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 download the most up-to-date software distributed by Nomadic.

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

To download (or upload) software from this FTP site, you can simply FTP to ftp.robots.com

Name: robots Password: N0madIC

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.

QUICK REFERENCE

Robot Commands

Communication Commands
connect_robot...................... connects to a robot
disconnect_robot................... closes connection with a robot
cnf_tm............................. sets the timeout period of the robot
real_robot........................ switches to real robot mode
simulated_robot.................... switches to simulated robot mode
quit_server......................... causes the server to quit
tk.............................. sends a character stream to the robot's voice synthesizer

Motion Commands
pr............................... moves the motors of the robot by a distance
vm............................... moves the robot at given velocities
mv............................... moves the three axes of the robot independently
st............................... stops the robot's motors
ws............................... waits for the stop of the robot's motors
lp............................... sets motor limp
zr............................... aligns steering and turret zero with bumper

Motion Parameters Setting Commands
dp............................... defines the position of the robot
da............................... defines the robot's steering angle
ac............................... sets the robot's accelerations
sp............................... sets the robot's speeds

Sensing Parameters Setting Commands
cnf_sn.......................... configures the sonar sensor system

Motion Parameters Retrieving Commands
get_rc........................... gets configuration data of the robot
get_rv........................... gets velocities of the robot

Sensory Data Retrieving Commands
get_sn........................... gets the sonar data of the robot
get_bp........................... gets the bumper data of the robot
gs............................... gets the current state of the robot

Local Map Display Commands
draw_line.......................... draws a line
draw_arc............................. draws an arc
draw_robot.......................... draws a robot
get_robot_conf............... gets interactively a point from robot's window

World Commands
add_obstacle ........... adds an obstacle to the current robot environment
delete_obstacle......... deletes an obstacle from the current robot environment
move_obstacle.............. moves an obstacle in the robot environment
new_world...................... clears all its obstacles from the map
CONVENTIONS

NAME
< Function name >

PURPOSE
<Purpose of the function>

SYNTAX
<C syntax of the function>

ARGUMENTS
<Type, meaning and range of the arguments, if any>

RETURNED VALUE
<Meaning of the returned value, if any>

UPDATEDGLOBALS
<Updated global vectors, if any>

DESCRIPTION
<Description of the function>

EXAMPLE(S)
<Refer to one of the examples in Appendix B where the function is used>

KNOWN BUGS
<Known bugs, or limitations of the function for the current release>

SEE ALSO
<Related functions>
CHAPTER 1

PROGRAM INSTRUCTIONS

NAME
ac

PURPOSE
It sets the left- and right-wheel accelerations of the robot.

SYNTAX
int ac (unsigned int r_ac, unsigned int l_ac, unsigned int unused)

ARGUMENTS
int r_ac - the right-wheel acceleration in 1/10 inch/sec;
int l_ac - the left-wheel acceleration in 1/10 degree/sec2; int unused - 0.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector

DESCRIPTION
This function sets the right- and left-wheel accelerations of the robot to \( r_{ac} \) and \( l_{ac} \) respectively. \( l_{ac} \) and \( r_{ac} \) are positive integers less than 390.
This function updates \( \text{State} \) according to the set \( \text{Smask} \).

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
sp
NAME

add_obstacle

PURPOSE

It adds an obstacle to the current robot environment.

SYNTAX

int add_obstacle (long obs[21])

ARGUMENTS

obs[0] - specifies the number (no greater than 10) of vertices of the polygonal obstacle. obs[1] to obs[20] - specify the x and y coordinates of the vertices, in counter-clockwise direction.

RETURNED VALUE

1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION

This function creates and adds an obstacle specified by obs to the current robot environment. Currently, an obstacle can have at most 10 vertices. obs[0] specifies the number of vertices of the polygonal obstacle. obs[2i+1] and obs[2i+2] specify the ith (i = 0, ..., 9) vertex of the polygon. The vertices of the polygon must be specified in counter-clockwise direction. The new obstacle appears in the world window of the graphic interface, if any, as soon as it is created.

EXAMPLE

World

KNOWN BUGS

SEE ALSO

move_obstacle, delete_obstacle
NAME
conf_sn

PURPOSE
It configures the sonar sensor system.

SYNTAX
int conf_sn (int firerate, int order[16])

ARGUMENTS
int firerate - firing rate of the sonar;
int order[16] - firing sequence of the sonar (#0 .. #15).

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function configures the sonar sensor system. The parameter firerate specifies the rate the sonars are fired (i.e. the time between two sonars are fired) at 4 milli-second time intervals. firerate should be set between 0 and 255. Note that firerate starts after the end of the processing of the previous sonar, which in turns depends on the time it takes for the sound to come back.

The parameter order specifies the firing sequence of the sonar sensors(#0, ..., #15). The sonar specified in order[i] will be fired before that specified in order[i+1]. The sonar sensors that are not specified in the order list will not be active. You can terminate the order list by a '255'. The sonar sensors are numbered counterclockwise, the front one being the first (opposite to the Emergency Stop button).

This function updates State according to the set Smask.

EXAMPLE
Sensing

KNOWN BUGS

SEE ALSO
gs, get_sn
NAME
conf_tm

PURPOSE
It sets the timeout period of the robot (in seconds).

SYNTAX
int conf_tm (int timeout)

ARGUMENTS
int timeout - timeout period in seconds. If 0, the timeout will be disabled. Maximum: 255 seconds

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function sets the timeout period of the robot (in seconds), such that if the robot has not received a command from the host for more than the timeout period, it will abort its current motion. This is a safety measure to prevent the robot from continuing its motion without control if for some reason the robot does not receive commands from the host.

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
NAME
connect_robot

PURPOSE
It requests the server to connect to the robot.

SYNTAX
int connect_robot (long robot_id)
int connect_robot (long robot_id, int model, char *dev, int conn)

ARGUMENTS
long robot_id - robot's identification number.
int model - model of robot may be one of MODEL_N200, MODEL_N150, MODEL_SCOUT, MODEL_SCOUT2.
char *dev - This character string is the serial port or the hostname of the robot. If left empty (""), localhost will be assumed. Passing NULL is not acceptable. If the string begins with "/dev" or ends with ":", a serial port will be assumed. Otherwise it will be used as a hostname.
int conn - TCP port for TCP/IP, baud rate for serial (probably 38400).

RETURNED VALUE
robot_id - if the connection is successful; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function requests a connection to the robot with robot_id in the server. In order to talk to the server with the single-parameter version of connect_robot, you must set SERVER_MACHINE_NAME (a char array of 80) and SERV_TCP_PORT (an int) properly in advance if you do not want to use the default values for them. The default value of SERVER_MACHINE_NAME is an empty string, which means that the current machine is the server; the default value of SERV_TCP_PORT is 7019. A robot id will typically be created by the Nserver. If a robot with robot_id exists, a connection is established with that robot. If no robot exists with robot_id, no connection is established. Once the connection is established, the subsequent commands are directed to that robot. Before your program sends any command to a robot, it must connect to it.

EXAMPLES
connect_robot(1, MODEL_SCOUT2, "/dev/ttyS0", 38400); will open the serial port (COM-port 1) at 38400 baud,
connect_robot(1, MODEL_SCOUT2, "/dev/ttyS1", 38400); will open COM-port 2,
connect_robot(1, MODEL_SCOUT2, "128.1.1.71", 4000); will connect to the machine at IP address 128.1.1.71, port 4000,
connect_robot(1, MODEL_SCOUT2, "myscout", 4000); will connect to the machine named "myscout", port 4000.

KNOWN BUGS

SEE ALSO
disconnect_robot
NAME
da

PURPOSE
It defines the robot’s steering and turret angles.

SYNTAX
int da (int theta, int unused)

ARGUMENTS
int theta - the orientation of the robot in 1/10ths of degrees. Angles increase in the counterclockwise direction, and zero is along the positive X axis.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function defines the robot’s steering angle to be $\theta$ and the turret angle to be $\tau$. It has no effect on the robot’s position. In the simulation mode, the encoder robot and the real robot will be given this configuration. In the real robot mode, the angles are reset without affecting the robot position, and without real motion: The robot internal counters for angles are reset to this value.

This function updates State according to the set Smask.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
dp
NAME
delete_obstacle

PURPOSE
It deletes an obstacle from the current robot environment.

SYNTAX
int delete_obstacle (long obs[21])

ARGUMENTS
obs[0] - specifies the number (no greater than 10) of vertices of the polygonal obstacle.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function deletes an obstacle specified by obs from the current robot environment. The obstacle to
delete is identified first by the number of vertices, second by the coordinates. Currently, an obstacle can
have at most 10 vertices. obs[0] specifies the number of vertices of the polygonal obstacle. obs[2i+1]
and obs[2i+2] specify the ith (i = 0, ..., 9) vertex of the polygon. The vertices of the polygon must be
specified in counter-clockwise direction.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
add_obstacle, move_obstacle
NAME
disconnect_robot

PURPOSE
It requests the server to close the connection with the robot.

SYNTAX
int disconnect_robot (long robot_id)

ARGUMENTS
long robot_id - robot's identification number.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function requests the server to close the connection with robot of robot_id.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
connect_robot
NAME
dp

PURPOSE
It defines the position of the robot.

SYNTAX
int dp (long x, long y)

ARGUMENTS
long x, y - the position coordinates.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function defines the robot's position to \((x, y)\). It has no effect on the robot's steering and turret coor-

dinates. In the simulation mode, the encoder robot and the real robot will be given this configuration. In

the real robot mode, the angles are reset without affecting the robot position, and without real motion: The

robot internal counters for angles are reset to this value.

This function updates State according to the set Smask.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
da
NAME
draw_arc

PURPOSE
It allows the client to draw an arc, a part of an ellipse, on the robot window.

SYNTAX
int draw_arc (long x0, long y0, long w, long h, int th1, int th2, int mode)

ARGUMENTS
long x0, y0 - specify the upper left corner of the rectangle bounding the ellipse in tens of inches in
world coordinates;
long w - width of the bounding rectangle in tens of inches;
long h - height of the bounding rectangle in tens of inches;
int th1, th2 - specify the angular range of the arc in tens of degree;
int mode - drawing mode

= 1: BlackPixel using GXcopy;
= 2: BlackPixel using GXxor;
> 2: color using GXxor.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function allows the client to draw an arc which is a part of an ellipse in the robot window of the cur-
rently connected robot. (x0, y0) specifies the upper left corner of the bounding box of the ellipse and (w, h)
specifies the width and height of the bounding box. (th1, th2) specifies the angular range of the arc. If
mode = 1, the drawing is done in black using GXcopy. If mode = 2, the drawing is done in black using
GXxor. If mode > 2, the drawing is done in color using GXxor.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
draw_line, draw_robot
NAME
draw_line

PURPOSE
It allows the client to draw a line.

SYNTAX
int draw_line (long x1, long y1, long x2, long y2, int mode)

ARGUMENTS
long x1,y1 - starting point of the line, tens of inches in world coordinates;
long x2,y2 - ending point of the line, tens of inches in world coordinates;
int mode - drawing mode

= 1: BlackPixel using GXcopy;
= 2: BlackPixel using GXxor;
> 2: color using GXxor.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function allows the client to draw a line from (x1, y1) to (x2, y2) in the robot window of the currently connected robot. If mode = 1, the drawing is done in black using GXcopy. If mode = 2, the drawing is done in black using GXxor. If mode > 2, the drawing is done in color using GXxor.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
draw_arc, draw_robot
NAME
draw_robot

PURPOSE
It allows the client to draw the shape of a robot.

SYNTAX
int draw_robot (long x, long y, int th, int mode)

ARGUMENTS
long x, y - x-y position of the robot;
int th, tu - steering and turret orientation of the robot;
int mode - drawing mode

= 1: BlackPixel using GXxor.
= 2: BlackPixel using GXxor -- a small arrow is drawn at the center of the robot using GXcopy.
= 3: BlackPixel using GXcopy.
> 3: color using GXxor.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function allows the client to draw a robot at configuration (x, y, th, tu), using the world coordinates. If mode = 1, the robot is drawn in black using GXxor (using this mode you can erase the trace of robot). If mode = 2, the robot is drawn in black using GXxor and in addition, a small arrow is drawn at the center of the robot using GXcopy (using this mode you can leave a trace of small arrow). If mode = 3, the robot is drawn in black using GXcopy. If mode > 3, the drawing is done in color using GXxor.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
draw_arc, draw_line
NAME
get_bp

PURPOSE
It gets the bumper data.

SYNTAX
int get_bp (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function gets the bumper data independently of Smask. However, data is valid only if the bumper is active (as specified by the previous conf_bp function call).

This function updates the State vector (state STATE_BUMPER).

EXAMPLE
Sensing

KNOWN BUGS

SEE ALSO
gs
NAME
get_rc

PURPOSE
It gets the configuration data of the robot.

SYNTAX
int get_rc (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function gets the configuration of the robot including its integrated x, y, and angle coordinates, independent of Smask.

This function updates the State vector (states STATE_CONF_X, STATE_CONF_Y, STATE_CONF_STEER, STATE_CONF_TURRET).

EXAMPLE
Sensing

KNOWN BUGS

SEE ALSO
get_rv, gs
NAME
get_robot_conf

PURPOSE
It interactively gets a point from robot’s window.

SYNTAX
int get_robot_conf(long *conf)

ARGUMENTS
long *conf - an array of 4 long integers; the configuration of the robot is returned in this array.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function interactively gets a robot’s configuration. When called, a robot shape, augmented with synchronization bars and handles appear in the MAP window:

- Clicking Left with the left button will set the new position of the robot
- 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 bar will align both turret and steering to the angle of that bar
- 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. The configuration is stored in the array given as argument as follows:

conf[0]: X position in tens of inches
conf[1]: Y position in tens of inches
conf[2]: Steering in tens of degrees
conf[3]: Turret orientation in tens of degree

EXAMPLE
World

KNOWN BUGS

SEE ALSO
NAME
get_rv

PURPOSE
It gets the translation, steering, turret velocities of the robot.

SYNTAX
int get_rv (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function gets the velocities of the robot including its translation, steering and turret rotation velocities, independently of Smask.

This function updates the State vector (state STATE_VEL_RIGHT, STATE_VEL_LEFT).

EXAMPLE
Sensing

KNOWN BUGS

SEE ALSO
get_rc, gs
NAME
get_sn

PURPOSE
It gets the sonar data.

SYNTAX
int get_sn (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function gets the sonar data, independent of Smask. However, only the active sonar sensor (as specified by the previous conf_sn function call) readings are valid.

This function updates the State vector (states STATE_SONAR_0 to STATE_SONAR_15).

EXAMPLE
Sensing

KNOWN BUGS

SEE ALSO
conf_sn, gs
NAME
    gs

PURPOSE
It gets the current state of the robot according to the mask Smask.

SYNTAX
int gs (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function gets the current state of the robot according to the mask of the communication channel. It simply updates State.

State and Smask values:

<table>
<thead>
<tr>
<th>Name</th>
<th>State Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  STATE_SIM_SPEED</td>
<td>speed of simulation</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>17 STATE SONAR_0</td>
<td>sonar data #0</td>
</tr>
<tr>
<td>18 STATE SONAR_1</td>
<td>sonar data #1</td>
</tr>
<tr>
<td>19 STATE SONAR_2</td>
<td>sonar data #2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>32 STATE SONAR 15</td>
<td>sonar data #15</td>
</tr>
<tr>
<td>33 STATE BUMPER</td>
<td>bumper data</td>
</tr>
<tr>
<td>34 STATE_CONF_X</td>
<td>x position</td>
</tr>
<tr>
<td>35 STATE_CONF_Y</td>
<td>y position</td>
</tr>
<tr>
<td>36 STATE_CONF_STEER</td>
<td>steering angle</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>38 STATE_VEL_RIGHT</td>
<td>translational velocity</td>
</tr>
<tr>
<td>39 STATE_VEL_LEFT</td>
<td>steering velocity</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>41 STATE_MOTOR_STATUS</td>
<td>motor status</td>
</tr>
<tr>
<td>44 STATE_ERROR</td>
<td>error number</td>
</tr>
</tbody>
</table>
EXAMPLE

Sensing

KNOWN BUGS

SEE ALSO
NAME
lp

PURPOSE
It sets the motor limp.

SYNTAX
int lp (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function stops all motors of the robot, that is, the robot will not hold its position; the old accelerations will be restored after the call to this function. This function will return without waiting for the stop to complete. Note that this function might not produce the desired effect if the accelerations are too small.

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
st
NAME
move_obstacle

PURPOSE
It moves an obstacle in the robot environment.

SYNTAX
int move_obstacle (long obs[21], long dx, long dy)

ARGUMENTS
obs[0] - specifies the number (no greater than 10) of vertices of the polygonal obstacle.
long dx, dy - the x and y distances to translate the obstacle.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function moves the obstacle specified by obs by the distance of dx along the x-axis and dy along the y-axis. Obs[21] will be modified to reflect the move.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
add_obstacle, delete_obstacle
NAME

mv

PURPOSE

Move the two axes of the robot independently.

SYNTAX

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

ARGUMENTS

int r_mode - the control law for the right wheel
int r_mv - the motion value for the right wheel
int l_mode - the control law for the left wheel
int l_mv - the motion value for the left wheel
int unused1 - 0
int unused2 - 0

RETURNED VALUE

TRUE - the command was executed successfully
FALSE - the command could not be executed or wrong arguments

UPDATED GLOBALS

State vector.

DESCRIPTION

mv

The motion commands vm (velocity move) and pr (position relative) allow to move all of robot’s axes by specifying either a velocity or a relative position, respectively. The motion command mv (move) allows to specify modes of motion control for each of the axes independently. The values that specify the modes are defined in Nclient.h:

MV_VM: specifies velocity mode similar to vm
MV_PR: specifies position mode similar to pr
MV_IGNORE: ignore the information for this axis
MV_LP: set this axis limp
MV_SP: set the speed for this axis
MV_AC: set the acceleration for this axis

The mode arguments t_mode, s_mode, and r_mode define how the corresponding values t_mv, s_mv, and r_mv are interpreted. In velocity mode they are treated like the arguments to vm, as velocities. If position mode is specified they will be interpreted as positions relative to the current configuration. Specifying mode MV_IGNORE for an axis will result in that axis to remain in its current state. Refer to the documentation of pr and vm for detailed information on the value arguments for the corresponding mode. Example: mv (MV_VM, 200, MV_PR, 100, MV_IGNORE, MV_IGNORE) will cause the robot to translate in velocity mode at a velocity of 20 inch per second, to steer 10 degrees, and to continue the previously specified turret motion (if a turret command was issued prior to the mv).
EXAMPLE

Motion

KNOWN BUGS

SEE ALSO

pr, vm
NAME
new_world

PURPOSE
It deletes all obstacles in the current robot world.

SYNTAX
int new_world(void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function deletes all obstacles in the current robot world.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
add_obstacle, delete_obstacle
NAME
place_robot

PURPOSE
It places the robot at a certain position.

SYNTAX
int place_robot (long x, long y, int th, int unused)

ARGUMENTS
long x, y - the x-y position of the desired robot configuration;
int th - the steering orientation of the desired robot configuration
int unused - 0.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function places the robot at position (x y), angle at th. In simulation mode, it will place both the Encoder-robot and the Actual-robot at this configuration. In real robot mode, it will reset the robot’s counters to the new values.

EXAMPLE
World

KNOWN BUGS

SEE ALSO
dp, da
NAME
pr

PURPOSE
It moves the motors of the robot by a distance, using the speed set by sp().

SYNTAX
int pr (int rpr, int lpr, int unused)

ARGUMENTS
int rpr - right wheel step in 1/10 inches, within [-32000, 32000];
int lpr - left wheel step in 1/10 inches, within [-32000, 32000];
int unused - 0.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function moves the robot's right and left wheel motors by \((rpr/10)\) inches and \((lpr/10)\) inches respectively, at the speeds specified by the previous function call to \(sp(rsp, lsp, 0)\) and at accelerations by the previous call to \(ac(...)\). The first two parameters specify the relative distances for the two motors: right wheel and left wheel. Both of the motors move concurrently if their speeds are not set to zero and the distances to be travelled are not zero.

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
vm, mv
NAME
quit_server

PURPOSE
It causes the server to quit.

SYNTAX

int quit_server (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - failure.

UPDATED GLOBALS

DESCRIPTION
This function causes the server to quit assuming this feature is enabled in the setup file of the server.

EXAMPLE
Motion

KNOWN BUGS
This function works only with ONE client program; it has the side effect of killing other clients connected to the server and cannot get the returned value 1.

SEE ALSO
NAME
real_robot

PURPOSE
It switches the server to the real robot mode.

SYNTAX
int real_robot (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATEDGLOBALS

DESCRIPTION
This function switches the server to the real_robot mode. All the commands will be directed to the real robot.

CAUTION: Make sure that the robot is in a safe position before switching to this mode.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
NAME
server_is_running

PURPOSE
It queries the server to see if it is up and running.

SYNTAX
int server_is_running (void)

ARGUMENTS
none

RETURNED VALUE
1 - the server is running; 0 - the server is not running.

UPDATED GLOBALS

DESCRIPTION
This function queries the server to see if it is up and running.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
quit_server
NAME
simulated_robot

PURPOSE
It switches the server to the simulated_robot mode.

SYNTAX
int simulated_robot (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS

DESCRIPTION
This function switches the server to the simulated_robot mode.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
real_robot
NAME
sp

PURPOSE
It sets the right and left wheel translation speeds of the robot.

SYNTAX
int sp (unsigned int rsp, unsigned int lsp, unsigned int unused)

ARGUMENTS
int rsp - the right wheel speed in 1/10 inch/sec, within [0, 400].
int lsp - the left wheel speed in 1/10 inch/sec, within [0, 400].
in unused - 0.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function sets the right and left wheel speeds of the robot to rsp and lsp respectively. The speeds are initially set to 200 and 200 for rsp and lsp respectively. Note: The defaults can be found in the file /etc/robot.cfg on the simulated robot.

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
ac
NAME
st

PURPOSE
It stops the motion of the robot.

SYNTAX
int st (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function brings the robot to a controlled stop with appropriate accelerations and holds its current position. If acceleration = 0, the robot will NOT stop.

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
lp, ws
NAME
tk

PURPOSE
It sends a character stream to the robot’s voice synthesizer.

SYNTAX
int tk (char *talk stream)

ARGUMENTS
char *talk stream - the character stream to be sent to the synthesizer.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function sends a talk stream in characters to the robot’s voice synthesizer to let the robot talk. This
function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS
It does not accept non-printable chars.

SEE ALSO
NAME
vm

PURPOSE
It moves the robot according to the velocities specified by its parameters.

SYNTAX
int vm (int rv, int lv, int unused)

ARGUMENTS
int tv - the desired right wheel velocity in 1/10 inch/sec, within [-400,400];
int sv - the desired left wheel velocity in 1/10 inch/sec, within [-400,400];
int unused - 0.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
This function moves the robot at right wheel velocity \( rv \) and left wheel velocity \( lv \). \( rv \) and \( lv \) are both integers between -400 and 400 (in units of 0.1 inches/sec). The robot will continue to move at these velocities until either it receives another command or it receives no command after timeout (in which case it will stop its motion).

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
pr, mv
NAME
ws

PURPOSE
It waits for the stop of the motors of the robot.

SYNTAX
int ws (unsigned char w_r, unsigned char w_l, unsigned char unused, unsigned char timeout)

ARGUMENTS
unsigned char w_r, w_l - These two parameters specify which axis or combination of axes (right wheel, left wheel) to wait: 1 for wait and 0 for not;
unsigned char unused - 0.
unsigned char timeout - specifies how long (in seconds) to wait before timing out.

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
The function waits for the stop of the motors of the robot. Which motor(s) to wait depends on which of the parameters w_r and w_l are set. This function is intended to be used in conjunction with pr() or st to detect the desired motion has finished.

Note: contrary to the standard behavior, this command will only return after the motors of the robot have actually stopped (usually commands return immediately regardless of whether the robot has completed the desired action or not).

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
st
NAME
zr

PURPOSE
It zeroes the robot - resets internal position and angle to zero.

SYNTAX
int zr (void)

ARGUMENTS
none

RETURNED VALUE
1 - success; 0 - otherwise.

UPDATED GLOBALS
State vector.

DESCRIPTION
Resets the internal coordinate system of the robot such that the current position is (0, 0) and angle is 0 degrees.

This function updates State according to the set Smask.

EXAMPLE
Motion

KNOWN BUGS

SEE ALSO
zr
CHAPTER 2

PROGRAMMING EXAMPLES

2.1 Motion example

/*
 * This program will connect to the robot, configure locomotion,
 * and move the robot using various commands.
 * It assumes that a server is running and connects to it
 * To compile: gcc -o motiontest motion.c Nclient-linux.o -DSIMULATION=1 OR
 * gcc -o motiontest motion.c Nclient-sparc.o -DSIMULATION=1
 */
#include <stdio.h>
#include <unistd.h>
#include "Nclient.h"
#define ROBOT_ID 1 /* Currently, only robot #1 allowed */
#ifndef SIMULATION
#define SIMULATION 1
#endif
#define DIAMETER 136 /* wheel-to-wheel diameter in 0.1in */

int main()
{
    /* Connection */

    SERV_TCP_PORT = 7019; /* Matches the number given in world.setup */
    strcpy(SERVER_MACHINE_NAME, "masai"); /* The machine the server is running on */

    if (!connect_robot(ROBOT_ID))
    {
        printf("Connection to robot failed\n");
        return(1);
    }

    if (SIMULATION)
        simulated_robot(); /* Commands will be sent to simulator */
    else
        real_robot(); /* Commands will be sent to real robot (CAUTION!!) */

    conf_tm(2); /* Robot will stop if no command from the server in 2 seconds */

    printf(" ****************************************");
    printf(" * NOMADIC HOST SOFTWARE ENVIRONMENT - MOTION DEMONSTRATION *");
    printf(" ****************************************


");
/* Initialize the robot */
    printf("Zeroing...
");
zr(); /* Zero the robot */

/* Initialize motion parameters */

ac(400,400,0); /* right and left translation accelerations in .1in/s2 */
sp(100,100,0); /* right and left translation speeds in .1in/s */
printf("Hit any key to translate blindly by 1000 tens of inch... will wait
for the motion to stop\n");

getchar();
pr(1000,1000,0); /* right, left, unused */
ws(1,1,0,20); /* Wait for the motion to stop */

printf("Hit any key to steer blindly by 1800 tens of degree... will wait for
the motion to stop\n");

getchar();
pr(DIAMETER * 3.14 / 2, -DIAMETER * 3.14 / 2, 0); /* rotate 180 degrees */
ws(1,1,0,50); /* Wait for the motion to stop */
printf("Hit any key to move in straight line until hitting something...\n");
if (SIMULATION)
    printf("Make sure that there is an obstacle in front of the robot, or cre-
ate it NOW !\n");
getchar();

gs(); /* Get state according to Smask */
while(!State[33]) /* Check bumpers */
    vm(100,100,0); /* If ok, velocity move (vm updates State vector) */
st(); /* Stop the robot, hold the position */
sleep(2); /* Wait (ws would do as well) */
printf("Hit any key to backup a little bit...\n");
getchar();

pr(-500,-500,0); /* position relative */
sleep(2);
get_rv(); /* Get current velocities, independently of the mask */
if (State[38])
    tk("ALERT! The robot did not stop in time !"); /* This robot talks !! */
lp(); /* Stop the robot without holding the position */

printf("End of demo, quitting the server...\n");
quit_server(); /* Kills the server; just disconnect_robot if the server is to
be used again */
return(0);
2.2 SENSING EXAMPLE

/*
 * This program will connect to the robot, configure sensing,
 * get sensor data, print and draw it.
 * It assumes that a server is running and connects to it
 *
 * To compile: gcc -o sensingtest sensing.c Nclient-linux.o -DSIMULATION=1 -DALL_SENSORS=0 -lm   OR
 * gcc -o sensingtest sensing.c Nclient-sparc.o -DSIMULATION=1 -DALL_SENSORS=0 -lm
 *
 */

#include <stdio.h>
#include <math.h>
#include "Nclient.h"

#define ROBOT_ID 1 /* Currently, only robot #1 allowed */
#define PI 3.1415
#ifndef SIMULATION
#define SIMULATION 1
#endif

#ifndef ALL_SENSORS
#define ALL_SENSORS 0
#endif

int main()
{
  int i;

  /* Sensing configuration */

  int sn_order[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}; /* Use all
  sonars, in order */

  /* Connection */
  SERV_TCP_PORT = 7019; /* Matches the number given in world.setup */
  strcpy(SERVER_MACHINE_NAME, "masai"); /* The machine the server is running on */

  if (!connect_robot(ROBOT_ID))
  {
    printf("Connection to robot failed\n");
return(1);
}

if (SIMULATION)
    simulated_robot(); /* Commands will be sent to simulator */
else
    real_robot(); /* Commands will be sent to real robot (CAUTION!!) */

/* Initialize sensing parameters */

conf_sn(2, sn_order); /* Sonar: firing interval, order */

/* Get sensor data */
printf(" *************************************************************
");
printf(" * NOMADIC HOST SOFTWARE ENVIRONMENT - SENSING DEMONSTRATION *
");
printf(" *************************************************************


");

if (SIMULATION)
{
    printf("Make sure that you have obstacles around, or get some NOW\n");
    printf("Then hide map in robot window, and move the robot to some interesting place\n");
}

printf("Hit any key to get sensor data...\n");
getchar();

if (ALL_SENSORS)
    gs(); /* Get sensor data according to Smask */
else
{
    get_sn(); /* To get sonar data independently of Smask */
    get_bp(); /* To get bumper data independently of Smask */
    get_rc(); /* To get configuration data independently of Smask */
    get_rv(); /* To get velocity data independently of Smask */
}

/* Print configuration sonar and bumper data on screen */

printf("Sonar data:");
for (i=0; i<16; i++)
    printf("%ld ", State[STATE_SONAR_0+i]);
printf("\n");

printf("Bumper data:");

for ( i = 0 ; i < 6 ; i ++ )
    if (State[STATE_BUMPER] & (1L << i))
        printf("1");
    else
        printf("0");

printf("\n");
printf("X Pos: %d Y Pos: %ld Steer Pos: %ld\n", State[STATE_CONF_X],
    State[STATE_CONF_Y], State[STATE_CONF_STEER]);
printf("Right Speed: %ld Left Speed: %ld\n", State[STATE_VEL_RIGHT],
    State[STATE_VEL_LEFT]);

printf("End of demonstration, disconnecting...\n");
disconnect_robot(ROBOT_ID);
return(0);
2.3 WORLD EXAMPLE

/*
 * This program will connect to a robot, illustrate obstacle manipulation
 * functions, and interactively get a new position for the robot
 * It assumes that a server is running and connects to it
 *
 * To compile: gcc -o worldtest world.c Nclient-linux.o OR
 * gcc -o worldtest world.c Nclient-sparc.o
 *
 */
#include <stdio.h>
#include "Nclient.h"
#define ROBOT_ID 1 /* Currently, only robot #1 allowed */
#define RANGE 500

int main()
{
    /* Obstacle definition: number of vertices, coordinates */
    long obs[21]={3,400,-100,700,-100,500,500};
    long conf[4];

    /* Connection */
    SERV_TCP_PORT = 7019; /* Matches the number given in world.setup */
    strcpy(SERVER_MACHINE_NAME, "masai"); /* The machine the server is running on */

    if (!connect_robot(ROBOT_ID))
    {
        printf("Connexion to robot failed\n");
        return(1);
    }

    printf("************************************************************************
    * NOMADIC HOST SOFTWARE ENVIRONMENT - WORLD MANIPULATION DEMONSTRATION *
    ************************************************************************
    ");
    new_world(); /* Clear the map */
printf("Hit any key to add a newly created obstacle\n");
getchar();
add_obstacle(obs); /* Add this obstacle */

printf("Hit any key to translate this obstacle\n");
getchar();
moved_obstacle(obs, 200, 200); /* Move it !! */

printf("Hit any key to delete this obstacle\n");
getchar();
delete_obstacle(obs);

printf("Position the robot in Robot window with the mouse: \n");
printf("Click LEFT to drop the robot in place\n");
printf("Drag handles to rotate/steer the robot\n");

printf("Click on sync bars to re-align steering and turret\n");
printf("Click on the gray square to exit\n");
get_robot_conf(conf);
printf("Setting position to %ld, %ld, steer to %ld\n", conf[0], conf[1],
conf[2]);

draw_robot(conf[0], conf[1], conf[2], 0, 2); /* Draw the future position of
the robot on the robot window */
draw_arc(conf[0]-RANGE, conf[1]+RANGE, 2*RANGE, 2*RANGE, 0, 3600, 2); /* Draw
safety range */

printf("Hit any key to reset robot encoders\n");
getchar();

/* Reset robot encoders */

dp(conf[0], conf[1]); /* x, y */
da(conf[2], conf[3]); /* steering, turret */

printf("End of demo, disconnecting...\n");
disconnect_robot(ROBOT_ID);
return(0);