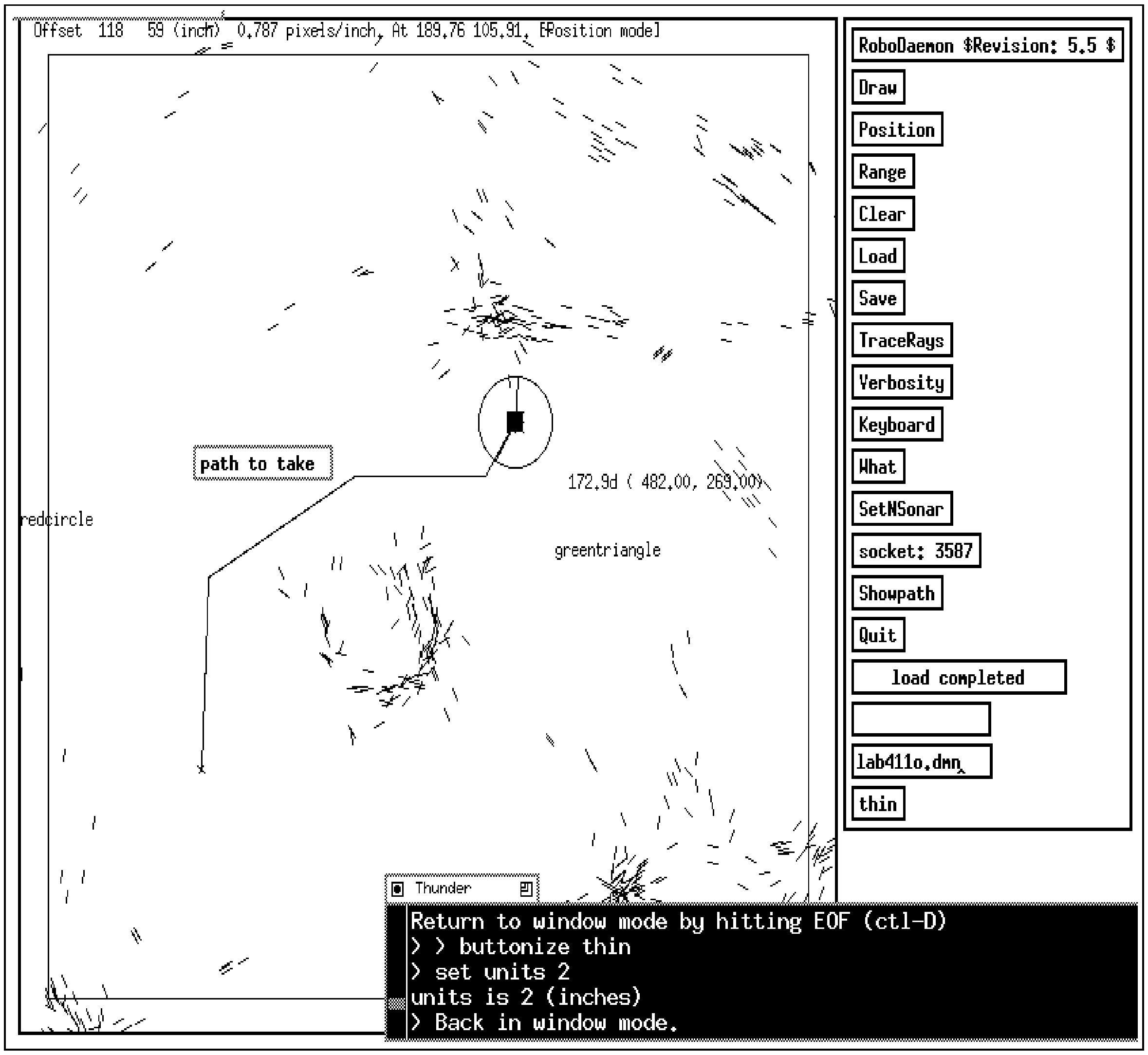

Robot
Simulated Robot
Serial Radio link
Daemon Control Interface
Running on a Sun Sparc 1+
Graphic User Interface
Transparent link
Command-Line Programmable Interface
User Input
Internet communication
Neural Network for Path Planning
Navigation in terms of high-level landmarks.
Others...
Running on a Sun Sparc SLC
Running on a SGI IRIS

Figure 1: Functional units of the interface



```
Return to window mode by hitting EOF (ctl-D)
> > buttonize thin
> set units 2
units is 2 (inches)
> Back in window mode.
```

Figure 2: Graphical display from main module.

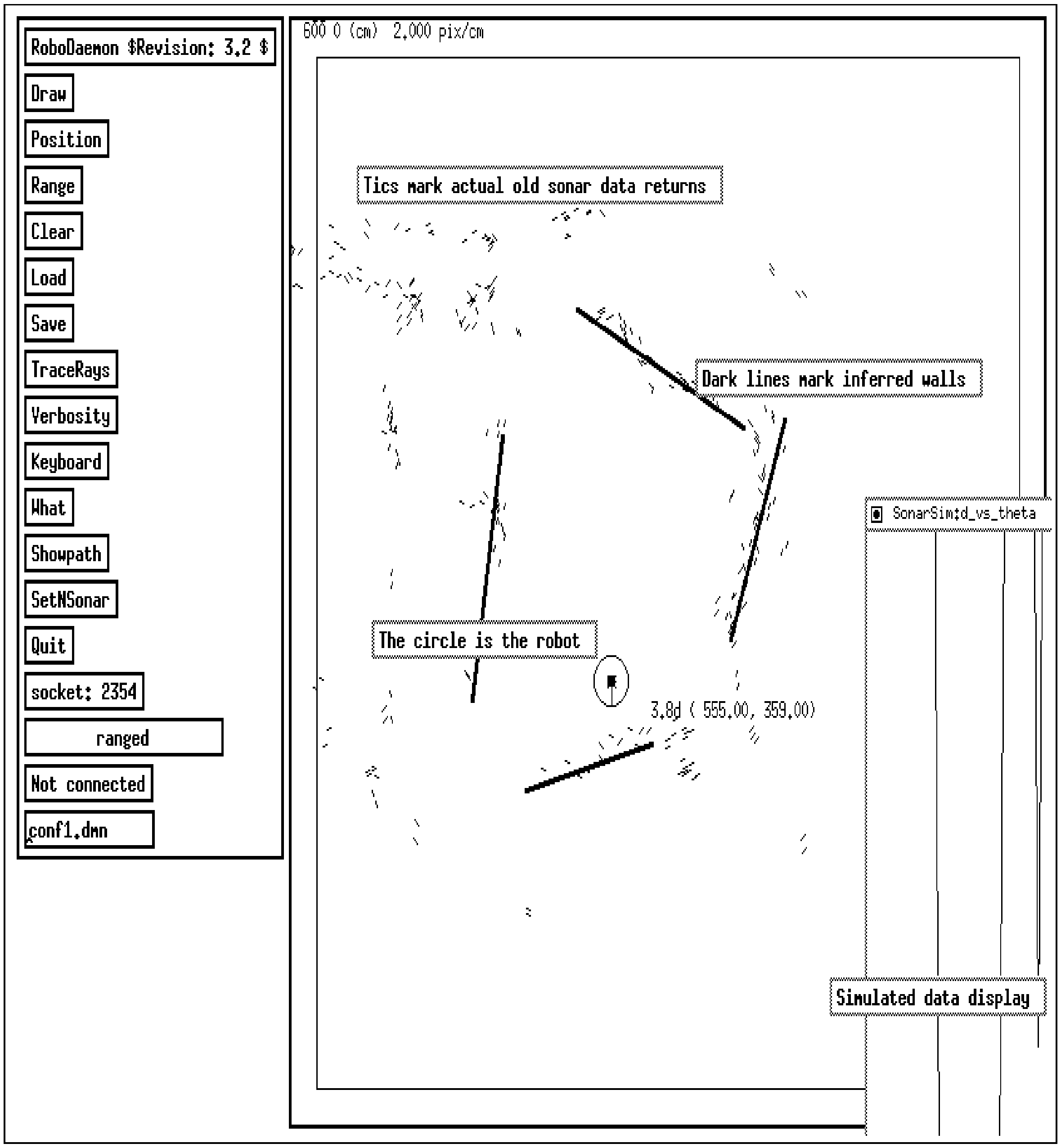


Figure 3: Graphical display from main module.