MiMicS: a Multi-robot simulator for teaching, rapid prototyping and large scale evaluations

Eduardo Ferrera
University of Duisburg-Essen
Essen, Germany 45127
eduardo.ferrera@uni-due.de

Jesús Capitán Fernández
University of Seville
Sevilla, Spain 41092
jcapitan@us.es

Merlin Stampa
University of Duisburg-Essen
Essen, Germany 45127
merlin.stampa@uni-due.de

Pedro José Marrón
University of Duisburg-Essen
Essen, Germany 45127
pjmarron@uni-due.de

ABSTRACT

This paper presents MiMicS, a novel multi-robot simulator for teaching, rapid algorithm prototyping and large scale evaluation. MiMicS works over MatLab, inheriting its portability and the smooth learning curve of its language, making it specifically suitable for teaching purposes. Contrary to many other solutions, this simulator provides easy-to-use methods for defining new scenarios through MatLab scripts. Thus, extensive batteries of simulations (e. g., Monte Carlo simulations) can be easily created and tested on the simulator. The system also provides several control mechanisms to easily manage and run large simulation batteries, while handling and recording failures of the algorithms tested on it. MiMicS provides a pseudo-realistic interface that includes robot models with second-order dynamics and noisy sensors. Finally, MiMicS incorporates a simple but efficient multi-thread and timing control mechanism. This reduces the need for using highly powerful computers to run long-term executions, alleviating problems caused by busy CPUs that can jeopardize the results of a simulation. The paper reviews the current state of the art on multi-robot simulators and introduces the architecture and main features of MiMicS. Then, it presents an evaluation of the running times and error handling of MiMicS, when exposed to large numbers of robots. To conclude, a comparison against the well-known Stage simulator in two benchmarking scenario is presented.

CCS CONCEPTS

•**Software and its engineering** → **Application specific development environments;** •**Computer systems organization** → *Robotic autonomy;* •**Computing methodologies** → Simulation environments;

KEYWORDS

Multi-robot simulators, second-order dynamics, Monte Carlo simulations

SAC'18, Pau,France

DOI: xx.xxx/xxx_x

ACM Reference format:
Eduardo Ferrera, Jesús Capitán Fernández, Merlin Stampa, and Pedro José Marrón. 2018. MiMicS: a Multi-robot simulator for teaching, rapid prototyping and large scale evaluations. In *Proceedings of ACM SAC Conference, Pau,France, April 9-13, 2018 (SAC'18),* 8 pages.
DOI: xx.xxx/xxx_x

1 INTRODUCTION

Nowadays, the use of simulators has become an essential part of the loop to design robotic algorithms. In the earliest state, when the algorithm is only a concept discussed on a blackboard, many researchers initiate the process of creation by selecting a simulator to work with. The selected platform will allow them to identify all required inputs and outputs that the algorithm will use; often forcing a review of the initial concepts and assumptions. This saves the researchers time, since the simulator provides a quicker and safer setup. The developer does not have to care about charging or repairing the real robot, placing it at its initial position or preparing communication channels. Moreover, simulators also allows students (or researchers with low resources) to test their algorithms without risking expensive platforms.

Even when real robots are available, the use of a simulator as a safe testing environment can be beneficial. In fact, algorithms such as collision avoidance are particularly sensitive to safety issues, mainly in their early stages. A non-detected failure could cause collisions, resulting in expensive material damages or – in the worst cases – risk of injury. Therefore, performing experiments with real robots without previous extensive simulations is not an option in many applications. With a simulator, tricky configurations can be repeatedly tested and analyzed in order to identify issues within an algorithm's concept or its implementation. If the algorithm proves its validity in simulation, the developers can decide to move on into the challenging real world, or to test it further with more realistic constraints.

The advantages of the multi-robot paradigm have turned it into a clear trend, leading many developers to work on simulators for that field. In this context, simulation earns even more relevance: charging and preparing several robots becomes extremely time expensive, repeating system failures is more complex, and the communication setup can significantly delay even simple tests. This may even preclude some researchers from working with multi-robot teams.

This paper presents a new **M**atlab based **M**ulti-robot **S**imulator called MiMicS[1]. In contrast to other existing solutions, MiMicS is developed to speed up the early stages of the design process for algorithms. Therefore, the simulator is prepared to describe multi-robot scenarios easily, and to test algorithms in pseudo-realistic environments in a systematic and intensive manner; allowing users to evaluate their algorithms by means of replicative and challenging simulations. Moreover, the simulator considers second-order dynamics and noisy sensors, approaching the simulations to the real world behavior.

The paper is organized as follows: Section 2 introduces alternative state-of-the-art simulators; Section 3 details the architecture of MiMicS; in Section 4 a study of the performance of the simulator is presented; Section 5 provides a comparison against the well-known simulator Stage; and Section 6 summarizes the conclusions.

2 RELATED WORK

Due to their advantages, there exist many simulators for robotics [8, 15]. However, the real world can be quite complex, so simulators are usually specialized. For instance, specific simulators for aerial [17] or underwater robots [6] behave better than combined solutions. Regarding multi-robot simulators, the variety is not so high [5]. The existing solutions need to find a precise balance between the amount of robots that they can support and the degree of realism achievable.

In 2000, the project TeamBots[2] [10] released its latest version. This bidimensional simulator served as an inspiration for many future works. Written in Java, the execution speed was compensated with its portability. The documentation available is not extensive and it is no longer widely used by the community.

The educational project Webots [3] [3, 9] integrates dynamics with the Open Dynamics Engine (ODE). Thus, it is able to simulate 2D kinematics with body collision detection. One of its main disadvantages is that it is not free of charge. The cheapest option is to acquire the Webots-MOD version: a model-based version where the simulator and the model of the robot are purchased separately.

The Urban Search and Rescue Simulator (USARSim) [4][2] builds the model of the robots and the physical world over the well-known *Unreal Engine* produced by Epic Games. This is a proprietary 3D video game engine that accepts an interface called Gamebots and is able to load the necessary models to simulate dynamic robots.

Similar to USARSim, the Delta3D[5] [12] is another open-source gaming engine. Some multi-robot systems have been simulated with Delta3D, but this engine is mainly focused on military applications and open-source games.

The Microsoft Robotics Developer Studio 4 (MRDS)[6] [4] is a Windows-based solution to model and code robotics applications. This initially successful simulator was released as a free tool in 2010 for academics and commercial purposes. However, due to a restructuring plan in Microsoft, its last version was released in 2012 and the project was canceled in 2014.

Due to the versatility of the Robotic Operating System (ROS) [13], some multi-robot simulators provide interfaces to be used through ROS. In this sense, USARSim [1], V-REP[7] [14] and Gazebo[8] [11] are relevant 3D simulators that make use of several physics engines, such as ODE. Moreover, the Stage simulator[9] [7] is another tool that was adopted by the ROS community as one of its main simulators. It can be used to simulate the behavior of crowded and bidimensional multi-robots environments, and its behavior has been deeply tested over benchmarking scenarios [16].

Many of the above simulators offer very realistic models for the physics of the robots and the environment, as well as good sensor models. However, their performance does not scale well with the number of robots in the simulation [11, 18]. Besides, they are not designed to easily run batteries of simulations with different multi-robot scenarios in a straightforward fashion. We tried to tackle these issues with our simulator, designing it with the following goals in mind:

- **Usability:** Implementing and testing new multi-robot algorithms should be as easy as possible. Accessing sensors or commands to move robots should require no more than a single line of code.
- **Scalability:** The simulator should be able to support large numbers of robots executing complex algorithms simultaneously. No circumstances (e.g., low CPU availability or slow responses due to pagination effects) should jeopardize the results of such simulations.
- **Flexibility:** A wide range of different robot types (with varying dimensions, dynamics and sensor configurations) should be supported. Partial realism in terms of sensor noise and dynamics should also be provided.
- **Autonomy:** No user intervention should be required when running batteries of simulations (e. g., for Monte Carlo experiments). Errors and exceptions should be handled and saved, allowing a posterior interpretation by the user. Punctual error codes should not stop the execution of the full battery.
- **Adaptability:** Users should be able to adapt MiMicS to fit their needs.

3 ARCHITECTURE OF MIMICS

MiMicS was developed over the framework MatLab (Matrix Laboratory) produced by The Mathworks, Inc.[10] MatLab provides its own script language, qualified as a forth-generation programming language (4GL). Thus, MatLab is designed to reduce development efforts for software engineers, thanks to a high level of abstraction. In the context of MiMicS, this eases the early stages of algorithm prototyping, allowing the developer to focus more on the algorithm's core concepts and less on its implementation. Furthermore, basing the simulator on MatLab is a major advantage when it is to be used for educational purposes: many universities already use MatLab in their computer science and engineering courses. Students can advance their knowledge in robotics in an environment

[1] The current version of MiMicS can be downloaded from https://drive.google.com/open?id=0B3gF5g1g0UuNdWMwbTFyQzBWZ3M
[2] http://www.cs.cmu.edu/~trb/TeamBots/
[3] http://www.cyberbotics.com/
[4] https://sourceforge.net/projects/usarsim/
[5] https://sourceforge.net/projects/delta3d/
[6] www.microsoft.com/robotics/
[7] http://www.coppeliarobotics.com/
[8] http://gazebosim.org/
[9] http://wiki.ros.org/stage
[10] https://www.mathworks.com/

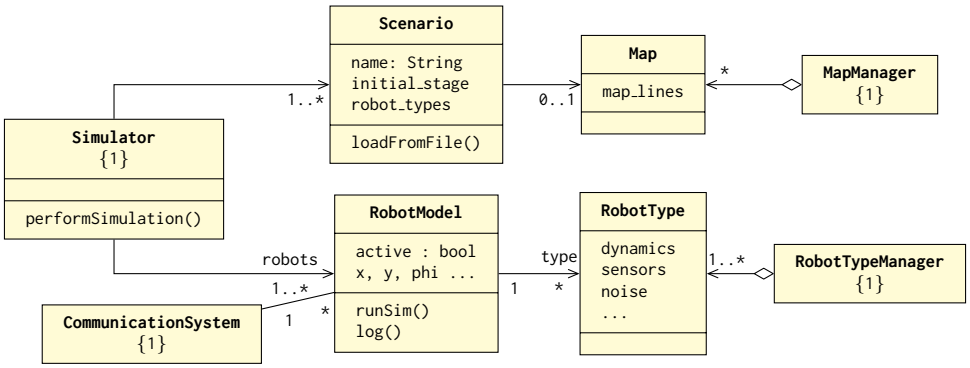

Figure 1: Overview of the architecture of MiMicS.

they are already familiar with, that is fully portable and that is easy to install and maintain. Moreover, the abstraction provided speeds up the time required from the students to complete tasks; allowing larger algorithm reviews in the constrained teaching hours.

The general architecture of MiMicS, graphically depicted in Figure 1, is based on the object-oriented designing tools included in MatLab. Specifically, multi-robot simulations consist of running multiple instantiations of a class called *RobotModel*. This class is in charge of tracking the necessary status of each individual robot, at the time that it interacts with other *RobotModel* objects to detect possible collisions between them or with any obstacle in the scenario.

When the main class *Simulator* is initialized, a world configuration file (a *Scenario*) is loaded, defining static obstacles, robot's starting positions, types and goals. Furthermore, the types of robots used for this scenario (physical dimensions, dynamic behavior or noise characteristics of the sensors) are loaded into instances of the *RobotType* class. Then, a set of unique *RobotModel* objects is created, with each robot possessing a link to its corresponding type. The simulation itself follows a simple step-to-step schema. For each robot, the following routine is executed:

(1) Locate obstacles with a continuous ray-tracing algorithm around the robot. Feed with this information the collision detector and the ranger finder.
(2) Execute the user's code to be tested (e. g., an algorithm for collision avoidance or exploration). This code has – via the *RobotModel* object – access to the robot's status, the ranger scans, the *CommunicationSystem* and the static world map. It should output two actuation variables: the speed and orientation references. Exceptions thrown by the user's code are recorded and handled.
(3) Calculate the robot's movement by feeding the references to two in-build PID controllers and considering the dynamics given by the robot's type.
(4) Log the robot's current status into a file.
(5) Deactivate the robot if it reached its goal, experienced a collision, escaped from the map or got trapped in a permanent deadlock situation.

When each robot's new position is known, several criteria indicating are checked to determine the end of the simulation. These criteria include full robot team deactivations, the occurrence of exceptions, the violation of an overestimated time limit for the simulation, etc. If no criterion applies, the simulation evolves and the loop continues. A plot that visualizes the simulation's progress can be updated in between simulation steps. After the simulation ends, a summary file is created and MiMicS can advance to the next simulation in the queue.

With this procedure, large batteries of simulations can be executed without intervention during nights or weekends, even when errors arise due to the user code (during step 2 of the routine described above). If this occurs, MiMicS records the error message, gracefully terminates the simulation and seamlessly continues with the rest of the battery. Later on, users can re-execute the erroneous simulations and determine if the errors are isolated or if the whole battery has to be simulated again.

This feature distinguishes our solution from other commonly used simulators such as Stage or Gazebo, which do not signal the end of the simulation. With those systems, in order to automatically process a batch of simulations, users would be required to set up scripts. Such scripts typically abort a simulation after an a priori guessed amount of time and relaunch the simulator with a new configuration. The selection of the appropriate time to terminate simulations can be specially tricky, since short times can abort simulations too early and long times waste computing time. More complex scripts can be programed to determine the right time to stop a simulation, but they can hardly detect nor record failures during the execution.

The following subsections describe some of the main features of the simulator in more detail, such as the dynamic models for the robots or the sensors. A full, detailed documentation is included in MiMicS. It also contains a number of code examples in order to support beginners with implementing their first algorithms.

3.1 Robot dynamics

MiMicS simulates 2D multi-robot scenarios. More specifically, all robots are assumed to be rectangles with no mobile parts. Moreover, we assume *unicycle* models, i.e., the linear speed always coincides with the direction of the robot orientation and all robots can rotate in place (differential drive). Making use MatLab's *Dynamic System Toolbox* combined with Laplace model representations, the robots are enhanced with translational and rotational inertia and second-order dynamics.

Equation 1 represents the relationship between the position on the bidimensional plane (x, y), the orientation θ and the linear and angular speeds of the robot, represented by v and w respectively.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \frac{-1}{\tau_v} & 0 \\ 0 & \frac{-1}{\tau_\omega} \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{k_v}{\tau_v} & 0 \\ 0 & \frac{k_\omega}{\tau_\omega} \end{bmatrix} \cdot \begin{bmatrix} u_v \\ u_\omega \end{bmatrix} \quad (2)$$

Equation 2 models the relationship between the velocities and the actuators of the robot. In the case of the linear velocity v, its representation depends on the specific actuator u_v. This variable, limited between -1 and +1 represents the percentage of energy that an electrical actuator can provide over the two parallel motors that makes the robot advance (differential robots). The angular velocity ω is also dependent on u_ω. That variable is limited between -1 and +1 and represents the amount of current that is derived to the left motor, in contrast to the right motor. The parameters k_v and k_ω model the saturation of the motors while achieving maximal speeds; while τ_v and τ_ω determine how fast the robots can accelerate.

Since every robot is under the influence of inertia dynamics, each robot includes two PID controllers that close the loop in orientation and speed. First, this eases the task of the programmers when designing new code: simple speed and orientation references can drive the robots. Second, this forces users to work under realistic dynamics from the earliest stage of the design.

The range of types of robots that can be simulated in MiMicS is limited by the above model (e. g., skid-steer robots). However, this simplifies kinematics, allowing the simulator to deal with a larger amount of robots while still considering acceleration constraints.

3.2 Range-finder sensors and collisions

In order to detect and avoid collisions, many real robots are equipped with a range-finder sensor. It is defined by a field of view fov_r, a number of beams and range of detection r_r. These parameters are part of the *RobotType* class. Using them, an array of so-called d-beams is created.

Robots and other obstacles in the map are defined as sets of simple segments, so-called l-obstacles. For instance, a robot is represented by a set of four l-obstacles. At each iteration, every d-beam is evaluated against all l-obstacles as if they were infinite lines, calculating intersections. This operation can result in three different outputs:

- No intersections. That l-obstacle is ignored.
- The intersection is a single point. The system checks whether that intersection is within the limits of the l-obstacle and range of the sensor. If so, the d-beam is shortened until the intersection to simulate that measurement.
- Both lines are coincident. The two extremes of the l-obstacle are evaluated along the d-beam. If the closer one is within the range of the sensor, the d-beam is shortened until that point to simulate that measurement.

This solution ensures a high level of precision while computing the simulated range-finder, with a fair computational load. However, the performance is dependent on the number of segments that represent the scenario, and not in the size of the scenario. This makes large maps with simple straight walls computationally cheap, but little scenarios with multiple vertices can become expensive.

If activated, MiMicS can corrupt each beam of the ranger with Gaussian noise of zero mean and σ_r variance. This responds to the necessity of simulating noisy rangers like sonar arrays.

To detect collisions between robots, a similar system is used. Each robot defines its perimeter as a set of d-beams, and makes the detection range r_r coincide with the longitudes of the vehicle. Each beam is evaluated against all l-obstacles. If an object is met, a collision is detected.

3.3 Positioning and other sensors

Each *RobotModel* object also includes a model of a positioning sensor to measure the position, velocity and acceleration of the robot. The class *CommunicationSystem* provides a matrix with this information for all existing robots in the simulation. This corresponds to many paradigms where the robot's position is assumed to be known and the information from the others is expected to be shared.

As the ranger sensor, the positioning systems can be noisy. Thus, when a robot is defined, the noise associated with its position, velocity and acceleration measurements is modeled as a Gaussian with zero mean and σ_{pos}, σ_{vel} and σ_{acc} variances, respectively. Users are encouraged to base their code on these noisy measurements, but the ground-truth information is still accessible.

Moreover, MiMicS includes an odometer sensor for each robot. A simple Gaussian noise with zero mean and standard deviations σ_{pos}^{odom} and σ_{θ}^{odom}, for translation and rotation, respectively. As for real robots, such noise only applies when robot translates or rotates (in an independent way). Positions are simple integrations of translations and rotation differences, therefore noise is inherited on each step.

Apart from the positioning systems and the ranger sensors, MiMicS does not simulate any other sensors explicitly. This is done to keep a low computational cost of the simulations with large teams of robots. Nonetheless, the ranger model covers a wide variety of robot sensors. Moreover, the positioning information included in the matrix that the robots share could be used to implement other sensors, such as those based on Received Signal Strength Indicator (RSSI) or bearing-based sensors.

3.4 Scalability

ROS and most simulators based on its architecture are optimized to work in multi-robot environments, where each robot usually has a dedicated computer to run its own algorithms. When we move it to simulated environments, all algorithms executed on each simulated robot are usually thrown simultaneously to the same CPU.

Moreover, the computational capacity is also shared with the simulator that emulates the dynamics of the robots themselves. Simulated algorithms, running many threads in parallel in a single computer could slow down the performance due to continuous context switching. In that case, the algorithms on board the robots would not be able to access the same computational resources as originally, and they may be simulated in a slower manner.

MiMicS tries to alleviate that issue by allocating the full computational resources from the CPU to each individual robot routine,

minimizing the context switching within each simulation step. For instance, if two robots are executed on the simulator, the whole CPU is assigned to the user's code running on the first robot, then it is used to emulate the physics of that robot, and later it is assigned to the second robot. This induces several advantages for the simulator:

(1) Each algorithm is executed in a virtually isolated computer. When a user codes hardware multi-thread solutions, simulations with n robots do not create n multi-core executions of the code running simultaneously. If this were done, the context switching between the different processes would invalidate the memory hierarchy of the system, producing cache misses; at the time that the inter-thread communication will decelerate the execution, making every single process run slower than when executed separately.
(2) It ensures that the physical simulated world does never run faster than the algorithms to test. When the physics model is set to run as fast as possible, it may happen that the time simulating the world overtakes the time that an algorithm requires to be executed. Our solution ensures that if the physics are fast, they will always wait for the algorithm to end its loop.
(3) If the user needs to measure the execution time of a single instance of her/his code, MiMicS will measure it truthfully and log it. There will not be parallel instances of the same code running and interfering the measurement.
(4) Since the simulation time will never increase until each robot routine is done, the simulator can assume as many simulated robots as necessary, making the final simulator scalable.

3.5 Performance

Since the simulator works over MatLab, its performance is limited to the advantages and disadvantages of the interpreted language. On the one hand, the level of abstraction that the language provides makes new code easy to develop and debug. At the same time, MatLab allows for the usage of other languages like C, C++, Fortran, Java or Python. It also provides portability, running on Windows, Linux and Mac OS.

On the other hand, the MatLab interpreted language makes by default the general performance of the simulator slower. To fight against big delays on this context, MiMicS implements critical functions like the raytracer with optimized C++ code. Thereby on the very first execution of MiMicS, all those functions are automatically compiled on the computer, speeding up the performance and reaching execution times close to fully compiled solutions.

MiMicS' performance may benefit from parallelization if many CPUs and MatLab's *Distributed Computing Toolbox* are available. In this situation, the user can opt to distribute the load of executing the individual robot's routines (ranger scan, algorithm execution and movement simulation) to different CPUs, each running an independent MatLab client. The results are collected by a controlling workload server before the subsequent steps (mainly, checking if the simulation should end) are performed, ensuring the validity of the advantages 2-4 described in Section 3.4. We found that this method reduces the overall running time of the simulation only when a relatively high number of robots is present, since MatLab's parallelization mechanisms exhibit a significant overhead.

3.6 MatLab-based scenarios

When the amount of robots to simulate is large, a simple action like moving all the initial positions of all robots $0.2m$ to the left can become a great waste of time. Therefore, MiMicS defines scenarios through matrices formatted with MatLab code. This saves the user time testing minimal differences between similar scenarios and it helps when adding complex degrees of randomness to the configurations.

Listing 1 presents a simple scenario example with 1000 robots. Note that if necessary, the placement of this amount of robots can be reduced to just 3 lines of code.

```
rng(5);      % Forcing a random-predictable result
x_offset = 0.2; %Allow general change of positions

% Defining 1000 robots on a line with:
%    - Separation of 1m between robots
%    - Random orientations on the origin
%    - Separated 20m to its destiny
for i = 1:1000
    initial_stage(i,:) = [ ...
      0.0+x_offset     1.0*i  ... % x-orig, y-orig
      2*pi*rand()-pi   0.0    ... % yaw-orig, v-orig
      20.0             1.0*i];   % x-goal, y-goal
end
```

Listing 1: Scenario definition file.

Static obstacles are also defined making use of well defined matrices that can be written by the users. The scenarios are defined by lines going from a position A to a position B. On the one hand, this eases the inclusion of the static scenario on the raytrace paradigm. On the other hand, although most environments can be represented by few or many straight lines, scenarios with many objects could be hard to encode. In order to mitigate this issue, any open-source vector graphics suite could be used to vectorize images representing maps.

3.7 User-friendly interface

Most simulators relegate some tasks to the user, like defining new scenarios or maps, or loading and executing the necessary files to test for a specific simulation. MiMicS provides the user with friendly graphical interfaces that guide her/him throughout the process:

(1) An editor is available to help the user with the creation or modification of scenarios, avoiding tedious coding for simple configurations.
(2) Similarly, a map editor is provided.
(3) In order to set up a battery of simulations, MiMicS can display the list of available algorithms and available scenarios. After choosing, each selected algorithm is simulated with each selected scenario. This eases the extensive testing of the algorithms over a multitude of scenarios, disclosing potential problems hard to find. Furthermore, this process allows comparisons between different algorithms or different versions of the same algorithm.
(4) The logs of finished simulations can be loaded, replayed and converted to videos.

Intensive Monte Carlo testing can take long computational times. Nonetheless, MiMicS is thought to work in background. One can configure a large simulation battery and run it over a secondary computer without external intervention. Besides, MiMicS implements a command-line interface which shortcuts the aforementioned options, for instance to jump directly to the simulation(s) the user specified (and saved) during a previous usage of the simulator.

3.8 Log files

In order to analyze, reproduce or display simulations, MiMicS creates human-readable and machine-readable log files. Each robot tracks its status into its own log file which includes performance dates, final status of the robot, and variables like positions, orientations or speeds. Listing 2 shows a part of one of those log files. A separate file summarizes the simulation's duration, the number of robots which reached their goals and other general information.

```
scenario = MyScenario
algorithm = example
creation_time = 01-Aug-2017 16:53:42
log_uuid = 904b074c-9e7d-4fc9-b108-086258bf8177
robot_id = 1
type = Pioneer3AT (default)
vehicle_width = 0.5
vehicle_length = 0.7

time;x_gt;y_gt;phi_gt;v_ref;phi_ref;user_msg;sys_msg
 [s]; [m]; [m]; [rad];[m/s];[rad/s];       [];      []
   0;   -3;   0;      0;     0;      0;        ;
 0.1;-2.98914;0;      0;     1;      0;        ;
 0.2;-2.9602;-0.000162143;-0.0112036;1;-0.0060328;;
 0.3;-2.91757;-0.000604075;-0.00952897;1;0.00397832;;
[...]
 3.1;-0.280864;0.0397515;0.0513278;1;-0.0124653;;
      Deactivation: Collision with robot 2
```

Listing 2: Excerpt of a robot's log file.

MiMicS pays special attention to deadlock and livelock situations (e.g., to test collision avoidance algorithms). Therefore, a top-tier log file is generated specifying the amount of collisions, deadlocks and livelocks that occurred during each simulation for the specified scenarios and with the specified algorithms. Stuck robots with no chance to move are considered to be into a deadlock, whereas robots moving along endless trajectories without reaching their goals are into livelocks. This log helps the user to quickly identify the simulations of the battery that did not behave as expected.

3.9 Integration with other platforms

MatLab is capable of interfacing with other languages like C/C++ or Python, but the interfaces provided are not intuitive. However, students may prefer to stick to a language they already know rather than to learn MatLab. For those reasons, we included code examples as a reference for users that want to use those languages for their algorithm implementations. They demonstrate the proper usage of the MatLab interfaces that can call C/C++ or Python code in the context of MiMicS.

When MatLab's *Robotics System Toolbox* is available, ROS [1] and MiMicS can be used in conjunction. This may ease the transition of the user's code from or to ROS. We successfully performed simulations in which the core parts of an algorithm were executed in ROS nodes communicating with MiMicS via ROS messages. The nodes received the robot's status, the map (converted to an occupancy grid map) as well as the ranger scan and sent calculated orientation and speed references back to MiMicS. The *tf2* package was used to perform coordinate frame transformations on the MatLab side.

4 SCALABILITY EXPERIMENTS

The key point of MiMicS is to provide the researcher easy tools to develop and evaluate new algorithms and concepts in a systematic manner. With that in mind, only few improvements to increase the speed and efficiency of the simulator have been made. However, it is important to remark that a simulator that is not able to run faster than a real experiment can put off the creation process. If that is the case, the time that the user saves while creating a new algorithm, will be wasted during the testing process.

Therefore, in order to test the scalability and the speed of the simulator, we created a simple scenario, similar to the one specified in Section 3.6. An increasing number of robots placed over the y-axis where commanded to reach in a straightforward motion a goal placed $20m$ away. Each robot was modeled with a version of the well-known Pioneer-3AT, a $0.7 \times 0.5m$ robot with $k_v = 1.0$, $k_w = 1.0$, $\tau_v = 0.5$ and $\tau_w = 0.2$. The robot range-finders were simulated with a simplified version of the Hokuyo UTM-30LX, where the amount of beams were abridged to 270, but the field of view of 270 degrees and the $30m$ beam length were kept. No noise in the system was added. A simple algorithm to drive the robots to their goals was used for the simulation. The tests were performed with two different computers, whose features are depicted in Table 1.

	i7	Duo 2
CPU	Intel Core i7-5500u 4× 2.4 GHz	Intel Core 2 Duo E8400 2× 3.0 GHz
GPU	Intel HD Graphics 5500	ATI Radeon HD 2400 Pro
RAM	8 GB DDR3	4 GB DDR2
OS	Ubuntu 16.04 LTS (64bit)	Windows 7 (32bit)

Table 1: Features of the computers used for the experiments.

Since the visualization of the simulation produces a deep impact on the execution times, three different representation times for the system were used: one for each simulation step ($T = 0.1s$), one for each simulated second and one each 10 simulated seconds. The first representation time allows the user to track in a detailed way the simulation; the second one still allows a good track but saves time; and the third one can help giving an intuition of the general evolution of an algorithm.

In Figure 2, the average running time required to perform 30 simulations is represented on the y-axis. Each of those simulations emulates 21.8 simulated seconds. The x-axis represents the increasing number of robots, going from 1 to 60. As it is possible to see, the time that a simulation requires to be performed increases linearly with respect to the number of robots in the simulation. This desired effect is not always present in nowadays multi-robot simulators [18]. For larger numbers of robots, a steeper slope is

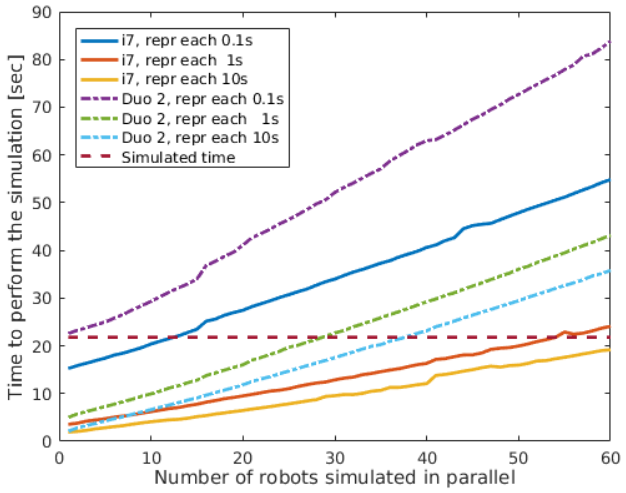

Figure 2: Averaged simulation time for a simple scenario with increasing number of robots. Results for two computers and three different configurations are shown. The real-time for the simulation should be 21.8 seconds.

Figure 3: Benchmarking scenarios at the second 600 of two simulations. On top, the *Cave* with 100 robots, at the bottom, the *Hospital* with 2000 robots.

appreciated, caused probably due to paging memory effects. The shorter the representation time, the higher the offset introduced in the simulation time. Moreover, as it was expected, the difference between using an old computer and a newer one mostly impacts the gradient of the simulation time.

Note that to plot this graphic, each computer performed 5400 simulations, requiring a total of 28.92 hours of computation for the i7 and 46.81 hours for the Duo 2. However, each simulation battery was performed with a single human intervention, demonstrating MiMicS capability to run large experimental sets during long periods of time.

5 BENCHMARKING EXPERIMENTS

In order to further assess the performance of MiMicS, we evaluated it against the benchmarking scenarios proposed in [16] (see Figure 3), where the multi-robot scenarios known as the *Cave* and the *Hospital* are executed with a simple distributed deployment algorithm.

5.1 Comparison against Stage

In order to compare the Stage simulator with MiMicS, the *Cave* scenario was selected. More specifically, the performance of the newest version of Stage (one of the nowadays most used simulators) running on ROS Kinetic, was compared against MiMicS running on MatLab R2016a; both on the i7 computer.

The *Cave* is a scenario of $16 \times 16m$ that models a moderately constrained environment with a population of 100 robots originally placed between the north and the west sides of the map. The robot model for the simulation was the Pioneer-2DX; a $0.325 \times 0.27m$ robot with $k_v = 0.3$, $k_w = 1.0$, $\tau_v = 0.05$ and $\tau_w = 0.2$ without noise. All robots executed a deployment algorithm making use of a simulated Sick LMS200 laser with $8m$ range, 180° field of view and 180 beams. We executed that scenario 30 times for each simulator over 600 simulated seconds. Both graphical interfaces were set to repaint after each simulated second.

The benchmarking scenario required in Stage a mean of 94.37 ± 4.88 seconds to be executed, being able to run at $6.35 s_{sim}/s$. MiMicS required 919.50 ± 15.33 seconds[11], $0.65 s_{sim}/s$. However, each simulation on Stage required the periodic intervention of the user, closing and relaunching the simulator. Taking all into account the experiment took more than 1 hour, on which the user could not deeply focus on another task. Conversely, MiMicS lasted 7.6 hours on a row, but the simulation battery was launched and left running during night hours. The time required from the user to perform such task lasted less than 10 minutes. Moreover, the post analysis of the battery showed that no collisions occurred during any of the 30 experiments, while Stage required a visual review of all 100 robots on each simulation.

Measuring the number of lines of code, Cloc[12] demonstrated that the C++ implementation of the code required 133 lines, while same code implemented on MatLab only required 54. This demonstrates that even if the simulation in MiMicS is slow compared to Stage, the time spent by the user to program and test similar solutions on it is remarkably faster. Moreover, assuming similar scalability performance in Stage and MiMicS (Section 4); a decrease on the number of simulated robots will only speed up the time to perform the experiments. The coding and user's time requirement from Stage will remain present under those circumstances. This demonstrates the advantages that MiMicS has while prototyping new algorithms.

[11] Video of the Simulation:
https://drive.google.com/open?id=0B3gF5g1g0UuNMGNqUllQRV94dm8
[12] http://cloc.sourceforge.net/

5.2 Scalability for huge teams of robots

To test the response of MiMicS against very large teams of robots, the *Hospital* scenario was selected. This challenging $140 \times 60m$ environment represents a side of the Hospital at Fort Sam Houston, San Antonio, Texas; and it is widely considered as a generic indoor scenario. It was populated with two sets of 5 rows and 200 cols of robots (2000 in total), each $0.1 \times 0.1m$ with similar dynamics and noises as in the aforementioned *Cave* experiment, placed across the corridor that connects the whole floor. As before, the experiment consisted of running the deployment algorithm for 600 simulated seconds.

Vaughan [16] originally executed this experiment to evaluate the performance of Stage using the robot server Player [7] as its back-end. He ran the C code of the algorithm within Stage itself, simulating 6 inaccurate sonar sensors per robot and completely disabled collision checks. In recent years, Player lost its popularity in the robotics community and has not been updated since 2010. Because of this, we tried to recreate the experiment (as in Section 5.1) with the more contemporary ROS framework serving as Stage's back-end. Following a common approach, each robot was to run the algorithm in its own ROS node. On our i7, we were not able to run this simulation. This was mainly due to the fact that ROS required the setup of 2×2000 sockets (robot's position and laser) for the node's communications. This, in conjunction with the sockets used by Stage to present all robot's messages made our computer incapable of handling the requested communication flow.

In the MiMicS version of the experiment, we aggravated the challenge by again simulating the LMS200 (with 2-meter range) for each robot instead of the 6 sonar sensors – i. e. 30 times more beams – and testing for collisions. The graphical interface was disabled. MiMicS was able to simulate 30 repetitions of the scenario in a mean time of 15.09 ± 0.018 hours[13] , achieving a speed of $0.011 s_{sim}/s$. Therefore, we can conclude that MiMicS can also tackle large-scale multi-robot simulations where ROS/Stage falls short on its default configuration.

6 CONCLUSIONS

This paper presented a novel multi-robot simulator called MiMicS. The simulator is based on MatLab and designed for teaching, rapid algorithm prototyping and evaluation. Contrary to other existing solutions, MiMicS is prepared to run extensive experiments without the intervention of the user during the execution, easing the configuration of large batteries of simulations. MiMicS also eases the creation of new multi-robot scenarios to test algorithms .

The paper described the main features of MiMicS and some examples on how to create scenarios or log files. Our experiments showed that running times scale well with the number of robots, growing linearly. Compared to Stage, MiMicS is slower in terms of running time, but more scalable, portable and simpler to create prototypes of new algorithms. In fact, its smooth learning curve makes it perfect for students with low knowledge in robotics, while the dynamics and sensor models included make it also valid for research purposes.

As future work, MiMicS will be extended to deal with 3D scenarios and with other robotic representation models. The exploitation of the parallelization of simulations with GPUs will also be further studied in order to enhance the simulator's performance.

[13] Video of the Simulation:
https://drive.google.com/open?id=0B3gF5g1g0UuNVFZqbGM3bENwMVE

REFERENCES

[1] S. Balakirsky and Z. Kootbally. Usarsim/ros: A combined framework for robotic control and simulation. In *Proceedings of the ASME 2012 International Symposium on Flexible Automation (ISFA 2012), St. Louis*, 2012.

[2] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. Usarsim: a robot simulator for research and education. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405. IEEE, 2007.

[3] R. A. Castillo, O. G. Rubiano, and G. A. Vargas. Cooperative robotic system simulation in webots. *International Journal of Applied Engineering Research*, 11(15):8714–8720, 2016.

[4] J. S. Cepeda, L. Chaimowicz, and R. Soto. Exploring microsoft robotics studio as a mechanism for service-oriented robotics. In *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American*, pages 7–12. IEEE, 2010.

[5] J. Craighead, R. Murphy, J. Burke, and B. Goldiez. A survey of commercial & open source unmanned vehicle simulators. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 852–857. IEEE, 2007.

[6] C. Georgiades, M. Nahon, and M. Buehler. Simulation of an underwater hexapod robot. *Ocean Engineering*, 36(1):39–47, 2009.

[7] B. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.

[8] A. Harris and J. M. Conrad. Survey of popular robotics simulators, frameworks, and toolkits. In *Southeastcon, 2011 Proceedings of IEEE*, pages 243–249. IEEE, 2011.

[9] K. Hungerford, P. Dasgupta, and K. Guruprasad. Distributed, complete, multi-robot coverage of initially unknown environments using repartitioning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1453–1454. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[10] J. Jia, W. Chen, and Y. Xi. Design and implementation of an open autonomous mobile robot system. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1726–1731. IEEE, 2004.

[11] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[12] P. McDowell, R. Darken, J. Sullivan, and E. Johnson. Delta3d: a complete open source game and simulation engine for building military training systems. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 3(3):143–154, 2006.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[14] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, Nov 2013.

[15] M. Torres-Torriti, T. Arredondo, and P. Castillo-Pizarro. Survey and comparative study of free simulation software for mobile robots. *Robotica*, 34(04):791–822, 2016.

[16] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2-4):189–208, 2008.

[17] R. Veloso, Z. Kokkinogenis, L. S. Passos, G. Oliveira, R. J. Rossetti, and J. Gabriel. A platform for the design, simulation and development of quadcopter multi-agent systems. In *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, 2014.

[18] H. Yang and X. Wang. A case study on the performance of gazebo with multi-core cpus. In *International Conference on Intelligent Robotics and Applications*, pages 671–682. Springer, 2017.