A framework for simulations and tests of mobile robotics tasks

Emanuele Frontoni*, Adriano Mancini, Fabio Caponetti, Primo Zingaretti

Dipartimento di Ingegneria Informatica, Gestionale e dell'Automazione (DIIGA)
Università Politecnica delle Marche, Ancona, Italy

***Abstract* – This paper presents an education framework, developed in Matlab, for studying and experimenting typical mobile robotics tasks such as obstacle avoidance, localization, navigation and SLAM. The most important characteristic of this framework is the ability to easily switch from a simulator to a real robot to tune and test algorithms and to evaluate results in simulated and real environments. The framework is being used with interesting results in robotic courses at the Università Politecnica delle Marche in Ancona, Italy. In the second part of the paper a test case to evaluate an optimization of a Monte Carlo Localization process with sonar sensors is presented.**

Index Terms* – *Robotics education, mobile robotics, localization, particle filtering.

I. Introduction

In robotics research we usually need a way to easily simulate algorithms, tune some parameters and, finally, perform a test of the tuned algorithm in a real system. This purpose is also useful in robotics education with the difference that in robotics research the goal is to develop and test novel approaches to certain robotics tasks, while in robotics education the goal is to understand and test well know algorithms or systems.

The main goal of this work is to present a mobile robotics framework targeted toward research and education needs. The framework was developed with the following characteristics: to switch easily between a robot simulator and a real robot; the use of Matlab as high level programming platform; the use of C/C++ SDK for robot control and time consuming/multithread processes; the reuse of existing solutions and algorithms implemented for different tasks in mobile robotics; the use of the same framework also for multirobot cooperation and control.

Matlab is an environment for linear algebra and graphical presentation that is available on a very wide range of computer platforms with many built-in functions available in various fields (i.e., Matlab toolboxes). In the last years Matlab has been used more and more frequently in the field of technical education, in particular owing to the very short time developing time needed to perform a simulation. The same reasons justify the use of Matlab for educational purposes in robotics and automation [2, 3, 4].

The proposed framework, developed using the Matlab environment and its powerful graphical functions, provides a very convenient approach to robotic simulations, tests and experimental analysis. Being open source it allows users to investigate framework's functionalities and to modify some of the modules. A mobile robot simulator is integrated in the framework with the purpose of quickly validating different approaches to solve mobile robotics tasks or to tune parameters. Low level control algorithms and interfaces with hardware are developed in C++ to allow real time control and to solve time consuming tasks. So, the framework is made by a low level control for real time applications and a supervisor implemented in Matlab.

In the second part of the paper an example of how to use the framework to implement, tune and test an efficient Monte Carlo Localization (MCL) algorithm [7] is presented. The MCL algorithm is applied to the problem of mobile robot localization using low cost sensors like sonar sensors. Robot localization is essential for a broad range of mobile robot tasks. The aim of localization is to estimate the position of a robot in its environment, given local sensorial data [5, 6].

Localization routines have been implemented by interfacing Matlab with ARIA (ActivMedia Robotics Interface Application), an open source collection of C++ classes able to ensure a high level access to Active Media Robotics' robots, in particular. The MCL localization algorithm was tuned and tested using the proposed framework firstly in simulation and finally on a real robot. Good global localization and tracking performances were obtained even using low cost sensors.

The structure of the paper is the following: next section presents the framework and its architecture; the MCL test case is presented in section III after a brief introduction to particle filters; finally, results for the test case are reported in section IV before conclusions.

* Contact author: Emanuele Frontoni
mail: frontoni@diiga.univpm.it
fax: +39(0)712204474

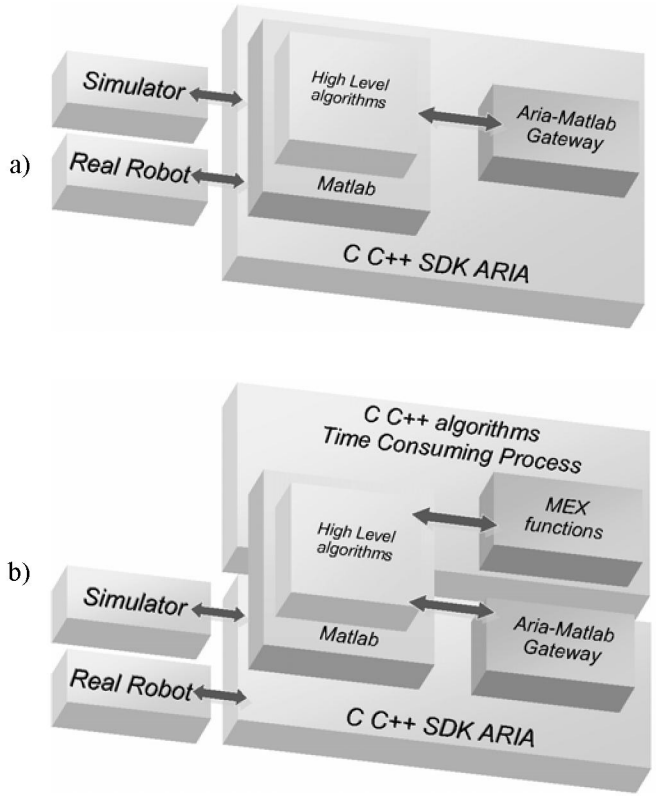

Fig. 1. Software architectures of the frameworks: a) layers of the basic framework; b) general architecture to solve real time and multi thread tasks.

II. The Mobile Robotics Framework

The framework was developed in the Matlab environment using a software layer to interface high level algorithms with low level controls and hardware interactions.
The framework incorporates two software architectures to be more flexible to user requests. Fig. 1a) reports the architecture used in the test case described in this paper, while Fig. 1b) is a modified architecture that is functional to the development of time consuming high level algorithms in C++ (i.e. image processing), the reuse of C++ code for high level robotics tasks and the execution of algorithms in real time or multithread (Matlab 7 is single thread) mode.
The two major features of the implemented framework are the integration of different software platforms (SDKs) and/or programming languages and, above all, the development and the integration of a mobile robot simulator.
An ARIA–Matlab gateway has been developed in C++ to interface the two platforms. The gateway is a DLL (Dynamic Link Library) using the Matlab External API (MEX functions). In addition, a class, named *robot,* has been created to call methods contained in the gateway to manage the real or simulated robot by different types of connections (Matlab–Simulator, Matlab–Robot using serial or TCP/IP communication).
More details about the mobile robot simulator are given in the last paragraph of this section.

A. ARIA Library

ARIA is an object oriented software for the management of mobile robots like Pioneer 3 or AmigoBot. This open source software, written in C++, allows the user to interact with the robot at a low level in an efficient, fast and modular way.
A set of classes, are available for the connection to a real or virtual robot. In particular, the *ArSerialConnection* class implements the management of a serial communication between a robot and a PC, the *ArTcpConnection* class allows the use of two different types of TCP connections: the Wi-Fi connection using the real IP address of the robot or the more interesting simulator connection using the loop back address 127.0.0.1. The *ArSimpleConnector* class implements all types of connections between a robot and a PC. The most interesting class is *ArRobot*, which is a real robot's abstraction with many methods and properties for the programmer, from motor speed control to sonar and laser readings. *ArRobot* is the heart of ARIA, acting as a client-server communication gateway for collecting and distributing many robot tasks and utilities, including packet synchronization, range-finding sensors and robot action classes.

B. MEX functions

MEX functions are used to set up the interaction between Matlab and ARIA. They allow calling C/C++ or Fortran subroutines from Matlab as built-in functions. Matlab callable C/C++ and Fortran programs are referred to as MEX-files. MEX-files are dynamically linked subroutines that the Matlab interpreter can automatically load and execute.
MEX-files have several applications:

i) large pre-existing C/C++ and Fortran programs can be called from Matlab without having to be rewritten as M-files (Matlab source files);
ii) bottleneck computations (usually for-loops) that do not run fast enough in Matlab can be recoded in C/C++ or Fortran for efficiency.
In general, most programming should be done in Matlab and MEX facility should be used only when the application requires it (bottlenecks of the developed algorithm can be found using a profiler).
MEX functions have the following prototype:

```
void mexFunction(int nlhs, mxArray *plhs[],
        int nrhs, const mxArray *prhs[]);
```

Matlab and C/C++ interact and exchange data using the parameters: *nlhs, *plhs, nrhs, *nrhs*. In addition, several special functions for type casting provided by Matlab (i.e., *mxGetChar, mxGetString*, etc.) are used to ensure data coherence.

C. Mobile Robot Simulator

A robot simulator has been integrated in the framework. It implements probabilistic models to simulate odometry errors and noisy sensor readings and provides a graphical interface to represent the environment where robots can be put. In fact, the development of localization algorithms that use range finder sonar sensors starts from the assumption that the map of the environment is known. The description of the environment is stored in a world file (.wld) following the Activ Media Robotics specifications; typical mobile robots simulation software as SRISim and MobileSim [8] also work with this kind of file.
Therefore, the simulator can use the same map data described before. The same functions used for ARIA are used to get information about robots coordinates and sensor data and to set up speed or movement. The simulator, differently from other robot simulators, is implemented in Matlab. This allows easily modifying sensors models or creating news models.

D. Framework basic functions

The framework has several functions to control robot movements and to get sensors readings. It can be also easily extended by using Matlab or C/C++ layers. Other useful ARIA functions can also be mapped using MEX files. Tab.1 reports a list of some basic functions we have developed.
An example of how using these functions to perform a simple robot movement is described below. First the robot is connected to the simulator, then a loop performs some basic movements and, finally, the robot is disconnected resetting the hardware.

```
% Connection to the simulator on localhost
r = robot('127.0.0.1');
for i=1:10
  % Move 0.5 meters forward
  move(r, 500);
  % Wait until the robot movement is completed
  while ismovedone(r)==0   end
  % Rotate 180 degrees clock-wise
  setdeltaheading(r, 180);
  % Wait until robot rotation of 180 degrees is
  % completed
  while isheadingdone(r)==0
  end
end
disconnect(r);
```

Tab. 1. Basic functions of the proposed framework.

Function call	Description
r = robot (address);	Connection of robot "r"
move (r, distance);	Forward motion of r for a distance
setheading (r, heading);	Robot rotation
ismovedone (r); isheadingdone (r);	Check of movement ending
data = readsonar (r);	Sonar reading (data is a vector)

As shown by this example the framework provides real high level programming functions. Please note that the example can easily be tested in a real robot only changing the IP address from localhost to the real IP address or to the COM port of the robot.
In conclusion the choice of a high level programming language is the real advantage of this framework: using Matlab the user can easily test and tune his robotics algorithms, modify the simulated robot model, easily interface sensors (including vision sensors), use a huge range of data visualization tools and functions.

III. Test case: Monte Carlo Localization using low cost sensors

In this section we describe a Monte Carlo Localization algorithm using low cost sensors. It represents a test case about the use of the proposed framework for educational purposes. In particular, this localization exercise is part of the work for the course "Tools and Methods for Simulation", in the Automation Engineering master level course at the Università Politecnica delle Marche in Ancona, Italy.
The main goal of this test case is to propose and tune the *particle weight generation* (starting from the sensor model) and the *re-sampling* processes in the MCL algorithm to find an optimal solution to global localization and position tracking problems. The tuning was performed using the simulator, while final results are obtained testing the whole localization process using a real robot.

A. Monte Carlo Localization algorithm

In this paragraph a short introduction to the probabilistic localization problem and to the Monte Carlo Localization algorithm are given.
To achieve the localization goal using only low cost sensors, like sonar sensors, we need a map and a set of sensorial measurements taken by the robot from its current position. These observations are noisy and potentially perceptually aliased (that is, different positions over the map can generate the same or very similar observations).

The variable that represents the robot's position over the map is x_k, where k denotes the time. If observations are represented by the vector z_k , denoting all sensorial readings at time k, and the movement done by robot is represented by the vector u_k, the goal of this approach is to find a posterior representation of the probability density function $P(x_k \mid z_k, u_k)$ able to describe the position of the robot over the map, ,given movements and observations at time k.
Monte Carlo is a particular particle filtering algorithm. The main goal of this kind of algorithm is to track a variable of interest, typically a not Gaussian and multi modal probability density function, as it evolves during time. This

method is based on the discrete representation of the probability density function of interest; every sample is called particle. To track the time evolution of the probability density function, every particle is updated according to system dynamics. At every step some observations are available and they act as constraints for the future evolution of the particle population. A weight proportional to the probability of representing the true variable is associated to each particle. This weight is a function of the sensor model. At every time step the tracked variable can be estimated as the weighted mean of the particle population.
In our particular case the variable of interest is the position of the robot; every particle represents a possible position over the map; a weight proportional to the probability to represent the true robot position is associated to each particle; the particle weight is function of observations (i.e., sonar readings in the real robot example).
Particle filters algorithms are divided in two phases: prediction and update. After every action taken by the robot, a model of this movement is applied to every particle. Each weight is updated on the basis of the observation taken by the robot on the new position. Through the re-sampling phase particles with lowest weights are eliminated to obtain a new population able to represent better the robot position.
In our problem the state variable (robot position and orientation) is described by the vector x_k, which is represented at every instant k by a set of *nParticles* samples $\left\{ S_k^j = \left[x_k^j, w_k^j \right] : j = 1...nParticles \right\}$, where w_k^j denotes the weight associated to particle j at time k.
The algorithm structure can be outlined as follows:

Initialization, k=0

$$x_0^i = [xMax \cdot rand; yMax \cdot rand; 2 \cdot \pi \cdot rand]$$
$$w_0^i = \frac{1}{nParticles}, i = 1..nParticles$$

While (Exploring)

Prediction

$$x_{k+1}^i = f(x_k^i, u), i = 1...nParticles$$
$$\tilde{w}_{k+1}^i = f(x_{k+1}^i, z_{k+1}), i = 1...nParticles$$
$$w_{k+1}^i = \frac{\tilde{w}_{k+1}^i}{\sum_{j=1}^{nParticles} \tilde{w}_{k+1}^j}, i = 1...nParticles$$

Re-sampling

$$x_{k+1} = Resample(w_{k+1})$$

Position estimation

$$\bar{x}_{k+1} = \sum_{j=1}^{nParticles} w_{k+1}^j \cdot x_{k+1}$$

$$k = k + 1$$

B. Weight generation

The generation of the weight that will be associated to each particle is a crucial aspect of the localization algorithm as the robot position depends directly from particle weights.
Each particle weight is assigned according to the difference between the actual sensor reading of the robot and the sensor reading that a robot would obtain from the position of the particle (evaluated trough the sensor model [1]).
An innovation term can be computed for each particle at every time step k as:

$$Innov_k^j = \left| \frac{RealSonarMeasure_k - SimulatedSonarMeasure_k^j}{RealSonarMeasure_k} \right|, j = 1...nParticles$$

Really, being, by hypothesis, the number of the available sensors more than one, the innovation term is a vector of differences and its number of columns is given from the number of sensors available.
To obtain a scalar representative of the difference between real and simulated observations we adopt the following expression:

$$difference_k^j = \frac{1}{nSensors} \cdot \sum_{i=1}^{nSensors} Innov(i) \quad , j = 1...nParticles$$

The weight associated with a single particle is then obtained from the evaluation of the following expression:

$$\tilde{w}_k^j = e^{-difference_k^j} + 1/nParticles, j = 1...nParticles$$

In the weight evaluation a quantity of *1/nParticles* is added with the goal that any particle has a starting weight that can be approximated to 0.
All these parameters have been evaluated and tuned using the simulator.

C. Re-sampling strategy

The operation of sampling consists in the generation of a new set of particles (population) from an existing one. The new population elements are selected from the old one on the basis of their characteristics. The selection of particles that will compose the new generation happens in a probabilistic way: each particle has a probability equal to its weight of being present n times in the new population.
The approach to re-sampling we adopted can be divided in three steps, each characterized by a specific operation taken over the population.
The first step is known in literature as "*selection with replacement*". Each particle has a probability, proportional to its weight, to be regenerated a casual number of times into the new population. In this way the elements with a high weight will be repeated a higher number of times than those with a low weight. During the application of this re-

sampling technique the number of particles remains constant, but the new population should represent in a better way the true position of the robot.
In the second step the population is resized (i.e. the number of particles decrease at every step), so that the computational load can be optimized to the localization process. We introduce an index to describe the quality of the localization process. This index represents the dispersion of particles on the map area.
The last step of the re-sampling process operates a control on the position of each particle to minimize the influence of any element of the population whose position will not be in the admissible area of the map. In the case that a particle has coordinates external to the admissible map region, it will be eliminated and then regenerated with a small weight in the map.
The whole re-sampling algorithm can be outlined as follows:

Step one : *resampling based on the particles weights*

$$x_{k+1} = SelectWithReplacement\left(x_k\right)$$

Step two : *resize of the population*

$$\sigma_{xP} = \left[\sigma_x, \sigma_y, \sigma_\theta\right]^T : \text{standard deviation of particle population}$$

$$Ip = \frac{\sqrt{\sigma_x^2 + \sigma_y^2}}{\sqrt{xMap^2 + yMap^2}}$$

where :

xMap and yMap represent the dimension of the smallest rectagle who can contain whole map

$$nParticles = int(nParticles \cdot e^{(-\lambda \cdot Ip)}) \qquad ,\lambda : parameter$$

$$x_{k+1} = ResizeParticlePopulation(x_{k+1}, nParticles)$$

Step three : *Check of the position of each particle*

$$x_{k+1} = CheckParticles\left(x_{k+1}, Map\right)$$

Also, all free parameters (*nparticles,* λ) have been evaluated and tuned using the simulator.

IV. MCL Results

The proposed framework was tested in the tuning of the particle weight generation and the re-sampling processes of a MCL algorithm in two different simulations: global localization and robot position tracking.
The whole MCL algorithm was implemented in M-file and used all the functionalities provided by the proposed framework.
Final results were obtained using an Active Media Pioneer3 DX robot, a differential drive robot with a high degree of mobility and the capability to climb over small obstacles. This robot is characterized by the following technical specifications: eight sonar sensors situated on the front part and characterized by a maximum range of 5 m and a visibility angle of 30°; two incremental encoders; two wheels controlled by independent motors; serial connection RS232.
The environment used for tests is a long corridor affected by high aliasing: two different positions in the map cause similar measures and, as a consequence, there is a high probability that the localization process could fail. This environment was chosen to validate the algorithm in critical situations and to better analyze its performances.

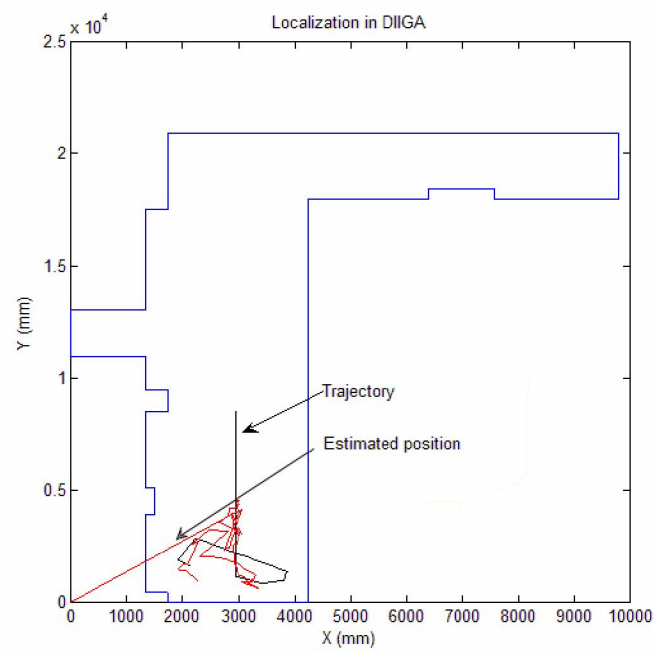

Fig. 2. Estimated position and real trajectory of the robot during a global localization and tracking test.

As said before, results were obtained both for the global localization process and the position tracking process. Using the Matlab simulator, the described algorithm worked efficaciously in both test processes, resulting in good robot position estimations. Fig. 2 shows the real and evaluated trajectories of the robot during a global localization and a robot position tracking test.
The number of particles was fixed to 300; a maximum number of 75 steps were defined to limit test duration, and after this threshold the localization procedure is considered to fail. Results showing localization accuracy and number of used particles are reported in Fig. 3 and 4. The localization process is successful starting from the 25th step. After this step the position tracking starts until the end of the experiment.
Correlated to the localization and position tracking is the number of particles; when the localization happens, the computational load decreases quickly. During the position tracking the mean error is about 380mm, while the standard deviation is 180mm.

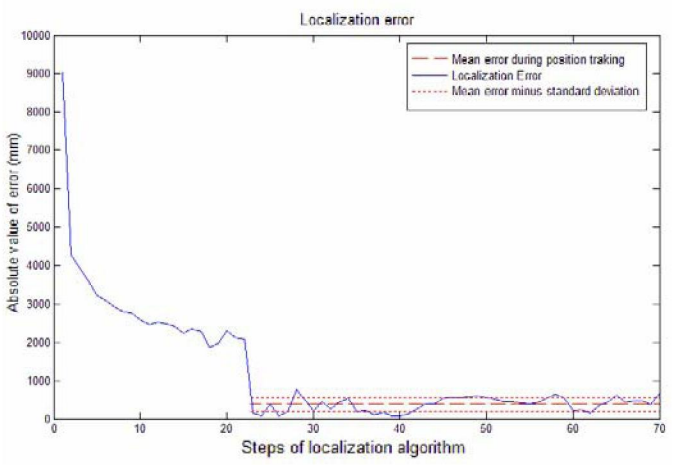

Fig. 3. Localization errors of a MCL test: dark line is the global error and the dashed line is the mean error during the position tracking.

Using the profiler embedded in Matlab to evaluate time performances of every function, we obtained that the most time-consuming process was the function *SonarModel* used to apply the sensor model to all particles of the population. The implementation was tested on an Intel Celeron 800 MHz laptop with 256 MB of RAM. The whole algorithm, starting with 300 particles for the MCL, takes about 5 seconds per step using the real robot and more than 8 seconds per step using the simulator (simulator time included). Time performances increase during the localization process due to the re-sampling results (Fig. 4).

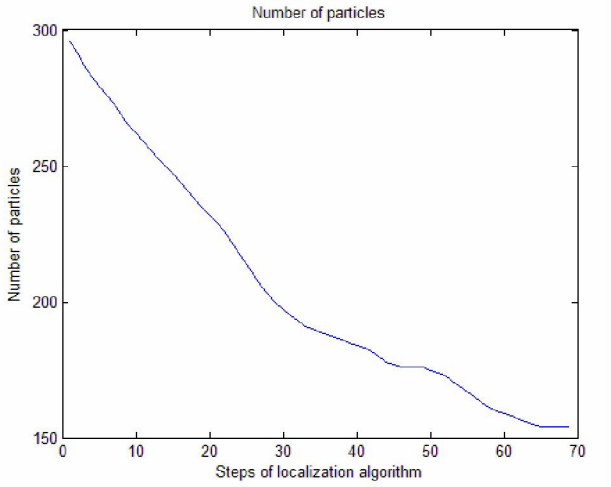

Fig. 4. Number of particles used during the global localization and the tracking processes.

V. Conclusions and Future Works

In robotics research and education we usually need a way to easily simulate algorithms, tune some parameters and, finally, perform tests of the tuned algorithm in a real system. The main goal of this work was the presentation of a mobile robotics framework targeted toward research and education needs. The framework was used with interesting results in robotic courses to make students able to solve mobile robotics tasks using the simulator and then testing the proposed solution with a real robot. A test case using a Monte Carlo Localization process using only sonar sensors was also presented. Final results show good performances of the proposed approach to the robot localization problem and, consequently, of the usefulness of the framework.

In conclusion, the whole framework allows to easily and quickly developing all typical mobile robotics tasks and, in particular, it is useful for parameter tuning and fast evaluation and choice of different approaches.

Future developments of the educational framework will integrate a method to simulate vision based mobile robotics tasks using datasets of real images and their position in a map.

Further improvements could rely on the development in C++ of particular time consuming parts (for example, implementing the sensor model using the optimized architecture proposed in Fig. 1b).

References

[1] B. Billur, K. Roman, Differentiating sonar reflections from corners and planes by employing an intelligent sensor, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol 12(6), 560-569, 1990.

[2] P. I. Corke, A robotics toolbox for Matlab, IEEE Robot. Automat. Mag., Vol. 3(1), 24-32, 1996.

[3] W.E. Dixon, D.M. Dawson, B.T. Costic, M.S. de Queiroz; A Matlab-based control systems laboratory experience for undergraduate students: toward standardization and shared resources, IEEE Trans. on Education, Vol 45(3), 218-226, 2002.

[4] S.L. Eddins; M.T. Orchard; Using Matlab and C in an image processing lab course, IEEE Int. Conf. on Image Processing ICIP-94, Vol 1, 515-519, 1994.

[5] E. Frontoni, P. Zingaretti, A vision based algorithm for active robot localization, Proc. of IEEE Int. Symposium on Computational Intelligence in Robotics and Automation, Helsinki, 347-352, 2005.

[6] E. Menegatti, M. Zoccarato, E. Pagello and H. Ishiguro, Image-based Monte-Carlo localization with omnidirectional images, Robotics and Autonomous Systems, Elsevier, Vol 48(1), 17-30, 2004.

[7] S. Thrun, D. Fox, W. Burgard, F. Dellaert, Robust Monte Carlo localization for mobile robots, Journal of Artificial Intelligence, 128 (1-2), 99-141, 2001.

[8] http://robots.mobilerobots.com/MobileSim/ (accessed 28 April 2006).