Innovative Production Machines and Systems
3rd I*PROMS Virtual International Conference, 2-13 July 2007
*Edited by D.T. Pham, E.E. Eldukhri and A.J. Soroka,*
*Manufacturing Engineering Centre, Cardiff University, UK*

# A Matlab-based Simulator for Autonomous Mobile Robots

Jiali Shen and Huosheng Hu
Department of Computer Science, University of Essex
Wivenhoe Park, Colchester CO4 3SQ, United Kingdom

**Abstract**

Matlab is a powerful software development tool and can dramatically reduce the programming workload during the period of algorithm development and theory research. Unfortunately, most of commercial robot simulators do not support Matlab. This paper presents a Matlab-based simulator for algorithm development of 2D indoor robot navigation. It provides a simple user interface for constructing robot models and indoor environment models, including visual observations for the algorithms to be tested. Experimental results are presented to show the feasibility and performance of the proposed simulator.

**Keywords:** Mobile robot, Navigation, Simulator, Matlab

## 1. Introduction

Navigation is the essential ability that a mobile robot. During the development of new navigation algorithms, it is necessary to test them in simulated robots and environments before the testing on real robots and the real world. This is because (i) the prices of robots are expansive; (ii) the untested algorithm may damage the robot during the experiment; (iii) difficulties on the construction and alternation of system models under noise background; (iv) the transient state is difficult to track precisely; and (v) the measurements to the external beacons are hidden during the experiment, but this information is often helpful for debugging and updating the algorithms.

The software simulator could be a good solution for these problems. A good simulator could provide many different environments to help the researchers to find out problems in their algorithms in different kinds of mobile robots. In order to solve the problems listed above, this simulator is supposed to be able to monitor system states closely. It also should have flexible and friendly users' interface to develop all kinds of algorithms.

Up to now, many commercial simulators with good performance have been developed. For instance, MOBOTSIM is a 2D simulator for windows, which provides a graphic interface to build environments [1]. But it only supports limited robot models (differential driven robots with distance sensors only), and is unable to deal with on visual based algorithms. Bugworks is a very simple simulator providing drag-and-place interface [2]; but it provides very primitive functions and is more like a demonstration rather than a simulator. Some other robot simulators, such as Ropsim [3], ThreeDimSim [5], and RPG Kinematix [6], are not specially designed for the development of autonomous navigation algorithms of mobile robots and have very limited functions.

Among all the commercial simulators, Webot from Cyberbotics [4] and MRS from Microsoft are powerful and better performed simulators for mobile robot navigation. Both simulators, i.e. Webots and

MRS, provide powerful interfaces to build mobile robots and environments, excellent 3-D display, accurate performance simulation, and programming languages for robot control. Perhaps due to the powerful functions, they are difficult to use for a new user. For instance, it is quite a boring job to build an environment for visual utilities, which involves shapes building, materials selection, and illumination design. Moreover, some robot development kits have built-in simulator for some special kinds of robots. Aria from Activmedia has a 2-D indoor simulator for Pioneer mobile robots [8]. The simulator adopts feasible text files to configure the environment, but only support limited robot models.

However, the majority of commercial simulators are not currently supporting On the other hand, Matlab programming that provides a good support in matrix computing, image processing, fuzzy logic, neural network, etc., and can dramatically reduce the coding time in the research stage of new navigation algorithms. For example, a matrix inverse operation may needs a function which has hundreds of lines; but there is a simple command in Matlab. To use Matlab in this stage can avoid time-wasting on regenerating existed algorithms repeatedly and focus on the new theory and algorithm development.

This paper presents a Matlab-based simulator that is fully compatible with Matlab codes, and makes it possible for robotics researchers to debug their code and do experiments conveniently at the first stage of their research. The algorithms development is based on Matlab subroutines with appointed parameter variables, which are stored in a file to be accessed by the simulator. Using this simulator, we can build the environment, select parameters, build subroutines and display outputs on the screen. Data are recorded during the whole procedure; some basic analyses are also performed.

The rest of the paper is organized as follows. The software structure of the proposed simulator is explained in Section II. Section III describes the user interface of the proposed simulator. Some experimental results are given in Section IV to show the system performance. Finally, Section V presents a brief conclusion and potential future work.

## 2. Software architecture

To make algorithm design and debugging easier, our Matlab based simulator has been designed to have the following functions:

- Easy environment model-building; including walls, obstacles, beacons and visual scenes;
- Robot model building, including the driving and control system and noise level.
- Observation model setting; the simulator calculates the image frame that the robot can see, according to the precise robot pose, the parameters of camera, and the environment.
- Bumping reaction simulation. If the robot touches “walls”, the simulator can stop the robot even when it is commanded to move forward by other modules. This function prevents the robot passing through the “wall” like a ghost, and makes the simulation running like the experiment on real robots.
- Real-time display of the running processing and observations. This is for users to track the navigation procedure and find out the bugs.
- Statistical results of the whole running procedure, including the transient and average localization error. This is detailed navigation result for offline analysis. Some basic and simple analysis has been done in these modules.

The architecture shown in Fig. 1 has been developed to implement the functions above. The rest of this section will explain the modules of the simulator in details.

### *2.1. User Interface*

The simulator provides an interface to build the environment, set the noise model; and a few separate subroutines are available for users to implement observation and localization algorithms. Some parameters and settings are defined by users, the interface modules and files can obtain these definition. As shown in Fig. 1, the modules above the dashed line are the user interface. Using Customer Configure files, users can describe environments (the walls, corridors, doorways, the obstacles and the Beacons), explain system and control models, define noises in different steps and do some simulator settings.

The Customer Subroutines should be a serious of source codes with required input/output parameters. The simulator calls these subroutines and uses the results to control the mobile robot. The algorithms in the Customer Subroutines are therefore tested in the system defined by Customer Configure Files (CCFs) in the simulator. The grey blocks in Fig. 1 are the Customer Subroutines integrated in the simulator.

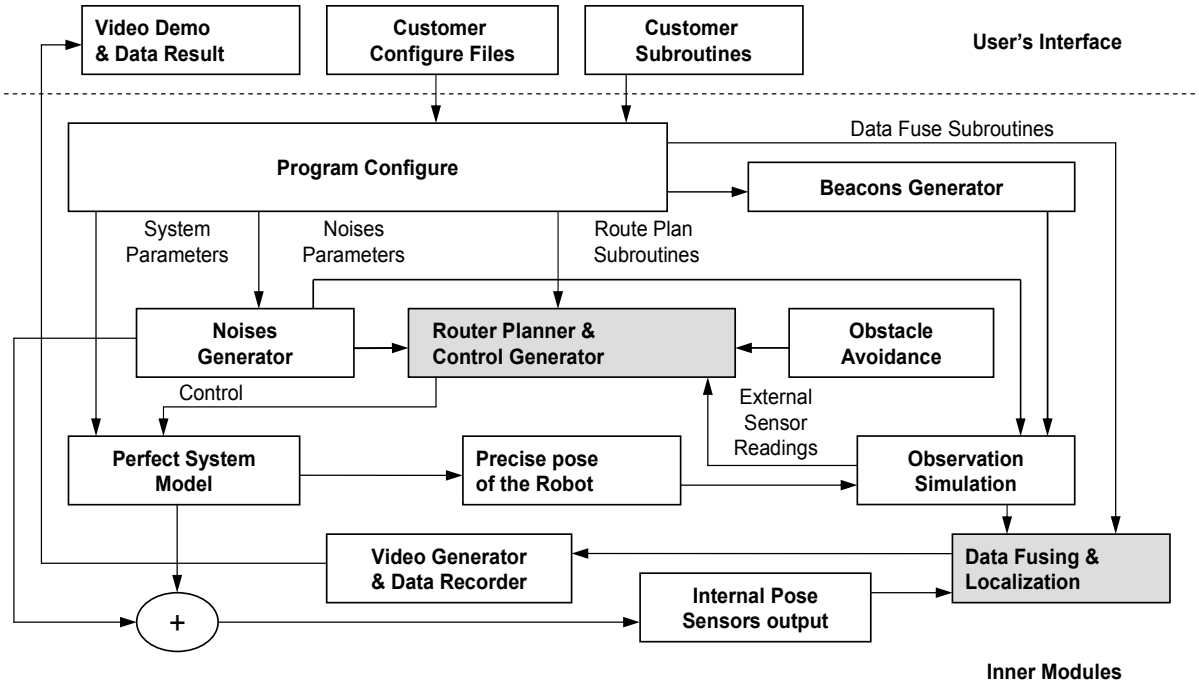



Fig. 1 Software structure of the simulator

The environment is described by a configure file in which the corners of walls are indicated by the Cartesian value pairs. Each pair defines a point in the environment and the Program Configure module connects points with straight lines in serials and regards these lines as the wall. Each beacon is defined by a four-element vector as $[x,y,\omega,P]^T$, where $(x,y)$ indict the beacon's position by Cartesian values, ω is the direction that the beacon faces to, and $P$ is a pointer refer to an image file reflect the venue views in front of a beacon. For a non-visual beacon, e.g. a reflective polar for a laser scanner, the element $P$ is evaluated, which is illegal for am image pointer.

Some parameters are evaluated in a CCF, such as the data of the robot (shape, radius, driving method, wheelbases, maximum translation and rotation speeds, noises, etc.), the observing character (maximum and minimum observing ranges, observing angles, observing noises, etc.) and so on. These data are used by inner modules to build system and observation models. The robot and environment drawn in the real time video also rely on these parameters.

The CCF also defines some setting related to the simulation running, e.g. the modes of robot motion and tracking display, the switches of observing display, the strategy of random motion, etc.

*2.2. Behaviour controlling modules*

A navigation algorithm normally consists of few modules such as obstacle avoidance, route planner and localization (and mapping if not given manually). Although obstacle avoidance module (OAM, safety module) is important, it is not discussed in this paper. The simulator provides a built-in OAM for users so that they can focus on their algorithms. But the simulator also allows users to switch off this function and build their own OAM in one of the customer subroutines. A bumping reaction function is also integrated in this module which is always turned on even the OAM has been switched off. Without this function, the robot could go through the wall like a ghost if the user switched off the OAM and the robot has some bugs in the program.

The OAM has the flowchart shown in Fig. 2. The robot pose is expressed as $X=[x,y,\theta]$, where $x$, $y$, and $\theta$ indicate the Cartesian coordinates and the orientation respectively. The $(x, y)$ pair is therefore adopted to calculate the distance to the "wall" line segments by using the basic theory of analytic geometry. The user's navigation algorithm is presented as the Matlab function to be tested, which is called by the OAM. It should output the driving information defined by the robot model, for example,

the left and right wheel speed for a differentially driving robot.

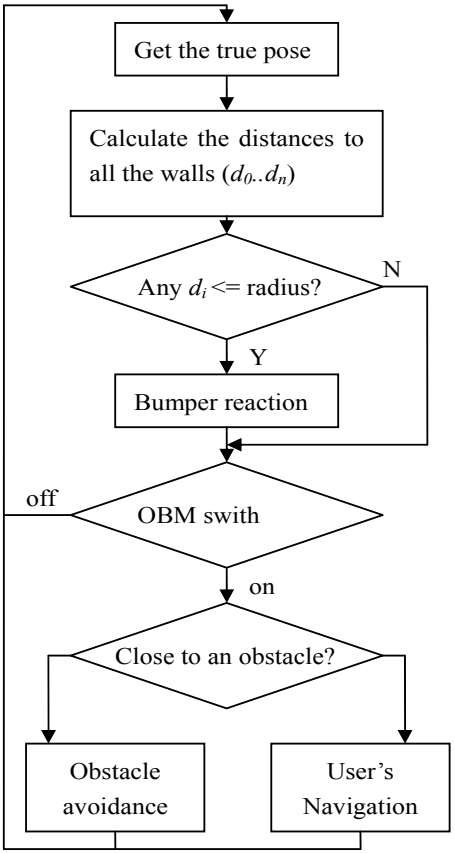



Fig. 2 Obstacle avoidance module

### *2.3. Data fusion subroutines*

The data fusion is another subroutine of the simulator, which is available to users. The simulator also provides all the information required and receives the output of this subroutine, such as, the localization result and the mapping data.

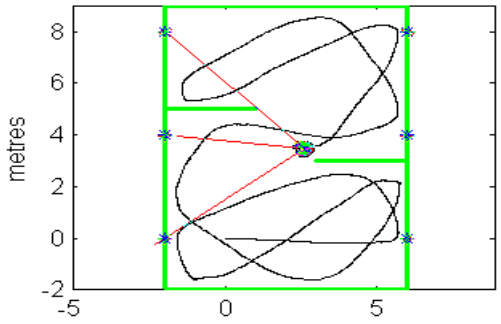
Normally, the robot acquires data using its onboard sensors, such as internal odometers, external sonars, CCD cameras, etc. In the simulator, these sensor data should be transferred to the subroutine as close to that of a real robot as possible. Thus the observation simulation module (OSM) is developed. The internal data includes the precise pose plus the noises generated with the parameters set by CCFs, which is easy to acquire.

According to the true robot pose and the arrangement of the beacons, it is easy to deduce the beacons that can be detected by the robot, as well as the distance and direction of the observation. The information of all observed non-visual beacon will be selected according to the CCFs and transferred to the data fuse subroutines. For visual based algorithms simulation, the CCFs of the environment contain the image files of the scenes at different places. Combined with the camera parameters defined in CCFs, the beacon orientation ω and the observing data such as distance and direction, the OSM can calculate and generate zoomed images to simulate the observations at a certain position.

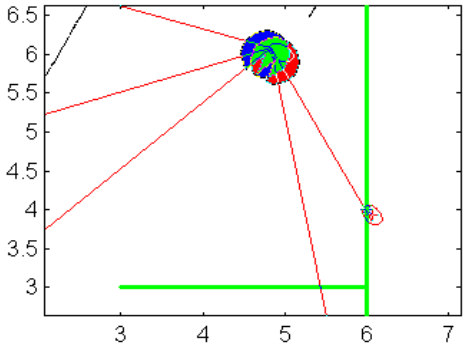
The user observation subroutines are therefore acquired the image just like that from an onboard camera in the real world.

### *2.4. Simulator output module*

The "Video Demo & Data Result" is the output module of the simulator. The real time video gives the direct view of how the algorithm performs, while the output data give the precise record during the simulation. Fig. 3(a) shows a frame of the real time video, i.e. the whole view, while Fig. 3(b) is the enlarged view of the middle part of Fig. 3(a).




(a) Whole view




(b) Enlarged part

Fig. 3 The view of the simulator

```
{Routine Refresh:    calculation the current drawing }
{all parameters} = getParameter(configure_file);
Rob=BuildRobot(robot_Para);
Wall = BuildWall(wall_Para);
Beacon = getBeacons(beacon_Para);
DrawRobot(Rob,[0,0,0],[0,0,0]);
DrawImage(Wall, Beacon);
loop for every 40 ms
        control = call(users_Control);
        tpose = getTruePose(control);
        obv = getObservation(tpose, beacons);
        [lpose,l_noise] = call(users_Localization);
        [map,m_noise] = call(users_mapbuilding);
        DrawRobot(Rob,tpose,lpose);
        DrawImage(l_noise, obv, map, m_noise);
end loop
```

Fig. 4 The output video

The wide straight lines denote the walls of the environment; the round on the left in Fig. 3(b) is the real position of the robot and the one on the right is the localization result. The thin straight lines are the feature observation at the certain moment, and the ellipses with crosses at the centres express the uncertainties of mapping. The ellipse around the centre of the localization result means the uncertainty of the localization. The source code about the plotting is based on Bailey's open source [7]. It should be noticed that the output data contains the estimated pose, true pose, covariance matrixes of each step, which can be processed and evaluated precisely after the experiment.

The Video is actually implemented by quick update of a serial of static images. In every 40 milliseconds, the simulator calculate all the state parameters, such as the true pose as the localization result of the robot, the current observations and the current mapping result. The simulator draws the image for the current frame with these data and refreshes the output image. Since the image is refreshed 25 times per second, it looks like a real video. The calculation and drawing of current frame is implemented with the method shown in Fig. 4

In each loop cycle, the DrawRobot function translates and rotates the shape stored in the vector *Rob*, according to the true pose and localization results respectively, and draws the results with different fill shadows or colours. During the processing cycles in Fig. 4, all data and parameters, e.g. the *lpose, t_pose, map,* etc, are recorded by another thread in a file. After the navigation, these data will be output as well as some basic statistic results.

## 3. Experimental result

The purpose of the experiment is to test the performance of the simulator. Therefore, the experiment is designed to test the functional module of the simulator separately and then run a real SLAM algorithm in the simulator to test the overall performance.

First of all, the OAM is switched off, and the user's navigation module can only provide a constant speed on both wheels. In other words, the robot can only move forward. During the experiment, when the robot bumped into the wall in its front, it stopped and wriggled at the place, because of the driving noise. The bumping reaction module works as designed, and makes the robot stops when bumping into any object in the environment.

Secondly, we remove all the beacons in the environment and the robot is running 100% on the internal sensors. By analysing the data and observations, the real time video and the noise generation module works perfectly and provides the result as we expected.

Thirdly, the OAM is switched on, and the user's navigation module keeps the same. That is to say, the robot is moving forward unless the built-in avoidance module takes the control to avoid obstacles nearby. During the experiment, the robot keeps moving for more than 10 minutes, and the route covers every corner of the environment. It avoids all the obstacles reliably. The path shown in Fig. 3(a) also clearly proved the performance of the obstacle avoidance module. Then, the observation simulation module is tested by off-line processing of the recorded images, which is generated during navigation. By using the triangulation method [9], the estimated position of each recorded image is deduced and compared with the true position. The error is acceptable, considering the uncertainties of the triangulation. That means the image zoom and projective operation in this module

is reliable.

Finally, a simple simultaneously localization and mapping (SLAM) algorithm presented in [10] is running in the developed simulator. All the function modules are evolved in this step. The simulator gives all the running results as designed and these results fit well to the result on a real robot given by the reference. The transient localization error result of the simulator is shown in Fig. 5.

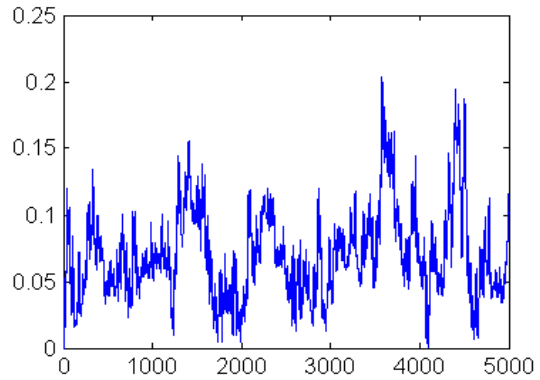



Fig. 5: Transient error output

## 4. Conclusion and future work

This paper presents a novel simulator that is based on Matlab codes, and allows users to debug their navigation algorithms with Matlab, build an indoor environment, set the observation models with noises, and build the robot models with different driving mechanism, internal sensors and external sensors. The visual observation is also calculated by some projective and zoom calculation based on the built environment view. This function is important for the experiment of visual based algorithms. To save the time, the simulator also provides some functional modules which can implement some navigation tasks such as obstacle avoidance and bumping reactions. All the above functions of the simulator have been tested by designed experiments, which show that the simulator is feasible, useful and accurate for 2D indoor robotic navigation.

The current version needs further improvement in the next stage since (i) this simulator cannot implement 3-D experiments; (ii) the input interface is text file based, which is easy to be used by experts but difficult to be used by new users. A graphic drag-and-set interface is needed; (iii) in some environments with complex visual views, the observation simulation is computationally expensive and makes simulation very slow; (iv) the scene of the onboard camera is not displayed in real time; and (v) only one robot is supported in this version.

## References


[1]. Mobotsim,http://www.mobotsoft.com/mobotsim.htm, accessed on 21 February 2007

[2]. Bugworks, http://www.cogs.susx.ac.uk/users/christ/bugworks/, accessed on 21 February 2007

[3]. Ropsimhttp://camelot.interlogic.com.ua/Ropsimintro.aspx

[4]. Webots, Commercial Mobile Robot Simulation Software http://www.cyberbotics.com

[5]. ThreeDimSim,http://www.havingasoftware.nl/index.htm

[6]. RPG Kinematix, http://www.robotics.utexas.edu/rrg/downloads/software/rrgkmax4.0/

[7]. T. Bailey , KFSLAM_SIM, http://www.acfr.usyd.edu.au/homepages/academic/tbailey/software/slam_sims/ekfslam_v1.0.zip

[8]. Mobile Robots, Aria, ActivMedia, USA; http://www.activrobots.com/SOFTWARE/aria.html,

[9]. J. Shen and H. Hu, "Mobile Robot Navigation through Digital Landmarks", *Proceedings of the 10th Chinese Automation and Computing Society Conference in the UK*, Liverpool, England, 18 September, 2004, pages 117-124

[10]. T. Bailey, J. Nieto, J. Guivant, M. Stevens, E. Nebot, "Consistency of the EKF-SLAM Algorithm", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2006, Beijing, China, pages 3562-3568