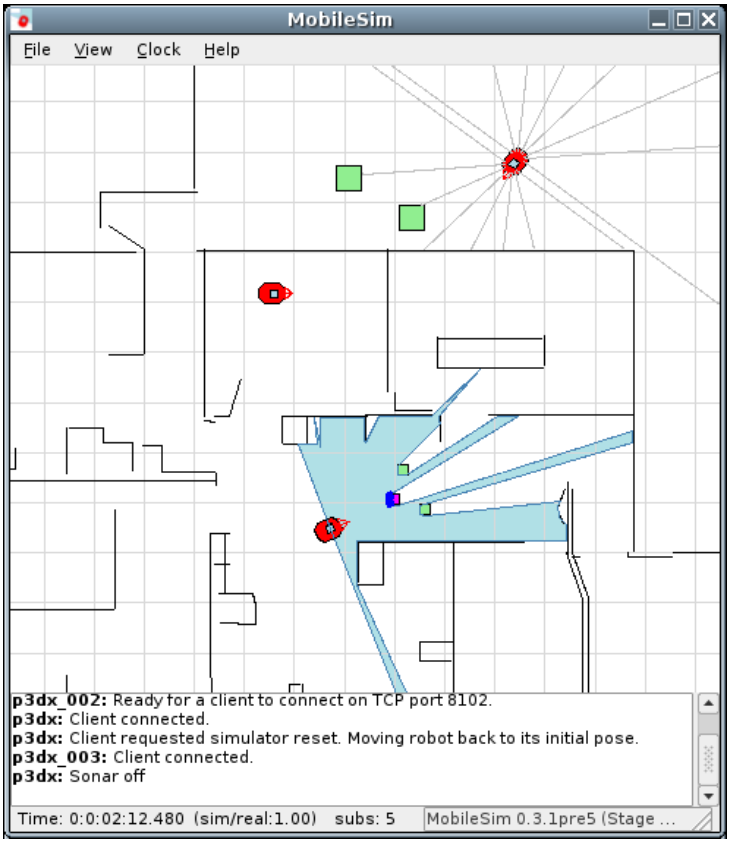
MobileSim

The Adept MobileRobots Simulator



Introduction

MobileSim is software for simulating mobile robots and their environments, for debugging and experimentation with ARIA or other software that supports Adept MobileRobots platforms.

MobileSim is based on the Stage library, created by the Player/Stage/Gazebo project. Stage and MobileSim are free software, distributed under the terms of the GNU General Public License: see the file LICENSE.txt for details.

MobileSim converts a MobileRobots/ActivMedia map (a .map file, created by Mapper3™, Mapper3-Basic, or other means) to a Stage environment, and places a simulated robot model in that environment. It then provides a simulated Pioneer control connection accessible via a TCP port (similar to the real Pioneer's serial port connection). ARIA is able to connect to TCP ports instead of serial ports (ArSimpleConnector, for example, automatically tries TCP port 8101 before the serial port).

MobileSim (powered by Stage) has the following features and device models. Device model parameters may be customized in a configuration file, or new models may be defined based on existing models.

- Various models of MobileRobots/ActivMedia differential drive mobile robot bases (Pioneer 3 DX and AT, PowerBot™, AmigoBot™, PeopleBot™, PatrolBot®, Seekur®, Pioneer 2, Pioneer 1).
- You can define custom models of differential drive or omnidirectional robots, laser and sonar, defined either as a Stage model definition, or by loading an Aria parameter (.p) file.
- Sonar range sensor model
- Laser range sensor (SICK LMS-200) model
- Robot odometry pose estimation (with cumulative error)
- Multiple robots (good performance is possible on a fast computer with more than 25 robots)
- Evironment and obstacles are loaded from a MobileRobots/ActivMedia/ARIA map file; use the same map with ARNL/SONARNL or your own software.
- Interactively move robots and obstacles (with customized map file, see below)
- Objects (robots or obstactles) may be configured to be detectable only by some sensors
- Objects (robots or obstacles) may be set as "highly reflective" for special detection by the laser range sensor

- Save image "snapshot" of simulation or save a sequence of images for making movies.
- Map files may be reloaded at run time
- Robots may be interactively repositioned through the GUI or by a command from client software
- Client software may send log messages to MobileSim. Log messages are timestamped and displayed both the GUI and console or log file, with optional HTML formatting.
- Visualization options for sensors, robots and obstacles, such as drawing robot "trails" to show history of motion, color, diagnostic values/visualization, and visualizations for debugging the simulator itself.
- Configurable space and time resolutions.
- Full source code included under the GPL for understanding the simulation implementation, customizing and improving it, or even adapting MobileSim's ARIA communication protocol layer for other simulators.

Installation

To install MobileSim on Windows, run the executable self-installer.

To install MobileSim from either the RPM or Debian (deb) package, use your system's appropriate installation tool (`rpm` on RedHat, SuSe, etc. and `dpkg` or Synaptic ("Add downloaded packages") on Debian, Ubuntu, etc.). The RPM was built on RedHat 7.3, and the Debian package was built on Debian 3.1 (sarge) for i386 (32-bit). MobileSim requires the GTK 2.0+ libraries on RedHat, and GTK 2.6+ on Debian. It also requires libstdc++ 2.2 for libc6. (These libraries are standard parts of most Linux installations, but if they are missing you may have to install packages which provide them). If you are running amd64 or ia64 Debian, try installing the `ia32-libs-gtk` and `ia32-libs` packages using `apt-get`, and then installing the MobileSim package with the `--force-architecture` option to `dpkg`.

To build MobileSim from source code, obtain the "MobileSim-src" tar/gz archive from http://robots.mobilerobots.com/MobileSim and follow the instructions in the README.src.text file found inside.

How to get started

To simulate a robot you need a map file containing lines for walls. A map file with lines can be created using a recent version of Mapper3 or Mapper3-Basic (See http://robots.mobilerobots.com), or by other means. See the ARIA documentation for the `ArMap` class for a full description of the format, or refer to `columbia.map` or `AMROffice.map`, distributed with MobileSim, for an example.

MobileSim may be launched from the Start menu (Windows), or from the Gnome or KDE menu (Linux), or from the command line (any platform).

Run MobileSim from a command line prompt like this:

```
MobileSim
```

or, to skip the initial dialog by specifying a map file and robot model to create on the command line:

```
MobileSim -m <map file> -r <robot model>
```

For example:

```
MobileSim -m columbia.map -r p3dx
```

The parameters are optional. The default robot model created, if -r is not given, is "p3dx". Some additional robot models include "p3at", "powerbot", "peoplebot", "patrolbot-sh", "seekur", and "amigo".

To create multiple simulated robots, use -r multiple times, naming the robots after the model type, seperated by a colon, if you wish:

```
MobileSim -m columbia.map -r p3dx:robot1 -r p3dx:robot2 -r amigo:robot3
```

If you are working with many robots, it can be inconvenient to create them all at MobileSim startup. If you use -R instead of -r, then a new robot will be created on demand for each client that connects, and destroyed when the client exits. (So the robot's state is not preserved.)

More command-line options are described below.

If you run MobileSim with no command line options, or from the desktop menu, you may select the map and one robot model from a dialog box. Or, you may choose to use no map (but note that in this case, the usable universe is limited to 200 x 200 meters -- gray area indicates the edge.)

Example maps are included with MobileSim. On Linux, you can find these files by navigating from the root file system to `/usr/local/MobileSim`. On Windows, these maps are in `C:\Program Files\MobileRobots\MobileSim`. `columbia.map` included with MobileSim is equivalent to `columbia.map` included with ARNL.

A window will appear, displaying your map environment and robot(s). The robot will start at a home position in your map, if present, or at the center of the map. The robot's body is drawn based on the aproximate length and width -- including wheels -- of the model selected. You can pan in the window by holding down the right mouse button and dragging. You can zoom with the mouse scroll wheel, or by holding down the middle mouse button and dragging towards or away from the center of the circle that appears.

The robot may be moved by dragging it with the left mouse button and rotated by dragging it with the right mouse button. Note that this will update the robot's true pose, but *not* its odometry -- it is like picking up the robot off the ground and carrying it to a new location.

You can turn on device visualizations using the View menu. Ranger or Sonar: if a client program has connected and enabled the sonar, sonar field centers will be drawn as grey lines. Laser: If a client program has enabled the laser rangefinder, its range will be drawn as a light blue field. Position: Axes are drawn for the robot's odometry coordinate system, and its true pose, odometric pose, and current velocity are displayed.

A snapshot or sequence of snapshots of the window can be saved as a bitmap file to disk with the Export item in the File menu.

The lower log message pane displays information and warning about what the robots are doing (some commands will result in a log message, but not all). SIM_MESSAGE commands also display messages here. The statusbar pane below the log message pane displays, from left to right: the total time since the simulator started; the ratio of real time per update step and simulation time, indicating how close to real-time (1.0) the simulation is running; the total number of "subscriptions" in the simulation (one per each robot plus one per each device being accessed); and the MobileSim and stage version numbers.

Once running, a program can connect to the simulator via TCP port 8101, and use the same protocol on that TCP port as it does with a real robot over its serial port link. Any ARIA-based program that uses ArSimpleConnector or ArTcpConnection will do this automatically. You can specify an alternate port number with the -p option. If multiple robots were requested with multiple -r options, each additional robot will use the next port number (i.e. 8102, 8103, etc.). For these additional robots, you will have to specify the port number when running its ARIA program, usually using the -rrtp (-remoteRobotTcpPort) command-line option.

A tool called `convertSfWorldToMap` is available in ARIA's collection of utility and example programs to convert a Saphira .wld file to a .map, as well as a tool called `addSfWorldLinesToMap` for importing lines into a .map from a .wld. The MobileSim source code also contains a simple command line tool that attempts to convert a PNM bitmap to a .map file.

Run MobileSim on the command line with the --help option for a full list of options.

Notes and Future Plans

The goal of Stage and MobileSim are to provide an efficient and sufficient simulation of multiple robots. While we have modified Stage to add some features and additional behavior (such as basic sensor noise), the simulation remains simplified. Don't forget to test your code on a real robot.

"All models are wrong. Some are useful." --George E. P. Box

However, since the full source code for MobileSim and Stage are available (in the "MobileSim-src" package available from the web), it is always possible to add new models or modify existing models to simulate any particular properties you need. Stage is not hard to understand and modify. If you need any help, ask on the aria-users mailing list, or the Player/Stage mailing list. If you do make any modifications that you think others would find useful, you may post them to aria-users, or send them to support@mobilerobots.com.

A note about data communications channels

On a real robot, ARIA may be communicating with different devices over separate communications channels. For example, ARIA communicates with the robot microcontroller over one RS-232 serial port, with a SICK LMS-200 laser rangefinder over a different RS-232 serial port, simultaneously using two different threads (one contained in ArRobot, the other in ArLMS200 which is a subclass of ArThreadedRangeDevice). Other additional devices may use additional parallel communications channels such as serial, USB and ethernet connections.

However, ARIA communicates with MobileSim over only one TCP socket connection. Therefore all communications with both the simulated robot and simulated devices must occur on this one channel, the same as the robot connection (ARIA's Connector classes detect MobileSim and use the appropriate code to handle data transfer with devices via the robot connection). Be aware of this when structuring programs; for example, laser data will not be received unless the ArRobot connection is active and running (using either `ArRobot::runAsync()` or `ArRobot::run()`).

Additional notes:

- See below for details on supported commands and specific known problems with the ARCOS/AROS/P2OS command emulation.
- Sonar reflection is not modeled
- Several devices are not available yet, including grippers, arms, general pan-tilt units, IR sensors, bumpers, analog or digital I/O ports, GPS units, or video cameras.
- MobileSim is graphics intensive and will compete with e.g. MobileEyes for graphics resources. Graphics display is not fully optimized yet.
- While it is possible to make the simulation run faster (or much slower) than near "real time", it is not well tested how this affects clients. Many clients (including those using ARIA) have timeout limits based on the real system time (it is possible to configure most of ARIA's timeout limits, though. See reference documentation or ask the aria-users group). Clients also may not be able to process packets fast enough if the simulation runs much faster than real time, resulting in buffering of data and possibly loss of data.

Future plans for include better sonar model, simulating additional devices, and various user interface features for creating new simulated robots, loading maps, and tweaking their properties. If you have any requests or would like to help add features, please join the discussion on the aria-users mailing list.

Advanced Usage

Command-line Arguments

A full listing of MobileSim's command line parameters follows:

Usage:

```
MobileSim [-m map] [-r robot model ...] [...options...]
```

`--map map`	Load map file (e.g. created with Mapper3)
`-m map`	Same as `-map map`.
`--nomap`	Create an "empty" map to start with. Same as -map "". Ignored if -m is also given.
`--robot model[:name]`	Create a simulated robot of the given model. If an ARIA robot parameter file is given for *model*, then MobileSim attempts to load that file from the current working directory, and create an equivalent model definition. May be repeated with different names and models to create multiple simulated robots. Name portion is optional. For example: `--robot p3dx --robot amigo:MyRobot -r custom.p:CustomBot` See PioneerRobotModels.world.inc for model definitions.
`-r model[:name]`	Same as `--robot model[:name]`.
`--robot-factory model`	Instead of creating one robot of the given model, accept any number of client programs on its port, creating a new instance of the model for each client, and destroying it when the client disconnects.
`-R model`	Same as `--robot-factory model`
`-p port`	Emulate Pioneer connections starting with TCP port *port* (Default: 8101)
`-W worldfile`	Use Stage worldfile *worldfile* instead of map, and instead of standard world configuration files.
`--stageworld`	Same as -W.
`--fullscreen-gui`	Display window in fullscreen mode.
`--maximize-gui`	Display window maximized.
`--minimize-gui`	Display window minimized (iconified)
`--noninteractive`	Don't display any interactive dialog boxes that might block program execution. Also restart MobileSim if it crashes (Linux only).
`--lite-graphics`	Disable some graphics for slightly better performance
`--no-graphics`	Disable all graphics drawing for slightly better performance
`--html`	Print log messages and other output in HTML rather than plain text
`--cwd dir`	Change directory to *dir* at startup. Client programs can then load maps relative to this directory.
`--log-file file`	Print log messages to *file* instead of standard error console.
`-l file`	Same as --log-file *file*.
`--log-file-max-size size`	If the amoount of data written to the log file exceeds *size* bytes, then rotate the log files (up to 5) and open a new log file. This option keeps the total size of all the log files under *size**5 bytes. If --noninteractive is given, the default for *size* is 5 MB. Otherwise, the default is not to limit the size of the log file at all.
`--update-interval ms`	Time between each simulation update step. (Default is 100 ms. less may improve simulation performance but impact client responsiveness and data update.)
`--update-sim-time ms`	How much simulated time each simulation update takes (Default is equal to update-intenval. More than update-interval results in faster-than-realtime simulation, which may cause problems for clients)
`--start x,y,th` OR `--start outside` OR `--start random`	Use *x*, *y*, *th* as robot starting point (even if the map has Home objects). Or, use the keyword "outside" to cause robots to start 2m outside the map bounds — it later must be moved within the map bounds to be used — or, use the keyword "random" to randomly choose a starting place within the map bounds.
`--resolution r`	Use resolution *r* (milimeters) for collisions and sensors. Default is 20mm (2cm)
`--ignore-command num`	Ignore the command whose number is given. Refer to robot manual and MobileSim documentation for command numbers. For example, to prevent client programs from moving the robot's true position with the SIM_SET_POSE command, number 224, use `--ignore-command 224`. May be repeated. Warning, MobileSim and/or your program may not function correctly if some critical commands are ignored.
`--less-verbose`	Be a bit less verbose about logging messages, especially things that might be frequently logged (such as unsupported commands), to avoid filling up log files with useless info.
`--log-timing-stats`	Log some simulation timing information every 30 seconds.

`--bind-to-address` *address*	Only listen on the network interface with IP address *address* for new client connections (default is to listen on all addresses). This lets you run more than one MobileSim process on the same TCP ports on a machine that has more than one network interface.
`--disable-crash-handler`	Disable the GDB crash handler (and just abort program on fatal signals). No effect on Windows, where there is no crash handler.

Windows' command shell unfortunately does not display the standard console output of GUI programs like MobileSim. To see MobileSim's output, you can either use the --log-file option, or run MobileSim from another command shell, such as the MSYS shell (download MSYS here).

Model Definitions

The standard Pioneer robot model definitions are in the file `PioneerRobotModels.world.inc`, which is installed with MobileSim (by default, in `/usr/local/MobileSim` on Linux and `C:\Program Files\MobileRobots\MobileSim` on Windows). This file is used to create the desired robot model when loading a MobileRobots/ActivMedia .map file, and you may include it in your custom Stage worlds as well. It must be present to run MobileSim. To define modified or custom robot model definitions, either modify this file, or create one of these directories:

- `~/.MobileSim/include/` if on Linux (where ~ is your home directory), or
- `C:\Documents and Settings\<Your User Name>\MobileSim\include` on Windows (Where <*Your User Name*> is your user account name.

Any files MobileSim finds in this include directory (except files that begin with a dot ".") are included into the world configuration after PioneerRobotModels.world.inc. (You can also put Stage world and GUI settings, and even model instantiations into these files as well. See the Stage documentation for complete documentation of the syntax of these files.) New model definitions from these files can be used with the -r option on the command line using the model definition name, or by typing the name into the Robot Model field of MobileSim's initial dialog box.

When you define a new model, you can change the `pioneer_robot_subtype` property; this is the robot subtype string that ARIA uses to load client-side parameters for the robot. If you do change this value, you must also create a parameter file (.p file) for that new type in ARIA's 'params' directory.

MobileSim expects common resources like the `PioneerRobotModels.world.inc` file to be in the standard installation location (`/usr/local/MobileSim` on Linux; `C:\Program Files\MobileRobots\MobileSim` on Windows) unless an environment variable named `MOBILESIM` is set, or on Windows, in the `\SOFTWARE\MobileRobots\MobileSim\Install Directory` registry key. Change these if you want MobileSim to use resources in a different directory than the standard installation location.

Robot speed limits

Model definitions for robots ("position" models) have speed limits defined. This is analogous to the limits set in a real robot's firmware.

Stage Worldfile Override

Instead of using a map file and the model definitions in `PioneerRobotModels.world.inc` and the files in `~/.MobileSim/include`, you may also simply provide a Stage-format world file with the -W or -stageworld command line options. If provided, MobileSim uses only this world file for world configuration, it does not use the common `PioneerRobotModels.world.inc`, *or* the files in your "include" directory.

Special Simulator Map Objects

MobileSim includes some "box" model definitions in its model definitions file `PioneerRobotModels.world.inc`. These are 0.5 x 0.5 meter objects that you can place anywhere in the map, and move with the mouse during the simulation. These boxes are created in the same way as robots, using the `-r` command line argument. Use the `--start` option to choose their locations. You can create your own model definitions based on the `box` model type as well (e.g. with different dimensions, color, etc.) MobileSim's predefined `box` model types are described below:

`box`
: A 0.5 x 0.5 m yellow box that is an obstacle which can also be sensed by laser and sonar.

`box-nolaser`
: A 0.5 x 0.5 m yellow box that is an obstacle which can be sensed by sonar, but *not* lasers.

`box-nolaser-nosonar`
: A 0.5 x 0.5 m yellow box that is an obstacle which can *not* be sensed by sonar or lasers (the robot will collide with it and trigger bumpers and stall).

`box-reflector`
: A 0.5 x 0.5 m yellow box that is an obstacle which can be sensed by sonar, and by laser with a high reflectance value (2).

For example, you can create a simulation with two movable boxes and a Pioneer 3-DX robot with the following command:

```
MobileSim -r p3dx -r box -r box
```

You can also define custom map object types in a map file, which will allow you to add them to that map file using Mapper3. MobileSim recognizes some additional attributes of the custom types in order to create extra objects in the simulation.

`Sim.Obstacle=`*yes*|*no*
: If this attribute is given to a `SectorType`, then MobileSim creates an extra movable box. (Not supported yet for `BoundaryType`) If the value is `yes`, then the object will be an obstacle that the robot will collide with. If `no`, then the robot will pass through it. This

attribute controls whether a box is created or not, it must be given with either a yes or no value.

`Sim.LaserReflect=`*int*
: If the *int* value is greater than 1, then it will be returned to the client with laser readings, if the laser sensor model is able to detect it according to its rules for detecting reflectors. (See `Reflector` below).

If 1, it will be detected as a normal object by the laser. If 0, it will not be detected by the laser at all.

`Sim.SonarReflect=`*yes*|*no*
: If `yes`, then the object will be detected by sonar. If `no`, then it will not be detected by sonar.

MobileSim also recognizes the following predefined Cairn objects:

`Reflector`
: This creates a line in the map that has a higher "reflectance" value than the normal LINES data. (Note, ARIA version 2.5 or later is needed to be able to read the reflectance value data from the simulater LRF). Both sides of the line will have high reflectance.

To add these custom object type definitions to a map file, first manually edit the map to define a custom type that has a simulator specific attribute. For example, the following defines five object types:

1. a normal box
2. a highly reflective box
3. an box that is invisible to laser and sonar but which the robot will collide with
4. a box that is invisible to laser and sonar and which the robot will not collide with (it is simply visible in the simulator, this is useful as a simple marker of a place in the map)
5. a reflector line

```
MapInfo: SectorType Name=BoxObstacle Label=Sim_BoxObstacle Sim.Obstacle=yes
MapInfo: SectorType Name=ReflectiveBoxObstacle Label=Sim_ReflectiveBoxObstacle Sim.Obstacle=yes Sim.LaserReflect=32
MapInfo: SectorType Name=InvisibleBox Label=Sim_InvisibleBox Shape=Plain Sim.Obstacle=yes Sim.LaserReflect=0 Sim.SonarReflect
MapInfo: SectorType Name=MarkerBox Label=Sim_MarkerBox Shape=Plain Sim.Obstacle=no Sim.LaserReflect=0 Sim.SonarReflect=no
MapInfo: BoundaryType Name=Reflector Label=Reflector Desc=Reflector "Color0=0xFF00FF"
```

You can also add attributes such as `Shape`, `Desc`, `Color0` and `Color1`. To include spaces in the `Desc` or `Label` attributes, surround the whole key=value part in quotes (e.g. `"Desc=Invisibel Box for Simulator"`).

Once these definitions are added to a map file which is then loaded into Mapper3, then Mapper3 makes them available: in the Boundary tools for Reflector lines, and the Sector tools for boxes. You can modify the Color and Shape parameters if you would like them to be displayed differently in Mapper3. When instances of the objects are added with Mapper3 they are then added to the map file as "Cairn" map objects (similar to goals etc.).

Default Map File

If you find that you are often loading the same map file into MobileSim over and over again, you can save a step by copying that map file into `~/.MobileSim` (Linux) or `\Documents and Settings\`*`<User Name>`*`\MobileSim` (Windows) and naming it `init.map`. If this map file exists, MobileSim will load this map at startup if no command-line option is given, rather than dislpaying the initial Load Map File dialog box.

Exit Codes

The MobileSim process has the following exit codes:

0	No error
255 (-1)	Error in command line arguments
254 (-2)	Error writing a temporary file on startup
253 (-3)	Error loading a map file
252 (-4)	Error initializing Stage
250 (-6)	Error loading a robot parameters file (.p file) to define a robot model
249 (-7)	Error initializing the Pioneer emulation thread
248 (-8)	Error opening a TCP socket for a robot or factory
236 (-20)	MobileSim crashed, but crash handler caught signal and logged information to log file.

Error codes may also be requested in the SIM_EXIT command by a client (use positive error codes less than 127, to differentiate from possible MobileSim codes).

Robot Protocol Support

Currently the following parts of the ARCOS/AROS/P2OS/AmigOS/PSOS protocols are supported. This covers all robot motion commands used by ARIA (including both direct motion requests and actions), and several other features.

Name	Command #	Argument Type	Description
PULSE	0	none	No-op (but reset watchdog timeout)

OPEN	1	int	Start up standard devices and send SIP packets
CLOSE	2	int	Stop standard devices
ENABLE	4	uint	Enable (1) or disable (0) motors
SONAR	28	uint	Disable (0) or re-enable (1) sonar (but using argument bits 1-4 to select particular arrays is not supported)
CONFIG	18	int	Request a configuration packet
STOP	29	int	Stops the robot from moving
VEL	11	int	Set the translational velocity (mm/sec)
ROTATE	9	int	Set rotational velocity, duplicate of RVEL (deg/sec)
RVEL	21	int	Set rotational velocity, duplicate of ROTATE (deg/sec)
VEL2	32	2bytes	Independent wheel velocities. High 8 bits are left wheel velocity, low 8 bits are right wheel velocity.
HEAD	12	uint	Turn to absolute heading 0-359 (degrees)
SETO	7	none	Resets robots odometry back to 0, 0, 0
DHEAD	13	int	Turn relative to current heading (degrees)
SETV	6	int	Sets maximum velocity and MOVE velocity (mm/sec)
SETRV	10	int	Sets the maximum rotational and HEAD velocity (deg/sec)
SETA	5	int	Sets forward translational acceleration or deceleration (mm/sec/sec)
SETRA	23	int	Sets rotational accel(+) or decel(-) (deg/sec)
MOVE	8	int	Translational move (mm)
LATVEL	110	int	Set lateral velocity (on robots that support it, i.e. Seekur)
LATACCEL	113	int	Set lateral acceleration or deceleration (on robots that support it, i.e. Seekur)
BATTEST	250	int	Artificially set battery voltage, in decivolts (e.g. 105 for 10.5 volts)
ESTOP/QSTOP	55	none	Stop with maximum deceleration.
BUMPSTALL	44	int	Configure bumpstall behavior.
TTY2	42	string	Log with string contents
TTY4	60	string	Log with string contents
RESET or MAINTAINENCE	253 or 255	none	Abort MobileSim for debugging. If running in noninteractive mode on Linux, restart program after logging some debugging informamtion using gdb.

Argument types: int=16-bit signed integer; uint=16-bit unsigned integer; 2bytes=pair of 8-bit bytes; int4=32-bit signed integer.

Simulator-only commands

For compatability with SRISim and old ARIA versions, equivalent commands with deprecated ID numbers are also accepted, with a warning.

MobileSim will always use the robot name "MobileSim". It will also report an OS version number starting with "S." and the string "SIM" for serial number in the CONFIGpac packet, and respond to the SIM_STAT command with a SIMSTATpac packet and the SIM_CTRL,6 command with a SIM_INFO packet. You can use these values in client software to detect whether it is connected to MobileSim or not.

SIM_SET_POSE	224	int4,int4,int4	Move robot to global pose in simulator (does not change odometry). The parameters are 4-byte integers for X, Y and Theta. The packet's argument type byte is ignored. Here is an example of how to send this command in ARIA, where `robot` is an `ArRobot` object, and `x`, `y` and `th` are `int` variables containing desired position in milimeters and degrees:
			`ArRobotPacket pkt;`
			`pkt.setID(ArCommands::SIM_SET_POSE);`
			`pkt.uByteToBuf(0); // argument type: ignored.`
			`pkt.byte4ToBuf(x);`
			`pkt.byte4ToBuf(y);`
			`pkt.byte4ToBuf(th);`
			`pkt.finalizePacket();`
			`robot.getDeviceConnection()->write(pkt.getBuf(), pkt.getLength());`

SIM_RESET	225	none	Move robot to original starting position in simulator (e.g. Home point or Dock)

<table>
<tr><td></td><td></td><td></td><td>and reset odometry to 0,0,0.</td></tr>
<tr><td>SIM_LRF_ENABLE</td><td>230</td><td>int</td><td><table><tr><td>1</td><td>to enable simulated laser rangefinder in-band (data packet ID=0x60).</td></tr><tr><td>2</td><td>to enable simulated LRF with extended information (data packet ID=0x61).</td></tr><tr><td>0</td><td>to disable LRF.</td></tr></table>See below for data packet formats.</td></tr>
<tr><td>SIM_LRF_SET_FOV_START</td><td>231</td><td>int</td><td>Set start angle of laser field of view</td></tr>
<tr><td>SIM_LRF_SET_FOV_END</td><td>232</td><td>int</td><td>Set end angle of laser field of view Currently start and end must be symetrical around 0. ARIA always does this, but if you want configure it differently it won't work.</td></tr>
<tr><td>SIM_LRF_SET_RES</td><td>233</td><td>int</td><td>Set degree increment between each reading; with FOV, this number determines the number of readings in each sweep</td></tr>
<tr><td>SIM_CTRL</td><td>236</td><td>int,...</td><td>Perform a simulator meta-operation. The initial 2-byte integer indicates the operation. The rest of the packet is operation-specific:<table><tr><td>1</td><td>Replace map</td><td>Remove existing map data and load a new map from the file given as a length-prefixed string folowing the operation code. Note: If the path is not absolute, it will be interpreted as relative to MobileSim's startup working directory, or the directory given with the --cwd command-line argument. For compatibility with Windows and other platforms with case-insensitive file naming, the case of the file name does not matter. A SIM_MAP_CHANGED packet will be sent back to the client when MobileSim is done loading the new map.</td></tr><tr><td>2</td><td>Master replace map</td><td>Similar to 1, but once this command is received, future control commands with operation 1 are ignored, only operation 2 is accepted. Use this when many clients are requesting different maps, but you want one of them (or a seperate program) to instead have control over which map is being used.</td></tr><tr><td>3</td><td>Master map clear</td><td>Disable master map mode if enabled by 2</td></tr><tr><td>4</td><td>Rotate logs</td><td>If logging to a file (--log-file command-line option given), then close current log file, move old log files to backup files (up to 5 are kept), and open a new log file. Only implemented for Linux currently.</td></tr><tr><td>5</td><td>Log detailed state</td><td>Write detailed internal simulation state for this robot. (For debugging, mainly.)</td></tr><tr><td>6</td><td>Request SIMINFO packet</td><td>Request SIM_INFO packet (see below).</td></tr></table></td></tr>
<tr><td>SIM_STAT</td><td>237</td><td>none, 1, 2 or 0</td><td>Request SIMSTAT packets (ID=0x62) to be returned. See below. Argument: none or 1 to return one packet, 2 to return a packet before each SIP, or 0 to stop sending.</td></tr>
<tr><td>SIM_MESSAGE</td><td>238</td><td>byte,string...</td><td>Display the length-prefixed string in the messages window.</td></tr>
<tr><td>SIM_EXIT</td><td>239</td><td>int</td><td>Exit the simulator with the given exit code. (use 0 to indicate "normal" exit).</td></tr>
</table>

The normal simulator laser packet is as follows:

Packet ID	ubyte=0x60
Robot X	int
Robot Y	int
Robot Theta	int
Total No. Readings	int
Current	int, index of the first reading given in this packet
Packet Len.	int, number of bytes of data that follow

For each reading:		
	Reading Range	int, Range for reading
...		

The extended simulator laser packet is as follows:

Packet ID	ubyte=0x61	
Total No. Readings	int	
Current	int, Index of first reading given in this packet	
Packet Len.	int, number of bytes of data that follow	
For each reading:		
	Range	int, Range for reading #i
	Reflectance	ubyte, Reflectance value, if supported
	Flags	2 bytes, Reserved
...		
Laser Device Index	ubyte, Reserved	
Flags	ubyte, Reserved	

The SIMSTAT packet is as follows. New fields may be added in future versions.

Packet ID	ubyte=0x62 (98)	
unused/reserved NULL byte	ubyte=0	reserved/unused
unused/reserved NULL byte	ubyte=0	reserved/unused
Flags	4ubyte	bit 0=have a map; bit 1=map has origin georeference (OriginLatLonAlt)
SimInterval	2ubyte	Configured simulated time one loop interval takes
RealInterval	2ubyte	Configured real time one loop interval should take
LastInterval	2ubyte	Real time the last loop interval was measured at
True X	4byte	Robot's "true" X position in the simulator
True Y	4byte	Robot's "true" Y position in the simulator
True Z	4byte	Robot's "true" Z position in the simulator
True Theta	4byte	Robot's "true" rotation on its vertical axis in the simulator
Geo X	4byte	Robot's latitude (degrees * 10^6). Only valid if map has origin georeference.
Geo Y	4byte	Robot's longitude (degrees * 10^6). Only valid if map has origin georeference.
Geo Z	4byte	Robot's altitude (cm). Only valid if map has origin georeference.

Here is an example of a packet handler function for ARIA that extracts some data from the SIMSTAT packet. Wrap this function in an `ArFunctor` object and use `ArRobot::addPacketHandler()` to register it with an `ArRobot` object.

```
bool handleSimStatPacket(ArRobotPacket* pkt)
{
   if(pkt->getID() != 0x62) return false; // SIMSTAT has id 0x62
   printf("SIMSTAT pkt received:\n"    );
   char a = pkt->bufToByte();  // unused byte
   char b = pkt->bufToByte();  // unused byte
   ArTypes::UByte4 flags = pkt->bufToUByte4();
   printf("\tFlags=0x%x\n", flags);
   int simint = pkt->bufToUByte2();
   int realint = pkt->bufToUByte2();
   int lastint = pkt->bufToUByte2();
   printf("\tSimInterval=%d, RealInterval=%d, LastInterval=%d.\n", simint, realint, lastint);
   int realX = pkt->bufToByte4();
   int realY = pkt->bufToByte4();
   int realZ = pkt->bufToByte4();
   int realTh = pkt->bufToByte4();
   printf("\tTrue Pose = (%d, %d, %d, %d)\n", realX, realY, realZ, realTh);
   if(flags & ArUtil::BIT1)
   {
      double lat = pkt->bufToByte4()/10e6;
      double lon = pkt->bufToByte4()/10e6;
      double alt = pkt->bufToByte4()/100;
      printf("\tLatitude = %f deg., Longitude = %f deg., Altitude = %f m\n", lat, lon, alt);
   }
   else
```

```
    {
        puts("No geoposition.");
    }
    return true;
}
```

The SIMINFO packet is as follows. New fields may be added in future versions.

Packet ID	ubyte=0x63 (100)	
Application name	NULL-terminated string	Name of simulator
Application version	NULL-terminated string	Version of simulator
Flags	4ubyte	Bit 0=Sim is interactive with user and has a GUI; Bit 1=Sim crashed last time but was automatically restarted.
Num. Devices	4ubyte	Number of devices attached to the robot
For each device:		
Device Name	string	Unique name for this device
Device Type	string	e.g. "laser"
Device Number	ubyte	To distinguish individual devices of the same type
Device Status	ubyte,ubyte,ubyte,ubyte	Device-dependent status code(s)
...		

The SIM_MAP_CHANGED packet is as follows. New fields may be added in future versions.

Packet ID	ubyte=0x66 (102)
User	ubyte. 1=Map was loaded by user command (e.g. via gui), 0=Map was loaded by client program SIM_CTRL command.
Loaded	ubyte. 1=A new map was really loaded. 0=Map was not reloaded because already loaded and file is unchanged since last load.
Filename	string. filename of map that was loaded.

Unsupported commands

The following commands are not supported, and will be ignored, with a warning message. Some are not applicable to the simulator (ENCODER) or pertain to devices that aren't implemented in the simulator yet (gripper, sounds).

ENCODER	19
POLLING	3
SAY	15
JOYINFO	17
DIGOUT	30
GRIPPER	33
ADSEL	35
GRIPPERVAL	36
GRIPPERPACREQUEST	37
IOREQUEST	40
PTUPOS	41
GETAUX	43
TCM2	45
JOYDRIVE	47
HOSTBAUD	50
AUX1BAUD	51
AUX2BAUD	52

GYRO	58		
CALCOMP	65		
TTY3	66		
GETAUX2	67		
SOUND	90		
PLAYLIST	91		
SOUNDTOG	92		
OLD_LOADWORLD	63	string	Load a 2-D world file from worlds/ directory
OLD_LOADPARAM	61	string	Load a robot definition from params/ directory
OLD_STEP	64	int	Steps through client program/simulation

The following commands will probably never be supported since they do nothing on modern robots or use is discouraged:

DROTATE	14
DCHEAD	22
KICK	34

Old simulator laser commands: These commands are supported for compatability with the previous simulator and ARIA (but may be disabled with the --no-srisim-laser-compat option):

OLD_LRF_ENABLE	35	int	Start/stop sending laser data in-band
OLD_LRF_PARMSTART	36	int	Sets start angle 0-180 degrees; also disables laser; with #37, determines "flipped" order of readings.
OLD_LRF_PARMEND	37	int	Sets scan end angle; also disables laser; with #36, determines "flipped" order of readings.
OLD_LRF_PARMINC	38	int	Sets inter-sample angle (100ths of degree)

Deprecated SRISim commands: These commands are normally disabled, but may be enabled with the --srisim-compat

OLD_END_SIM	62	none	Exit simulator (MobileSim exits completely). Note, this command is also used for another use on real robots and newer version of ARIA send it after connecting (so the simulator exits immediately). You can enable the other SRISim compatability commands with `--srisim-compat`, but disable this command using `--ignore-command 62`
OLD_SETORIGINX	66	int	Set the robot's current true X, (Note, conflicts with TTY3 command)
OLD_SETORIGINY	67	int	...Y
OLD_SETORIGINTH	68	int	...Theta
OLD_RESET_TO_ORIGIN	69	none	Reset true X Y, TH to 0,0,0

Copyright (C) 2006, 2007, 2008, 2009, 2010 MobileRobots Inc.
Copyright (C) 2011, 2012 Adept Technology
All Rights Reserved.

Patrolbot® and Seekur® are registered trademarks of Adept Technology