AMORsim – A Mobile Robot Simulator for Matlab

Toni Petrinić, Edouard Ivanjko, Ivan Petrović

Department of Control and Computer Engineering, Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
E-mail: toni.petrinic@gmail.com, edouard.ivanjko@fer.hr, ivan.petrovic@fer.hr

Abstract: *In the development of mobile robot control algorithms the researchers are often faced with the problem of ensuring safety operation of used mobile robot during the real mobile robot experiment phase. For this reason implemented control algorithms have to be firstly tested in various simulation scenarios. Appropriate simulation tools have to enable easy creation of different simulation setups, usage of different sensors, collection and evaluation of sensor measurements, examination of noise influence, etc. This paper describes a developed simulation tool for such a purpose – AMORsim (**A**utonomous **MO**bile **R**obots **sim**ulator), a mobile robot simulator that's able to simulate a three-wheeled user-defined mobile robot in a two-dimensional environment. It's written in Matlab, which is a common, and well-known simulation environment.*

Keywords: *Mobile robot, Simulation, Matlab*

I INTRODUCTION

When developing control algorithms for mobile robots or any other expensive/complex systems, a simulation tool is often of significant importance in order to reduce development time, avoid damages due control algorithm failure, find errors in the implemented control algorithm, etc. A software simulation can be easily manipulated and monitored and is offering access to data that are hard to measure on a real mobile robot. Also different mobile robot types can be tested with no need of significant adaptations of the tested algorithm. An algorithm that proved to be successful in simulation has a good chance to be successful when implemented on a real mobile robot. The experiments on a real system must also be done because simulators provide only a simplified model of a real robot and its environment, that often doesn't take into account problems like process noise, real behavior of sensors, etc.

Many mobile robot manufactures offer simulators of their products but they are often limited in the capabilities. System and non-system error simulation is missing or can't be adjusted; external tools have to be used for data evaluation; etc. Also they don't enable access to real measurements, which makes them not suitable for education purposes because many students have problems understanding the difference between real, measured and estimated values, for example.

The AMORsim (**A**utonomous **MO**bile **R**obots **sim**ulator) simulator development was started at the Department if Control and Computer Engineering in order to take the mentioned drawbacks into account and to alleviate mobile robotics courses to students [1]. Matlab was taken as the platform of choice because it's a high-featured and user friendly environment for technical computing such as graphical user interface building, modeling, simulation, algorithm development, data analysis and graphical presentation of collected data [2]. It also offers a programming language, which is suitable for algorithm implementation without the need for complicated variable definition and memory allocation like in the C/C++ programming language. This allows students to focus more on the problem that has to be solved than on the program implementation.

In the following sections we give a short overview of current available mobile robot simulators and then describe the AMORsim simulator.

II EXISTING SIMULATORS

In the past, several mobile robot simulators have been developed. Most of them are written in C/C++. The main disadvantages of simulators written in C/C++ is that people have to posses a good programming knowledge in order to do some changes or to adapt the simulator to their particular needs. Also for some more advanced data processing like operation with matrices appropriate libraries have to be used. Some examples of existing simulators are: Stage [3], Khepera [4] and SIMROBOT [5].

Stage simulator can simulate one or more mobile robots in a two-dimensional environment. Various sensor models are provided, including sonar, laser range finder, a pan-tilt-zoom camera, and odometry. This simulator comes with some predefined robot models such as the Pioneer 2DX and the Segway RMP. It also offers the possibility of creating and integrating own robot models as plug-ins but this has to be done by code-based modeling in C/C++.

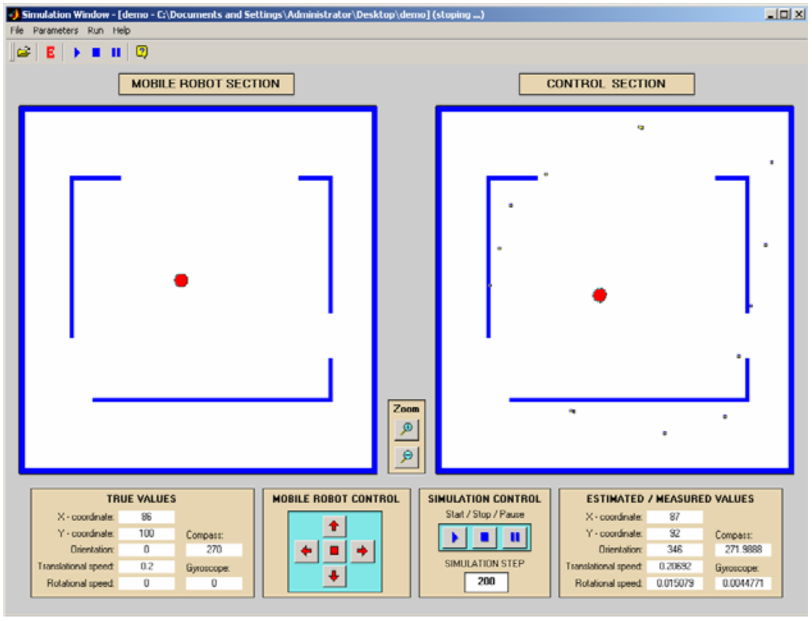
Khepera simulator simulates one or more Khepera mobile robots in a two-dimensional environment. The mobile robot is equipped with 8 infrared sensors, which is the only type of implemented sensor. There is no possibility of creating own mobile robot models. The user can write own control algorithms in the C/C++ language. A Matlab interface is also available.

SIMROBOT is a mobile robot simulator for Matlab. It has been developed as part of a diploma thesis at the Department of Control, Measurement and Instrumentation, Faculty of Electrical Engineering and Computer Science, Brno University of Technology. It simulates one or more mobile robots in a two-dimensional environment. The user can write own control algorithms and setup sensors parameters for each mobile robot. There are two types of implemented range sensors - sonar and laser range finder. This simulator fulfils some of the mentioned requirements and as such it is used as the starting point for AMORsim development.

III THE AMORSIM MOBILE ROBOT SIMULATOR

When we used the above-mentioned simulators, they showed fairly limited options for education purposes. None of them had options to display the true and estimated mobile robot pose, to visually present what happens with range measurements when a pose drift occurs, to present data available to an operator using a control unit to control the mobile robot, include systematic and non-systematic errors simultaneously enabling students an easy implementation of localization, mapping, path planning and obstacle avoidance algorithms. Whence our department owns a Pioneer 2DX mobile robot [6], we needed also a development environment for this three wheeled mobile robot that contains the above mentioned features and is more suitable for students without a very good knowledge of the C/C++ programming language.

To meet these requirements the following steps were performed. A model of the Pioneer 2DX mobile robot was created. The kinematic model corresponds to a three cycle mobile robot with two drive wheels and one castor wheel for stability [7]. Modeled sensors are drive wheels encoders, compass, gyroscope, sonar, and laser range finder. Every sensor measurement can be corrupted whit white Gaussian noise. To include a presentation of a teleoperation unit the application window was divided into a mobile robot section and a control section (Fig. 1). The mobile robot section presents the simulated mobile robot location in the simulation environment, and the control section presents data available to a teleoperator in case of a real mobile robot (estimated pose, measured velocities and orientation, range measurements). In this way the students can observe the pose drift,



Simulation Window - [demo - C:\Documents and Settings\Administrator\Desktop\demo] (stoping ...)

TRUE VALUES			
X - coordinate:	86		
Y - coordinate:	100	Compass:	
Orientation:	0		270
Translational speed:	0.2	Gyroscope:	
Rotational speed:	0		0

SIMULATION STEP: 200

ESTIMATED / MEASURED VALUES			
X - coordinate:	87		
Y - coordinate:	92	Compass:	
Orientation:	346		271.9888
Translational speed:	0.20692	Gyroscope:	
Rotational speed:	0.015079		0.0044771

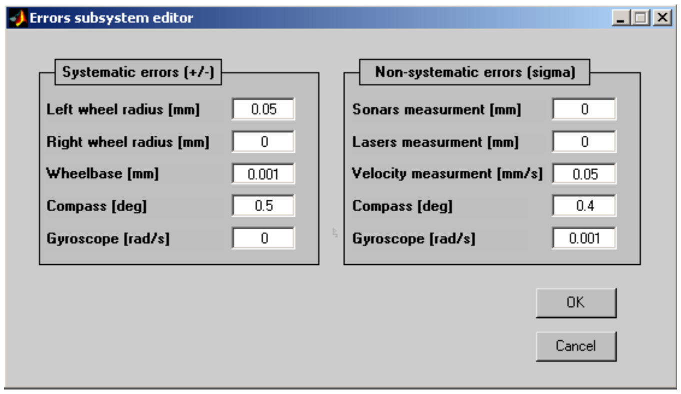
Figure 1
AMORsim graphical user interface

range measurements that don't mach the obstacle that caused them, and different values of real/measured/ estimated values. To allow a manual mobile robot motion control a keyboard interface and appropriate buttons are implemented.

In order to model systematic and non-systematic pose errors two mobile robot kinematics models are used. One presents the simulated mobile robot and uses kinematic models with all parameters exactly known. Resulting data is presented in the mobile robot section. The second one uses a kinematic model with nominal parameters and noisy measurements. Resulting data is presented in the control section.

The simulation consists of the following stages. First the true pose is computed. Then noise is added to drive velocities and using these noisy measurements the estimated pose is computed. The discrepancy from the nominal parameters and measurement noise values can be changed in a dialog box shown in Fig. 2. To present the difference between real, measured and estimated values all real uncorrupted values are made available (real mobile robot pose, drive wheel velocities, etc.). Furthermore, this allows plotting of the real and estimated mobile robot trajectory after simulation end and comparison of implemented localization algorithms quality.

Range sensors measurements are simulated using the *Bresenham's algorithm* [8]. Detection of obstacles in the simulation is based on detection of a cell at a given location in the occupancy grid simulation environment map with value '1'. Value '1' in the occupancy grid map means that this environment part is occupied and '0' means that's free. To alleviate the collision detection the internal occupancy grid map, that's also used for the visual representation can take a third value ('2') to denote that the mobile robot occupies this particular cell. If the mobile robot collides with an obstacle, it's stopped. Collision detection is done using overlap checking i.e. if any part of the mobile robot overlaps with an occupied grid cell, collision is detected. These routines are written in


Errors subsystem editor

Systematic errors (+/-)	
Left wheel radius [mm]	0.05
Right wheel radius [mm]	0
Wheelbase [mm]	0.001
Compass [deg]	0.5
Gyroscope [rad/s]	0

Non-systematic errors (sigma)	
Sonars measurment [mm]	0
Lasers measurment [mm]	0
Velocity measurment [mm/s]	0.05
Compass [deg]	0.4
Gyroscope [rad/s]	0.001

Figure 2
Graphical interface for systematic and non-systematic error definition

C/C++ and then compiled into a dynamically linked library (a Matlab 'mex' file) to ensure their faster execution during simulation.

Simulated environment model can be loaded in two formats. First format presents an occupancy grid map saved in a two-bit bitmap file and second format presents a feature map saved in a line based 'wld' model [6]. The occupancy grid map can so be created in any graphical editor, it just have to be saved as a black/white bitmap image and the feature map can be created in any textual editor. Only line features are used in the feature map and they are defined by line begin and end *x*, *y* Cartesian coordinates. When a feature map based simulation environment is loaded into the simulator an appropriate occupancy grid model is created, with the cell size automatically set to 1 [mm].

The simulation is done with fixed time steps without any dynamics included. In each time step first the mobile robot control algorithm is executed (implemented as a standard 'm'-function), and motion commands (left and right drive wheel velocities) are sent to the simulated mobile robot. Motion commands are the result of manual mobile robot control or of a navigation algorithm execution implemented as part of the mobile robot control algorithm. Then using the sent motion commands new mobile robot pose is computed, and using the new pose sensors reading are updated. At the end every time step the simulator window is redrawn. If the mobile robot collided with an obstacle, control algorithm execution is stopped. During simulation, all relevant data such as true and estimated pose, true and measured data (range, orientation, rotation speed) are stored in text files for off-line analysis. Total time of one simulation step is equal to the time needed to execute all described actions.

IV SIMULATION EXAMPLE

The simulator includes two independent applications - *Editor* and *Simulator*, which allows the user to create and/or modify a simulation and then run it.

The *Editor* application is started with the command ''*createsim*'' from Matlab. It allows the user to create a new or to modify an existing simulation setup. Creation or modification of a simulation setup includes creation/loading of a simulation environment map, mobile

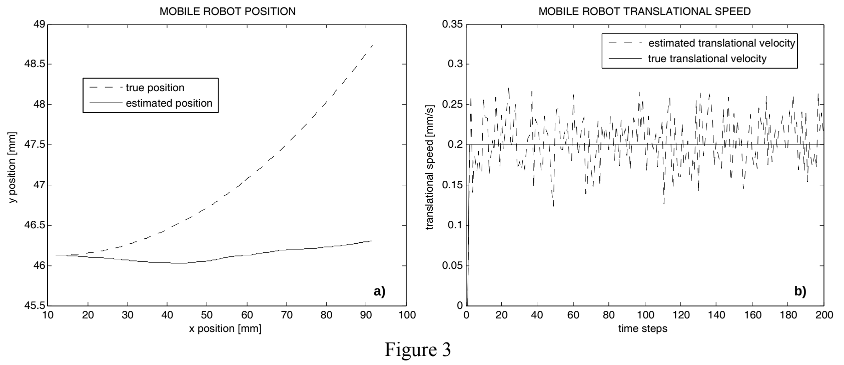


Figure 3
Mobile robot position (a) and translational speed (b) during simulation

robot model, or a control algorithm, and assignment of an initial mobile robot pose. The simulation setup can be saved and then loaded next time or just started using the ’*run*’ menu command. Previously saved simulation setups can be directly loaded and started.

The *Simulator* application allows the user to run a previously created and loaded simulation (Fig. 1). It’s started with the command ’*runsim*’ from Matlab or from the editor part using ’*run*’. During simulation relevant data are presented in the simulation window and also saved for evaluation. Example of collected sensor data is presented in Fig. 3.

Conclusion

A mobile robot simulator for Matlab is presented in this paper. Its visual appearance and algorithm implementation is adapted in order to represent the crucial problems in the field of mobile robotics to students in a more friendly way. It also provides a simple development environment for testing and implementation of mobile robotics related algorithms appropriate for students or researchers with poor C/C++ knowledge.

Future work will include a module for adding models of other mobile robot drive systems, like Ackerman or synchro drive. Apart from this, a module for collected data analysis, as well as a library of basic algorithms for localization, mapping, path planning and obstacle avoidance will be added. In order to achieve a more realistic simulation of real world conditions, ability of defining moving obstacles will also be considered.

Acknowledgement

This research has been supported by the Ministry of Science and Technology of the Republic of Croatia under grant No. 0036018. The authors also thank Marija Seder and Andreja Kitanov for their helpful suggestions and comments.

References

[1] Toni Petrinić: Implementation of navigation algorithms in mobile robot simulator for Matlab (in Croatian), diploma thesis No. 1451, University of Zagreb, 2005

[2] Matlab: Creating Graphical User Interfaces, The MathWorks Inc., 2005

[3] Richard T. Vaughan: Stage: a multiple robot simulator, Technical Report IRIS-00-394, University of Southern California, 2000

[4] Oliver Michel: Khepera Simulator Package version 2.0, http://diwww.epfl.ch/lami/team/michel/khep-sim/SIM2.tar.gz

[5] Jakub Hrabec: Autonomous mobile robotics toolbox SIMROBOT, http://www.uamt.feec.vutbr.cz/robotics/simulations/amrt/simrobot.zip

[6] Pioneer 2 Operations Manual, ActivMedia Robotics, LLC, 2000

[7] P. Muir: Modeling and Control of Wheeled Mobile Robots, doctoral dissertation, Technical Report CMU-RI-TR-88-20, Robotics Institute, Carnegie Mellon University, August, 1988

[8] Tom Ootjers: Line Drawing Algorithm Explained, http://www.gamedev.net/reference/articles/article1275.asp