NOMAD SCOUT USER'S MANUAL

July 12, 1999
Software Version: 2.7
Part Number: DOC00004

Nomadic Technologies, Inc.
2133 Leghorn Street
Mountain View, CA 94043
TEL 650.988.7200
FAX 650.988.7201
Email: nomad@robots.com

WHERE CAN I GET HELP?

1. By email: send a description of your problem, if possible with the source code of your program, to nomad@robots.com.
2. By phone: call +1.650.988.7200 and ask for Technical Support.

WHERE CAN I GET SOFTWARE?

For the convenience of timely software distribution, Nomadic has set up an FTP site for this and future Nomadic software releases. From this FTP, you can down load the most up-to-date software distributed by Nomadic.

In addition, we have setup a directory for you to up-load software that you want to share with other Nomad users.

To down load (or up-load) software from this FTP site, you can simply FTP to ftp.robots.com

Name: robots Password: N0mad1C

Note the 0 (zero) and 1 (one) in Nomadic.

Once you have logged in, cd to the "pub/files" directory. Within this directory, there is a NOMAD-README file and an AGREEMENT file. Please read these two files carefully. In addition, there are seven subdirectories. The NOMAD-README file provides descriptions of the subdirectories (what they contain) and instructions on how to obtain and to extract the software in these sub-directories.

If you have any questions regarding how to obtain the software or how to run the software, please email them to: software@robots.com.

To order additional copies of this manual or other manuals, please call +1.650.988.7200 and ask for the Sales Department.

DISCLAIMER AND WARRANTY INFORMATION

Thank you for purchasing a Nomadic Technologies product. The Nomad and Sensus products are warranted to the original purchaser, to be free from defects in materials and workmanship for a period of one year from the shipping date. During this period Nomadic Technologies, Inc. will repair or replace, at our discretion, any defective components.

This warranty does not apply to any Nomad or Sensus products which have been damaged by accident, abuse, negligence, improper use, power surges, acts of God or have been repaired, altered, or modified in any way by anyone other than Nomadic Technologies. This warranty does not apply to the batteries or antennae.

Nomadic Technologies, Inc. expressly disclaims and excludes all other warranties, express, implied, and statutory, including without limitation, the warranty of merchantability and fitness for a particular purpose.

Nomadic Technologies, Inc. expressly disclaims and excludes all liability for incidental and consequential damages, including lost profits or savings, and the cost of recovering or reprogramming lost data. The Customers maximum entitlement shall in no event exceed the cash value of the covered item(s) at the time of the item(s) breakdown.

If you have any questions or problems with your Nomad or Sensus products contact Nomadic Technologies Customer Service at +1.650.988.7200 for instructions.

In the event that service is required, after notifying Nomadic Technologies and receiving a RMA, ship your product, together with all accessories, in its original packaging, fully prepaid and insured, to Nomadic Technologies, Inc. Nomadic Technologies, Inc. is not responsible for any damages incurred during shipping. We will notify you of repair costs, if they are not covered by the warranty, before undertaking them and will notify you before return shipping your product. The customer is responsible for all shipping and shipping insurance costs.

CONVENTIONS

Here are the typographical conventions used in this manual:

1 Typewriter characters denote user input at a terminal, as well as code examples, as in:

```
machine:~/Nomad200/host$ Nserver
my.world.setup my.robot.setup
```

2 ☑ is a remark, note, or tip, as in:

☑ *You can have up to 6 robots controlled simulta*neously from the same GUI.

3 🏳 denotes trouble shooting information, as in:

🏳 **The robot does not move.**

- ■ Is the emergency stop released?
- ■ Are the batteries in place?

4 💣**WARNING!** signals an important point, as in:

💣 **WARNING!**

Running the robot with batteries under 11V of charge may damage the CPU

CONTENTS

List of Figures

CHAPTER 1

INTRODUCTION

This User’s manual describes the use of the Nomad mobile robot. It contains:

- Some quick notes explaining the difference between the Scout and Super Scout, as well as the difference between the Scouts and earlier Nomad robots
- A tutorial introduction to the robot use, including start-up procedures, the graphic interface and simulator, elementary programming and trouble shooting
- A detailed description of the graphic interface and the associated robot simulator
- An introduction to the programming language.
- A description of the robot advanced features: sensors usage, configuration files
- A reference section for the optional vision system

In addition to this "*User’s Manual*", the "*Language Reference Manual*" contains a detailed description of each robot command.

CHAPTER 2

THE SCOUT AND SUPER SCOUT

Overview

The Nomad Scout is an integrated mobile robot system with ultrasonic, tactile and odometry sensing. It uses a special multiprocessor low-level control system that controls the sensing, motion, and communitations. At a high level, the Scout is controlled either by a laptop mounted on top or a remote workstation communicating view radio modem. Alternatively, the Super Scout is controlled via an on-board PC computer. Currently, the host computer software must run under Linux. The Scout is code compatible with the Nomad 200 class robots.

Reuse of the API

In order to maintain the Application-Programmer Interface (API) between the Nomad Scout and the Nomad 200, some functions have extra unused parameters. This is because the Scout has one fewer degree of freedom in its motion system than the Nomad 200. The N200 used a synchro-drive system, with one axis for translation, one axis for steering, and one axis for a turret containing the sonar sensors as well as other sensor packages. The Scout employs a differential drive system, in which the user controls the right and left wheels independantly. The extra unused parameters in the motion functions (`ac,mv,pr,sp,vm`) should be passed as zero.

Hardware

Located on the top and the front of the Scout are the following.

Power buttons

Pressing the gray button with either batteries connection and / or AC power turns the Scout on. Pressing the red button will turn the Scout off.

For Super Scout owners, it is important to do a "clean shutdown" before turning the robot off to ensure file-system integrity. So please make sure that your computer is in power-down safe read-only mode before powering the robot off. See *Chapter 3, Shutdown*

"Host" Serial Port

Control of the Scout is accomplished through this serial port, which defaults to 38400 baud, 8 data bits, 1 stop bit, DCE. Your host computer (e.g. laptop, radio modem) running client control software connects to this port.

For Super Scout owners, this port is disabled. Instead, control is accomplished through a TCP / IP connection from Nserver. See "*Chapter 3, Connecting to the Robot*".

"Joystick" Port

This allows user control of the Scout. Simply hold down one of the buttons and move the joystick up / down for forward / reverse and left / right for left turn / right turn motion.

"Console" Port

This port offers a text-based interface to the Scout's command set (listed in the Language Reference Manual). It powers up at 9600 baud, 8 data bits, 1 stop bit, DCE. Use of this port is self-explanatory by typing "help" at the "Robot->" prompt.

For Super Scout owners, the console port functionality may also be accomplished by using telnet to connect to port 65000.

VGA Port

Found on Super Scouts only, this port is an access to the on-board PC computer's VGA port.

Keyboard Port

Found on Super Scouts only, this port is an access to the on-board PC computer's keyboard port.

Radio Modem Power

If applicable, this is a 9V power cable for plugging into the radio modem power jack. Power to this cable is switched off with the main power switch.

AC Receptacle

This is located on the front panel of the Scout and is used to supply AC power to the Scout for simultaneous AC power and battery charging. This Scout's batteries charge only when the Scout is in the on state.

Status LEDs

These consist of three LEDs located on the front of the Scout. The green LED indicates power; the yellow LED indicates that the batteries are charging; the red LED indicates that the battery is low.

CHAPTER 3

GETTING STARTED

This chapter is a general introduction to the basic operation of the Nomad Scout. You will:

- Install batteries in the robot
- Boot up the robot
- Joystick the robot around
- Configuring the Radio Ethernet
- Start a server
- Connect to the robot
- Move the robot from the server

This Quick Start section is followed by an introduction to programming, and instructions on proper battery management.

Quick Start

Installing the batteries

The Nomad is powered by a set of batteries that together provide 432 watthours of usage. The set consists of a pair of 12V-18AH batteries. The batteries are accessible by unhitching the two latches on door on the back of the robot below the bumper.

- If the joystick is plugged in, remove it before accessing the batteries.

Starting the robot

- Turn the robot on by pressing the grey button on the front panel.

Robots have an alarm that will beep when the onboard power gets too low. In that case, you should stop using the robot and recharge the batteries. See *"Chapter 3 - Battery Maintenance Guidelines"* for indications on how to recharge the batteries.

The robot will accept joystick commands immediately after you turn it on. However, you cannot connect to a Super Scout via software until the internal PC finishes booting. When the Super Scout is ready for communication, it will beep. Linux takes about three minutes to boot.

- The robot is now operational and can be moved using the joystick.

WARNING! Do not boot close to a wall! After booting, the robot will test motor response by sending extremely small motion commands (so small that the robot won't noticeably move). However, if the encoders have been accidentally disconnected, the robot will move a few inches, with the possibility of hitting something. In this case, contact Nomadic Technologies, Inc. for assistance.

Moving the robot around using the joystick

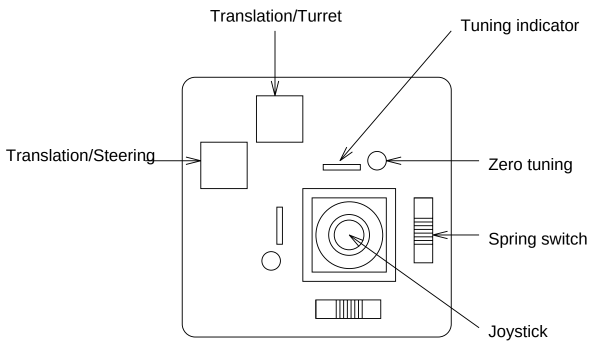


Figure 3.1 Joystick

The robot comes with a two-axis joystick depicted in *Figure 3.1*. It should be plugged into the 15-pin connector on the top of the robot. The joystick is to be used in conjunction with either of the two motion buttons in the top left corner. Always push the motion buttons BEFORE moving the joystick.

- While holding either button down:

  Pushing the joystick forward (backward) will move the robot forward (backward). The further you push, the faster the robot will go.

  Pushing the joystick left (right) will rotate the robot to the left (to the right). The further you push, the faster the robot will turn.

  Pushing the joystick in a composite direction will both translate and rotate, resulting in a somewhat circular motion.

Contrary to popular belief, the bumpers are NOT wired to the motors: the robot will keep moving even if the bumpers are hit. However, when running the robot from a program, you can monitor the state of the bumpers and stop the robot if they are hit.

The spring switches (see *Figure 3.1*) cause the joystick to go back to the zero (middle) position when released. If they are not engaged, the joystick will be free when released, sending motion commands to the robot. Since this may cause unwanted motions, it is highly recommended to always have the spring switches engaged on both axes.

The zero tuning is used to adjust the joystick. If the robot is still moving with the joystick in the zero position, turn the small knob until the robot is immobilized. The tuning indicator gives the direction of the adjustment.

WARNING! The robot doesn't move when you joystick it.

- Are the batteries present, connected, and charged (10.8 V minimum)?
- Is the joystick connected?
- Is the Emergency Stop button released (if installed)?

Running the Sonar Bounce Demo

This section is only applicable to Super Scouts. Basic scouts can also run the sonar bounce demo, but they must first be set up (see below).

Hook a VGA monitor and keyboard to the VGA and keyboard ports on the top plate of your Super Scout. Turn the Super Scout on and wait for the computer to finish booting. Log on as `"root"` and change directories into `/usr/local/Nscout/client` and run `"./sbdirect"`. The Scout should then start to move around and avoid obstacles.

3.1.5 Configuring the Network

This section describes how to configure the networking devices on your Scout or Super Scout. Refer to the section that describes your robot.

Regular Scouts with a Mercury Radio Modem

In regular Scouts, the Mercury modem listens on the radio Ethernet for client program socket communication and forward this data to it's serial port which is connected to the serial port of the motor controller board. For the socket communication to be established, the Mercury must be configured with an IP address. To do so, you remove the cover of the robot, attach a serial cable, and configure it for `Passthrough` mode using the provided serial cable and standard terminal emulation software running on a desktop computer. Please refer to "*Configuring the Mercury for Passthrough mode*" below for instructions.

Super Scouts without a Mercury Radio Modem

Super Scouts with no radio modem have an ethernet cable running from the internal PC to an external 10BASE -T jack which will accept any standard ethernet cable. One can create a simple network by connecting the ethernet device of an external computer such as a laptop to this jack. Then the onboard computer can be configured as a node on that network as described in *"Configuring the onboard computer"* below.

Super Scouts with a Mercury Radio Modem

If you purchased a Super Scout with an internal radio modem, your Mercury is by default configured to act as a network bridge from the internal PC's ethernet to the wireless access point on your network backbone. As such, the Mercury will need no configuration unless you have configured your Access Point to a different domain than the default one (if this is the case, please refer to *"Configuring the domain of your Mercury"* below.

Configuring the Mercury for Passthrough mode

In order to bring up the configuration menu of the Mercury, you must first access its serial port. To do so, remove the top place of the scout by undoing the screws around its perimeter. The Mercury is mounted on the bottom-side of the top plate. On the Mercury you will see a DB9 serial connector. Once the serial port is located, you can plug in the black serial cable provided with the Mercury and bring up the configuration menu using a desktop computer and standard terminal emulation software as described in: "*Chapter 6 - Configuration, Mercury User's Guide*". Make sure you understand all of Chapter 6 before continuing. Note that you can provide power to the Mercury either by turning the Scout on, or by removing the power jack from the radio modem and replacing it with the power jack from the AC adapter provided with the robot.

Once you have the main menu up, the following changes must be made from the default configuration (if you think that for any reason your unit has been changed from the default, select "Reset configuration to "default" from the main menu): Lines in the configuration files can be sections, denoted by the brackets which enclose them (For example, the `[hardware]` section) and key / value pairs, separated by an equals sign. In the configuration changes below, we list the key / value pairs which must be edited under the section to which they belong. If there is a key / value pair in the default configuration that is not mentioned below, it should be left as found and not edited; we only list below those key / value pairs which must be changed.

First the following changes must be made to the `uart0` file:

```
[hardware]
baud = 38400
[software]
input timeout = 40
delimiters = 0x5c
[passthrough]
socket = tcpbind2
```

To configure the `lan0` file, you must have the following information which you can obtain from your network administrator:

- The wireless network domain number used by the Proxim access point(s) you will be connecting to. The default domain as shipped by Nomadic is one. This number is referred to below as `<YOUR DOMAIN>`.
- The robot's IP address referred to below as `<YOUR IP>`.
- The subnetwork mask referred to as `<YOUR NETMASK>`.
- Your gateway address if you have one, referred to as `<YOUR GATEWAY>`.

From these parameters, you should be able to determine (if not already provided) your network's broadcast address. This is a four-octet IP address, obtained by replacing every '1' bit in the subnetwork mask with the corresponding bit from your robot's IP address, and replacing every '0' bit in the subnetwork mask with a '1' bit. Consult your network administrator if you need help finding this value. We will refer to this value below as `<YOUR BROADCAST>`.

With this information, the following changes should be made to the `lan0` file:

```
[hardware]
domain = <YOUR DOMAIN>
[ip]
ip address = <YOUR IP>
netmask = <YOUR NETMASK>
broadcast = <YOUR BROADCAST>
[ip_route1]
destination = <YOUR IP>
genmask = <YOUR NETMASK>
```

If you have a gateway, you should also make these changes:

```
[ip]
#route = ip_route1
route = ip_route1 ip_route2
[ip_route2]
gateway = <YOUR GAMEWAY>
```

That's it. Exit the editor and reset the Mercury by selecting "Reset the Mercury-EN" from the main menu. Once reset, you should see the station LED come on, indicating that the unit has attached to its access point. Now try to ping the IP address you assigned to the unit from a desktop computer. If you have trouble, please refer to your "*User's Manual*" or contact Nomadic Technologies technical support. When everything is configured properly, remove power from the system and replace the top plate of the robot.

Configuring the domain of your Mercury

If your Scout robot will be communicating via wireless with a Proxim Access Point which has a different wireless domain number than the default of one, you will need to configure the Mercury to communicate on that domain as well. To gain access to the configuration, please read the relevant parts of the section: *"Configuring the Mercury for Passthrough Mode"* above.This describes how to remove the top plate of your robot and bring up the serial configuration menu using standard desktop computer terminal emulation software. Once you are at the main menu, the key / value pair which you must change can be found in the `lan0` file. Select that file and make the following change:

```
[hardware]
domain = <YOUR DOMAIN>
```

Once it is done, you can power cycle the Mercury and you should see the Station LED come on after two to four seconds, indicating that the Mercury has associated with its access point. If you have trouble, please refer to your "*Mercury User's Guide*" or contact Nomadic Technical Support. Once everything is configured properly, replace the top plate of the robot.

Configuring the on-board computer

The following applies only to Super Scouts.

The on-board computer on your Super Scout robot runs the RedHat 6.0 Linux operating system, which stores its network configuration in the `/etc/sysconfig/directory`. You must boot up in read / write mode and log in as the superuser (root) in order to change the configuration. Before you begin, you must have the following information, which you can obtain from your network administrator:

- The robot's hostname referred to below as `<YOUR HOSTNAME>`
- The domain referred to below as `<YOUR DOMAIN>`, which together with the hostname make up a full Internet name
- The robot's IP address referred to below as `<YOUR IP>`
- The subnetwork mask referred to as `<YOUR NETMASK>`
- Your gateway address if you have one, referred to as `<YOUR GATEWAY>`

From these parameters, you should be able to determine (if not already provided), the following values:

Your subnetwork's broadcast address

This is a four-octet IP address, obtained by replacing every '1' bit in the subnetwork mask with the corresponding bit from your robot's IP address, and replacing every '0' bit in the subnetwork mask with a '1' bit. For example, if your IP address is `205.162.4.162` and your subnetwork mask is `255.255.255.0`, your broadcast address would be `205.162.4.255`. Consult your network administrator if you need help finding this value. We will refer to this value below as `<YOUR BROADCAST>.`

Your subnetwork's network address

This is a four-octet IP address, obtained by replacing every '1' bit in the subnetwork mask with the corresponding bit from your robot's IP address, and copying every '0' bit in the subnetwork mask to the network address directly. For example, if your IP address is `205.162.4.162` and your subnetwork mask is `255.255.255.0`, then your network address would be `205.162.4.0`. Consult your network administrator if you need help finding this value. We will refer to this value below as `<YOUR NETWORK>`.

To change your Super Scout's network parameters:

- Plug in a keyboard and monitor into the Nomad SuperScout's top panel.
- Turn on the robot. After the initial BIOS screens go by, you will be left at the prompt LILO boot:. Within five seconds, type `linux-rw` and press Enter.
- After the boot sequence completes, you will see a new.robots.com login: prompt. Log into the robot's console as `root`. By default, there is no password.
- Change to the directory `/etc/sysconfig/network-scripts`.
- Using an editor such as vi or emacs, load the file `ifcfg-eth0`. If you are unfamiliar with UNIX-style editors, contact your network administrator for help.

Inside this file, you should see the following text:

```
DEVICE=eth0
ONBOOT=yes
IPADDR=<YOUR IP>
NETMASK=<YOUR NETMASK>
BROADCAST=<YOUR BROADCAST>
```

```
NETWORK=<YOUR NETWORK>
```

- Change to the `/etc/sysconfig` directory.
- Edit the file "`network`".
- The file should be modified to look like the following:

  ```
  NETWORKING=yes
  FORWARD_IPV4=false
  HOSTNAME=<YOUR HOSTNAME>.<YOUR
  DOMAIN>
  GATEWAY=<YOUR GATEWAY>
  GATEWAYDEV=eth0
  ```

- Change to the `/etc` directory.
- Edit the file "`hosts`". You should modify as follows:

  ```
  127.0.0.1 localhost
  ```

  <YOUR IP> <YOUR HOSTNAME>.<YOUR DOMAIN> <YOUR HOSTNAME>

- Once you have edited these files, you will need to reboot your computer for the changes to take effect. Then, you should be able to ping the IP address which you assigned to the robot from another computer on the network.

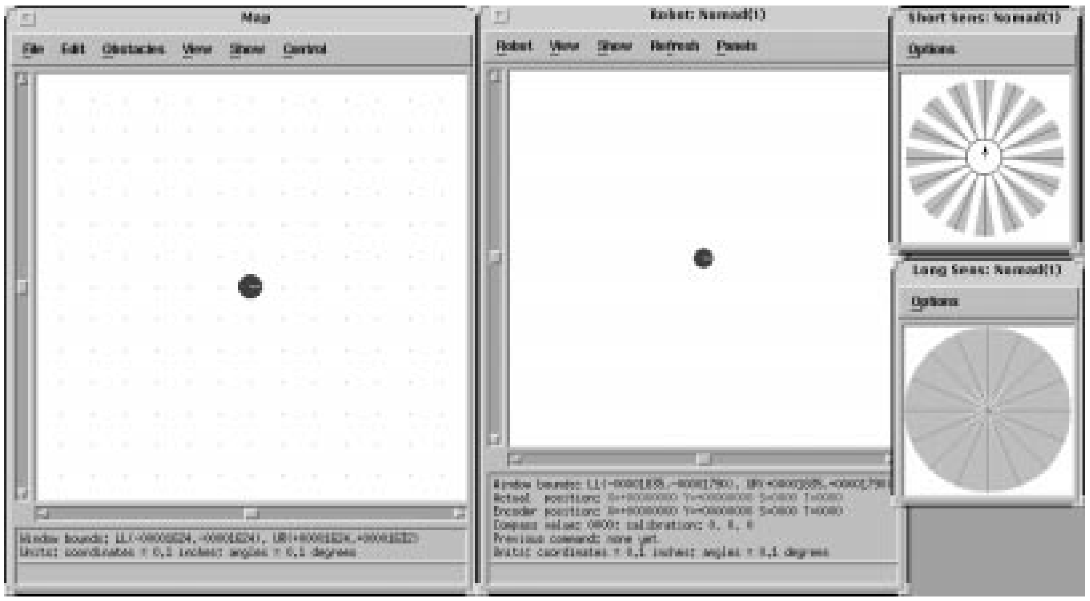
Starting a Server

The Server is a convenient way to send commands to the robot and to receive sensing data from the robot. It provides an elaborate graphic interface and simulation capabilities. You run the server as a separate process on a workstation. The server process communicates with the robot process through radio ethernet, using the TCP/IP protocol. To start a server on your workstation:

- Make sure that you have, in the same directory, the `Nserver` executable, and the two configuration files `world.setup`, `robot.setup`.
- Type the command:

  ```
  /Nserver
  ```

The graphic interface should appear as shown in *Figure 3.2*. The xterm that you used to run `Nserver` will be used to display messages; it will be referred to as "the xterm" in the sequel. You should now see the following in the xterm:



```
Window bounds: LL(-[illegible]), UR(+[illegible])
Units: coordinates = 0.1 inches; angles = 0.1 degrees
```

```
Window bounds: LL(-[illegible]), UR(+[illegible])
Actual   position: [illegible]
Encoder  position: [illegible]
Compass value: 0000: calibration: 0, 0, 0
Previous command: none yet.
Units: coordinates = 0.1 inches; angles = 0.1 degrees
```

Figure 3.2 The Entire Graphic interface

```
================================================================
Nomadic Host Software Development Environment (Version 2.6.2)
Last revision date: 12-Aug-1995
Copyright 1992,93,94,95, Nomadic Technologies, Inc.
================================================================
Using server tcp port #7019.
```

The command doesn't work. Possible causes:

- Nserver has not been compiled for your machine.

  Check how to get the proper executable in the section "Where can I get help?" at the beginning of this manual. We currently support:

  - Silicon Graphics: MIPS / IRIX5.3
  - SUN: SPARC / SUNOS4.1, SOLARIS2.3
  - Linux 2.0, 2.2 systems / X11R6

- You are not allowed to display on this machine.

  Check your DISPLAY environment variable, and xhost if necessary

- Some configuration files are damaged or missing.

  Restore the original files from the distribution, or edit the files (refer to *Chapter 6 - The Setup Files section*).

Connecting to the Robot

For safety reasons, the server is by default configured in simulation, which means that actions do not have physical consequences. You are now going to connect to the real robot.

- Make sure that the robot is ON.
- Select the ROBOT item in the robot window (entitled Robot:Nomad(1)) by clicking with the left button.
- Select REAL ROBOT and release the left button: the communication starts

The REAL ROBOT menu item turns into SIMULATED ROBOT: if you select it, the connection to the real robot will be closed, and your actions will be directed to the simulation again. When connected, you can check the information bar of the robot window, which should show the actual position grayed out (the actual position is a simulation concept; see *"Chapter 4 - The Graphic Interface*. The xterm should look like the following:

```
Robot <-> Host TCP/IP communication
setup (machine nomad on port 4000)
```

The server doesn't connect to the robot. Possible causes:

- The robot is OFF.
- The machine name in `robot.setup` does not match the robot name. Edit the file `robot.setup`, paragraph [connect], item machine, and type the correct name.
- The robot is too far away. The range limit within one single bridge is about 50 meters (150 feet) under normal conditions.

Moving the robot

You are now going to initialize the robot, and to move it around using the soft joystick. Initialization, or "zeroing", resets the encoders to the "zero" position.

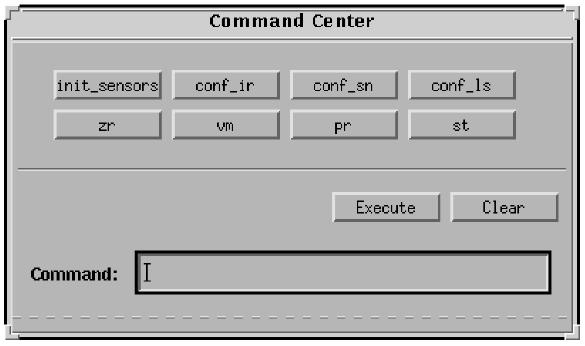


Figure 3.3 The Command Line Panel

- Select the PANELS item in the robot window, and choose COMMAND LINE. The window of *Figure 3.3* will appear.
- Select the `zr` (zeroing) button: `zr` appears in the command line.
- Click on `Execute` to start zeroing.
- Select COMMAND LINE again from PANELS to close the window.

The robot doesn't initialize. Check the following:

- Check if the robot is ON.
- The Emergency stop button is released.

- The Server is connected to the robot (see previous section).

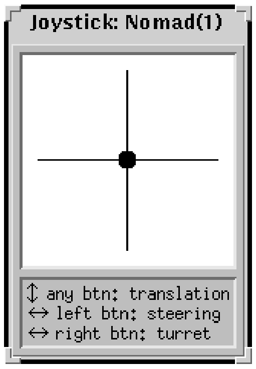


Figure 3.4 The Soft Joystick Figure

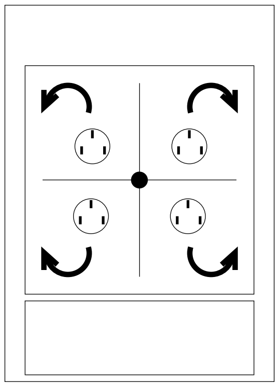


Figure 3.5 Combined Motions

After zeroing, you could move the robot by sending motion commands with the command line panel. However, it is usually easier to use the soft joystick:

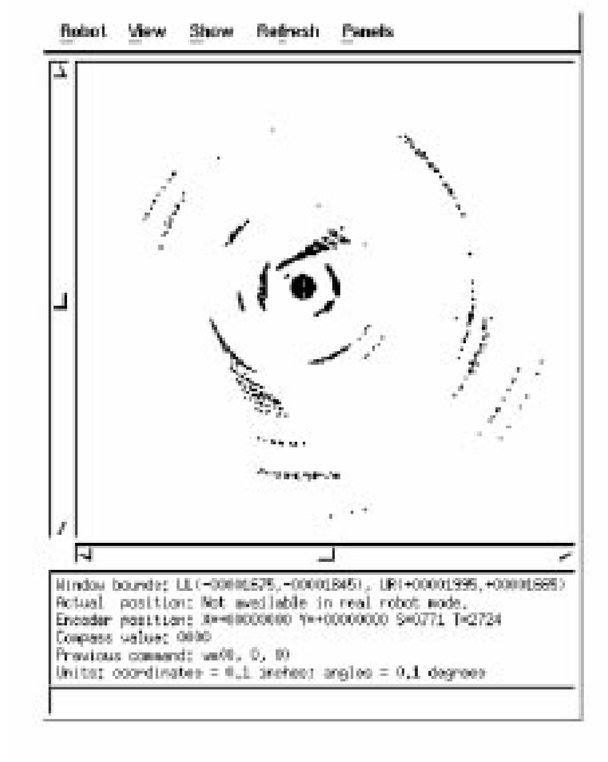
- Select the PANELS item in the robot window, and choose JOYSTICK. The window of Figure 3.4 will appear. The crosshairs represents the two degrees of freedom of the physical joystick. The central dot is the joystick: the robot moves according to its position.
- Make sure that the robot is free to move and that no equipment (keyboard, monitor,...) is connected to it.
- With any mouse button, click and drag the dot to the middle of the top vertical line: the robot will move forward. Release the left button to stop.
- Click and drag the dot to the middle of the left horizontal line: the robot will steer. Release the left button to stop.
- The further you move the dot from the center of the crosshairs, the faster will the robot go.
- As with the physical joystick, combined motions are possible, as illustrated in *Figure 3.5*.
- To get an idea of the robot motions, you can select SHOW in the robot window. Select then ROBOT TRACE, and choose SOLID. Move the robot: the path is shown. Select NONE to remove it.

☑ A common mistake is to give an absolute meaning to the soft joystick directions, like "North", "East", etc. The motions are always relative to the current orientation of the wheels. Moving the dot to the top will move forward, which may be "South" if the robot is so oriented.

The dedicated Short Sensor and Long Sensor windows display robot-centered sensor information. This is an instantaneous information. It is also possible to accumulate these data on the screen and to get a sensory image of the environment. This allows you to navigate from your workstation.

- Before you can see sonar data, you must turn on the sonar. This can be done from PANELS/COMMAND LINE by clicking the `conf_sn` button. Fill in the values with A:8, 0:0, 1:1, 2:2, ... 15:15. The meaning of these parameters will be explained later. Once you fill in all the fields, click the Execute button.
- In the robot window, select SHOW, and choose SONARS; dots representing sonar readings appear (default color is blue) in the robot window, around the robot

.



```
Window bounds: LL(-00001675,-00001845), UR(+00001995,+00001665)
Actual  position: Not available in real robot mode.
Encoder position: X=+00000000 Y=+00000000 S=0771 T=2724
Compass value: 0000
Previous command: vm(0, 0, 0)
Units: coordinates = 0.1 inches; angles = 0.1 degrees
```

Figure 3.6 Surroundings of the robots as shown by traces

- Using the soft joystick, rotate the robot by moving the dot on the horizontal axis, left or right: sensing data will be accumulated, and the surroundings of the room the robot is in will start to appear as in *Figure 3.6*.
- To stop the accumulating of the data, select SHOW and SONARS, INFRARED again. To erase the accumulated sensory data, select REFRESH in the robot window, and choose ALL SENSOR HISTORY.

Nothing appears, or strange data; check the following:

- Is the sonar ON? These sensors must be started by sending a `conf_sn` using the Command Center. See *"Chapter 4 - The Graphic Interface"* for a description of how to fill in the fields, as well the *"Language Reference Manual"* for a description of the meaning of the fields.
- Is anything in the way of the sensors?
- Sonars won't see smooth objects unless they are presenting a surface normal to the direction of the sound.
- Dots aligned along circles at some distance of the robot are NORMAL; when a sensor doesn't see anything, it returns a constant maximum distance value, whose accumulation produce circles.
- Sensor data is not displayed when the position of the robot is not available. If a program has set the sensor mask so as to exclude position, sensor readings are not printed.

You have successfully completed this Quick Start section. Your robot is now fully operational and ready to use. Subsequent sections will introduce programming and proper battery care. Follow the instructions of the next section to shutdown your robot.

Shutdown

This section is applicable only to Super Scouts. Basic Scout owners need not worry about shutting down their robot, and can simply turn it off. Since the Super Scout's computer system is a complete Unix operating system, it must be powered down cleanly, otherwise damage can result to the file-system. The Super Scout is configured at the factory to boot up in "read-only" mode, which means that the file-system is mounted such that it can only be read. This allows the Super Scout to power down without first unmounting the file-systems. That is, it can be simply turned off.

Typically, however, users want to install, copy or develop software on the computer system, which requires write access, or "read / write" mode. After the Super Scout has booted into read-only mode, read / write mode can be entered by simply typing `"exit"` at the shell prompt. After various operations finish, you will get a login prompt. Log on as `"root"` and you will have full file privileges. Returning to read-only mode can be accomplished by typing `"init 1"` at the shell prompt. Of course, once the computer has re-entered read-only mode, the robot can be safely shut off. Or alternatively while in read / write mode, you can run `/sbin/shutdown`, `/sbin/reboot`, or type `ctrl-alt delete` on a keyboard plugged into the computer to do a clean shutdown. After the shutdown is complete, the robot can be safely shut off.

In summary:

- If the computer is in read-only mode (default upon boot up), the robot can be safely powered down at any time.
- If the computer is in read / write mode, it can be put back in read-only mode by typing "`init 1`" at the shell prompt.
- Executing `/sbin/shutdown` or `/sbin/reboot` while in read / write mode is another way to bring the computer to a safe power down state.

When the robot is not in use, it is best to leave it on and plugged in, so that the batteries can charge (see the section"*Power System"* in this chapter).

DIP Switches inside the Scout configure the Host serial port baud rate. To access these switches, remove the top plate of the robot by removing the eight screws around the perimeter. The set of four switches is located in one of the far faces of the controller board inside. The switches are numbered 1 through 4 with the following settings:

DIP Switch Settings

1	2	3	4	Baud Rate
on	on	on	on	9600
off	on	on	on	19200
on	off	on	on	38400 (default)

Introduction To Programming

<figure showing direct and client modes>

Figure 3.7 Programming in Direct and Client mode

You are now going to run the robot from a program instead of manually sending commands to it. There are two ways to run the Nomad robot from a program, as illustrated *Figure 3.7*.

- **Direct mode**
  Your program communicates directly with the robot daemon (the program constantly running on the robot that accepts commands from outside, executes them, and sends data back).
- **Client mode**
  Your program communicates as a client to the server: the server accepts the commands of your program (exactly as it accepts your commands from the graphic interface) and transmits them to the robot daemon. The server can also transmit your commands to a simulation module instead of the real robot.

The client mode is the preferred way of testing and debugging a program: you first run your program in client mode using the simulator. When your program works correctly, you run it still in client mode, but using the real robot. You do not have to modify anything in your program: a simple switch in the server redirects your commands from the simulator to the real robot.

The direct mode is used when your program is completely correct, to minimize the communication overhead.

We are going to test the following simple C program:

```
#include "Nclient.h"

void main()
```

```
{
 connect_robot(1);
  zr();
  sp(50,50,0);
  pr(1000,1000,0);
  while(State[STATE_CONF_X] < 1000)
    gs();
  disconnect_robot(1);
}
```

This program:

- Connects to the robot
- Initializes it using the command `zr` that you already know
- Sets the translational speed to 5 inches/s
- Translates the robot by 100 inches (1000 tenths of inches)
- Gets the robot state during the motion
- Disconnects from the robot

The include file `Nclient.h` contains the prototypes of the robot commands: `zr,pr,` etc. Let us first compile this program:

- Create a file `myprog.c` containing the above program, and put it in the same directory as `Nclient.h`.
- Compile it: for instance,

  ```
  gcc -o myprog myprog.c Nclient.o
  ```

- Run a server as explained above, but DO NOT connect to the real robot. If a server is already running, disconnect from the real robot by toggling REAL ROBOT in the ROBOT item of the robot window.
- From another xterm, run `myprog.`

The program `myprog` will connect to the server, initialize the simulated robot and move it. You can check the motion progress by looking at the "Encoder Position" in the information bar of the robot window. You can also see which command is in progress in "Previous Command".

☑ Note that the robot information on the server is updated at each `gs` command. If there were no such call in your program, you wouldn't see any motion, because the interface wouldn't be updated. A simple way to update the graphic display is to repeatedly send `gs`() from the command line panel.

☑ When using `pr`, you must be aware that if you request a long motion (for instance `pr(10000,1000,0)`) without sending further commands to the robot, the motion will stop before the motion is complete because of the time-out mechanism. Once again, sending `gs`() is a good way to keep in touch with the robot.

The simulated robot does not move when you run the client program.

- The Server displays the following on the xterm:

  ```
  ERROR: bind call failed in server,
  tcp port #7019 already in use
  ```

You (or someone else) have a process using the same port. Most likely it is a `Nserver` process; you can kill it, or you can change the communication port by setting the `TCP_SERV_PORT` variable to another value in your program, and accordingly the serv port item in the [connect] paragraph of the `world.setup` file.

- Your program says:

  ```
  Not connected to any robot
  ```

Your program couldn't connect to the server: either you forgot the `connect_robot` command, either you do not have a Server running, either the communication ports of your server and your program do not match.

If your program works correctly in simulation, you can now use it with the real robot.

- Make sure that there is nothing in the way of the robot, and that no equipment (keyboard, monitor,...) is connected to the robot
- Select REAL ROBOT in the ROBOT menu.
- Run `myprog.`

Battery Maintenance Guidelines

Power System

The Scout has two large 12 Volt 17 Ampere-Hour lead-acid batteries. These batteries are shipped in the Scout unplugged. To access the batteries, open the battery compartment by undoing the silver latches on the right side of the robot. The batteries can then be plugged into the connectors found between the two battery compartments.

Fully charged, these batteries are capable of supplying power to a Scout and radio modem for up to 24 hours of normal operation or 5-8 hours for a Super Scout. These batteries can also power an on-board laptop if desired. The Scout also has advanced power and battery management features which are listed below.

On-board Battery Charging and AC Power

The Scout is equipped with an AC to DC converter that is capable of simultaneously charging the batteries and supplying power to the Scout electronics. Simply plug the supplied AC power cord into the AC power receptacle on the side of the Scout while it is powered on. You will hear a short double beep as the Scout acknowledges AC power. The Scout's batteries will be charged and power will be supplied through the AC power cord as long as it is plugged in. When the Scout's batteries are fully charged, four short beeps can be heard.

Battery Monitoring

The Scout has three battery states that are periodically checked to ensure that the on-board batteries are not damaged. When the battery monitor senses a very low condition, it will automatically turn the Scout off. The three battery states can be read through the state vector described above in the "*Power System"* section. These states are as follows:

- **Full**
  Indicates that the batteries are nearly full.
- **Medium**
  Indicates that the batteries are about half-full. A transition from Full to Medium is indicated by two long beeps.
- **Low**
  Indicates that the batteries are very low and should be charged soon. This state is indicated by a continuous beep. The battery monitor will only allow the Scout to be in this state for 60 seconds before it powers the Scout off.

Configurable Auxiliary Power

- The Scout has a built-in 9V power supply for a radio modem. Additionally, it has provision for a factory configured 50 watt supply available in voltages between 5 and 48V, which is intended for supplying laptop power. The Super Scout does not have this option, as the auxiliary power supply is used to provide power to the on-board PC computer.

The Status Beeper

The following is a summary of the possible beeper sounds and their meanings:

- **Two short beeps of 50ms each**
  This indicates that the AC power cord has been plugged in.
- **Two long beeps of 400ms each**
  This indicates that the Scout's batteries are about half-full and the battery monitory is in the Medium state.
- **Four short beeps of 100ms each**
  This indicates that the Scout's batteries are now fully charged and the battery monitory is in the Full state. The AC plug can be disconnected if desired.
- **Continuous beep**
  This indicates that the batteries are low and the battery monitor will shut the robot down in 60 seconds.

Usage

A fully-charged Scout will generally get 24 hours or more of battery life. However, a fully-charged Super Scout usually lasts only 5-8 hours, depending on usage. These estimates assume a Nomad Scout that has standard equipment installed in it. The power system has a low voltage alarm system connected to it. Whenever the voltage drops below 21.6V an alarm will sound. When the alarm is heard the robot should be recharged as soon as possible. The Scout will shut itself down automatically if it is not plugged in within 60 seconds of the low-battery alarm sounding.

Storage

When the Nomad is not in use the batteries should be disconnected from the robot. This is due to the fact that a fuse status LED is connected and will slowly drain the battery if it is left connected. If the batteries will not be used for an extended period of time it is best to charge them every 6 months.

Battery Lifetime

When the batteries are maintained according to the above specifications they should yield approximately 200 charge / discharge cycles. When they begin to discharge very quickly, or will not hold a charge at all it is time to buy a new set. Here are the battery specifications for replacing the batteries.

BATTERY	VOLT	AMP. HOUR	INTER. RESIST APPROX.	DIMENSION(MM/INCH)			OVR TER	WEIGHT KG/LBS	TERM
				LEN.	WID.	HEIGHT			
MAIN	12	18	11	181/	76/	167/	167/	6.2/	FASTON
				7.13	2.99	6.57	6.57	13.67	250 SER.
CHARGING VOLTAGE STAND BY USE = 13.5 TO 13.8V AT 20°C (68°F)									
CHARGING VOLTAGE CYCLIC USE = 14.4 TO 15V AT 20°C (68°F)									
CROSS REFERENCE BY MANUFACTURER									
TEMPEST	YUASA	POWER SONIC*	JOHNSON CONTROL S	GS	PANA SONI C	EAGLE			
						PICHER			
TR 18-12	NPG18-12	PS-12170	JC12150	PE12 V15	LCR1 2V17 BP	CFM12V1 8			

*Power-Sonic has a web site: http:www.power-sonic.com

Summary

- Proper maintenance and use of your batteries will ensure the proper operation of your Nomad, and lengthen the life of your batteries.
- Low voltage alarms will sound off if the battery voltage goes too low. When the alarm sounds, the robot should be immediately plugged in for recharging.
- When not in use for an extended period of time, the batteries should be disconnected from the robot.
- If not used for an extended period of time the batteries should be charged every 6 months.
- When used properly the batteries should yield approximately 200 charge / discharge cycles. Longer battery lifetime is possible if usage is light.

Where to go next?

You have completed this introduction to the basic operation of the Nomad. You may now:

- get a more detailed description of the use of the graphic interface and the simulator: see "*Chapter 4 - The Graphic Interface*".
- get an introduction to programming concepts and command language features: see "*Chapter 5 - Programming the Nomad*"
- get some information on advanced topics like the use of position data, sensors characteristics, setup files, etc: see "*Chapter 6 - Advanced Features*".

CHAPTER 4

THE GRAPHIC INTERFACE

Introduction

The Graphic User Interface (GUI) provides a convenient access to the real and simulated robots, and to the representation of the world, as illustrated in *Figure 4.1*. Through the GUI, the user can send command to robots, monitor command execution by seeing the robot actually moving on the screen, visualize instantaneous and cumulated sensor data. The user can also create and modify a simulated environment, and use it to test robot programs.

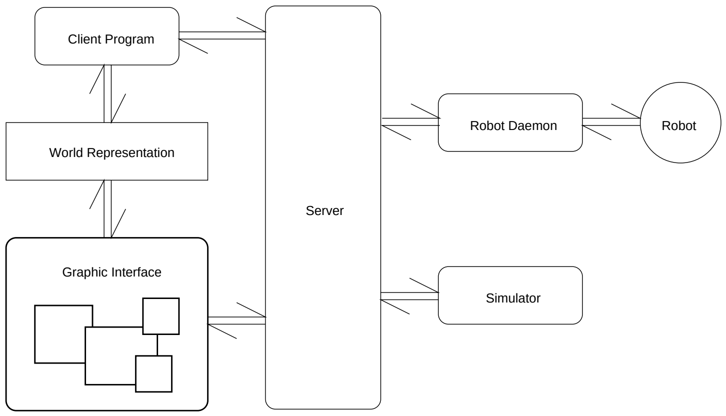


Figure 4.1 The Graphic Interface In the Graphic Display

To run the GUI, open an xterm, cd to the directory where the Nserver is (it should also contain files `world.setup, robot.setup` and `license.data`) and type:

```
./Nserver
```

You will see (Figure 4.2):

- The Map window
- One robot window
- The Short Sensors and LongSensors windows

The xterm is then used for displaying text messages from the server.

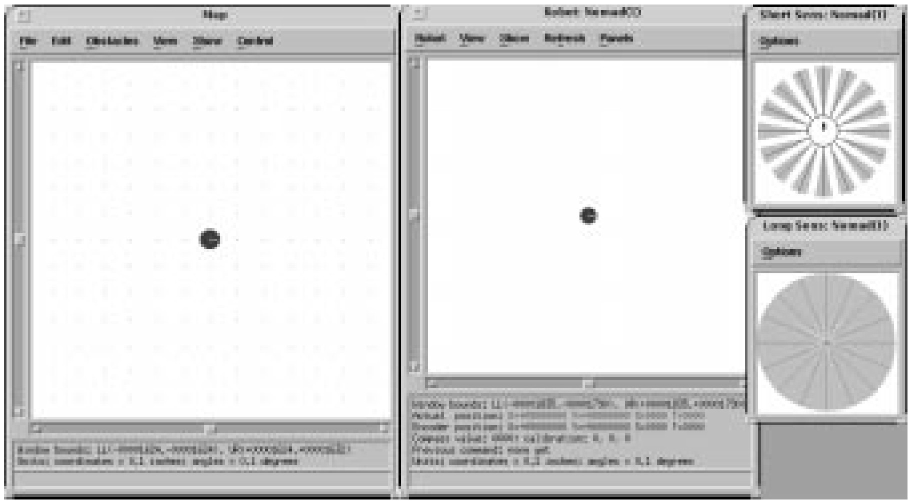


Figure 4.2 The Four Main Windows

The GUI presents 2 different views of the world the robot(s) are in:

- A global view ("God's eye"): the Map window, in which all the robots are represented
- A local view: the Robot window, that displays the world as seen by one robot only

Thus, there is only one Map window, but as many Robot windows as there are robots (real or simulated). The Map window gives access to the world representation, with functions such as: creating obstacles, editing obstacles, etc. The Robot window allows the display of sensor history, the path of the robot, as well as the execution of robot commands. Both windows support usual display functionalities like zooming and unzooming, scrolling, centering, etc. Several display parameters are controlled by values set up in the file `world.setup`. See "*Chapter 6 - the Setup Files section*" for an explanation of the display parameters.

The simulator

There is no difference in the graphic display whether the interface is dealing with a real or a simulated robot. By default, the GUI is in simulation mode. The REAL ROBOT option of the ROBOT item in the Robot window is used to toggle between the real and the simulated robot.

The Simulated Environment

The simulated robot moves in a two-dimensional world populated with other robots, and convex polygonal obstacles. These obstacles can be added to the current environment from the Map window, or by program using the world manipulation functions. The sensor data displayed on the robot window under simulation comes from modules simulating each of the sensors: these modules compute the sensor's answer according to the current (simulated) robot environment.

Actual/Encoder Robot

In the physical world, the robot knows its position by integrating the encoders' information. This is not exact information however, since there is some slippage of the wheels on the floor. The position of the physical robot is different from the position given by the encoders, and the difference keeps growing. In simulation, the actual robot represents this phenomenon. The position of the actual robot is equal to the encoder position, plus some error. The actual robot and the encoder robot are identical just after zeroing (the encoders are reset to zero, as well as the integrated position), and diverge according to the error model afterwards. For example, if a `gs()` command is issued in simulation, the position information returned in the State vector will come from the encoder robot, but the sensing information will be computed (simulated) from the actual robot's position.

On the GUI, you can display Actual and Encoder robots in different colors and check for their behavior (when you are in real robot mode, no actual robot is displayed).

The parameters describing the behavior of the simulation modules are in the `robot.setup` file. See "*Chapter 6 - the Setup Files section*" for an explanation of the simulator parameters.

Running the Nserver with arguments

When you run `Nserver`, the program looks for arguments on the command line: the first argument is expected to be a configuration file for the environment, containing among others display parameters for the Map window. All the other arguments are expected to be configuration files for robots. When you run `Nserver` without arguments, the programs looks for a default Map initialization file in the current directory: `world.setup`. Thus, calling

```
/Nserver
```

is equivalent to calling

```
/Nserver world.setup robot.setup
```

if `robot.setup` is the declared robot configuration file in `world.setup` (see below). If you want to automatically create several robots windows corresponding to several real or simulated robot (instead of creating them by hand from the graphic interface), you do so by providing several setup files at the command line level:

```
/Nserver world.setup robot.setup_A
robot.setup_A robot.setup_B
```

In the example above, three robots will be created, two using the setup file `robot.setup_A`, and one using the setup file `robot.setup_B`.

If you always want to create several robots when you run a server, then you can specify their setup files in the environment configuration file (`world.setup` by default), section robots, item setup files. For instance, if you have the following in your `world.setup` file:

```
[robots] setup.files = robot.setup_A
robot.setup_A robot.setup_B
```

then three robots will be automatically created each time you call `Nserver` (without arguments), two using the setup file `robot.setup_A`, and one using the setup file `robot.setup_B`.

☑ Command line arguments override the specifications given in the environment setup file.

The Map Window

The Map window allows an application to interactively define and modify the map of the world where a robot moves around. The robot world is an abstract coordinate system; Its dimensions are set in the world.setup file under `[physical]`, item size. The two pairs of coordinates at the bottom of the window reflect the current positions of the window's lower-left corner and the upper-right corner in the world, respectively. Look for units information also at the bottom of the window.

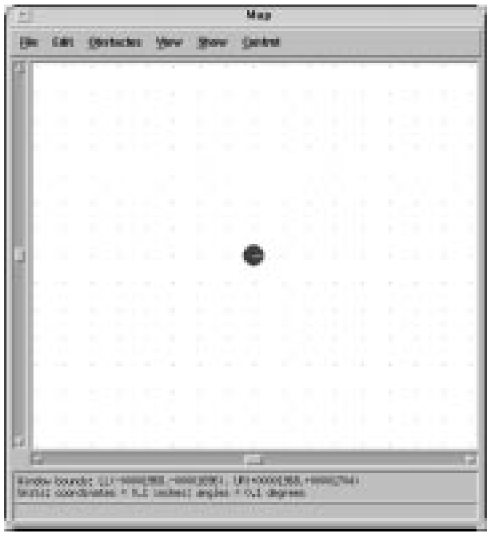


Figure 4.3 The Map Window

Initially, the (0,0) point of the coordinate system is the center of the Map window.

This window has six menus: File, Edit, Obstacles, View, Show, Control.

The File Menu

The robot map is treated as a disk file. The environment is described one obstacle per line, each line beginning with the number of vertices, followed by the *x-y* coordinates of the vertices enumerated counterclockwise.

Choose the menu to select one of the following five operations:

New Map will create a blank map with no obstacles. All the robots will remain unchanged.

Open Map will prompt you to open a map file, which was previously saved by a Save Map or a Save Map As operation. After selecting the map file, click OK. The map name, along with its path, will appear as the name of the window in the top bar.

Save Map will save the current map to the same file that is currently open. If this is the first time to save a file, it will prompt you to select the path and give a file name.

Save Map As allows you to save the current map in a different file. It will prompt you to select the path and give a file name.

Quit will quit the whole current server application. All windows will be closed.

The Edit Menu

The Edit menu allows you to edit the obstacles in the map. Before performing the following operations, you have to select an obstacle by clicking on the obstacle, in Grab Obstacle mode from the Obstacles menu.

Copy will store the selected obstacle in the internal clipboard.

Cut will delete the selected obstacle and store it in the internal clipboard.

Paste will paste on the map the obstacle currently in the internal clipboard. It requires a previous Copy or Cut operation. The pasted obstacle will overlap the one it was copied from, and can be moved.

Clear will delete the selected obstacle, without storing it in the internal clip board.

The Obstacles Menu

The Obstacles menu allows you to add, delete, and select obstacles in the map. It has the following modes:

Add Rectangles is used to create rectangular obstacles. To add a rectangular obstacle, click on a position where you want one corner of the rectangle to be, hold and drag the mouse, and release on a second position where you want the diagonal corner of the rectangle to be. You can repeatedly add rectangles when in this mode.

Add Polygons is used to create polygonal obstacles. To add a polygonal obstacle, click on a position to be the starting vertex of the polygon, move and click on a position to be the second corner of the polygon, and continue counterclockwise until finally click on a position close enough to the starting vertex. You can repeatedly add polygons when in this mode.

☑ Currently, only convex polygons are allowed. However, overlapping obstacles are allowed, so that you can create non-convex shapes by overlapping convex polygons.

Grab Obstacle allows you to select an obstacle by clicking on it. A selected obstacle has handles at its vertices. It can be moved by click and drag, and reshaped by moving the handles.

Delete Obstacles will delete obstacles by clicking on them. You can repeatedly delete obstacles in this mode.

☑ New Map option in the File menu will delete all the obstacles at once.

The View Menu

The View menu performs the following operations:

Graphics On displays the robots' motions in the Map and Robot windows during simulation. If this option is off, the graphic overhead will be avoided. The default for this option can be set on or off in the `world.setup` file, under `[graphics]`, item graphics.

Zoom In enlarges the map and centers it on the click-on point. You will see a map twice as large.

Zoom Out reduces the map and centers it on the click-on point. You will see a map half as large.

Center designates a point in the map as the center of the Map window. After selecting Center in the View menu, click on a point on the map and the point will be the center of the Map window.

Clip allows you to selectively zoom in a region of the Map window. After selecting this option, click and drag to form a rectangle to select the desired region to zoom in.

Slide allows you to move the whole map in the window. After selecting this option, position the mouse in the map and move the map by dragging the mouse.

The Show Menu

The Show menu performs the following operations:

Map is to switch on and off the display of the obstacles. **Actual Robot** is to switch on and off the display of the robot at its actual position (which only has meaning in simulation mode) in the Map window.

Encoder Robot is to switch on and off the display of the robot at its encoder position in the Map window.

The Control Menu

The Control menu performs the following operations:

Create Robot creates another robot in the current simulation. It will prompt you for a configuration file.

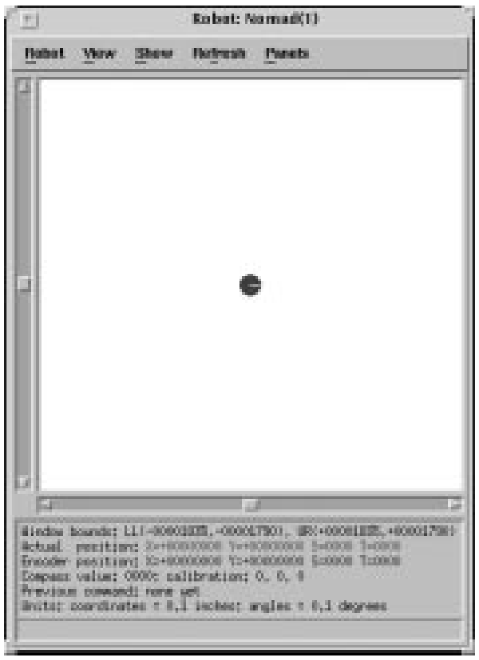
The robot id of the created robot will be given sequentially. A new Robot window will show up with its attached Sensors windows. The maximum number of robots currently allowed is 6.

☑ This is a "manual" creation of a robot. Automatic creation can be achieved using the setup files option in the environment setup file

☑ Removing a robot is done through the DESTROY ROBOT item of the ROBOT menu in the corresponding robot window

Speedup Simulation speeds up the simulation process by making the simulation process twice as fast.

Slowdown Simulation slows down the simulation process by making the simulation process half as fast.



```
Window bounds: LL(-00002035,-00001790), UR(+00001035,+00001790)
Actual  position: X=+00000000 Y=+00000000 S=0000 T=0000
Encoder position: X=+00000000 Y=+00000000 S=0000 T=0000
Compass value: 0000; calibration: 0, 0, 0
Previous command: none yet
Units: coordinates = 0,1 inches; angles = 0,1 degrees
```

Figure 4.4 The Robot Window

The Robot Window

The Robot window allows interactive control of a robot. This window has five menus: Robot, View, Show, Refresh, Panels.

At the bottom of the window are information about the current robot position, compass value, and the last command issued. In position information, X and Y are the coordinates, S is the steering direction in degrees, T is the turret direction in degrees. Note that the turret is maintained for backwards-compatibility with older Nomad robots. On the Scout and Super Scout, the turret always faces the same direction as the steering angle. Degrees range from 0 to 360, with 0 as the horizontal right.

The Robot Menu

The Robot menu has the following options:

Real Robot switches between the real robot and the simulator. When you are switching to the real robot, you must make sure that the appropriate communication (radio modem or radio ethernet) is set up between the robot and the host computer.

WARNING!!
After switching to the REAL ROBOT, every action (commands, joystick motions) performed in the GUI will be directly executed by the robot: make sure that the robot's environment is safe.

Place Robot allows you to place the robot in a given configuration. Synchronization bars and handles appear as shown in Figure 4.5. Only the robot that are currently displayed in the window according to the SHOW menu will be affected by place robot operation. If only the Encoder robot is shown, then the Encoder position and orientation will be reset. Let's assume that this is the case:

- **Clicking Left** with the left button on a (free) place will set the new position of the robot to this place
- **Click-and-Dragging Left** the farthest handle will set the turret angle. Dragging the closest handle will set the steering angle. You can monitor the value of the angle in the Position display at the bottom of the window
- **Clicking Left** on one of the synchronization bars will align both turret and steering to the angle of that bar
- **Click-and-Dragging Right** on one of the synchronization bars will move both the steering and the turret, keeping their relative angle

To finish, click in the gray label: bars and handles disappear.

☑ When connected to a real robot, the only information modified is the encoder information: placing the robot will only have an effect on where the robot thinks it is, but won't incur any motion. Only in simulation is the actual robot modified.

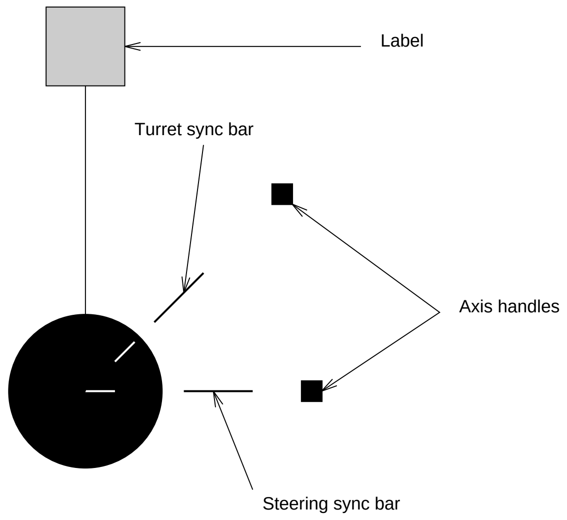


Figure 4.5 Place Robot

There are some specific operations when the two robots, Encoder and Actual, are displayed at the same time; you can monitor the effects of your actions by having both encoder and actual robot displayed in the Map window. Note however that clicking left will always set the position of both Encoder and Actual robots, even if only one is displayed.

- When Encoder and Actual robot are identical, the handle and synchronization bars are grayed. Gray handles and bars move both robots at the same time. To separate them, hide one of the robots in the SHOW menu.
- Clicking Left on some position will reset Encoder robot AND Actual robot x-y position. Dragging Left will move only one robot at a time.

Destroy Robot destroys the robot. All the three windows Robot, ShortSensors, and LongSensors associated with the robot will disappear. You can create a new robot from the Map window Control Menu.

The View Menu

The View menu has the following options:

Zoom In enlarges the map and centers it on the click-on point. You will see a map twice as large.

Zoom Out reduces the map and centers it on the click-on point. You will see a map half as large.

Center designates a point in the robot world as the center of the Robot window. After selecting this option, click on a point on the map and the point will be the center of the Robot window.

Clip allows you to selectively zoom in a region of the Robot window. After selecting this option, click and drag to form a rectangle to select the desired region to zoom in.

Slide allows you to move the whole robot world in the Robot window. After selecting this option, position the mouse in the Robot window and move the robot world by dragging the mouse.

The Show Menu

The Show menu has the following options:

Robot Trace sets various ways to display a trace of the robot as it moves. Currently available are Solid and Outline. Select None to remove the trace. Trace accumulated so far can be removed using the REFRESH options.

Map switches on and off the display of the obstacles of the map.

☑ Superimposing the obstacles to the traces of the sensors gives an idea of how well sensors capture the environment

Actual Robot switches on and off the display of the robot at its actual position (which only has meaning in simulation mode) in the Robot window.

Encoder Robot switches on and off the display of the robot at its encoder position in the Robot window.

Bumper switches on and off the display of the bumper when the robot runs into an obstacle.

Infrared is not supported on the Scout, so this menu option does nothing.

Sonar switches on and off the display in the Robot window of what the sonar sensors see.

Laser is not supported on the Scout.

The Refresh Menu

The Refresh menu allows you to get rid of the graphic robot traces and sensor history accumulated so far. The display of traces and sensor history will continue on a blank background (use the

SHOW menu options to stop these). The Refresh menu has the following options:

All clears everything mentioned in the rest of the menu - traces and histories.

All Traces clears traces of the actual robot and the encoder robot. See the Show menu for setting robot traces.

Actual Robot Trace clears the trace of the actual robot.

Encoder Robot Trace clears the trace of the encoder robot.

All Sensor History clears all the histories (successive sensor marks) mentioned in the remaining of the menu. See the Show menu for setting sensor display.

Bumper History clears the bumper-related marks in the Robot window.

Infrared History clears the infrared-related marks in the Robot window. Note that the Scout does not have infrared sensors.

Sonar History clears the sonar-related marks in the Robot window.

Laser History clears the laser-related marks in the Robot window. Note that the Scout does not have Laser sensors.

Client Graphics clears the drawings made by a client program, using instructions such as draw line or draw robot.

The Panels Menu

The Panels menu has the following three robot monitoring/control methods:

Recorder displays a Record and Playback center for recording robot motion and sensing sequences. This option also allows you to play back recorded sequences. To use the recorder:

- Select the items to record by Left-clicking (the Recorder uses only the left mouse button)
- Click on the record button (red dot) to start recording. A record is made and time-stamped each time the `State` vector gets updated (almost all robot commands update the `State` vector: See "*Chapter 5 - Programming the Nomad*").
- Click on the stop button (black square) to stop recording
- Click on the left double-arrow to rewind (the single arrow rewinds one recording at a time)
- Click on the play button (green triangle) to play the recording. You can move forward and fast-forward, backward and fast-backward in the recordings.

Records can be saved to and loaded from a text file using LOAD and SAVE options. To dismiss the Recorder panel, select Recorder again.

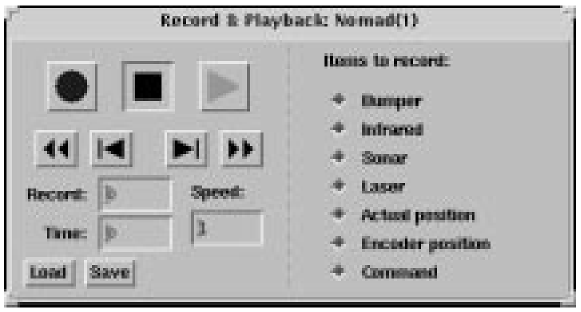


Figure 4.6 The Recorder Panel

Command Line allows you to type in a robot command from a console. All of the robot commands mentioned in the Language Reference Manual are supported.

☑ However, you cannot set the individual values of the `SMask` vector from this panel (there is no robot command to do this as well); you can only reset it with init sensors to one everywhere (all available sensor data gets returned). The only way of setting individually the `SMask` vector values is through a client program.

The top-half panel contains pre-defined calls to standard robot instructions like `zr` or `pr`. When you click on one of these, the bottom-half panel drops and displays a number of text fields corresponding to the command (zero for `zr` that takes no argument, sixteen for `conf_sn`) (see *Figure 4.8*).

Other commands can be entered by typing their names in the Command line. Typing a space after the command name will drop the corresponding number of editing fields. Type a space after each value entered to jump to the next editing field.

The command will be actually sent to the robot after clicking on EXECUTE. Click on CLEAR to cancel, and select COMMAND LINE again to dismiss the command panel. The COMMAND LINE does not support the user-packet commands, the position attachment commands, and the `get_rpx` command.

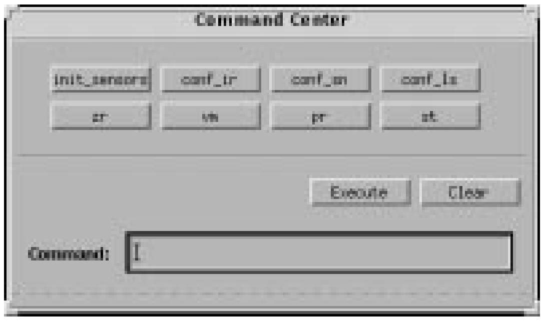
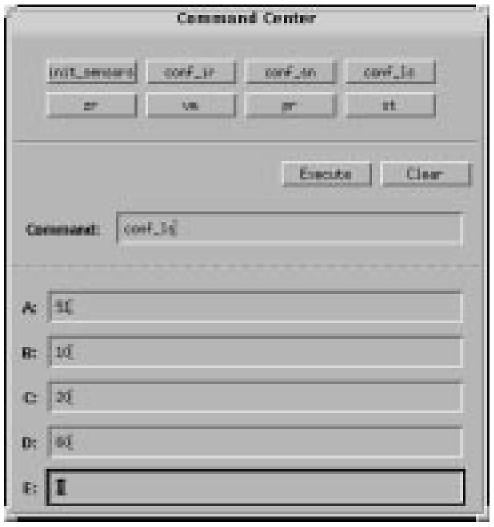


Figure 4.7 The Command Line Panel



Command: conf_ls
A: 51
B: 10
C: 20
D: 60
E:

Figure 4.8 Entering parameters in the command center

Joystick allows you to move the robot or the simulated robot around by moving the mouse in the Joystick window (see *Figure 4.9*) which will appear once this option is selected.

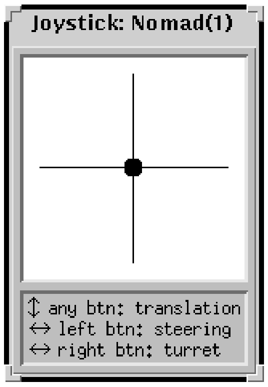


Figure 4.9 The Soft Joystick

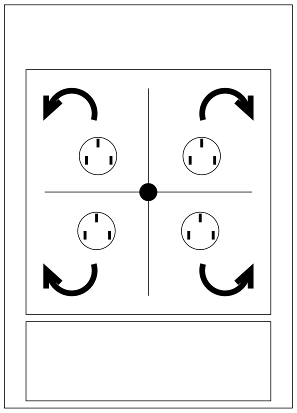


Figure 4.10 Combined Motions

- The crosshairs represent the two degrees of freedom of the physical joystick. The central dot is the joystick: the robot moves according to its position.
- Using any mouse button to click and drag the dot to the top or the bottom will move the robot forward or backward by sending repeated `vm` commands. The speed value depends on how far from the center you move the joystick dot (the farther, the faster).
- Release the button to stop the motion.
- Using any mouse button to click and drag the joystick dot to the left or the right will steer the robot to the left or the right. The farther you go, the faster the steering. Releasing the button will stop the motion.
- As with the physical joystick, combined motions are possible, as illustrated in *Figure*

4.10.

☑ A common mistake is to give an absolute meaning to the soft joystick directions, like "North", "East", etc. The motions are always relative to the current orientation of the wheels. Moving the dot to the top will move forward, which may be "South" if the robot is so oriented.

To dismiss the Joystick panel, select **Joystick** again.

The ShortSensors Window

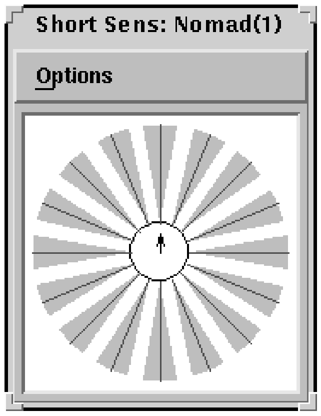


Figure 4.11 The ShortSensors Window

The ShortSensors Window has only one menu: Options.

The Options Menu

The Options menu has the following options:

Show Bumper allows you to switch on and off the display of the bumpers when the robot runs into an obstacle.

Show Infrared Rays allows you to switch on and off the display of the infrared sensor data as radius lines. Note that since the Scout does not have infrared sensors, this data will not be meaningful.

Show Infrared Cones, IR Cones Are Arcs, IR Cones Are Filled, and **Show Infrared Connections** similarly have no meaningful function on a Scout.

Global View allows you to switch between global view and local view of the robot. In local view, the robot's forward direction is always aligned with the upward vertical direction of the window. In global view, the robot will rotate with respect to the existing environment, as the turret of the robot rotates.

The LongSensors Window

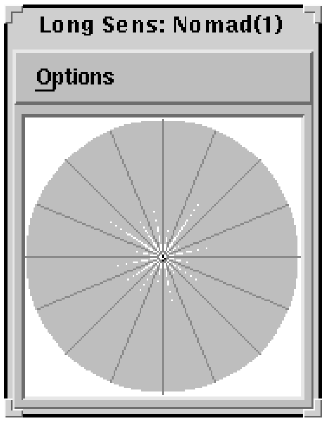


Figure 4.12 The LongSensors Window

The LongSensors Window has only one menu: Options.

The Options Menu

The Options menu has the following options:

Show Sonar Rays allows you to switch on and off the display of the sonar sensor data as radius lines.

Show Sonar Cones allows you to switch on and off the display of the sonar sensor data as cones, which is closer to the physical reality of sonar sensors but longer to draw.

Sonar Cones Are Arcs allows you to switch on and off the display of the sonar sensor data as cones with arcs at the outside ends. Arcs are a better representation, but take longer to draw.

Sonar Cones Are Filled allows you to switch on and off the display of the sonar sensor data as cones shaded or outlined. Shaded cones are a better representation, but take longer to draw.

Show Sonar Connections allows you to switch on and off the display of the sonar distance points connected using line segments.

Show Laser is not supported on the Scout.

Show Robot Proximity allows you to switch on and off the display of dots when another robot is nearby. This display will occur only if a client program sends a `get_rpx` command.

Global View allows you to switch between global view and local view of the robot. In local view, the robot's forward direction is always aligned with the upward vertical direction of the window; all the sensors are fixed. In global view, the robot will rotate with respect to the environment shown in the Map window, as the turret of the robot rotates.

CHAPTER 5

PROGRAMMING THE NOMAD

In this chapter the concepts and techniques required to write application programs for the Nomad robot are presented. It is recommended to read this chapter completely before starting to develop software.

To exploit all features of the robot and the programming environment the programmer has to be familiar with the basic architecture of the system and the entailed programming concepts, which are presented in the following section. The subsequent section gives an overview of the different classes of commands that can be used in application programs. How to use these commands is demonstrated in the last section of this chapter with example programs.

PROGRAMMING CONCEPTS

Programming Modes

An application program can communicate with the Nomad in two different ways (see *Figure 5.1*):

- **Client mode**: the application communicates with a server, which in turn communicates with the robot daemon or the simulator.
- **Direct mode**: the application communicates directly with the robot daemon.

Which of these two modes should I use?

While developing an application program it is usually advisable to use the client mode. When using the client mode it is possible to switch between real robot and simulated robot. This is convenient during debugging because the robot cannot be damaged due to a programming error.

Once a program is running stably the direct mode can be used: the program can run independently of the server.

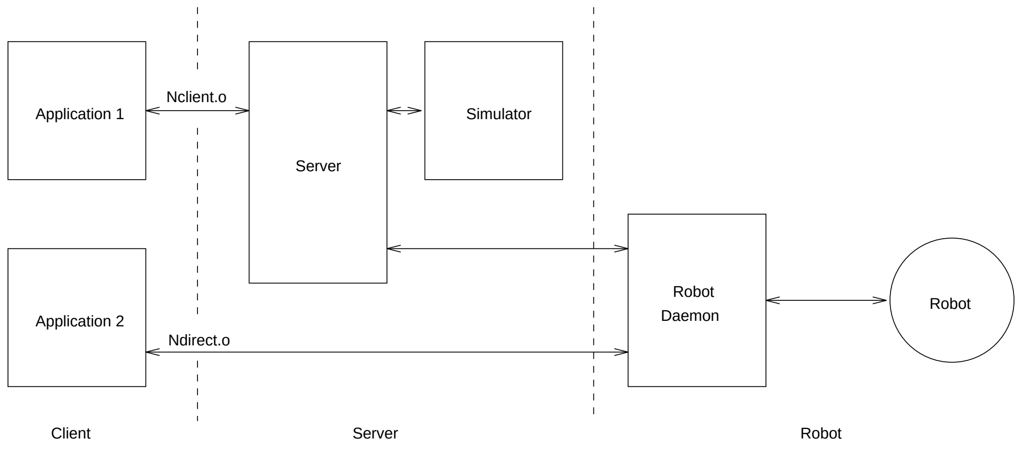


Figure 5.1 Programming in Direct and Client Mode

How can I switch between these modes?

To switch between the two modes the program simply has to be relinked. The object file `Nclient.o` is used for the client mode and the object file `Ndirect.o` is used for the direct mode. In general the source code does not need to be changed.

Are the modes compatible?

The two modes are fully compatible. However, commands that are specific to the server / simulator will have no effect in direct mode. Nevertheless, the procedure call will succeed and return an appropriate value. When writing applications that are supposed to run in client mode as well as in direct mode the programmer has to take these differences into account; programs must not rely on functionality provided by the server. In the file `Nclient.h` functions that have no effect in direct mode are marked. The *"Server Commands"* section in this chapter discusses these commands in detail.

How can I specify to which robot/server I want to connect?

How to establish a connection with a specific robot or server is explained in the section "*Communication Commands*".

Simulated vs. Real Robot

From the programming point of view, there is no difference between the simulated robot and the real robot: you do not have to change your program to switch between them. However, since simulation is always an idealization of data, especially regarding sensing, you should be ready to experience some variations between simulated and real data.

When running in client mode, you can switch between real and simulated robot using the REAL ROBOT item of the ROBOT menu in the ROBOT window. You can also switch from a program using the commands real robot and simulated robot. Again, these commands will have no effect in direct mode. Direct mode always uses the real robot.

If the application relies on realistic sensory data the direct mode should be used, since the simulation cannot capture all physical aspects of the real world.

C and C++

To control the robot from within a C program, you include the `Nclient.h` header file and link your program with the `Nclient.o` or `Ndirect.o` library. You can also use the robot from C++ by declaring all the commands appearing in `Nclient.h` as external C.

The Global Vectors

Information about the current state of the robot, its configuration and the readings of the sensors can be obtained by an application program through a global array, called the `State` vector. This structure is updated after the execution of a robot command (see the "*Commands" section* in this chapter for a detailed description). This structure is described in detail below. Table 4.1 provides a quick reference to `State`.

The name fields are values that are defined in `Nclient.h`; they should be used in application programs rather than the index into the array. This increases readability and is invariant to changes in the `State` vector.

The State Vector

In this section the fields of the state vector are explained in more detail. See also Section 6.1.

`STATE_SIM_SPEED` Simulation speed. The value is a factor to the speed of realtime; its unit is 1/10. Therefore, 10 corresponds to realtime, with a setting of 5 the simulation of one second will take two seconds (half the speed) and with a setting of 20 it will take 1/2 second (twice the speed).

`STATE_SONAR_0 - 15`
The readings of the sixteen sonar. The sonar are numbered counter-clockwise consecutively beginning with the front of the robot. The readings correspond to distances in inches (see "*Chapter 5 - Programming the Nomad*").

	Name	State Vector	Name
0	STATE_SIM_SPEED	speed of simulation	SMASK_POS_DATA
...	...	...	...
17	STATE_SONAR_0	sonar data #0	SMASK_SONAR_0
18	STATE_SONAR_1	sonar data #1	SMASK_SONAR_1
19	STATE_SONAR_2	sonar data #2	SMASK_SONAR_2
...	...	...	...
32	STATE_SONAR 15	sonar data #15	SMASK_SONAR_15
33	STATE_BUMPER	bumper data	SMASK_BUMPER
34	STATE_CONF_X	x position	SMASK_CONF_X
35	STATE_CONF_Y	y position	SMASK_CONF_Y
36	STATE_CONF_STEER	steering angle	SMASK_CONF_STEER
...	...	...	...
38	STATE_VEL_RIGHT	translational velocity	SMASK_VEL_TRANS
39	STATE_VEL_LEFT	steering velocity	SMASK_VEL_STEER
...	...	...	...
41	STATE_MOTOR_STATUS	motor status	--
44	STATE_ERROR	error number	--

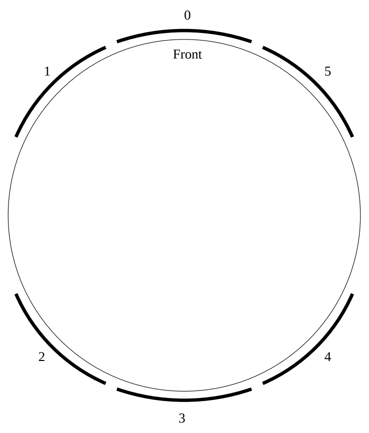


Figure 5.2 The Arrangement of the Bumper Sensors.

STATE_BUMPER
The readings of the bumpers. There is a total of six individial bumper sensors on the robot that are arranged in a ring. Refer to *Figure 5.2* for an illustration of their arrangement. Bumper sensor number n is represented by the nth bit in this value of the state vector. The 0th bit is the least significant one. A bit is set to one when the corresponding bumper is hit.

STATE_CONF_X
The integrated x-coordinate of the robot in 1/10s of inches with respect to the start position. This value is reset by the commands `zr` and `dp` (see the *"Motion Commands"* in this chapter).

STATE_CONF_Y
The integrated y-coordinate of the robot in 1/10s of inches with respect to the start position. This value is reset by the commands `zr` and `dp` (see *Sec-*

tion 5.3.2).

STATE_CONF_STEER
The orientation of the steering in 1/10s of degrees with respect to the start orientation, in the range [0; 3600). This value is reset by the commands `zr` and `da` (see the *"Motion Commands"* in this chapter).

STATE_VEL_RIGHT
The velocity of the right wheel in 1/10s of inches per second.

STATE_VEL_LEFT
The velocity of the left wheel in 1/10s of inches per second.

STATE_MOTOR_STATUS
The status of the motors. The lowest two bits correspond to the two motors. The next five bits apply to the new power management and sensing features of the Scout. *Figure 5.3* shows a bitmap of the status value.

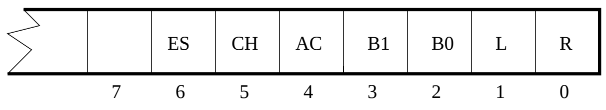


Figure 5.3 Status Value Bitmap

R: This bit is set when the right wheel is in motion.

L: This bit is set when the left wheel is in motion.

B1, B0: These bits have the following meaning:

B1	B0	Meaning
0	0	Low Battery
0	1	Med Battery
1	0	High Battery
1	1	Reserved

AC
This bit is set when the Scout is plugged into an AC source.

CH
This bit is set when the Scout is plugged into an AC source and the batteries are charging (i.e. the batteries are not fully charged.)

ES
This bit is set when the E-Stop is down. Some models of the Scout do not have an E-Stop, in which case this bit is always 0.

STATE_ERROR Error number
The state vector is updated by all robot motion and sensing commands (see the "*Commands"* section in this chapter). Note, however, that the family of commands that starts with get only updates a part of the state vector. For example, `get_sn` (get sonar) only updates the sonar readings. The command `conf_cp`, which configures the compass, is an exception in that it does not affect the state vector at all. If the user wants to force an update of the state vector, the command `gs()` (get state) can be issued (see the "*Commands"* section below).

Commands

This section introduces the robot language. It is intended as a general overview. For a detailed description of the commands please refer to the "*Language Reference Manual"*. Programming the Nomad requires the following steps:

- Establish communication with a robot
- Initialize the robot and its sensors
- Repeat until done:
  - Send motion and sensing commands to the robot
  - Get motion and sensing data from the robot
- Disconnect from robot

This illustrates the use of the three basic classes of robot commands:

- Communication commands to establish a connection to a robot.
- Motion commands to move the robot and to obtain its current configuration.
- Sensing commands to configure the sensors and to receive the sensory data.

In the remainder of this section one subsection is dedicated to each of these groups of commands. A common property of almost all of the commands is that they update the global vector State (see the *"State Vector"* section in this chapter). The value returned by the functions themselves is TRUE if the command was successfully transmitted to the robot, and state information came back correctly. It is not an indication that the command was successfully completed. Usually, the user's program will have to monitor the state of the robot (for instance, the robot integrated coordinates, or the robot speeds) to make sure that a command has been successfully executed.

The reason why monitoring of commands is required to determine success is that commands are executed asynchronously: the function itself will return immediately, and while the intended action starts on the robot, the program will move on to the next instruction. For instance, if you send `pr(1000,1000,0)`, a command that tells the robot to move forward by 100 inches, the command will return immediately (and probably even before you see the robot actually moving). If your program's next line is `pr(-1000,-1000,0)`, this will cause the robot to stop the previous motion and start this next one, requesting a move into the opposite direction. The only exceptions are the commands `zr` and `ws`, that initialize the robot's encoders and wait for the robot to stop, respectively. These functions will only return after the execution of the command has been completed.

Communication Commands

The application program can connect to the server or directly to the robot. This is determined by the object file the application is linked with. In case of `Nclient.o` the connection will be established with the server and in case of `Ndirect.o` with the robot.

If the user decides to use the server two variables can be used to specify the connection:

`SERVER_MACHINE_NAME` is a string that should contain the name of the machine the server is running on or its IP-address. If that name is computer, for example, you have to issue the C-command `strcpy` (`SERVER_MACHINE_NAME`, "computer"); before connecting to the server. Furthermore, the variable `SERV_TCP_PORT` determines to which socket the application program will connect to. For the server and the application to be able to communicate they both have to connect to the same socket. The socket number the server connects to is specified in the file `world.setup`. Assign the number given there to the variable `SERV_TCP_PORT` before connecting to the server.

When connecting directly to the robot the variable `ROBOT_MACHINE_NAME` has to contain the name or the IP address of the robot. This variable can be assigned by using the C-command `strcpy` (`ROBOT_MACHINE_NAME`, `"128.12.24.8"`);, assuming that the IP-address of the robot is `128.12.24.8`.

For the case of communication failure, the user can specify a timeout for the robot. This is done with the command `conf_tm`. After the specified time has elapsed the robot will stop; if the application program crashes this hopefully prevents the robot from crashing into a wall.

`int create_robot (long robot_id)`

Causes the server to create a new robot with the ID `robot_id`. A robot window and the sensor windows will be created. After the creation of the robot the application program can connect to it. The return value is the argument upon success, otherwise zero. This command has no effect when connecting directly to the robot using `Ndirect.o`.

`int connect_robot (long robot_id)`

Establishes a socket connection between application program and server (if using `Nclient.o`) or between application program and robot (if using `Ndirect.o`). Only after this command has been issued successfully commands can be sent to the robot. The return value is the argument upon success, otherwise zero.

`int disconnect_robot (long robot_id)`

Disconnects the application program from the server (if using `Nclient.o`) of from the robot (if using `Ndirect.o`). The return value is `TRUE` upon success, otherwise `FALSE`.

`int tk (char *talk_string)`
`(talk)`

If speech synthesis is present this command will send the string `talk_string` to it. With this command you can make the robot speak. This is particularly helpful for debugging.

`. unsigned int conf_tm (int timeout)`
`(configure timeout)`

Sets the timeout of the robot to timeout seconds. If the robot does not receive any commands within this period the robot is stopped.

Motion Commands

To fully understand the robot motion commands it is helpful to have some knowledge about the drive system of the robot, which consists of two independent axes. These axes are the right and left wheels. The combination of these two wheel motions can produce translational motion, rotation, or a combination of these to describe an arc.

These two axes can be controlled in two different control modes: velocity and position control. In velocity control the goal is to maintain a given velocity, whereas position control attains and maintains a given position relative to the current position of the robot.

For all motion commands velocities are specified in 1/10s of inches per second. Positions are specified relative to the current position. All commands return `TRUE` upon successful transmission to the robot, `FALSE` otherwise.

The format of commands which control each axis independently is (right wheel, left wheel, unused), where the unused value should be passed as zero and ignored on return. It is included for API backwards-compatibility with models that have three axes of control.

`int ac (int r_ac,int l_ac,int unused) (acceleration)`

This command sets the accelerations for the two axes in units of 0.1in/s^2. The acceleration must be lower than 800 0.1in/s^2. When doing only very small motions with position control the accelerations should be set to small values; otherwise the robot will accelerate very abruptly, move further than the desired position, accelerate abruptly into the opposite direction to compensate for the error, and so on.

`. int sp (int r_sp, int l_sp, int unused)(speed)`

This command sets the maximum speeds for the two axes. The maximum velocities must be lower than 400 0.1in/s^2.

`. int vm ( int r_vm, int l_vm, int unused) (velocity move)`

This command controls the two axes of the robot in velocity mode. The arguments to this function are the desired velocities for the two axes; they can be negative but their absolute value has to stay below the maximal value given above (see `sp`). A velocity of zero will maintain the current position.

When a `vm` command is issued the robot will move its axes at the requested velocity and will continue moving unless another motion command is issued, or a timeout occurs (see `conf_tm`).

`. int pr (int r_pr, int l_pr, int unused) (position relative)`

This command controls the three axes of the robot in position mode. The arguments are the desired position relative to the current one.

Note that if the distances are different, one axis may complete its motion before the other. Note also that the function call will return immediately. To wait for the motion to be completed the user should issue the command `ws` after the `pr`. A parameter of zero will maintain the current position.

```
int mv (int r_mode,int r_mv,int
l_mode,int l_mv,int unused1,int
unused2)            (move)
```

This command allows the user to drive both axis independently from each other in velocity or position control mode. There are two arguments for each axis:

- The first argument determines the control mode by specifying

  - `MV_VM` for velocity control,

  - `MV_PR` for position control, and

  - `MV_IGNORE` if the motion for that axis should remain unaltered, and

- The second argument is a velocity if velocity control is the specified mode or a position if position mode was requested.

For example, `pr (MV_VM, 100, MV_VM, 100, MV IGNORE, MV IGNORE)` will cause the robot to start translating at 10 inches per second.

`int st (void )` (stop)

The robot will stop after this command has been sent. However, since the function call returns immediately, the robot can still be decelerating after the termination of the command. It is recommended to issue a `ws` (see below) after an `st`.

`int ws (unsigned char r_ws, unsigned char l_ws,unsigned char unused, unsigned char time-out)`
`(wait for stop)`

This allows the user to wait for some or all of the axes to be stopped. The first three arguments are `TRUE` if the command should return after the stopping of that particular axis; the argument should be `FALSE` if the status of an axis does not need to be monitored. If not all of the specified axes have stopped before timeout seconds have elapsed the function returns.

. `int lp (void)`
`(limp)`

After the robot has stopped it will still try to maintain its position. That means that any attempt to push it will cause the motors to drive the robot into the opposite direction. This command cause the robot to go in limp mode. After executing it, position will no longer be maintained and the robot can be pushed around. This control mode is sometimes referred to as floating.

`int zr (void) (zero)`

This function resets the internal coordinates of the robot to (*x*, *y*, *theta*) = (0, 0, 0). It has no external effects, however.

`int dp (int x, int) (define position)`

Sets the robot position to (*x*,*y*).

`int da (int th, int unused) (define angles)`

The orientation of the steering is set to th 1/10s of degrees.

`int get_rc (void)`
`(get robot configuration)`

Issuing this command updates the following entries in the `state` vector: `STATE_CONF_X`, `STATE_CONF_Y`, `STATE_CONF_STEER`, `STATE_CONF_TURRET`.

`int get_rv (void)`
`(get robot velocities)`

Issuing this command updates the following entries in the state vector: `STATE_VEL_RIGHT`, `STATE_VEL_LEFT`.

Sensing Commands

For each of the sensors there exists a command to configure it and another one to obtain the readings. In general the sensory readings will be obtained with the `gs` command (see below) that stores readings in the `State` vector. However, dedicated functions exist to save bandwidth.

`int gs (void) (get state)`

Updates the `State` vector.

`int conf_sn (int rate, int order[16])`
`(configure sonars)`

This command will configure the sonar; please refer to the *Language User Manual* for details on the configuration of the sonar.

`int conf_tm (unsigned int time - out)`
`(configure timeout)`

Sets the timeout of the robot to timeout seconds. If the robot does not receive any commands within this period, the motion will be stopped.

`int get_sn (void)`
`(get sonar)`

This command will update the sonar data in the `State` vector.

`int get_bp (void)`
`(get bumper)`

This command will update the bumper data in the `State` vector.

5.3.4 Server Commands

The commands introduced in this section require the application to be connected to a server. Called from programs that connect directly to the robot (linking with `Ndirect.o`) they will have no effect. For a more elaborate description of these commands please refer to the "*Language Reference Man-*

ual."

Commands related to the server:

`server_is_running (void)`

Returns TRUE if a server is connected to the same socket the application is connected to.

`quit_server (void)`

Called from an application that is connected to a server, this command will cause the server to exit.

Commands related to the world representation: The application program can modify the world data base of the server. The polygons that are passed as arguments to the following functions have to be convex and the points are given in counterclockwise order.

`add_obstacle (long obs[2*MAX VERTICES+1])`

Add an obstacle to the world representation of the server.

`delete_obstacle (long obs[2*MAX VERTICES+1])`

Delete an obstacle from the world representation of the server.

`move_obstacle (long obs[2*MAX VERTICES+1], long dx, long dy)`

Translate an obstacle of the server's world representation by (*dx, dy*).

Commands related to drawing in the map window:

`draw_robot (long x, long y, int th, int unused, int mode)`

This command draws a robot at the given configuration into the ROBOT window of the server. For a description of the arguments and mode refer to the "*Language Reference Manual*".

`draw_line (long x_1, long y_1, long x_2, long y_2, int mode)`

This command draws a line into the ROBOT window of the server. For a description of the arguments and mode refer to the "*Language Reference Manual*".

`draw_arc (long x_0, long y_0, long w, long h, int th1, int th2, int mode)`

This command draws a circular arc into the map window of the server. For a description of the arguments and mode refer to the "*Language Reference Manual*".

`place_robot (int x, int y, int th, int unused)`

With this command the client program can request the user to input a robot position via the graphic interface.

Commands to switch between real and simulated robot:

`real_robot (void)`

Called from an application program, this command causes the server to send the commands (from the application program) to the real robot instead of sending them to the simulated robot.

`simulated_robot (void)`

This command has the inverse effect of `real_robot`.

Some Examples

In this section, the development of a small application program is presented. Step by step, we are going to increase its capabilities to demonstrate how to program the robot. The first program in *Figure 5.4* just connects to the robot, initializes it, moves 10 inches forward, and finally disconnects again. It will be the starting point for further refinement. For information on how to compile programs and how to run them in connection with the server please refer to the section in "*Chapter 3 - Starting a Server*"; you will have to adapt the first two lines of the procedure main to adjust the communication parameters to your specific environment. This program should be used in conjunction with a server. The user can then choose if the real or the simulated robot will be used.

If your program connects to the real robot you should see the robot move ten inches forward. Nothing will happen if you use the simulated robot. The reason for this is that the simulator only updates the screen when the `State` vector is updated. The simulator basically displays the robot as represented by the state vector. If you manually issue a `gs` command in the command line window you will be able to see the result of this little program.

Now we will extend this program to move the robot in a square. While it is moving it will monitor the bumpers and if one of the bumpers is hit the program will abort.

To make the robot move in a square we will issue eight motion commands: first a translation, then a steering rotation, repeated for the four sides of the square. However, `pr` does not wait for the current motion to be completed, so the user must be careful not to overwrite a commanded motion before it is completed. We will implement a function that monitors the velocity of a given axis. This will allow the program to determine when a motion has been completed. In order to do so we could use the command `ws`, but we want our program to issue the command `gs` while waiting so that the simulator gets updated and we are able to follow the motion on the screen.

```
void wait_for_stop (int axis)
{
/*
 * Example 1 Version 1
 *
 * This program will connect to the robot, initialize it,
 * move forward monitoring the bumpers, and disconnect.
 *
 */

#include "Nclient.h"
#define ROBOT_ID 1 /* the ID of the robot */

void main (void)
{
  int i;

  /* setup connection parameters */
  SERV_TCP_PORT = 7019;
  strcpy (SERVER_MACHINE_NAME, "computer");

  /* establish connection */
  if (!connect_robot (ROBOT_ID) )
    exit (0 );

  /* initialize robot */
  conf_tm (2);
  zr ();
  ac (200, 200, 0);
  sp (100, 100, 0);

  /* move robot 10 inches */
  pr (100, 100, 0);
  /* disconnect */
  disconnect_robot (ROBOT_ID);
```

Figure 5.4 The First Example.

```
void wait_for_stop(int axis)
{
  /* wait for motion to begin */
  while (State [axis] == 0)
    gs();
  /* wait for motion to end */
  while (State [axis]!= 0)
    gs();
}
```

The above procedure requires as an argument the index to the `State` vector that contains the velocity of the axis to be monitored. It first waits for the motion to be started by remaining in the first `while`-loop until the velocity differs from zero. Then it waits for the motion to be completed; this is recognized by the velocity becoming zero again. Inside both `while`-loops the command `gs`() is executed to update the `State` vector, and therefore the simulator screen.

We will now extend the procedure `wait_for_stop` to monitor the bumper while waiting for the motion to be completed. This can be realized using the bumper reading in the state vector.

```
int wait_for_stop_or_bumper (int axis)
{
  /* wait for motion to begin */
  while ((State [axis] == 0) && (State [STATE_BUMPER] == 0))
    gs();
  if (State [STATE_BUMPER]!= 0)
    return (FALSE);

  /* wait for motion to end */
  while ((State [axis]!= 0) && (State [ STATE_BUMPER ] == 0))
    gs();
  if (State [STATE_BUMPER] != 0)
    return (FALSE);
  return (TRUE);
}
```

Now the `while`-loop is also terminated when the bumper data assumes a value different from zero, which indicates a bumper hit. The new procedure wait for stop or bumper returns TRUE if the motion was completed without a bumper hit and it returns FALSE if a bumper was hit. This allows us to write the second version of our example program, listed in *Figure* 5.5. In that listing the code for the procedure wait for stop or bumper has been left out for brevity.

```
/*
 * Example 1 Version 2
 *
 * This program will connect to the robot, initialize it,
 * move in a square monitoring the bumpers, and disconnect.
 */

#include "Nclient.h"
#define ROBOT_ID 1 /* the ID of the robot */

int wait_for_stop_or_bumper (int axis)
{ . . . }

void main (void)
{
  int i;

  /* setup connection parameters */
  SERV_TCP_PORT = 7019;
  strcpy (SERVER_MACHINE_NAME, "zulu");

  /* establish connection */
  if (!connect_robot (ROBOT_ID))
    exit (0);

  /* initialize robot */
  conf_tm (2);
  zr ();
  ac (200, 200, 0);
  sp (100, 100, 0);

  /* move in a square */
  for (i = 0; i < 4; i ++)
 {
    /* move straight */
    pr (200, 200, 0);
    /* we can only wait for one wheel using this function.
     * consider the advantages and disadvantages of the ws() function
     * over our wait_for_stop_or_bumper()
     */
    if (!wait_for_stop_or_bumper (STATE_VEL_RIGHT))
      break;

    /* turn 90 degrees */
    scout_pr (0 , 900);
    if (!wait_for_stop_or_bumper ( STATE_VEL_RIGHT))
```

```
      break;
 }
  /* stop robot */
  st();
  /* disconnect */
  disconnect_robot (ROBOT_ID);
}
```

Figure 5.5 The Second Version of the First Example

Having gained some familiarity with the robot, we are now going to develop a wanderer program. This program will cause the robot to wander around in an unknown environment. It will move forward until the front sonars detect an obstacle, then it will choose a new direction for motion based on the reading of the sonar to the rear and on the side. First, we will develop a few procedures that will be helpful for the wanderer program.

```
/* returns the minimum sonar reading of the front 5 sonars */
int get_min_dist (void)
{
  int i;
  int min_value = 255; /* max. sonar reading */

  /* use sonars 0,1,and 2 and remember minimum value */
  for (i = 0; i < 3; i++)
    if (State [STATE_SONAR_0 + i] < min_value)
      min_value = State [STATE_SONAR_0 + i];

  /* use sonars 14 and 15 and remember minimum value */
  for (i = 0; i < 2; i++)
    if (State [STATE_SONAR_14 + i] < min_value)
      min_value = State [STATE_SONAR_14 + i];

  return (min_value);
}
```

The function get min sonar uses sonars 1, 2, 3, 15, and 16. It determines the shortest sonar reading and remembers it in min_value. Once all five sonar are examined, the smallest value is returned. This will help us in determining if an obstacle is in our way. Once we have determined that this is the case we want to choose another direction to wander. That can be done with the following procedure:

```
/* returns the angle that is the freest direction */
int get_best_dir (void)
{
  int i;
  int max_value = 0; /* min. sonar reading */
  int best_dir; /* to remember the direction */

  /* use sonars 3 through 13 */
  for (i = STATE_SONAR_3; i <= STATE_SONAR_13; i++)
```

```
    if (State [i] < max_value)
    {
      /* remember value and direction */
      max_value = State [i];
      best_dir = i - STATE_SONAR_0;

  }
  /* do we want to rotate left or right? */
  if (best_dir < 8)
    best_dir = best_dir - 16;

  /* return angle to rotate in 1/10 of degrees if enough space */
  if (max_value < 30)
    return (best_dir * 225);
  else
    return (-1);
}
```

This function returns the angle in 1/10s of degrees that we should rotate in order to be facing into the direction with the most free space. In the for-loop we find the largest sonar reading among the sonars 4 through 14; we remember that value in the variable max value. The variable best dir stores the number of the sonar that had this value as its reading. In order to do so we have to subtract `STATE_SONAR_0`, otherwise we would store the index of that sonar in the state vector.

After these values have been determined, we can compute the angle that we have to rotate the robot by to face into the freest direction. However, since it is faster to rotate clockwise by 10° that it is to rotate counterclockwise by 350° we change best dir to the appropriate negative number if necessary.

Since there should be a way to indicate that there is no free space at all, the function will return -1 if the largest sonar reading does not exceed 30. Otherwise the angle is returned. As two neighboring sonar have an angle of 22.5°, we simply have to multiply by 225 to obtain the value in 1/10s of degrees.

Using these two functions and the `wait_for_stop` routine from the previous example the wanderer looks as follows (again, for brevity the bodies of the procedures are omitted):

```
/*
 * Example 2 The wanderer
 *
 * This program will let the robot wander around.
 */

#include "Nclient.h"

#define ROBOT_ID 1

/* returns the minimum sonar reading of the front 5 sonars */

int get_min_dist (void)
```

```
{ . . . }

/* returns the angle that is the freest direction */
int get_best_dir (void)
{ . . . }

/* wait for a certain axis to stop */
void wait_for_stop (int axis)
{ . . . }

void main (void)
{
  int i;
  int angle;

  /* setup connection parameters */
  SERV_TCP_PORT = 7019;
  strcpy (SERVER_MACHINE_NAME, "computer");

  /* establish connection */
  if (!connect_robot (ROBOT_ID))
    exit (0);

  /* initialize robot */
  conf_tm (2);
  zr ();
  ac (200, 200, 0);
  sp (100, 100, 0);

  /* as long as there is free space */
  while ((angle = get_best_dir ())!= -1)
  {
    /* turn into that direction */
    pr (0, angle);
    wait_for_stop (STATE_VEL_RIGHT);

    /* start wandering and monitor sonars */
    vm (100, 100, 0);
    while (get_min_dist () < 30)
      gs();

  /* stop the robot and wait for it to be stopped */
  st();
  ws (TRUE, TRUE, TRUE, 3);}
  /* disconnect */
  disconnect_robot (ROBOT_ID);
}
```

This little program tries to keep a safe distance from obstacles. You might want to try to improve it.

CHAPTER 6

ADVANCED FEATURES

Using the Sensors

The Sonar Sensors

Sensor Description

The Sensus 200 is a ring of 16 Polaroid 6500 sonar ranging modules. The Polaroid 6500 is an acoustic range finding device that has been widely used in the mobile robotics community. It can measure distances from 6 inches to 35 feet, with a typical absolute accuracy of +- 1 percent over the entire range.

The standard Polaroid system in long-range mode (the one currently used on the Nomad robot) emits 56 cycles of a 49.4 kHz square wave through a transducer. A blanking period follows, during which the internal circuitry is reset and stabilized. The transducer then acts as a receiver, feeding the detected echoes in a time variable gain amplifier. The gain factor of this device increases with time to compensate for spreading loss and the attenuation of sound in air. The output of the amplifier then goes to a thresholding circuit. As soon as the threshold is exceeded, the time elapsed since the beginning of the transmission of the pulse is measured, and converted into distance through an appropriate calibration factor.

Sensor Physics

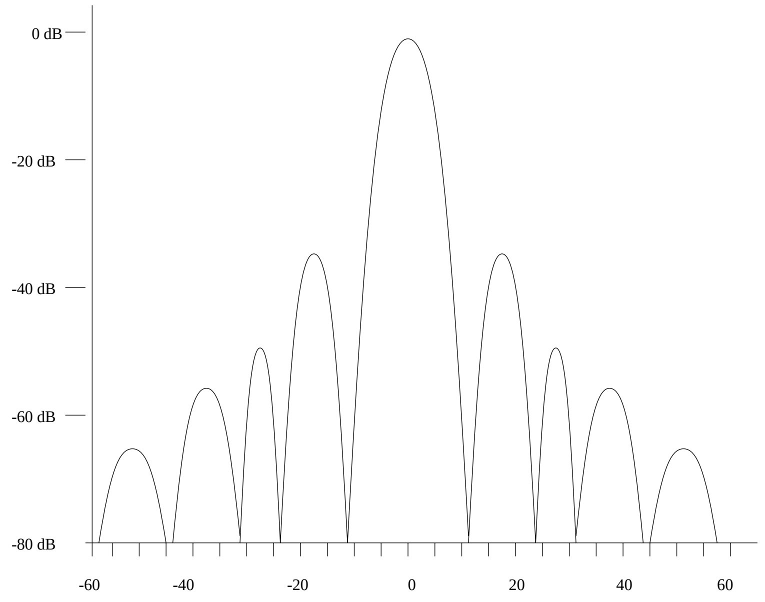


Figure 6.1 Modelized sound radiation for the Polaroid Transducer (from Leonard)

Sound characteristics The transducer does not emit energy homogenously in all directions, but instead form lobes of decreasing intensity, as illustrated in the chart of *Figure 6.1*[1].

According to the results of extensive experiments conducted by various researchers (Lang, Kuc, Leonard), the radiation pattern of Polaroid Transducers is not symmetric, and varies from one transducer to another. These effects are more significant for the side lobes.

The attenuation of ultrasound in air increases with frequency, and depends on temperature and humidity. Typical values for 50 KHz are an attenuation of 0.6 to 1.8 dB/m for variations in temperature from 17 to 28°C and variation in relative humidity from 15 to 70%. The speed of sound in air is expressed as c = 331.4 * sqrt(T/273) m/sec, T

1. Most of the information relative to sonars in this section come from Leonard J., DurrantWhyte H., Directed Sonar Sensing for Mobile Robot Navigation, Kluwer Academic Publishers, 1992. Thanks to Dr. Billur Barshan, Bilkent University, Turkey for useful comments on the physics of the Sonar transducers.

being the ambient temperature in degrees K.

Electronics characteristics

The following three sources of errors in range measurements are related to the specifics of the sensor circuitry (Leonard):

- **Transmitted Pulse duration**

  All the timing is based on the assumption that the start of the transmitted pulse is the part of the returned echo that actually exceeds the detector threshold. If this is not the case, the error can be as much as 8 inches.

- **Time Variable gain amplifier**

  The ideal exponential curve that would exactly cancel beam spread and attenuation losses is approximated by a 12-step only piecewise constant function. Even if this function was exactly given, it should be different according to temperature and humidity conditions. Since the returned energy is a function of the incident angle (as shown in the radiation pattern), the visibility angle changes with range.

- **Capacitive charge-up in the threshold circuit**

  For strong reflected signals, 3 cycles are enough to charge up to the threshold: the calibration usually accommodates that delay. For weaker signals, charge-ups can take place over a considerably longer time, resulting in erroneously elongated range values.

  One major source of uncertainty in distance estimation coming from these characteristics is the existence of weak returns, as opposed to strong returns:

- Strong returns possess enough energy to exceed the threshold promptly, giving very accurate measurements,
- Weak returns cause time-delay range errors: the threshold is reached only after a long charge-up and changing gain in the amplifier. The threshold is only exceeded by the random combination of a slow charging-up period and jumps in the non-linear time variable gain amplifier.

Target characteristics

Targets can be divided in two groups:

- Reflecting objects, of dimensions larger than the wavelength (6.95 mm at 20°C),
- Diffracting objects, of dimensions smaller than the wavelength.

Objects whose overall size is smaller than the wavelength are usually rare (one can think of wire fences for instance), but rough surfaces like concrete, or textured walls present small asperities which actually behave as diffractors, as the experimental data will show. Smooth surfaces like metallic desks, painted walls, doors, etc, are reflectors. Those are the most common in indoor environments, which supports the commonly heard assertion that most indoor surfaces act as mirrors with respect to sound waves. Slightly rounded convex edges with radius of the order of the wavelength produce weak specular echoes: some ornamental (carved) table legs and cardboard boxes often have this character.

One consequence of the reflective properties of surfaces is the multiple echo effect: the sound wave bounces around, and eventually reaches the receiver after several reflections: because of attenuation, the energy level of the incoming wave is very likely to be quite low, one possible origin of what is called a "weak return".

Typical Sonar Data

A good representation of sonar data is the sonar scan: a dot representing the measured distance is drawn on the sonar axis. Repeated measurements over a given angular range, for instance one every degree over the whole circle, give a sonar scan. Sonar scans are the basic data that can be used for map building or navigation purposes. Since the sonar are mounted horizontally on the robot, which can itself be oriented to any angle, one of them can be used directly to make measurements, using the robot to set it at desired angle and distance with respect to the targets.

- **Smooth flat surface**

  The scan of *Figure 6.2* is typical of the response of a flat, smooth surface. The central feature is an arc of circle, centered with respect to the surface's normal. This pattern is caused by the main lobe: although the sonar is rotating, as long as there is in the active cone of the main lobe a patch of surface normal to the direction of the wave, it gets reflected back. Since we are in the main lobe, where energy is high, the thresholding circuit is promptly triggered, and the measurement is very precise: we get an arc of circle of constant radius.

When the normal to the surface leaves the main lobe, the energy is reflected away from the source. However, the secondary lobe comes into effect, with the same property: if the normal to the surface is within the secondary lobe, the sound gets reflected. Since the secondary lobe carries less energy than the main lobe, we have a time-delay range error because the thresholding circuit takes more time to be triggered. We also get an arc of circle, but of slightly greater radius. Note that the secondary lobe dots may be accidentally aligned with the actual wall, which in the past have induced people to try to match lines to sonar readings, but this is purely coincidental: physics of the sonar dictates that circles should be fit to sonar scans.

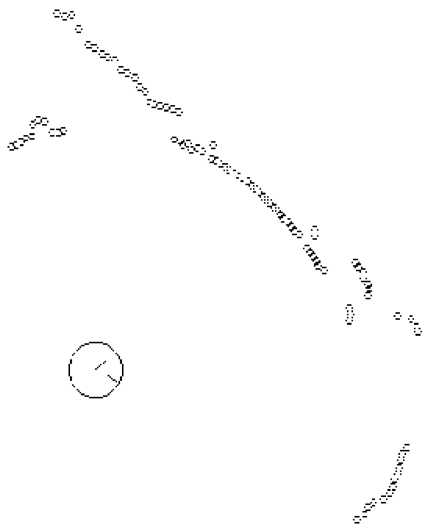


Figure 6.2 Sonar Scan on a Flat and Smooth Surface

The sonar rotating further, the secondary lobe also leaves the normal to the surface: nothing can be seen any more (there are actually other lobes to follow, until the whole half circle, but they are too weak to give an answer), and the plots are drawn to infinity (equal to the maximum detectable distance). One interesting phenomenon can be observed at the junction of the main and secondary lobe: provided that the scan is detailed enough, one consistently gets readings that are extremely erroneous, more than twice the actual distance. These are caused by some low energy coming at the junction of the two lobes. The pattern of *Figure 6.1* shows a single frequency (f=49.9KHz), with zero energy at the junction of two lobes. The Polaroid sensor has a finite bandwidth around this resonant frequency. Therefore, the sidelobes of a range of frequencies are superposed. Each frequency produces zeroes at slightly different locations, and the net superposed effect is not zero.

■ **Rough flat surface**

The scan of *Figure 6.3* is typical of a rough surface. It can be understood if we consider that a rough surface is made of numerous small asperities, of dimension close to the sound wavelength. Thus, the primary lobe always picks a small patch normal to its axis, and return the corresponding distance. This is the only case where the sonar scans bears a strong resemblance with the actual map of the room.

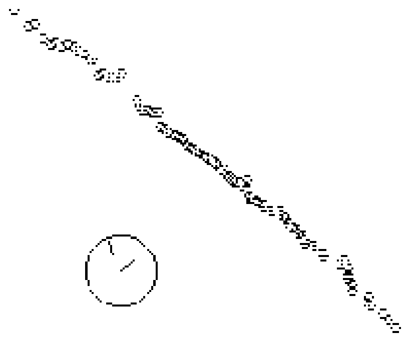


Figure 6.3 Sonar Scan on a Flat and Rough Surface

■ **Corner**

The scan of Figure 6.4 is typical of a smooth corner. The central pattern is an arc of circle centered about the corner. It comes from a double specular reflection at the corner (one from each surface). We also have two neighboring arcs that we can attribute to the secondary lobes. The leftmost and rightmost arcs come from specular reflection on the corner's sides. One interesting point is that the corner's side reflections could also give secondary lobe arcs, if the sonar was a bit farther from the corner. In the present case, it happens to be that these secondary lobe arcs actually come from the corner, as we verified by unfolding it (then the arcs disappeared).

Note that when the sonar is close to the corner, it is difficult to predict where the returned echo is actually coming from, because there is a competition between a weak, but specular echo (the secondary lobe on the corner side), and a strong, but distant echo (the main lobe on the corner). Additionally, there may be multiple echo phenomena that can create additional error, depending whether the first echo to trigger the threshold circuit comes from a single-hit echo, or from multiple reflections on the corner sides.

The language commands related to sonar sensors are `conf sn`,`get_sn`; the sonar data is stored in the `State` vector, at indexes ranging from `STATE_SONAR_0 to STATE_SONAR_15`. Refer to "*Chapter 5 - Programming the Nomad*", and to the "*Language Reference Manual*".

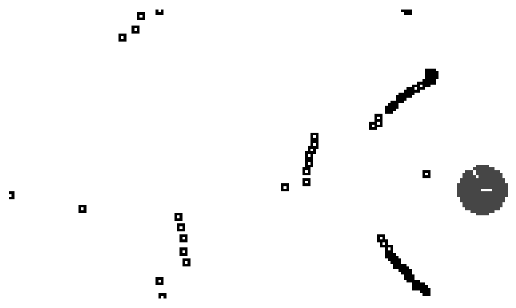


Figure 6.4 Sonar Scan on a Corner

The Setup Files

The setup files are a convenient way to set a number of configuration parameters for the applications using the Nomad robot. The file world.setup mostly sets graphic parameters for the MAP window of the graphic interface. It also sets the communication port (Unix socket) that will be used by the client programs to connect to the server, and the default configuration file(s) for the robot(s). The `robot.setup` file sets graphic parameters for the ROBOT window, and simulation parameters. It also sets the connection parameters for the robot.

You can have only one setup file for the MAP window; its default name is world.setup: this name is used when you start an `Nserver` without arguments on the command line. If you do use arguments, the first one should be the name of the MAP window setup file (any name), and the following arguments should be the name(s) of ROBOT setup files. The server will automatically create as many robots as there are ROBOT setup files. There is also an option in the MAP setup file that allows you to specify ROBOT setup files as well. Any command line specification will override the setup file specification.

If you want to change any of the parameters, you simply edit the file and modify the values. Upon restart of the Nserver, your new values will take effect. The syntax of both files is the same: after the Copyright banner, the file is subdivided in sections with a section title in brackets:

```
[graphics]
```

In each section appear items with their values, separated by the equal sign:

grid origin = +0+0

Comments start with a semi column and end at the end of the line. Line continuation (if the data is too long to fit on one line) is possible by putting a backslash at the end of each line except the last one.

The world.setup file

```
[physical]
```

- `size` encodes the physical size of the world for simulation and graphic representation purposes. The world is rectangular; its size is expressed in tenth of inches. World units are also tenths of inches. After the size comes the origin. An origin of (0,0) will be at the left bottom corner of the rectangle. Shifting the world by -Width/2, -Height/2 will set the origin to the middle of the rectangle, which is the default. The default value of 87360x87360-43680-43680 is a square world of 1/64th sq. mile, with origin at the center.

[graphics]

- `graphics` is off if the graphic interface (Map window, robot window) is to be disabled by default, for instance to speed up simulations. This can also be done by toggling the Graphics ON switch in the VIEW menu of the MAP window.
- `world_geom` is the pixel size and position of the MAP window, expressed in X fashion, upon creation of the graphic interface. The position is the position of the left top corner of the MAP window relatively to the left top corner of the screen. These can be modified using the window manager.
- `world_zoom` is the default zooming factor; 100% is 1 pixel per world unit (1/10th inch). 50% is 0.5 pixel per world unit. The default 12.5% is 0.125 pixels per world unit, which is 1 pixel for 8 tenths of an inch. Zooming can also be achieved using the zooming options of the VIEW menu.
- `world_center` is the world position of the center of the MAP window upon creation of the graphic interface. It is expressed in world coordinates (10th of inches) in the world coordinate system. This position can also be changed in a variety of ways: the horizontal and vertical slides of the MAP window, and all the options: ZOOM IN, ZOOM OUT, CENTER, CLIP, SLIDE, of the VIEW menu.
- `grid_origin, grid_incr` relate to a rectangular grid laid on the MAP window. This grid is a visual aid for the user: it provides the world dimensions, and can be used (for instance) when creating obstacles. The origin and increment size of the grid are expressed in world coordinates, relative to the world origin.
- `grid_dot_color` sets the colors for the grid dots.
- `grid_shad_color` sets the colors for the shadow dots. These shadow dots are offset by 1 pixel right and down relative to the regular grid dots. They allow the grid to be seen even in the presence of obstacles.
- `default_map` is used as a short cut if you use frequently the same map and want to load it automatically upon creation of the server. It is set to a path to the map you want to use, for example: `/maps/my office.map`. The default value is `none`.

[menu toggle]

- `display` is set to `solid` if you want the robot to be represented as a solid circle in the MAP window, to `outline` if you want it to be represented as a hollow circle.
- `encoder` and `actual` enable the display of the encoder and real robots (see *Section 4.1.2*). The default value is `on`. This display can be dynamically changed using the SHOW menu of the MAP window.
- `grid` enables the display of the grid; its default value is `on`.

[connect]

- `serv_port` is the number of the port that client programs may use to connect to the server. This number has to match the number set by the client program (variable `SERV_TCP_PORT`).

[robots]

- `setup_files` sets the names of the robot setup files. This is used if you usually work with several robots, and want to avoid creating them by hand each time you run an `Nserver`. Each name is separated from the next by a white space. The server will create as many robots as there are setup files. The setup files specification can be overridden by command line arguments.

☑ If you want to create three identical robots, put the name of your setup file three times on this line.

The robot.setup file

[simulation]

- `sim_speed` is the ratio simulated time/clock time. If set to 2.0, the simulation will run twice as fast as the real time. The simulation can also be accelerated / decelerated using the CONTROL menu of the MAP window.

- `timeout` is the default timeout value for the simulated robot. It is equivalent to the default value given in the file `timeout.cfg` on the real robot (`/usr/local/share/robotd`). See also the command `conf_tm`.
- `translation` sets the default speed for both the right and left wheels. See also the command sp.
- `steering and rotation` are ignored when the simulator is in Scout mode.

`[infrared]`

Everything in the infrared section is ignored when the robot is a Scout.

[sonar]

- `firing_rate and firing_order` have the same meaning as the corresponding parameters of the function `conf_sn`: the interval between two firings in slices of 0.004 secs, and the firing order. However, since dynamic simulation of the sonars is not implemented, only the list firing order is actually used to detect which are the active sonars. As with the parameter order[] of the function `conf_sn`, a marker 255 can be used to stop the list.
- `dist_min` and `dist_max` set the range of the simulated sonars in tenths of inches.
- `halfcone` is half the angular range of the main lobe of the sonar, in tenths of degrees. If this value is set to 0, a simple ray model of the sonar is used, in which the actual distance to the object hit by the beam is simply modified by the error coefficient described below. If the value is not zero, a cone is used to simulate the sound beam, and the incidence angle of the beam to surfaces is considered. Using the simple model increases the speed of the simulation.
- `critical` is the maximum angular difference in tenths of degree between the sonar axis and the normal to the surface for the sensor to return a value. If the difference is larger, the simulated sonar is assumed not to have got any echo back, and the maximum distance will be returned.
- `overlap` sets the minimal apparent size of a surface to be detected when using the conical model. It is expressed as a ratio (0.5 = 50%) to the angular range of the main lobe. If the angular sector connecting the two endpoints of a segment within the sound cone is larger than *overlap* * (2 * *halfcone*), then this segment is considered visible.
- `error` is an absolute random error factor. It is expressed as a percentage of the real value: the resulting distance is computed as *value* * (1 + *random*[-1,1] * *error*), so if the item error is set to 0.2, the simulated values will be randomly distributed between 80% and 120% of the geometric values.

`[laser]`

This section is ignored when the simulator is in Scout mode.

`[other]`

- `bumper_mode` is `on` if the simulated robot has bumpers, `off` if not.
- `compass_mode` is ignored by the Scout.

`[mask]`

This value is ignored on the Scout.

`[motion]`

- `bounce` sets the distance that the simulated robot should go back after hitting a wall. The value is the time in seconds during which the bouncing motion occurs. The distance bounced is the current speed multiplied by this value. Note that a simplified control model is used: the wheels are supposed to be still spinning forward during the bouncing motion, so that the encoder robot will move on (and will probably cross the obstacle) while the actual robot will be pushed off the obstacle.
- `pos_*_bias, zero_*_bias and neg_*_bias` set the error model for the motion of the two axes: Right wheel and Left wheel. They are percentage errors on robot velocities. For both axes, the same model is used:

  - systematic bias: zero_*_bias

- directional bias: pos_*_bias and neg_*_bias

- directional random error interval: pos_*_int and neg_*_int;

Directional bias means a bias related to a particular direction of motion, positive or negative. If commanded * is the commanded value (translation, steering or rotation), then the simulated value will be:

```
if (commanded_* > 0)
simulated_*=commanded_*(1+zero_*_bias
    +pos_*_bias)(1+random[-
    1,1]*pos_*_int)
else
simulated_*=commanded_*(1+zero_*_bias
    +neg_*_bias)(1+random[-
    1,1]*neg_*_int)
```

The provision for a difference in negative and positive is to take into account gravity when the robot has to move on slanted surfaces. It is actually something that has been implemented for future releases when the client program will be able to set simulation parameters. For now, the difference can be ignored.

`[graphics]`

- `wind_name` is the name of the ROBOT window for the associated robot.
- `robot_geom` is the pixel size and position of the ROBOT window, expressed in X fashion, upon creation of the graphic interface. The position is the position of the left top corner of the ROBOT window relatively to the left top corner of the screen. It can be modified using the window manager.
- `robot_zoom` is the default zooming factor; 100% is 1 pixel per world unit (1/10th inch). 50% is 0.5 pixel per world unit. The default 12.5% is 0.125 pixels per world unit, which is 1 pixel for 8 tenths of an inch. Zooming can also be achieved using the zooming options of the VIEW menu.
- `robot_center` is the world position of the center of the ROBOT window upon creation of the graphic interface. It is expressed in world coordinates (1/10th of inches) in the world coordinate system. This position can also be changed in a variety of ways: the horizontal and vertical slides of the ROBOT window, and all the options: ZOOM IN, ZOOM OUT, CENTER, CLIP, SLIDE, of the VIEW menu.
- `ssen_geom, lsen_geom, joy_geom` set the pixel size and position of the SHORT SENSORS, LONG SENSORS and JOYSTICK windows respectively. These can be modified using the window manager.
- `com_coor` and `rec_coor` set the pixel position of the COMMAND LINE window and the RECORDER window. These can be modified using the window manager.
- `*_color` set the colors of the different graphic entities displayed in the ROBOT, SHORT SENSORS and LONG SENSORS windows, including actual and encoder robot.

`[menu toggle]`

- `bumper, infrared_ray, ..., long_local` set the default values of the toggle switches that govern graphic displays in the SHORT SENSORS and LONG SENSORS windows. See Chapter 4 for a description.
- `robwin_bumper, ..., robwin_laser` set the default values of the toggle switches that govern graphic display of sensor data in the ROBOT window. See also the SHOW menu for this window.
- `display` is set to solid if you want the robot to be represented as a solid circle in the ROBOT window, to outline if you want it to be represented as a hollow circle.
- `trace` is set to outline or solid if you want the robot to show its path as it moves, to none else. See also the SHOW menu for this window.
- `encoder` and `actual` enable the display of the encoder and real robots (see Section 4.1.2). The default value is on. This display can be dynamically changed using the SHOW menu of the ROBOT window.

`[recorder]`

- `infraredp, .., commandp` set the default values of the item that will be recorded when starting the recorder. See also Section 4.3.5.

- `tape_len` sets the maximum length (in recordings) of a recorder "tape".

`[connect]`

- `conn_type` select between serial and TCP/IP (ethernet) communication.
- `machine` is the machine name of the robot. It is the name you use to login or telnet to the robot. The robot comes with its name set as nomad.
- `tcpip_port` is the port used by the server to communicate with the robot via ethernet.
- `serial_port` and `serial_baud` configure the serial connection (usually radio modem)
- `retrans` sets the repeat of every message that generates a time out. It should be used only with serial port connection.
- `normal_timeout` is the timeout for all commands that are supposed to return immediately (zr and ws are not: they have their own internal timeout).

Using LILO, the LInux LOader

The LILO loader allows you to select different bootable partitions on the robot's hard drive. The robot comes with three different partitions: `linux-ro` (which is the default), `linux-rw`, and `ms-dos`.

- `linux-ro`
  This partition is read-only. The files cannot be written. A clean shutdown of the system is not required, and the robot daemon robotd is started automatically.
- `linux-rw`
  Files are writable on this partition, but a clean shutdown of the system is required. The robot daemon is started automatically.
- `ms-dos`
  This partition is not used for the robot (no robot daemon or robot support), but is sometimes useful for some subsystems (like vision systems).

When the robot is powered up, the

```
LILO boot:
```

prompt appears after boot-up messages. You will have 5 seconds to start typing the name of the partition you want to boot from. TAB will display the list of available partitions. If no partition is entered, the default partition will be used. Refer to the Linux documentation for indications on how to change the boot default partition.

CHAPTER 7

VISION REFERENCE

This chapter describes the use of different vision system available for the Nomad Super Scout robot. The Sensus 460 is a color frame grabber that merely discretizes the video signal and makes the resulting memory image available to the user.

Sensus 460

The frame grabber needs to be connected to the robot for power and video signal using the connectors on the top plate. Mount the camera and point it to some meaningful scene. Then plug a VGA monitor and a keyboard in the robot. You can run the demo by executing:

```
/usr/local/robot-devices/bttv/exam-
ples/video
```

if your robot is configured with a bttv (VideoLogic) video capture card. If your robot is configured with a Matrox Meteor video capture card, execute:

```
/usr/local/robot-devices/meteor/exam-
ples/video
```

Please refer to the supplied video capture card documentation if you are unsure which card your robot is configured with. When running the video program, you will see a 5 frame-per-second update of the video data from the camera. Make sure that the iris is not closed, which could result in little or no image.

APPENDIX A

BUG REPORT FORM

Thank you for reporting bugs, malfunctions and suggested improvements regarding this release of Nomadic's software. Please email the following information to bugs@robots.com. This information will help us to figure out your problem faster.

Information About Yourself

- Your Name
- Your Organization
- Your E-mail address
- Your robot(s) serial number(s)
- The date of this report

Information About Your Environment

- Your machine type (example: Sparc10)
- Your operating system (example: SunOS)
- The program and version you are using (example: Nserver version 2.6.1)

Description Of Your Problem

Include the answer to the following questions:

- What were you doing when the problem occurred?
- Is the problem reproducible?
- Does the problem occur on the robot and / or the simulator?

Additional Information

- Copy of the `world.setup` and `robot.setup` files
- Core file, if there is one
- Smallest sample of code that reproduces the problem

You can also mail this form to:

Nomadic Technologies Software Department
2133 Leghorn St.
Mountain View, CA 94043-1605
USA

APPENDIX B

APPLICATION NOTE

Setup and Configuration of Scout Communications Using a Pair of Mercury Radio Modems

By Jake Sprouse
Nomadic Technologies, Inc.
August 1999

I. Intended Audience

This application note applies if you own a regular Nomad Scout mobile robot, along with a pair of Mercury radio modems with which you intend to do serial cable replacement. This document does not apply if you have a Nomad SuperScout robot (i.e. a robot with an onboard PC), or if you are using one Mercury radio modem along with a Proxim RangeLAN2 access point.

II. Recommended Reading

This document references the Scout Beta 1.2 "READ THIS FIRST" document (referred to below as the Scout README) which you should have received with your Scout. You also should have received a manual with your Mercury radio modems. Both of these documents will be helpful in understanding the instructions below. You will also need to understand enough about serial communications to know the difference between serial ports which are wired as DCE (data communications equipment) and DTE (data terminal equipment).

III. Connecting to the motor controller

The controller board on Nomad Scout-series robots is called the Intellisys 175 (I175) board. This board has two serial ports via which it can accept commands. The first is the CONSOLE port and is labeled as SERIAL 1 on the board itself. This port presents a text-based interface which can be brought up for debugging purposes by connecting a terminal or terminal-emulator to it at 9600 baud, 8 data bits, 1 stop bit, and no parity. The CONSOLE port is a female plug and is wired as DCE, meaning one can connect a desktop PC to it using a standard serial cable (no gender changers or null-modem adapters).This serial port is usually brought out to an equivalently-wired serial port on the top front panel of the robot.

Normally, however, commands are accepted over the HOST port which is labeled as SERIAL 2 on the I175 board. Data is transmitted through this port in a binary packetized format at 38400 baud, 8 data bits, 1 stop bit, and no parity, giving much more efficient communications.

This port is wired at DTE although it is also a female plug; this means that to connect to it using a desktop PC one would need to use a standard serial cable along with a null-modem adapter.

On the top front panel of the robot there is a serial port adjacent to the CONSOLE port which is intended to connect to the HOST port. It uses a female connector and is wired as DCE, such that you could connect to it from a PC using a serial cable just as you would with the CONSOLE port. Depending on how your robot was shipped, this external port may or may not actually be connected to the SERIAL 2 port on the I175. If your robot was shipped with no Mercury radio modem installed, then it is connected via a special cable. Otherwise, this special cable is left unconnected under the top panel and the SERIAL 2 port is connected directly to the pre installed Mercury.

The following table describes how one would connect to the HOST port of your robot from both a desktop PC (wired as DTE) and a Mercury (wired as DCE); use a standard RS232 plus the listed adapter:

Connecting from:	to:	Adapters needed:
SERIAL 2 on I175	Desktop PC	Null Modem Adapter

Connecting from:	to:	Adapters needed:
HOST port on top panel	Desktop PC	(none needed)
SERIAL 2 on I175	Mercury	Gender Changer
HOST port on top panel	Mercury	Null Modem Adapter + Gender Changer

IV. Configuring Nserver

The best way to approach the problem of configuring your communications system is to break it down into parts. First we will make sure that `Nserver` is configured correctly. We will test the configuration with the workstation connected directly to the robot via serial cable. First, you may want to lift the wheels of the robot off of the floor so that it is not able to move if `Nserver` sends a motion command; you can use a block of wood or a couple of textbooks to support it. Now, connect your workstation to the SERIAL 2 port on the I175 (or the HOST port on the front top panel if it is connected) using a standard RS232 cable plus the adapters listed in the table above. You may need to remove the top plate of your robot in order to access the I175; this procedure is described in the Scout README.

Once you have the cable connected, turn the robot on and proceed with configuring `Nserver`. `Nserver's` configuration is stored in the `robot.setup` file. At the end of a file is a section called `[connect]`; you should edit that section to look like the following:

```
[connect]
conn_type  = serial ; Taipei or serial
machine  = nomad  ; for tcpip
tcpip_port  = 65001  ; for tcpip
serial_port  = 1   ; for serial: 1 = ttyS0, 2 = ttyS1
serial_baud  = 38400 ; for serial
retrans = off
normal_timeout = 10.0 sec
special_timeout = 10.0 sec
server_delay  = 0.002 sec; the avg roundtrip delay between client and server
robot_delay = 0.01 sec ; the avg roundtrip delay between client and robot
```

The setting for serial_port may be different if you are connecting to something other than the first serial port on your workstation.

Now start `Nserver` and select "Real Robot" from the "Robot" menu. Once connected, you should be able to joystick the robot using the joystick menu.

V. Configuring the Mercury radio modems

Now, the only thing left to do is to establish a wireless link in place of the serial port. To do this, we will configure your radio modems for RMP passthough mode. RMP (Radio Modem Protocol) is a protocol designed by Nomadic specifically for data transmission over wireless links. Any data received on the serial port is packetized into Ethernet frames and sent to the opposite unit. How the data is packetized and what opposite unit to send to is completely configurable by you.

The default configuration for your Mercuries puts them into RMP passthrough mode. However, we will discuss all of the relevant settings.

First, look in the `lan0` configuration file. This is where we will set up radio communications. The important parameters here are the radio domain and the station type. These are both found in the `[hardware]` section of this file. The domain parameter must be the same on both of your radio modems. You may want to set it to something other than the default of 1 if you have other Mercury radio modems in your lab; this will serve to isolate the pair from the others. The "station type" describes whether your units should be masters, or stations which associate with them. Configure the Mercury which will be attached to your workstation as a master, and configure the one to be attached to your Scout as a station.

Now, go to the `uart0` file. This file describes what the Mercury does with data destined to or received from the serial port; we will configure it to use RMP passthrough mode for this data. In the `[hardware]` section, set the baud rate to 38400, and make sure the data bits are set to 8, the parity is set to none, and the stop bits are set to 1; this matches the configuration of the I175's HOST port.

The `[software]` section describes how to packetize data received on the serial port; i.e. how to determine when a packet of data has been received, and how that packet of data should be sent to the wireless network. To accommodate the data protocol used in communicating to the I175 board, we set the "input timeout" parameter to 40 (characters, which is the maximum packet size) and we set the "delimiters" parameter to `"0x5c"`, which is the character used to signify and end of packet. We also set the "protocol" parameter to "passthrough" which tells the unit to send these data packets to the configured network protocol verbatim (i.e. pass them through).

Now, scroll down to the `[passthrough]` section. The only parameter here is called `"socket"` and it tells us which network binding should be used to pass the data to. Set this to `"rmpbind"`.

Next, find the `[rmpbind]` section. This is where we configure how RMP will work. The default settings will be adequate here (if you think you need more configuration, please consult the "*Mercury User's Guide*"). The protocol should be set to `"rmp"` meaning that the RMP protocol should be used for this network binding. The destination address parameter should be set to "dynamic"; this means that packets are sent to all RMP units until an RMP packet is received, after which packets will be sent to that unit only (this has the effect of making the two units in your system "find" each other automatically).

Save this configuration in both of your radio modems and reset them. You should see the MASTER LED light up on the master radio modem, after which the STATION LED will light up on the other unit. Now connect the "master" radio modem to your workstation.

You can run a simple test here if you like. Using a terminal emulation package on your workstation configured for 38400 8N1, type some random characters. You should see the SERIAL RX light on the Mercury flash, indicating that the characters are being received properly (if you get an ERROR LED, check the `[hardware]` section of the `uart0` file). After you've typed about 40 characters, you should see the RADIO Tx / Rx LED light up on both Mercuries and the SERIAL TX light on the second Mercury, indicating that the data was successfully passed through (it may be difficult to see these lights as the flash very briefly).

Now, connect the "station" radio modem to SERIAL 2 on the I175 (or the HOST port on the front top panel if it is connected) using a standard RS232 cable plus the adapters listed in the table from Section III.

Again, you may need to remove the top plate of your robot in order to access the I175. Apply power to this Mercury using the supplied jack.

At this point the serial cable is replaced, and you should be able to joystick the robot from `Nserver` as described in Section IV.