Smart Teams: Simulating Large Robotic Swarms in Vast Environments *

Stephan Arens[1], Alexander Buss[1], Helena Deck[1], Miroslaw Dynia[1,3], Matthias Fischer[1,4], Holger Hagedorn[1], Peter Isaak[1], Alexander Krieger[1], Jaroslaw Kutyłowski[1,2], Friedhelm Meyer auf der Heide[1], Viktor Nesterow[1], Adrian Ogierman[1], Jonas Schrieb[1], Boris Stobbe[1], Thomas Storm[1], and Henning Wachsmuth[1]

[1] Heinz Nixdorf Institute, University of Paderborn, Germany [4] `mafi@upb.de`
[2] International Graduate School of Dynamic Intelligent Systems
[3] DFG Graduate College Automatic Configuration in Open Systems

Summary. We consider the problem of exploring an unknown environment using a swarm of autonomous robots with collective behavior emerging from their local rules. Each robot has only a very restricted view on the environment which makes cooperation difficult. We introduce a software system which is capable of simulating a large number of such robots (e.g. 1000) on highly complex terrains with millions of obstacles. Its main purpose is to easily integrate and evaluate any kind of algorithm for controlling the robot behavior. The simulation may be observed in real-time via a visualization that displays both the individual and the collective progress of the robots. We present the system design, its main features and underlying concepts.

1 Introduction

The scenario where a team of exploring robots – we call it a *Smart Team* – has to organize itself in order to explore an unknown terrain and execute work is of high interest and importance. Practical applications include rescue expeditions to dangerous areas or expeditions into the oceans or to planets. To develop local, distributed algorithms for controlling robot behavior, a powerful simulator is needed. There are two challenges in creating such a simulator. First, the robots can only have restricted local knowledge about the global state of the system. This implies that the simulator has to provide robots with an abstracted view of the terrain, different for every robot. For a terrain with 1 million obstacles, each represented by four 32-bit float values

* Partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS) and by the DFG Priority Programme 1183: Organic Computing, project "Smart Teams: Local, Distributed Strategies for Self-Organizing Robotic Exploration Teams".

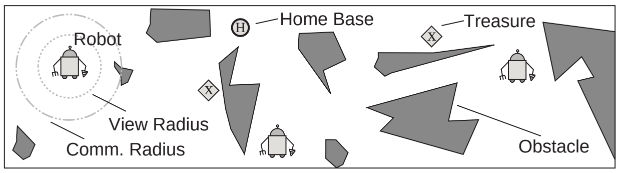


Fig. 1: The map with its important elements

(x-/y-position, length, width), this would result in ca. 15 GB data for 1000 robots, all needed to be held in memory. Current simulators are not able to tackle this demand. Very space-efficient data structures that provide efficient updates and queries are required. The second challenge lies in allowing rapid prototyping of strategies. This can be ensured by designing a simple interface for plugging in strategies and providing these strategies with a framework that can handle routine tasks.

We present a software simulator which allows to test arbitrary strategies for a large number of robots on terrains with dimensions ten orders of magnitude greater than the size of a robot and consisting of over a million obstacles.

1.1 Related Work

Several so-called multi-robot simulators have already been developed. *Stage* and *Gazebo* (see [6, 4]) are constructed to simulate mobile robots on a low level of abstraction, in case of Gazebo even with a realistic simulation of rigid-body physics. Even though Stage is designed to provide robots with a simpler view of the environment than Gazebo, it still focuses on physical properties of sensors. This high focus on realism and details conflicts with our algorithmic goals. Moreover, published results on the performance of Stage do not provide any information on its handling of complex terrain. Similar simulators have been developed for special hardware and different purposes (e.g. *Swarmbot3D* in [3] and *TeamBots* in [1]). A related category are multi-agent systems. The *Breve* simulator [5] focuses on the physical properties of a solid body. *GeoGraph 3D* [2] simulates moving agents on a realistic geographical map. However, multi-agent systems are not specifically designed for robot simulations which increases implementation efforts to adapt our model. To conclude, no system is currently able to meet all of our requirements, especially concerning the number of heterogeneous robots, their local views and the map complexity.

2 Required Model and Features of the System

The model's main elements are depicted in Fig. 1. We assume the *map* to be a two-dimensional rectangular cutout of a plane. Numerous arbitrarily shaped *obstacles* are spread over it, designating areas robots cannot enter. In addition, there is a single *home base*, the starting point for all robots. In contrast to those static elements, *treasures* represent dynamic objects that can be excavated and transported by *robots*. Treasures can be located at various positions on the map in different quantities. Robots may freely move within the accessible areas. The robot's *energy* decreases with each performed action but it can be recharged at the home base. Running out of energy disables a robot for the rest of the simulation. Robots perceive information about the environment within a *view radius* and are able to exchange information in a *communication radius*. Different *robot types* exist, each with certain restrictions. For instance, a transporter cannot excavate but only carry treasures.

Each robot has to obtain information locally and create its own representation of the environment. Shared knowledge is only achieved by explicitly exchanging information via communication. All robots act autonomously, that is, each robot has an individual *strategy* that defines its behavior. Their main goals are to explore the whole map, to excavate all treasures and to transport them to the home base.

2.1 Features of the simulator

To realize the model we have developed a highly configurable and extensible simulator in Java. It runs on a typical single workstation from 2007, giving researchers the advantage to use it in almost any situation without special hardware requirements. Our simulator allows arbitrary maps of the size of 10,000 km^2 consisting of more than 1 million obstacles, each with a minimum size of 1 m^2. Up to 1,000 robots, each having a diameter of 25 cm, can move on it to any continuous position. The simulation runs in discrete time steps (round based). However, simultaneity is achieved by providing all robots with updated data from the beginning of a time step.

Using an editor, maps may be created from scratch or with specific patterns (e.g. white noise, fractals). The number of robots as well as their abilities can be easily configured via an XML file. New strategies can be written in Java and are then dynamically invoked by the simulator. Moreover, a set of extra features is accessible to developers like an abstraction of the environment as a planar graph (for the use of graph algorithms), automatic collision avoidance, group building or energy management. All the above enables rapid prototyping. Finally, a real-time visualization allows the user to continuously observe the whole simulated environment in 2D or 3D. It may run on other computers to save memory and processor time. Additionally, real-time measurements can be evaluated, for instance the percentage of the explored map or of collected treasures. Representations like charts may display these statistics.

3 System Architecture

We developed our simulator on an architecture that includes four main components (see Fig. 2 left). As the system's core, the *Kernel* coordinates the

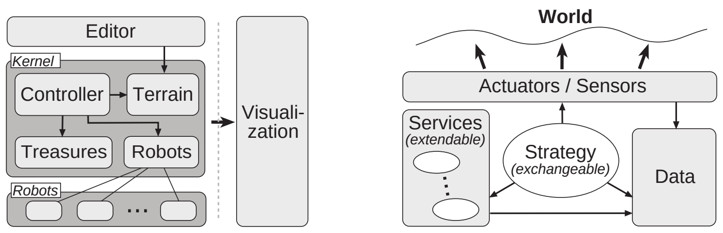


Fig. 2: *Left:* Architecture of the system *Right:* Architecture of a single robot

Robots maintaining all data and the strategy of the single robots (Sect. 3.1). The *Visualization* (Sect. 3.2) enables users to keep track of the simulation field and, finally, the *Editor* allows comfortable editing of the simulation input.

Inside the Kernel, a *Controller* executes simulation steps and updates the data structures for robots, treasures and the terrain (discussed in Sect. 4.1) with the data emerging from the robots' actions. Moreover, the Controller is responsible for establishing and controlling the communication between the Kernel and the Visualization.

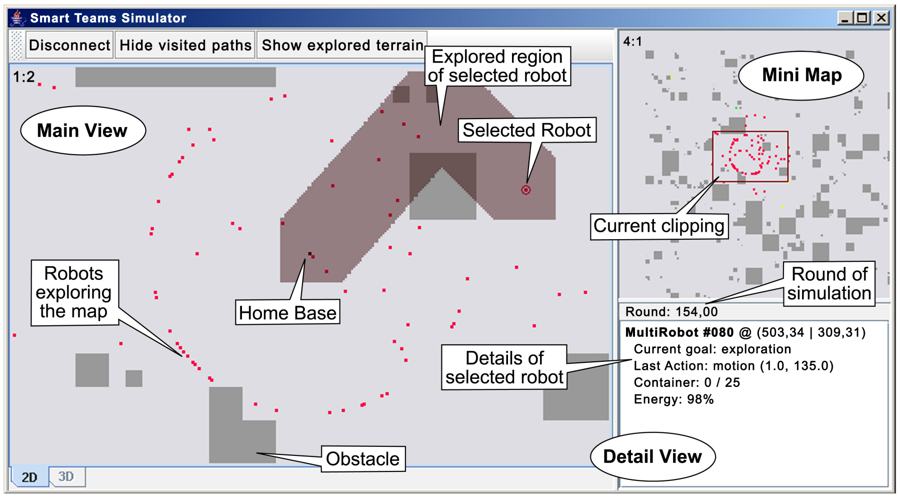
3.1 Robot

The architecture of a single robot (see Fig. 2 right) is based on an unmodifiable physical unit that interacts with its environment via actuators and sensors. We thereby simulate the physical conditions of the robot. An exchangeable strategy controls the robot's behavior. It has access to a data module storing all the relevant information and a tactics module providing a predefined, extendable set of local tactics.

The separation between the robot's physical unit and its strategy enables us to guarantee that a robot never acts incorrectly while executing its main loop. The overall process distinguishes between an evaluation, a goal formation and an execution stage. At the beginning of each round, every robot receives messages and perceives information about robots, treasures and obstacles in its local surroundings. For simplicity reasons, robots are equipped with a 360° view. After analyzing the data, the robot's strategy decides whether to pursue its current goal or to form a new one. Besides, messages may be broadcasted or sent to specific robots within the communication radius. To allow deterministic behavior, sending takes uniform time. At last, the elementary action to be executed next (e.g. "move") is automatically computed from the current goal. The physical unit checks whether that action is possible at the robot's position and, if not, prevents the robot from behaving wrong.

3.2 Visualization

A 2D and 3D real-time visualization allows the user to easily test strategies and to observe the robots' behavior. With its own independent data structures,



```
Round: 154,00
MultiRobot #080 @ (503,34 | 309,31)
  Current goal: exploration
  Last Action: motion (1.0, 135.0)
  Container: 0 / 25
  Energy: 98%
```

Fig. 3: The 2D visualization

it can be run from a different machine (and connected via RMI) to grant the actual simulation more memory. Fig. 3 depicts what the 2D visualization looks like. It allows the user to watch robot-specific data, e.g. the areas a robot has already seen. A detail view displays information like the last action of the currently selected robot or the remaining energy. To easily change the observed region a mini-map can be used, showing the clipping of the map the user zoomed in. Additionally, the Visualization may display statistical data on the performance of single or multiple robots as well as global values, e.g. the percentage of the currently explored map.

4 Efficient structures and methods

We need efficient data structures for representing terrain and robots. Most importantly, we must give the robots a local view on the map without storing it multiple times. Sect. 4.1 and 4.2 deal with these topics. We show how to provide a framework where arbitrary strategies can easily be built (Sect. 4.3).

4.1 Data structures for static and mobile objects

A typical terrain might consist of about a million obstacles. The corresponding map has to be held in memory as long as the simulation is running. Further, many range queries have to be answered very fast since each robot requests the obstacles in its view radius at least once in every round. For this, we use an implementation of the data structure *Region Quadtree* (based on [7]). It "bundles" obstacles not storing each of them separately and, additionally, obstacle-free areas occupy no memory. Therefore, its space complexity is very small in the average case. However, this also results in an approximation of non-rectangular obstacles by rectangles. Thus, the more winding the obstacles are, the more the possible amount of obstacles is reduced. We modified the quadtree by adding information to each node on its position and size. Thus, we can perform fast queries by rapidly deciding whether a subtree has to be traversed recursively. Given a map with ca. 750,000 quadtree leaves, distributed in a white-noise-manner, 30,000 random queries take less than a second on a typical workstation.

To store the large number of robots during the whole simulation in a way that queries can be answered quickly (e.g. the robots in a robot's view radius), we use two data structures, one which maps robot IDs to their corresponding robot instances and another one which maps the IDs to the robot's position. Though the theoretical running time is rather slow, it practically outperforms more sophisticated approaches.

Storing treasures is even more demanding as there may be much more treasures than robots. In every round, each robot asks for positions of treasures within its view radius, causing many queries in total. To handle this, a *PRQuadtree* is used as proposed in [7]. It allows to store treasures with low memory usage while keeping the response time for the queries low.

4.2 Maintenance of local maps

A challenging task in the development of the simulator was to find a way of handling the enormous amount of data resulting from up to 1000 different local views on the map. To avoid data overhead, we decided to give the robots direct, though restricted access to the Kernel's storage of the map.

Every robot only knows the regions of the map already explored by itself, in general. Therefore, a quadtree is used again that says "1" if a region is known and "0" otherwise. If a robot needs to have detailed information on an explored region, the appropriate obstacles are determined from the Kernel's map in background. The quadtree is efficiently updated after each robot motion, which may even reduce the memory needed when parts of explored regions are combined to larger ones. The example of Fig. 4 left supports our strong conjecture that the needed memory does not increase proportionally to the simulation time. Besides, it is linear in the number of robots and a single robot's need is merely a fraction of the Kernel's need (Fig. 4 right).

Quadtrees can easily be merged which allows to exchange partial maps via communication. Moreover, the quadtree helps solving the problem of enabling the use of graph algorithms (e.g. computing shortest paths). In the preprocessing of the simulation, a planar graph is computed representing possible paths on the map. Since Robots could only have created a graph based on their current knowledge, only graph nodes matching the quadtree are accessible.

Robots also have to manage dynamically changing data. That is, treasures and robots in the view and communication radius may possibly vary each round. However, it might, for instance, be useful to remember treasures seen earlier depending on the chosen strategy. For this purpose we have developed a scalable list that can efficiently store and update the last n seen treasures (or robots, respectively) or the treasures (robots) seen in the last m rounds.

Finally, strategies can enable or disable the different components of the data module with respect to their own needs. For example, an explorer is not interested in excavating treasures and, therefore, they are neither stored nor even determined at all. Using this approach, we reduce the amount of stored data and we decrease the time complexity of the evaluation stage.

4.3 Providing additional services

A *strategy* is the decision making process of a single robot, that is, an algorithm to control a robot's behavior. For instance, an explorer may have the strategy *Explore the Map with Breadth-First-Search.* A *tactic* is, on the contrary, a local decision or procedure that can be used by any strategy (e.g. the decision to join a team of robots or to continue moving on the original path).

Strategies and tactics can easily be implemented by deriving a class from the abstract class *Strategy* or *TacticProvider*, respectively. Different interfaces to control the robot or to obtain information about the environment can then automatically be used. Some build-in tactics are already available, including

- detection and avoidance of collisions with robots and obstacles,
- energy management that continually keeps a valid path to securely return to the home base when energy is low,
- automatic navigation along the boundaries of an obstacle,
- shortest path calculations in general, and
- team behavior with automatic grouping and breaking up.

A new strategy class may be assigned to one or more robots via an XML configuration file and is dynamically invoked at runtime. Consequently, no changes on the original code are needed. This applies for tactics as well. On the right is a short code example which uses all the above mentioned functions, illustrating the simplicity of implementing research results.

```
class MyStrategy extends Strategy
// initialize required tactics
IMovementTactic coll =
  tacticMgr.use("CollisionDetection");
// for each round
notifyTactics(); Treasure[] treasures =
  dynamicData.getTreasuresInViewRadius();
if(coll.hasAction()) coll.performAction();
else if(treasures.length > 0)
  actions.move(treasures[0].getPosition());
else actions.move(0.0, 1.0);
```

5 Current State and Future Work

The described functionality of our software system is mostly implemented. However, some parts are still under development. This primarily includes the

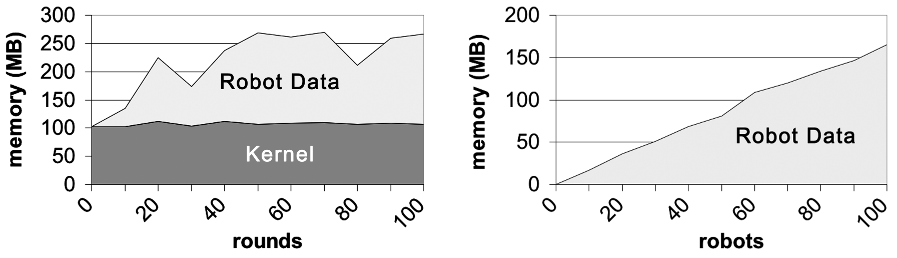


Fig. 4: Memory need for a map with 500,000 quadtree leaves. *Left:* 100 robots wrt. the number of rounds. *Right:* After 50 rounds wrt. the number of robots.

automatic generation of pattern-shaped maps, the 3D visualization and the measurement framework. We plan to simulate and render a terrain which may be compared to a Mars scenery or the Monument Valley: it is made up of arbitrarily shaped obstacles spread over a flat plane. Obstacles are small rocks and mesas of different size and topology, each modeled with up to 1000 triangles. So, for a million obstacles, we have to render up to 1 billion triangles in real time. We also still continue to develop more sophisticated approaches regarding tactics and basic strategies for robot actions. By now, our data structures and their related algorithms are able to run a simulation adequately for 1000 robots on maps represented by quadtrees with 1.3 million leaves. Nevertheless, we still enhance computation times and reduce the needed space since we seek to cope with even bigger maps. The simulator and further information can be obtained from the website `http://www.upb.de/cs/smartteams`.

References

1. T. Balch and A. Ram. Integrating robotics research with javabots. In *Working Notes of the AAAI 1998 Spring Symposium*, 1998.
2. C. Dibble and P. G. Feldman. The GeoGraph 3D Computational Laboratory: Network and Terrain Landscapes for RePast. *Journal of Artificial Societies and Social Simulation*, 7(1), 2004.
3. M. Dorigo, E. Tuci, R. Groß, V. Trianni, T. H. Labella, S. Nouyan, J. L. Deneubourg C. Ampatzis, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano, and L. M. Gambardella. The SWARM-BOTS Project. In *Proc. Swarm Robotics: SAB 2004*, volume 3342 of *LNCS*, pages 31–44, 2004.
4. B. P. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proc. International Conference on Advanced Robotics*, pages 317–323, 2003.
5. J. Klein. BREVE: a 3D Environment for the Simulation of Decentralized Systems and Artifcial Life. *Proc. Int. Conf. on Artificial life*, pages 329–334, 2002.
6. N. Koenig and A. Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *Proc. Intelligent Robots and Systems*, pages 2149–2154, 2004.
7. H. Samet. *Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.