[CNW:Counter]

AUTONOMOUS MOBILE ROBOTICS TOOLBOX
User's manual

Last update : 01.06.2001

Back to AMRT Homepage

TABLE OF CONTENTS

INSTALLATION

The installation procedure is very simple. Copy the contents of downloaded zip archive into any directory on your disk and add it to the matlabpath (easy way is to use Path Browser). It is also necessary to add the directory with DLLs to current MATLAB's search path.
That's all!
Note: There were some problems with directory names with # (hash symbol) as the first character

QUICK START

First of all, it is STRONGLY recommended to close all your MATLAB tasks and save your data. Although the SIMROBOT toolbox was tested, there can be some bugs hidden. The second reason to do that (especially on slower computers) is that sensor simulation and control algorithms execution (esp. more complex or based on fuzzy logic or neural networks) is very CPU intensive.

For quick start, you can have a look at included demo by following these simple steps:

1. Install the SIMROBOT toolbox
2. Type `simview` in the MATLAB Command Window, this will start Simulator
3. Click on `File`, then select `Open..` and then open the file `demo.mat`
4. Click on `Run`
5. You should see four robots moving in virtual world, avoiding obstacles and each other by steering left when they are near any obstacle, and going straight on, when they have free way ahead. The length of this demo is 300 steps. When the simulation ends, you can replay it by clicking on *Run replay* or draw tracks of all robots by clicking on *Draw tracks* (both in *Run* menu).

Index

SIMULATION IN DETAILS

All robots are stored in a vector (as objects of the SIMROBOT class type) called the *list*.

The simulation is run *step by step*. In each step, for all robots in the list, their control algorithms are executed (they are standard m-function), new position and heading of each robot is computed and the simulation window is redrawn. Only non-crashed robots with "power" set to ON are simulated ! Algorithms of crashed or stopped robots´ are not executed.

Name of the file with control algorithm is stored *without* extension (.m) and *without* path. Therefore the file must be in any directory on MATLAB search path.

Each virtual sensor can be either *ultrasonic sensor* or *laser scanner*. It depends only on the function, which is used to obtain data from the sensor.

The total time of one simulation step is equal to the time needed to finish all described actions. As it is easy to see, if any algorithm contains an error, e.g., endless loop, the simulation will not run.
When the simulation contains a lot of robots with complex algorithms or is run on a slow computer, one simulation step can be pretty lengthy. For that reason, there is the possibility to replay the simulation. The position and heading of each robot in each step is stored in robot object (in *history* field). When replaying, these stored coordinates of each robot in given step are used (only for visualisation) - this is much faster. The replay data can also be saved to disk and viewed later.

The minimum length of each replay step can be set by the user. This is necessary on faster computers. The simulation is always run at maximum possible speed.

Index

OBJECT STRUCTURE

Both toolbox applications are *object-oriented* programs. Each robot is an *object*,i.e. , structured variable with all important data stored within.

Object structure allows to store all important data. See *Fig. 1*.

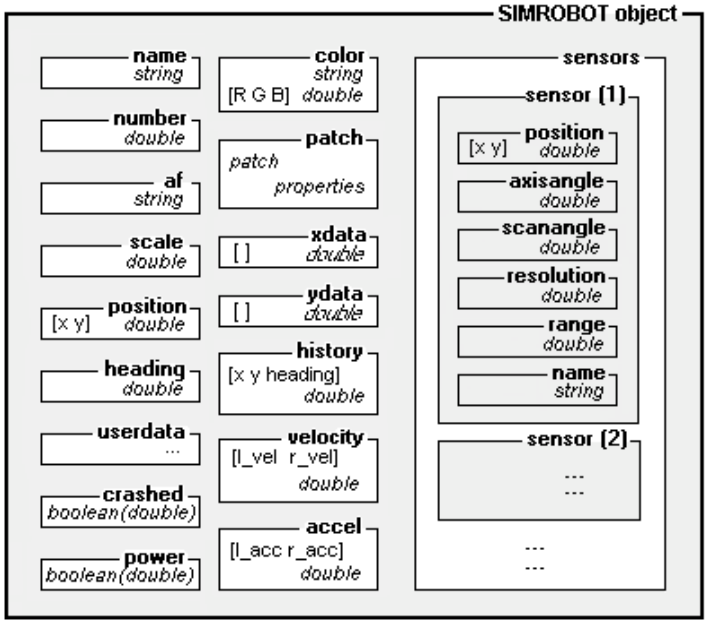

Figure 1: Object structure

A brief description of the structure

name - user-defined name of the robot
number - unique ID number
af - control algorithm file name
scale - scale of the robot - size of the robot can be adjusted without new shape definition
position - actual position of the robot in virtual environment map
heading - heading of the robot (in degrees)
userdata - this is the robot´s "memory" and any user data can be stored here
crashed - flag, set when a collision is detected
power - on/off switch flag
color - color of the robot ([R G B] vector)
patch - data for a patch (polygon) visually representing the robot
xdata - data of *x*-coordinates for the patch definition
ydata - data of *y*-coordinates for the patch definition
history - record of position and heading data of the robot in each simulation step
velocity - angular velocity of left and right actuated wheel
accel - angular acceleration of left and right actuated wheel
sensors
- structure defining sensorial subsystem of the robot (sensor data = position on the robot, angle of axis, beamwidth (in degrees), range, sensor name)

Index

VIRTUAL ENVIRONMENT MAP

To create a **new map**
of the virtual environment, use any graphics editor, which can save the image as a 1-bit (black/white) bitmap. The drawn points will be represented as solid obstacles (walls).
The virtual environment map is internally represented by *uint8* (unsigned 8-bit integer) matrix. The maximum value of each matrix element is therefore 255 (the maximum value of 8-bit integer minus 1), and minimum is 0. See *Tab. 1*.

Table 1

0	empty space
1	solid obstacle
2-255	robot

It is **necessary**
to enclose the whole map with walls, because the robots would otherwise run out of the screen, which could possibly lead to a program crash.

Index

COORDINATE SYSTEM ASSIGNMENTS

The robot coordinate system assignments is shown at *Fig. 2*. The robot is in "basic" position (heading = 0°). The position of virtual sensors placed on the robot is given in this coordinate system. When creating user-defined shape of a robot, it could be useful to keep an eye on this figure.
Arrows show the direction of positive and negative angle increase in value.
The basic unit of length is 1 (one) centimetre - one map point (one matrix element) represents 1 (one) square centimetre in real-world environment.

Note: The basic unit of length within the toolbox can be set arbitrary (e.g. one inch). The distances measured by sensors are in the toolbox units (when converted to real-world ones). Please read the note in Kinematic modeling section for more information.

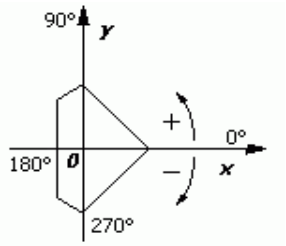

Figure 2: Coordinate system of the robot

KINEMATIC MODELING

The Simulator assumes wheeled mobile robot(s) consisting of one or two conventional, steered, unactuated and not-sensed wheels and two conventional, actuated, and sensed wheels (conventional wheel chair model). This type of chassis provides two DOF locomotion by two actuated conventional non-steered wheels and one or two unactuated steered wheels (i.e., one (two) castors).
Robot has two degrees of freedom (DOFs): *y*-translation and either *x*-translation or *z*-rotation.

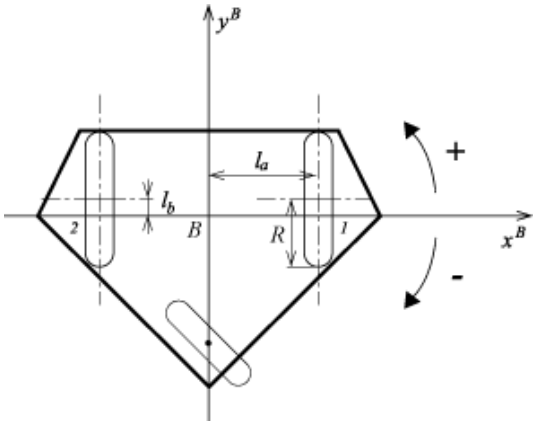

Figure 3: Kinematic coordinate system assignments

The sensed forward velocity solution is (see [2]):

$$\begin{pmatrix} v_{B_x} \\ v_{B_y} \\ \omega_{B_z} \end{pmatrix} = \frac{R}{2l_a} \begin{pmatrix} -l_b & l_b \\ -l_a & -l_a \\ -1 & 1 \end{pmatrix} \begin{pmatrix} \omega_{W_1} \\ \omega_{W_2} \end{pmatrix}$$

and the actuated inverse velocity solution

$$\begin{pmatrix} \omega_{W_l} \\ \omega_{W_r} \end{pmatrix} = \frac{1}{R(l_b^2+1)} \begin{pmatrix} -l_a l_b & -l_b^2-1 & -l_a \\ l_a l_b & -l_b^2-1 & l_a \end{pmatrix} \begin{pmatrix} v_{Bx} \\ v_{By} \\ \omega_{Bz} \end{pmatrix},$$

where (in metric system)

v_{B_x}, v_{B_y} are translational velocities of the robot body [m.s^{-1}],

ω_{B_z} is the robot *z*-rotational velocity [rad.s^{-1}],

$\omega_{W_l}, \omega_{W_r}$ are wheel rotational velocities [rad.s^{-1}],

R is actuated wheel radius [m],

l_a, l_b are distances of wheels from robot's axes [m].

This kinematic velocity solution is implemented by m-function called `mmodel.m`. Constants l_a, l_b and R should be set according to the robot proportions. Implicit values are l_a=3 cm, l_b=0 cm a R=1 cm. Keep this values consistent with those ones in `invmodel.m`
function (inverse velocity solution function). One simulation step represents one second in real-time (i.e. the robot movement in one simulation step is equal to the robot movement during one-second real-time period).

Note: The basic unit of length within the toolbox can be set arbitrary, but it affects the above kinematic equations. Therefore , when the basic unit is that 1 centimetre, all velocities will be in cm.s^{-1}. Important thing is to enter the correct robot proportions converted to these basic units of length (see constants mentioned above). Wheel velocities are still in rad.s^{-1} (radians per second).

Index

SENSORS MODELING

Detection of obstacles in the simulation is based on deciding, if there is a non-zero element (indicating an obstacle or robot) at given location in the matrix (virtual environment map).
Simulation of sensors uses the Bresenham's algorithm (see [1], here or here). Sensor beam is interlaced by *rays*. The number of rays is given by the *resolution*
parameter. Coordinates of points lying on these rays are computed using mentioned algorithm. If the ray collides with an obstacle (i.e., there is a non-zero element in the matrix), the endpoint of that ray is stored and computation of new ray is started. For the *ultrasonic sensor*, the nearest point is selected and its distance is returned (with the obstacle number (see Tab. 1)). If there is no obstacle within sensor range, the value 999999 is returned. For *laser scanner*, distances to all detected obstacles are returned (with the obstacles numbers).

Index

ROBOTS DETECTION

Besides solid obstacles detection, the robots should also be detected by the sensors. This is ensured by placing each robot into the virtual environment map as an obstacle in each step. In addition, for the possibility of distinguishing between the robots, each robot has the unique ID number. This number is generated by Editor when a new robot is created. When the robot is deleted, the robots in the list are renumbered.
But, sensors usually cannot detect the type of an obstacle (or number of detected robot). This can be ensured by, e.g., reading some kind of bar code placed on each robot. The main advantage of using the same routine for obstacles and robots detection is the fact that the simulation is faster and the program less complex.

Index

COLLISION DETECTION

If a robot collide with an obstacle or other robot, it is stopped. Collision detection is performed when the robot is placed as an obstacle into the matrix. This routine has been written in C and compiled into dynamically linked library

(DLL) - (the *mex* file).

Index

EDITOR

Editor allows the user to create and modify the simulation easily. It can be controlled using the window menu and/or the context menu, which can be invoked by right-clicking within the main window axes (white square/rectangle in the middle). Right-click on a robot brings up the context menu, which allows the user to modify properties of the robot.

STARTING EDITOR

To start Editor, type `simedit` in the MATLAB Command Window.
The main Editor window should appear on the screen. Note that some menus are disabled (some actions cannot be allowed unless there is any simulation opened/created). Actions allowed after Editor start are : "open simulation" (File > Open ..) and "import bitmap" (virtual environment map) (File > Import Bitmap ..). These actions can be activated also from the context menu.

Index

EDITOR MENUS

EDITOR WINDOW TOOLBAR MENU

Window menu is shown in *Fig. 4*.

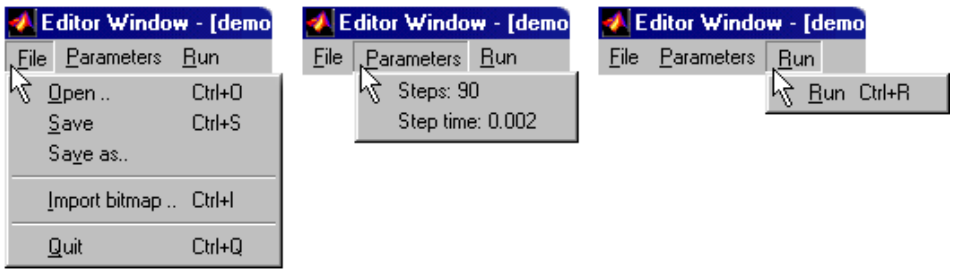

Figure 4: Editor menu

- FILE (Alt+F)
  Open .. (Ctrl+O)
  Opens the simulation or replay file for editing. The file structure is checked before opening. In case of incorrect file structure, warning message is displayed, and file is not opened. Replay data will be lost (the user is asked before proceeding).

  Save (Ctrl+S)
  Saves currently edited file to disk (under the same name).

  Save as ..
  Saves edited file under new name (using standard OS dialog box).

  Import bitmap (Ctrl+I)
  Allows the user to create a new virtual environment map. This action DELETES (after confirmation) all robots (cannot "move" the robots from one environment to another). The bitmap must be black&white (1-bit) bitmap. The map is then displayed.
  File name of just created simulation is set to `untitled.mat`. For some purposes, *no saved simulation* can have this name, otherwise it would be deleted, when importing a new map next time !! Therefore, Editor does not allow the user to save the simulation named this way. If the *Save* command is selected after importing a new bitmap, *Save as* .. dialog box opens and the user should enter different name (not `untitled.mat`).

  Quit (Ctrl+Q)
  Quits Editor (no questions, no auto-saving).

- PARAMETERS (Alt+P)

Displays and sets the simulation/replay parameters.
Steps limit
is the maximum number of simulation steps. This parameter can be used to restrict the simulation length. Set this value to *Inf* for unlimited number of steps - the simulation runs until it is stopped by the *Stop (Ctrl+T)* command. It is not possible to enter negative number, zero or NaN (Not-a-Number - a result of undefined mathematical operation, such as 0/0). Decimal numbers are rounded.
Step time
is minimum replay step time, in seconds. This parameter is used to slow down the replay on faster computers. The simulation is not affected by this parameter. It is not possible to enter negative number or NaN. Fraction mark can be either a period (.), or a comma (,).

- RUN (Alt+R)
  Run (Ctrl+R)
  Starts Simulator with currently edited simulation. Before that, the simulation is saved automatically. If this is the first saving after bitmap import, the user is asked to enter new file name (cannot be `untitled.mat`).
  Note: Simulator cannot be started, until there is at least one robot placed into the virtual environment map. Also, it is not possible to save this "empty" simulation.

Index

CONTEXT MENU

Context menu is shown in *Fig. 5*.

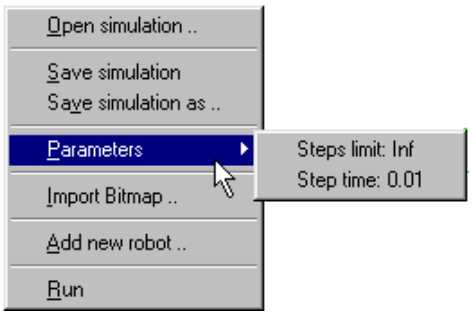

Figure 5: Editor context menu

The actions available from this menu are (almost) the same as described above. The only difference is in the command

- *Add new robot ..*
  This opens a window for adding a new robot, depicted in *Fig. 6*.

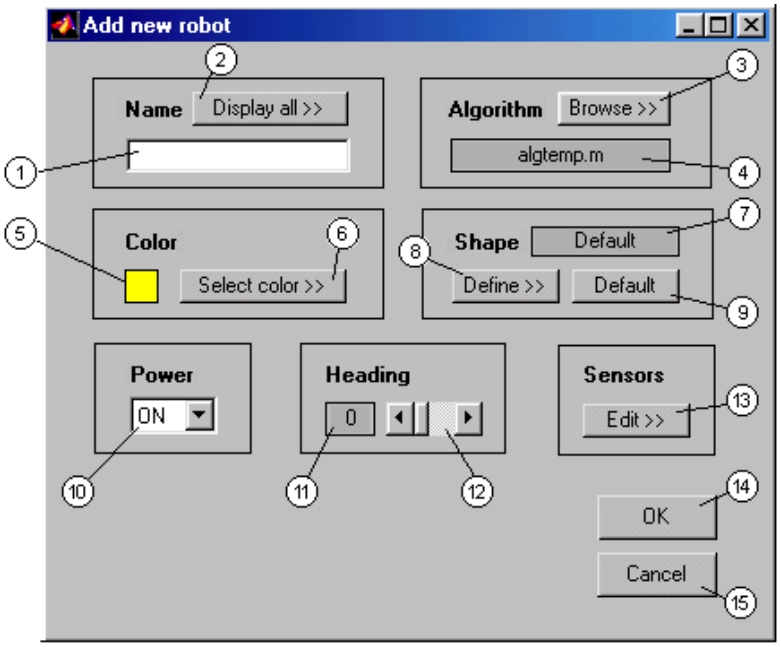

Figure 6: 'Add new robot' window

NAME

1. Name-input editable text. Name of created robot can be entered here.
2. *Display all* button. This button displays names of all robots in the simulation.

ALGORITHM

3. *Browse* button. Pressing this button allows the user to assign an algorithm to the robot. The default algorithm file for new robot is a "template" (empty algorithm file).
4. Algorithm file name. In this field, the algorithm file name is displayed. Context-sensitive help contains the full path.

COLOR

5. Color preview. Displays color of the robot.
6. *Select color* button. Opens system dialog box allowing the color change

SHAPE

7. Shape type indicator. Can be either *Default* (for default shape - see e.g.Fig. 2), or *User Defined* (if the user modifies the shape)
8. *Define* button. Opens dialog box for shape definition. The shape is defined by coordinates of polygon apexes (please refer to MATLAB help). The robot's centre (i.e., the origin of the robot's coordinate system - the *pivot*) must be on coordinates [0,0]. Scale of this new-created robot is set to 1.
9. *Default* button. This button sets the default shape of the robot (after confirmation by the user).

POWER

10. (Power pop-up menu). This pop-up menu defines the initial state of the "power switch".

HEADING

11. Heading in degrees.
12. (Heading slider). This slider can be used to adjust the initial heading of the robot.

SENSORS

13. *Edit* button. Invokes the Sensorial subsystem editor.

14. *OK* button . Accepts data and creates a new robot.
15. *Cancel* button

. Closes the window.

Index

CONTEXT MENU (FOR ROBOTS)

Context menu associated with each robot in Editor is depicted in *Fig. 7*. Right-click on the robot to display this menu.

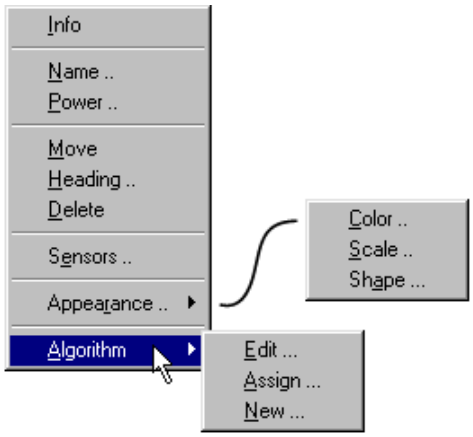

Figure 7: Context menu for robots in Editor

- *Info* - Displays information window (see Fig. 8).
- *Name* .. - Allows the user to change the name of the robot. Actual name of the robot is displayed implicitly.
- *Power* .. - Selecting this option allows the user to "switch the power switch" .
- *Move* - Changes the cursor shape. The crosshair indicates new position of the robot. Click on the desired location to move the robot here.
- *Heading* .. - displays the dialog box for heading adjustment, similar to that one in *Add new robot* dialog box).
- *Delete*
  - Deletes selected robot from the simulation. This action must be confirmed first. After that, all robots are renumbered.
- *Sensors* .. - Invokes *Sensorial subsystem editor*.
- *Appearance*
  - *Color* ..
    - Displays standard system dialog box for color change. The selected color is applied to the robot immediatelly.
  - *Scale* .. - This option is useful for adjusting the size of the robot.
  - *Shape* .. - Changes the robot´s shape by entering coordinates of *patch* apexes (please see section Coordinate system assignments).
- *Algorithm*
  - *Edit* ...
    - Opens the algorithm file associated with the robot in Windows Notepad (MATLAB Editor/Debugger cannot be used when any MATLAB m-file is running). If the algorithm file is used for other robots, the user is asked before editing.
  - *Assign* ...
    - Allows the user to select a new algorithm file. Note: The algorithm file should be in a directory on MATLAB search path.
  - *New* ...
    - Creates a new algorithm file in Windows Notepad editor. It is necessary to assign this file to the robot later!

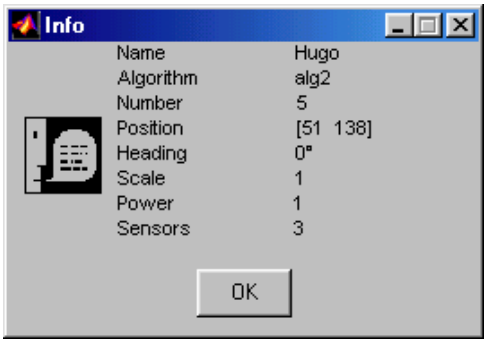

Figure 8: Information window

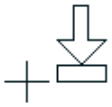

Figure 9: "Put-here" cursor

Index

SENSORIAL SUBSYSTEM EDITOR

This editor is a tool for editing sensors placed on a robot. It can be started by selecting *Sensors* .. from context menu of the robot or by clicking on *Edit* button in *Sensors* field of the *Add new robot* dialog box.
The window contains a Preview area, which displays the robot with all sensors´ beams. This area is redrawn (updated), whenever a sensor is selected from pop-up menu (1), deleted, renamed or *Update preview* button is pressed. To view the changes made, press this button.
Entered (modified) parameters are stored automatically (e.g., by clicking outside an editable text field). The OK button accepts all changes made to the sensorial subsystem of the robot and closes the editor!

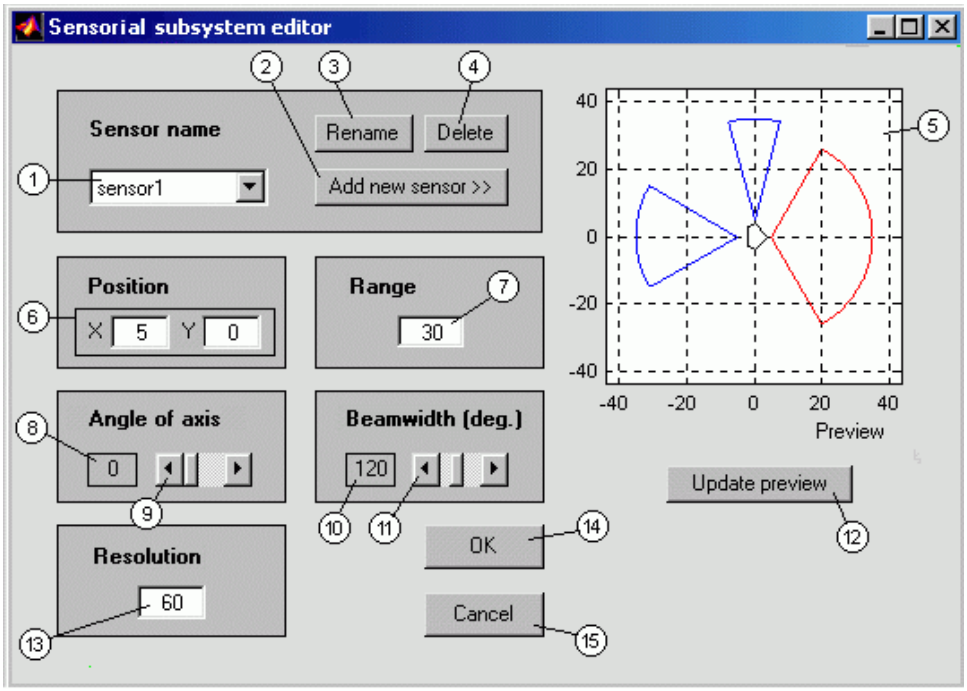

Figure 10: Sensorial subsystem editor

SENSOR NAME

1. This pop-up menu contains names of all sensors placed on selected robot. Parameters of the selected sensor can be edited and the sensor is highlighted (drawn in red color) in Preview area (5).
2. *Add new sensor* button. Allows the user to add a new sensor by typing its name. Default parameters of a new sensor are: position = [5,0], range = 30, resolution = 30, beamwidth = 60°, angle of axis = 0° (front sensor).
3. *Rename* button. Opens a dialog box for changing the name of the sensor. Actual name of the sensor is displayed.

4. *Delete* button. After user confirmation deletes the selected sensor.

5. Preview area. Displays the robot together with beams of all sensors. The selected sensor is drawn in red color, the rest of sensors are blue.

POSITION

6. Sensor coordinates. Position of the sensor on the robot. See coordinate system assignments.

RANGE

7. Range. Only obstacles nearer than this distance will be detected.

ANGLE OF AXIS

8. Angle between sensor axis and the *x*-axis of the robot coordinate system (in degrees).
9. Slider, sets the value.

BEAMWIDTH (deg.)

10. Beamwidth in degrees.
11. Slider, sets the value.

12. *Update Preview* button. This button redraws the *Preview area* (5).

RESOLUTION

13. Number of rays, which will be used for sensor simulation (see Sensor modeling)

14. *OK* button. Accepts changes and closes the editor.
15. *Cancel* button. Closes the editor without any changes to the sensors.

Index

SIMULATOR

Simulator allows the user to run created simulation or view saved replay. Overall appearence of the Simulator GUI is similar to Editor one. In lower right corner of the window is a step-counter, which displays the processed-step ordinal number.

STARTING SIMULATOR

Simulator can be started from Editor (using menus) or directly from the MATLAB Command Window by typing `simview`. As in Editor, some menus are disabled, when running Simulator from the MATLAB Command Window.

SIMULATOR MENUS

SIMULATOR WINDOW TOOLBAR MENU

Window menu is shown in *Fig. 11*.

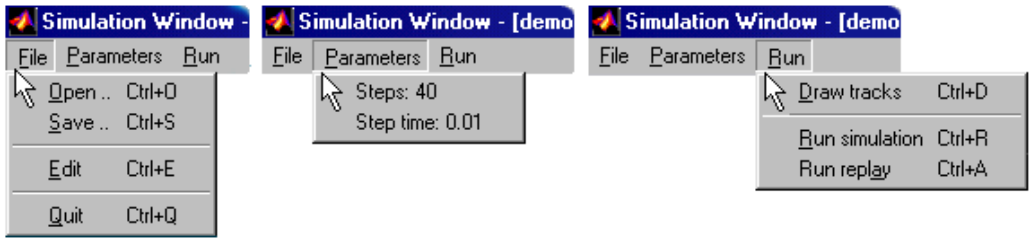

Figure 11: Simulator menu

- FILE (Alt+F)

  Open .. (Ctrl+O)
  Opens a file. It can be either simulation or replay. For the first one, it is possible to select the *Run simulation* command only. For the second one, the simulation or replay can be started. The simulation runs at maximum speed, the minimum replay step time is affected by *Parameters* settings. The file structure is checked before opening. In case of incorrect file structure, warning message is displayed, and file is not opened.

  Save .. (Ctrl+S)
  Saves opened file using the Save as .. system dialog box.

  Edit (Ctrl+E)
  Opens currently edited simulation/replay in Editor. All replay data will be lost without warning.

  Quit (Ctrl+Q)
  As it says, quits. No questions, no auto-saving.

- PARAMETERS (Alt+P)

  This menu displays the actual simulation parameter values. Click on a menu item brings up a dialog box, where the value can be changed.
  When the number of steps is changed, the *Run replay* command is deactivated. To activate it again, it is necessary to re-run the simulation.
  Steps limit
  is the maximum number of simulation steps. This parameter can be used to restrict the simulation length. Set this value to *Inf*
  (implicit) for unlimited number of steps. It is not possible to enter negative number, zero or NaN (Not-a-Number). Decimal numbers are rounded.
  Step time
  is minimum replay step time, in seconds. This is used to slow down the replay on faster computers. It is not possible to enter negative number or NaN (Not-a-Number). Fraction mark can be either a period (.), or a comma (,).

- RUN (Alt+R)

  From this menu, the simulation/replay can be run, stopped or trajectories of all robots can be drawn. When repaying, position/heading data recorded during simulation are used. To break the simulation/replay, use the *Stop (Ctrl+T)*
  command. It is also possible to stop the simulation by typing Ctrl+C, but it is NOT recommended. After that, it will be necessary to use the *Stop*
  command indeed, because almost all menus will stay deactivated. Moreover, when the simulation is stopped using this shortcut, all replay data from this simulation will be lost.
  Re-simulating is useful when developing algorithms. Right-click on a robot and select *Edit algorithm* .. from the context menu to edit the algorithm. This will run Notepad (in MS Windows). After saving modified algorithm, the simulation can be re-run using *Run - Run simulation* or Ctrl+R keyboard shortcut.
  To draw tracks of all robots, select *Draw tracks*
  from a menu or press Ctrl+D. These tracks are deleted before new simulation/replay.

Index

SIMULATOR CONTEXT MENU

Simulator context menu is depicted in *Fig. 12*.

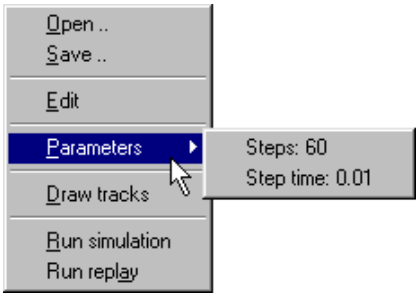

Figure 12: Simulator context menu

This menu can be invoked by right-click within the main window axes (white area). For menu description, please refer to previous section.

CONTEXT MENU (FOR ROBOTS)

Context menu for robots in Simulator is shown in *Fig. 13*.

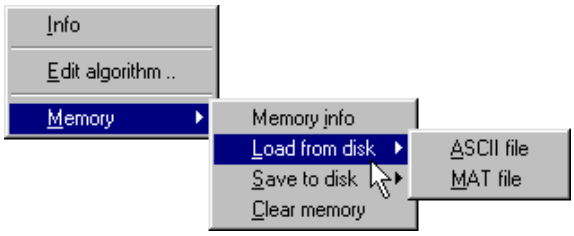

Figure 13: Context menu for robots in Simulator

- *Info* - Displays some information (see Fig. 8).

- *Edit algorithm ..* - Opens algorithm file in Notepad (MS Windows).

- MEMORY

  Memory info
  Displays information about the memory contents (matrix/structure size, size in bytes and class)

  Load from disk
  - Loads memory contents from disk (file import into robot's memory). Supported file formats are MAT and ASCII. MAT-file (allowing the user to store MATLAB variables using `save` command) **has to** contain a variable (or structure) called `mem`
  - contents of this variable will be imported into memory. ASCII files should have any of the formats supported by MATLAB (see `load` and `save` commands).

  Save to disk - Saves the memory contents to MAT (into `mem` variable) or ASCII file. Contents stored in this way can be re-loaded into robot's memory later (see previous option).

  Clear memory - Clears memory (after user´s confirmation).

  Note: Memory contents of all robots is stored into MAT-file when saving the simulation. These commands can be used to export/import data from/to memory of the robot.

Index

CONTROL ALGORITHMS

Control algorithms are standard MATLAB m-functions. Each robot has one control algorithm, but one control algorithm can be used by more robots. This control algorithm should be in a dirtectory on MATLAB search path, because only the file name is stored.

When a new robot is created, it is equipped with "algorithm template" (`algtemp.m`), which is actually an "empty" algorithm. Let´s have look at it:

```
function new = alg_name(simrobot,matrix,step)

% your algorithm starts here


% end of your algorithm

new = simrobot;
```

The variable *'step'* is the actual simulation step ordinal number and can be used in the algorithm.
The control algorithm should be written to indicated lines. For list of functions useful for writing control algorithms, please refer to the next section List of Simrobot commands.
Very easy algorithm is an example algorithm, `alg2.m`, which is used in the example simulation.

```
% sensor reading
   [dist,num] = readusonic(simrobot,'sens1',matrix);

% num, the nearest obstacle number, is not used


   if dist<20
      simrobot = setvel(simrobot,[0 0.5]);  % turn left
   else
      simrobot = setvel(simrobot,[0.5 0.5]); % go straight on
   end
```

Index

LIST OF SIMROBOT COMMANDS

When working with objects, it is possible to use specialized functions only. These are known as *methods*. In MATLAB environment, the methods for a class must be collected together in one directory with name formed with class name preceded by the character @. In the following section will be described the commands included in the *@simrobot* directory, which can be used in control algorithms. The other methods in this directory are *system* and should not be used in the algorithm (e.g., changing ID number of the robot by the user). Because the objects are MATLAB variables, working with variables and objects is very similar. The object altered by a method must be returned and stored.
Most of "set..." methods have the following syntax `new = method_name(simrobot,param1,param2,...)`, where `new` is the returned object, and `simrobot`
is the original object. These parameters will not be further described in the following list of commands.

Tip: It could be useful to use the `persistent`
command in algorithms. This makes a variable persistent in scope, i.e. it maintains its value from one call of the algorithm file to the next. Type `help persistent` in the MATLAB Command Window to get more detailed help.

Command

- *clearcf*

Purpose
Clears the crash-flag of the robot.

Syntax
`new = clearcf(simrobot)`

Command

- *getaccel/setaccel*

Purpose
Returns/sets the angular acceleration of left and right wheel.

Syntax
```
[left_accel,right_accel] = getaccel(simrobot)
new = setaccel(simrobot,[left_accel right_accel])
```

Returned value is two-element vector (double).

Command

- *gethead/sethead*

Purpose
Returns/sets the heading of the robot.

Syntax
```
heading = gethead(simrobot)
new = sethead(simrobot,name)
```

Returned value is in degrees. The angle 0° is returned, when the robot is "facing east".
"name" should be any string.

Command

- *getname*

Purpose
Returns the name of the robot.

Syntax
```
name = getname(simrobot)
```

Returned value is name of the robot (string).

Command

- *getnum*

Purpose
Returns the ID number of the robot.

Syntax
```
number = getnum(simrobot)
```

Returned value is ID number of the robot (double, but integer).

Command

- *getpos*

Purpose
Returns the actual position of the robot.

Syntax
```
position = getpos(simrobot)
```

Returned value is actual position of the robot ([x y], double). Lower left corner of the virtual "world" has coordinates [0,0].

Command

- *getpower/setpower*

Purpose
Returns/sets the "power switch state" of the robot (the robot is active/not active).

Syntax
```
power = getpower(simrobot)
new = setpower(simrobot,power)
```

Returned value is power state of the robot (boolean/double).
"power" can be either 0/1 or 'on'/'off'.

Command

- *getvel/setvel*

Purpose
Returns/sets the angular velocities of left and right wheel.

Syntax
```
[left_vel,right_vel] = getvel(simrobot)
new = setvel(simrobot,[left_vel right_vel])
```

Returned value is two-element vector (double).

Command

- *invmodel*

Purpose
This is the inverse kinematic model.

Syntax
```
[left_angular_vel,right_angular_vel] = invmodel(movspd, rotspd)
```

where
`left_angular_vel,right_angular_vel` are angular velocities of wheels,
`movspd` is the translational velocity of robot´s body,
`rotspd` is the *z*-rotational velocity.

Command

- *iscrashed*

Purpose
Returns the crash-flag.

Syntax
```
flag = iscrashed(simrobot)
```

If the robot is crashed, returns 1, otherwise 0.

Command

- *readlaser*

Purpose

Reads data from a laser scanner.

Syntax
```
[data,num] = readlaser(simrobot,sensor_name,matrix)
```

where
`data`
is a vector containing distances to the obstacles measured by given ray. The rays are read counterclockwise (i.e., the first returned distance is from the rightmost ray (when looking from behind the sensor)).
`num`
are the numbers of detected obstacles (1 for solid obstacle (the "wall"), other number for the robot. If there is no obstacle, 0 is returned),
`sensor_name`
is the name of the sensor. When there are more than one sensor with given name, only the first sensor in the list of sensors is read.
`matrix` is the matrix representing the virtual environment.

Note: If it is not necessary to get the detected obstacle/robot number, then the return parameter "num" can be omitted.

Command
- *readmem/writemem*

Purpose
Reads/writes to the "memory" of the robot.

Syntax
```
data = readmem(simrobot)
new = writemem(simrobot,data)
```

where `data` is read/written memory contents.

Command
- *readusonic*

Purpose
Simulates reading from an ultrasonic sensor.

Syntax
```
[distance,num] = readusonic(simrobot,sensor_name,matrix)
```

where `distance` is distance to the nearest obstacle.
`num`
is the detected obstacle number (1 for solid obstacle (the "wall"), other number for the robot. If there is no obstacle, 0 is returned). `sensor_name`
is the name of the sensor. When there are more than one sensor with given name, only the first sensor in the list of sensors is read.
`matrix` is the matrix representing the virtual environment.

Note: If it is not necessary to get the detected obstacle/robot number, then the return parameter "num" can be omitted.

REFERENCES

1. Wu, X. and Rokne, J.G. : Double-Step Incremental Generation of Lines and Circles", Computer Vision, Graphics and Image Processing, (37), 1987, pp. 331 - 334
2. P. Muir, Modeling and Control of Wheeled Mobile Robots, doctoral dissertation, tech. report CMU-RI-TR-88-20, Robotics Institute, Carnegie Mellon University, August, 1988, pp. 196-198
3. Miller, M.K. - Winkless, N. - Bosworth, J. : Personal robot navigator, Robot Press, Conifer, Colorado, 1998
4. Building GUIs with MATLAB, The MathWorks, Inc. 1997

5. Using MATLAB, The MathWorks, Inc., 1999

Top